REACHABILITY-BASED ROBUSTNESS VERIFICATION OF DEEP NEURAL NETWORKS WITH EMPHASIS ON SAFETY-CRITICAL TIME-SERIES APPLICATIONS

By

Neelanjana Pal

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical and Computer Engineering

May 10, 2024

Nashville, Tennessee

Approved:

Dr. Taylor Johnson, Ph.D.

Dr. Gabor Karsai, Ph.D.

Dr. Alan Peters, Ph.D.

Dr. Forrest Laine, Ph.D.

Dr. Hoang-Dung Tran, Ph.D.

To Maa and Baba, who imbued in me the importance of education, and to Dida, who had to stop her education for being a girl.
To all those who have experienced or are still experiencing challenges with their mental well-being; take your time and believe in yourself, trust me, everything will eventually come together in its own time.

ACKNOWLEDGMENTS

As I reflect on my PhD journey, words seem insufficient to capture the depth of my experience. This journey, it seems, was not merely a choice I made, but rather a destiny that beckoned to me. Throughout this path, I've faced a myriad of challenges and obstacles, each presenting its own test of will and determination. There were countless instances where surrendering felt like the easiest path, where the end of the road seemed just a step away. However, I found the strength to continue, a resilience that was fueled in no small part by the support and encouragement of so many individuals who appeared at critical junctures along my path. Their unwavering belief in me often reminded me of my capabilities, especially when my conviction wavered. They offered not just words of encouragement, but also practical support and understanding, which were invaluable in navigating the complexities and hurdles that marked this journey. To each and every one of them, I extend my deepest and most heartfelt gratitude, acknowledging that their support was a crucial element in helping me reach where I am today.

First and foremost, I am filled with profound gratitude towards my advisor and mentor, Dr. Taylor T. Johnson. His steady support and insightful guidance have been the cornerstone of my research endeavors and professional growth. Dr. Johnson is more than just an exceptional mentor; he is a staunch advocate for mental health and a proponent of maintaining a healthy work-life balance. His approach to mentorship goes beyond academic or professional realms; it encompasses a holistic concern for the well-being of his students. During a particularly challenging phase of my PhD journey, Dr. Johnson's unwavering belief in me was a beacon of support. In moments filled with doubt and uncertainty, his encouragement was the linchpin that enabled me to find my footing and persist. Dr. Johnson's exceptional mentorship and ability to recognize and nurture potential during difficult times speak volumes about his character and dedication. His influence has been transformative, leaving a lasting impact on my personal growth. His confidence in me was instrumental in helping me overcome challenges and emerge stronger, and for this, my gratitude to him is earnest and lasting.

I am particularly thankful to the members of my dissertation committee for their steadfast support and invaluable guidance throughout this process. Special mention must be made of Dr. Hoang-Dung Tran, whose assistance was pivotal both as a colleague and later as a mentor after his graduation. My gratitude also extends to Dr. Gabor Karsai, Dr. Alan Peters, Dr. Forrest Laine, and Dr. Hoang-Dung Tran for their roles on my Ph.D. committee. Their insightful discussions and constructive feedback greatly contributed to the development and refinement of this dissertation.

My journey would not have been the same without the invaluable assistance and encouragement from my colleagues, Preston, Diego, Serena, Patrick, and Xiaodong, at the veriViTAL Lab.

I also owe a heartfelt debt of gratitude to Dr. Shyamali Mukherjee and my friends in Nashville, Purboday, Anabil, Tonuka, Deepayan, Gourab, Soumita, Rudro, and Debjit. They have been a second family to me, making Nashville feel like a home away from home. Dr. Mukherjee, in particular, has been as close to me as a mother during my time in Nashville. Moreover, I'm grateful for the joy and serenity brought into my life by Eeshan, the adorable son of Soumita and Rudro. His playful antics, smiles, and hugs were the perfect antidote to stress, bringing lightness and happiness when it was most needed.

I would like to thank my professors from Jadavpur University, Dr. Smita Sadhu, Dr. Tapan Kumar Ghoshal, Dr. Sugato Munshi, and Dr. Ranjit Kumar Barai. Their mentorship during my final year project and thorough support during the Ph.D. application process were instrumental in my academic journey. Equally deserving of my thanks is my group of friends from Jadavpur University, Pritha, Achintya, Mrinmoy, Meghdoot, Koustav, Kunal, and Rudro. They have been my steadfast companions for the past 12 years and continue to be my strongest pillars of support. Additionally, I cherish my childhood friendship with Nabamita, with whom I share a uniquely special bond.

Lastly, but most crucially, I extend my heartfelt appreciation to my family, particularly my parents and grandparents, who have been the bedrock of my values and aspirations. They instilled in me the importance of education, even amidst the myriad of challenges they encountered in their own lives. My grandparents, uprooted during the tumultuous times of the partition and independence of British India, had to leave behind their homeland and start anew. My Dida (maternal grandmother), a bright scholar, had to forego her education due to societal norms and circumstances. My Maa encountered similar obstacles, with societal gender biases and financial limitations impeding her educational prospects despite her exceptional academic abilities. My

Baba, too, could not pursue further education due to financial hardships. Despite these personal sacrifices, they all emphasized the paramount importance of education in my life, instilling this value in me from my earliest memories. My brother also deserves heartfelt gratitude for his constant support. In moments of solitude and challenge, his presence and unwavering support provided me with the strength and guidance I needed. His role in my life has been that of a steadfast pillar, offering support and companionship when I felt most alone. Their collective sacrifices, struggles, and unwavering belief in the power of education have profoundly shaped my journey and aspirations.

Acknowledgement of Support

The material presented in this paper is based upon work supported by the National Science Foundation (NSF) through grant numbers 1910017, 2028001, 2220426, and 2220401, and the Defense Advanced Research Projects Agency (DARPA) under contract number FA8750-18-C-0089 and FA8750-23-C-0518, and the Air Force Office of Scientific Research (AFOSR) under contract number FA9550-22-1-0019 and FA9550-23-1-0135. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of AFOSR, DARPA, or NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

Introduction

1.1 Motivation

Deep neural networks (DNNs) have become a foundation of modern technological advancements, marking a significant shift in how complex problems across various domains are approached and solved. Today, DNNs are integral to a wide range of applications, from natural language processing (7; 188) in digital assistants to modern image recognition systems. They have the unique ability to learn from vast amounts of data, identifying patterns and insights far beyond human capabilities (87; 84; 91). This capacity has revolutionized industries, enabling automation and efficiency at unprecedented levels. In areas such as finance (44; 20; 111), healthcare (80; 135; 102), and entertainment (62; 122), DNNs have transformed traditional methodologies, leading to more informed decision-making processes and enhancing user experiences.

The application of DNNs in safety-critical systems has been both transformative and challenging. These systems, which include autonomous vehicles, medical diagnosis tools, and critical infrastructure management, rely heavily on the accuracy and reliability of DNNs. In these applications, the stakes are extraordinarily high, as any error or malfunction could lead to severe consequences, including risks to human life. For instance, in autonomous driving, DNNs must process and interpret complex sensory data to make real-time decisions, requiring an incredibly high level of precision and reliability; otherwise, there will be more mishaps like the Tesla, Uber, and Cruise crashes causing human lives (22; 50; 67). Similarly, in healthcare, DNNs used for diagnostic purposes must exhibit exceptional accuracy and robustness to ensure patient safety and effective treatment plans.

As the demands from various sectors grow, so does the complexity of DNN architectures. From basic neural networks, we have moved to more intricate and specialized ones designed to tackle case-specific problems. The transition from straightforward feedforward networks (FFNNs) to more sophisticated models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs), for instance, marks a significant progression in the discipline. These complex architectures allow for a more nuanced understanding and processing of data, be it in the form of images, speech, or sequential patterns.

Unfortunately, it has been seen that state-of-art well-trained networks can easily be fooled into resulting in erring predictions, by very small perturbations in the input (106; 152; 60). So the increasing reliance on DNNs in critical applications necessitates a robust framework for safety and formal verification. Safety verification involves ensuring that a system operates within the bounds of safety under all circumstances, while

formal verification is the process of proving or disproving the satisfiability of a certain formal specification or property, which translates to ensuring that the networks function as intended without unexpected or erroneous behaviors, especially in situations where failure could result in severe consequences.

Research in the field of neural network verification is diverse, reflecting the varied applications and complexities of DNNs. One direction involves stress-testing networks under various conditions to identify potential vulnerabilities or failure modes. The other focuses on developing methodologies for verifying the robustness of neural network outputs given specific inputs. Additionally, there is significant interest in creating frameworks that can formally verify the NNs against adversarial inputs. This multi-faceted research approach indicates the field's complexity and the critical importance of ensuring the safety and robustness of neural network applications.

Despite the progressions in DNN applications and verification techniques (168; 9; 26; 78; 104; 169), there remains a noticeable gap in the verification of NNs used in time-series applications. NN-based safety-critical time-series applications hold immense importance, especially considering the wide range of sectors they impact. While input perturbations in NNs have traditionally been associated with image-based networks, their extension to time series data and inputs with various noises underscores a broader, critical scope (40; 171). In the manufacturing industry, for instance, NNs play a pivotal role in providing invaluable insights for future operational strategies (133; 53). The significance of these applications extends beyond manufacturing, touching on vital areas; for example, the application of autoencoders in time-series data is particularly noteworthy. Autoencoders are instrumental in tasks like anomaly detection, feature extraction, and data compression, where they efficiently encode complex time-series data into a more manageable form for analysis and reconstruction. Additionally, time-series-based audio and video processing applications are gaining prominence, e.g., speech recognition, audio event detection, and video analytics. Due to their complexity and critical nature, these areas present unique challenges. In this 'not-so-explored' domain, NNs assist in managing and interpreting complex data, which often contains a range of errors, uncertainties, or environmental noise factors. The ability to accurately process this 'noisy' input data is crucial, as it directly impacts decision-making and operational efficiency in these critical fields. Therefore, ensuring the safety and robustness of NNs in handling time-series data is not just a technical necessity but a fundamental requirement for the advancement and safety of various industries and societal infrastructures. In these domains, the dynamic nature of time-series data demands a level of verification that accounts for temporal dependencies and rapid changes in data, a challenge that is yet to be fully addressed in current verification methodologies.

While this thesis primarily focuses on safety-critical time-series applications, it also delves into several experimental areas involving various data and network types. This includes investigating the safety verification and benchmarking of semantic segmentation datasets and networks. Additionally, the thesis examines

various adversarial attacks, analyzing their diverse impacts on NNs even when the extent of adversarial influence is consistent.

1.2 Research Challenges and Contributions

Therefore, the research problems addressed in this thesis emerge from two critical needs in the field of NN verification: (I) the need for robustness verification in time-series safety-critical application areas and (II) experimenting on other complex network architectures and layer types.

While the research to date has predominantly focused on the safety verification of NNs with image inputs in classification tasks, this study aims to shift the emphasis to other input data types as well as to non-classification models, e.g., time-series classification and semantic segmentation, among others. The work is developed as an extension of a pre-existing Neural Network Verification (NNV) tool. However, the scope of the work extends beyond various critical application areas to broaden the tool's capabilities, extending support to various layer types like Concatenation and Addition, thereby enabling verification of complex architectures such as U-Net, DenseNet, and ResNet.

The proposed contributions, addressing the dual needs of robust verification and support for complex architectures in the NNV tool, are comprehensively detailed in Chapters 3 to 8. For a better understanding, the structure of this thesis is divided into two distinct segments: (I) The first segment, encompassing the first four chapters (Chapters 3, 4, 5, and 6), concentrates on safety-critical time-series application studies. (II) The latter segment, comprising the next two chapters (Chapters 7 and 8), transitions to examining areas outside time-series applications, yet maintaining a focus on robustness verification.

1.2.1 Works Focused on Safety-Critical Time-Series Applications

1.2.1.1 Safety and Robustness Verification of Autoencoder-Based Regression Models [Chapter 3]

1.2.1.1.1 Research Challenge

Autoencoders, known for their ability to encode and decode data efficiently, play a crucial role in applications like anomaly detection, data compression, and generative tasks. However, adversarial noises can severely compromise their effectiveness, and such distortions can lead to faulty reconstructions or misinterpretations, posing risks in critical applications like medical imaging analysis or cybersecurity. This vulnerability underscores the need for rigorous verification protocols to ensure that autoencoders can withstand these adversarial perturbations and maintain accuracy and reliability in their outputs, thus making robustness verification not just beneficial but essential.

1.2.1.1.2 Research Contribution

This research introduces the first formal verification methodology for autoencoders, also marking the first of its kind for any time-series application. It involves a practical case study, a simulated regression-based autoencoder model that processes time-instanced signals from a device and potentially gets affected by spike faults. The analysis focuses on the reconstructed output compared to an uninterrupted input signal. This work contributes significantly to the field of autoencoder verification by checking whether the output of an autoencoder remains within a predetermined safe threshold relative to the corresponding uninterrupted input, particularly in the presence of specific faults. The effectiveness of this method is assessed using two distinct threshold values, offering insights into the robustness and reliability of autoencoder models in handling input perturbations.

1.2.1.2 Reachability Based Formal Verification of Time-Series Safety-Critical Applications in Prognostics and Health Management (PHM) [Chapter 4]

1.2.1.2.1 Research Challenge

Another significant challenge arises from input disturbances in time series data or signal inputs impacted by different intentional and unintentional noises, especially those encountered in predictive maintenance (40; 171) applications. One such use case is the manufacturing industry, where data from process systems, such as IoT (Internet-of-Things) sensors and industrial machines, are stored for future analysis (133; 53). Data analytics in this context provide insights and statistical information and can be used to diagnose past behavior (192; 98), and predict future behavior (150; 25; 93), maximizing industry production. This application is not only limited to manufacturing, but is also relevant in fields like healthcare digitalization (187; 158) and smart cities (149; 146). Noisy input data, here, refers to data containing errors, uncertainties, or disturbances caused by factors like sensor measurement errors and environmental variations among the potential noise sources.

1.2.1.2.2 Research Contribution

This research investigates a new case study on time-series-based NNs within two industrial predictive maintenance domains. A key aspect of the study involves employing star-set-based reachability methods to determine whether the output set's upper and lower bounds conform to the permissible bounds set by industrial guidelines. This work facilitates the formal analysis and verification of regression-based neural networks for time series data, utilizing sound and deterministic reachability methods, and is presented as an extension of the NNV tool. This research addresses the vital need for formal verification of time-series regression models, especially in Prognostics and Health Management (PHM), which is crucial in safety-critical systems where accurate fault prediction and anomaly detection are imperative.

1.2.1.3 Reachability Based Formal Verification of Time-Series Safety-Critical Applications in Audio Classification [Chapter 5]

1.2.1.3.1 Research Challenge

Models utilizing NNs for audio classification have found application in diverse tasks, ranging from Music Genre Classification (43; 35; 51) and Environmental Sound Classification (63; 13; 42) to Audio Generation (110; 125). Therefore, formal verification of audio classification systems holds paramount importance in ensuring their reliability and safety, particularly in safety-critical applications such as autonomous vehicles (173; 121), medical diagnosis (68; 103), and industrial monitoring (174). Therefore, the necessity to validate the performance and trustworthiness of these systems in such critical settings underscores the urgency of addressing this research challenge.

1.2.1.3.2 Research Contribution

In response to this significant challenge in formal verification, the current study marks a substantial progression. It builds upon the foundational insights of two recent studies, the star-based verification approach for basic vanilla RNNs (163) and the strategies for verifying CNNs in time series data applications (115) and then focuses on employing the over-approximation set-based reachability techniques for the verification of audio classification models, specifically those utilizing Long Short Term Memory (LSTM) and CNN-LSTM architectures.

1.2.1.4 Extending the Usability of the Neural Network Verification (NNV) tool [Chapter 6]

1.2.1.4.1 Research Challenge

As state-of-the-art neural network verification tools evolve, they increasingly spotlight a range of network architectures characterized by their complex layer structures. This progression calls for significant updates to the Neural Network Verification (NNV) tool to remain aligned with the swiftly progressing domain of neural network verification. While NNV has shown efficacy with simpler networks such as FFNNs and CNNs, it now encounters challenges when dealing with more intricate architectures like U-Net, Res-Net, and Dense-Net. This situation underscores an urgent need to augment NNV's functionality to accommodate these complex network models.

Furthermore, the application of formal verification across diverse areas has underscored the importance of addressing not just spatially distributed inputs but also those with temporal characteristics. This necessitates the integration of various RNN layers, e.g., Long Short-Term Memory (LSTM) layers, into the NNV framework. It also highlights the need for input layers capable of handling these specific data distributions. The challenge, therefore, also lies in expanding NNV's reach to include support for these RNN layers and

architectures, ensuring the tool's relevance and effectiveness in a broader spectrum of neural network applications.

1.2.1.4.2 Research Contribution

A key aspect of this research is enhancing the NNV tool to accommodate complex and advanced neural network layers. This includes integrating concatenation and addition layers for networks such as U-net, Res-Net, and Dense-Net, over-approximated LSTM layers for RNN applications, and sequence input layers for handling sequential input data. By undertaking these enhancements, the NNV tool is poised to remain a relevant and effective resource in verifying the performance and safety of cutting-edge neural network architectures, aligning with the evolving demands of the VNN Comp competitions.

1.2.2 Research Works in Other Directions

1.2.2.1 Robustness Verification of CNN Models against Multiple Attack Scenarios [Chapter 7]

1.2.2.1.1 Research Challenge

Understanding the nuances of different adversarial attack models presents a significant research challenge in the field of neural network security. The utilization of tools like Foolbox (123; Foo) is instrumental in this endeavor, as it provides a comprehensive platform to simulate and analyze a wide array of adversarial attacks on neural networks. Each attack model encapsulates unique strategies to manipulate input data subtly, aiming to deceive the network into erroneous predictions or classifications. By leveraging Foolbox, researchers can systematically evaluate how different attack methodologies impact the robustness and integrity of a neural network. This analysis is crucial not only for revealing potential vulnerabilities in neural network architectures but also for developing more robust and secure models. Therefore, the challenge for this chapter lies in deciphering the complex interactions between these adversarial inputs and the neural network's response mechanisms.

1.2.2.1.2 Research Contribution

This research undertakes a comprehensive experimental study involving two distinct Convolutional Neural Network (CNN) models, subjecting them to ten different adversarial attacks following Foolbox. The core contribution lies in presenting a comparative analysis of how these neural networks respond under identical adversarial influence. Then, using reachability-based robustness verification, the behavior of the CNN models was examined across a wide spectrum of attacks, revealing crucial insights into the models' resilience and vulnerabilities. This comparative approach not only highlights the varying impacts of different adversarial strategies on the same network but also underscores the nuances in network behavior under identical

adversarial conditions.

1.2.2.2 Benchmarking on Formal Verification of Semantic Segmentation Neural Networks [Chapter 8]

1.2.2.2.1 Research Challenge

The field of semantic segmentation, pivotal in applications like autonomous driving and medical imaging, faces a significant challenge in formal verification due to its critical role and data complexity. Formal verification entails a thorough analysis and validation of neural network algorithms against specific specifications and properties, a task complicated by the high-dimensional nature of data and the intricate architectures of these networks.

Robustness verification, particularly in semantic segmentation, faces a significant research challenge due to the lack of specialized benchmarks for evaluating different verification tools. This gap hinders the ability to assess and compare the effectiveness of state-of-the-art verification methodologies accurately. The integration of semantic segmentation tasks into prestigious competitions like 'VNNComp' could be an important step in addressing this issue. By including these tasks, VNNComp would not only broaden its scope but also stimulate the development and refinement of verification tools tailored for semantic segmentation.

1.2.2.2.2 Research Contribution

This research addresses the lack of suitable semantic segmentation network verification benchmarks to be tried and tested by the state-of-the-art formal verification tools and contributes a thorough benchmark study on the application of formal verification techniques to Semantic Segmentation Networks (SSNs). The study encompasses a variety of semantic segmentation datasets, i.e., MNIST (89) and M2NIST (a multi-digit variant of MNIST suitable for segmentation evaluation) datasets, leading to the development of neural network models, including fully-convolutional networks and encoder-decoder architectures. A key focus of the investigation is on a broad spectrum of verification properties, specifically targeting the robustness of these models against bounded adversarial vulnerabilities. Furthering the benchmark proposal, this research also enhances the study by providing sample verification results.

1.3 Thesis Organization

This thesis is structured as follows: Chapter 2 lays out the foundational background necessary for comprehending the work presented. Chapters 3, 4, and 5 delve into the robustness verification of various DNN models specifically in the safety-critical time series domain, focusing on autoencoders, PHM time series applications, and audio classifiers, utilizing reachability analysis. Chapter 6 discusses enhancements made to

the NNV tool, including the integration of complex layer types tailored for time-series applications and the inclusion of skip connections for Res-nets, Dense-nets, and U-nets. Chapter 7 shifts to an exploration of neural networks' responses to ten distinct adversarial attacks. Following this, Chapter 8 examines robustness verification and benchmarking in the context of Semantic Segmentation Neural Networks. Chapter 9 concludes the thesis, summarizing the research contributions and outlining potential future research paths. Lastly, Chapter 10 highlights the publications that have underpinned the material in this thesis.

1.4 Copyright Acknowledgements

The required copyright statements for permission to reprint portions of (112; 115; 113; 114) are included in the following:

- For portions of (112) reproduced in this dissertation, we acknowledge the EPTCS copyright: ©2022 EPTCS. Reprinted, with permission, from Neelanjana Pal and Taylor T. Johnson, "Work In Progress: Safety and Robustness Verification of Autoencoder-Based Regression Models using the NNV Tool," 7th International Workshop on Symbolic-Numeric Methods for Reasoning about CPS and IoT, SNR, August 2021, EPTCS 361, 2022.

- For portions of (115) reproduced in this dissertation, we acknowledge the Springer copyright: ©2023 Springer. Reproduced with permission from Springer Nature, and also from the authors Neelanjana Pal, Diego Manzanas Lopez, and Taylor T. Johnson, "Robustness verification of deep neural networks using star-based reachability analysis with variable-length time series input," In International Conference on Formal Methods for Industrial Critical Systems (FMICS), Springer, 2023, pp. 170–188, September 2023.

- For portions of (113) reproduced in this dissertation, we acknowledge the EPTCS copyright: ©2023 EPTCS. Reprinted, with permission, from Neelanjana Pal, and Taylor T. Johnson, "Formal Verification of Long Short-Term Memory based Audio Classifiers: A Star based Approach." In the 5th Fifth International Workshop on Formal Methods for Autonomous Systems (FMAS), EPTCS 395, 2023, pp. 162-179, November 2023.

- For portions of (114) reproduced in this dissertation, we acknowledge the the Springer copyright: ©2023 Springer. Reprinted, with permission, from Neelanjana Pal, Seojin Lee, and Taylor T. Johnson, "Benchmark: formal verification of semantic segmentation neural networks." In International Conference on Bridging the Gap between AI and Reality, pp. 311-330. Cham: Springer Nature Switzerland, 2023.

CHAPTER 2

Background and Related Works for Safety Verification of Deep Neural Networks

2.1 Neural Network

A neural network (NN) is a computational model inspired by the structure and functions of biological neural networks in human brains (75; 32; 19). It consists of interconnected groups of artificial neurons, which process information using a connectionist approach to computation. Neural networks are typically organized in layers, with input layers receiving the initial data, followed by hidden layers, and an output layer.

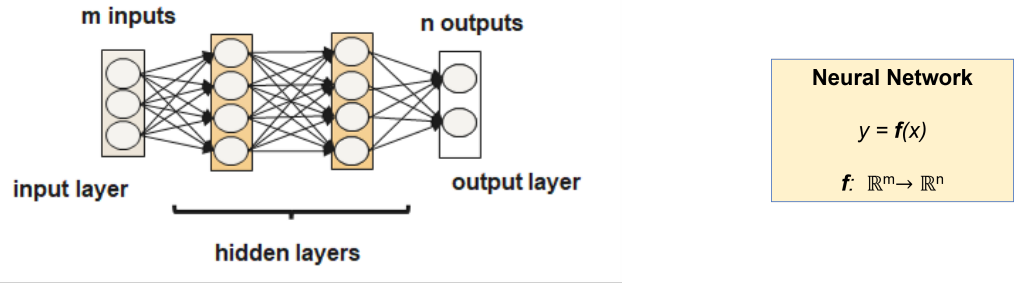

Figure 2.1: Example of a Neural Network with with 3 input nodes, 2 output nodes and 2 hidden layers

Definition 2.1.1. A **neural network (NN)** f is a nonlinear/partially-linear function that maps each input $x \in \mathbb{R}^m$ to the output $y \in \mathbb{R}^n$.

$$f : \mathbf{x} \in \mathbb{R}^m \rightarrow \mathbf{y} \in \mathbb{R}^n \tag{2.1}$$

2.1.1 Categorization Based on Output Layers and Type of Problems Solved

Depending on the output layers and the type of problems solved, NNs can be broadly categorized as (i) Classification NNs and (ii) Regression NNs.

2.1.1.1 Classification Neural Networks

In tasks involving classification, neural networks categorize input data into specified classes. For multi-class classification applications, the output layer of a classification neural network commonly employs a softmax activation function to generate a probability distribution across the classes. In contrast, binary classification tasks typically utilize a sigmoid activation function. In these classification neural networks, the output dimension n of Eq. 2.1 is invariably set to 1. Examples of such classification tasks include image classification, face recognition, and sentiment analysis, among others.

2.1.1.2 Regression Neural Networks

Unlike classification networks that predict discrete classes, regression neural networks aim to forecast numerical values at the output layer. This approach is particularly relevant for tasks like data forecasting, where the network predicts future feature values. In the context of regression neural networks, the output dimension n in Eq. 2.1 is always ≥ 1. Common applications of such regression tasks include weather forecasting, predictive maintenance, time series forecasting, etc.

$$f : \mathbf{x} \in \mathbb{R}^m \mapsto \mathbf{y} \in \mathbb{R}^n, \text{ where } \begin{cases} \text{for Classification Task:} & n = 1, \\ \text{for Regression Task:} & n \geq 1. \end{cases}$$

2.1.2 Categorization Based on Architectures

Depending on the architectures or the specific layers employed, NNs can be categorized into various types; the most well-known networks include feed-forward neural networks (FFNNs), convolutional neural networks (CNNs), semantic segmentation neural networks (SSNs) and recurrent neural networks (RNNs). Recent advancements have introduced architectures like U-nets, Res-nets, and Quantized Neural Networks (QNNs), which have become fundamental in various DNN applications.

This section will delve into some of these existing architectures in detail.

2.1.2.1 Feed-Forward Neural Networks (FFNNs)

Feedforward Neural Networks (FFNNs), among the earliest and most effective learning algorithms, are structured with an input layer, several hidden layers, and an output layer. Each layer comprises multiple neurons connected forward to the next layer via weights. The output of each neuron is determined by its input weight ω, input bias b, and an activation function f_a, as expressed in the equation:

$$y_i = f_a\left(\sum_{j=1}^{n} \omega_{ij} x_j + b_j\right) \tag{2.2}$$

Here, ω_{ij} and b_j represent the weight and bias from the j^{th} neuron of the preceding layer to the i^{th} neuron of the current layer, x_j is the input to this neuron, and y_i is its output.

FFNNs typically incorporate a variety of activation functions f_a, like ReLU, Tanh, or Sigmoid. The output y_k of the k^{th} layer, given an input x_0, is determined by successive applications of these functions:

$$\begin{aligned} L(x_k) &= f_a(\omega_{k,k-1} x_k + b_k), \\ y_k &= (L_k \circ L_{k-1} \circ \cdots \circ L_1)(x_0). \end{aligned} \tag{2.3}$$

In this formulation, $\omega_{k,k-1}$ and b_k denote the weight matrix and bias vector between the $(k-1)$th and kth layers, with x_{k-1} as the input. For the first hidden layer, its input x_0 is also the network's input. Furthermore, the output of one layer serves as the input for the subsequent layer, leading to the final output y_k as shown in Eq. 2.3.

2.1.2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks (CNNs) are structured with a series of layers, encompassing convolutional layers (L_c), pooling layers (L_p), batch-normalization layers (L_b), ReLU activation layers (L_r), and feed-forward layers (L_f). For a CNN denoted as C with l layers, let L_i represent its i^{th} layer, where i ranges from 1 to l. Consequently, the formulation of a CNN can be represented as:

$$C = L_1 \otimes L_2 \otimes \cdots \otimes L_l \tag{2.4}$$

Here, L_i belongs to the set $\{L_c, L_p, L_b, L_r, L_f\}$, and the symbol $\otimes$ indicates the sequential connection between adjacent layers. The inputs to the CNN are processed in order, passing through each layer in sequence, following Eq. 2.4.

2.1.2.3 Semantic Segmentation Neural Networks (SNNs)

Semantic segmentation networks are specialized neural network architectures designed to assign a class label to each pixel in an image, effectively segmenting the image into meaningful parts. This process involves the network initially downsampling the image to extract features (using convolution and pooling layers) and then upsampling (using deconvolutional or transposed convolutional layers) back to the original resolution. The final output is a per-pixel classification of the original image, effectively segmenting it semantically.

A semantic segmentation network S with l layers, where L_i represents the i^{th} layer which be either of the convolutional layers (L_c), pooling layers (L_p), upsampling or deconvolutional layers (L_u).

The operation of the network can be formulated as:

$$S = L_1 \otimes L_2 \otimes \cdots \otimes L_l \tag{2.5}$$

where $L_i \in \{L_c, L_p, L_u, L_s\}$ and $\otimes$ denotes the sequential and functional connections between the layers.

2.1.2.4 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of neural network specifically designed for processing sequential data, such as time series or text. Unlike feedforward networks, RNNs have a loop within them that allows

information to persist, enabling them to process sequences of inputs. This makes RNNs suitable for language modeling, text generation, speech recognition, and more tasks.

In an RNN, each neuron or unit takes input not just from the previous layer but also from itself from the previous time step. This recurrent nature allows the network to maintain a form of 'memory' over the input sequence. Mathematically, the operation of a standard RNN unit can be described as follows:

For an RNN with n time steps, at each time step t, the hidden state h_t of the RNN is updated based on the current input x_t and the previous hidden state h_{t-1}. This update can be represented by the equation:

$$h_t = f(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.6}$$

Here h_t is the hidden state at time t. x_t is the input at time t. W_{hh} is the weight matrix for the hidden state. W_{hx} is the weight matrix for the input. b_h is the bias. f is the activation function, often a non-linear function like tanh or ReLU.

The output y_t at each time step t is then computed based on the current hidden state:

$$y_t = W_{yh}h_t + b_y \tag{2.7}$$

where W_{yh} is the weight matrix from the hidden state to the output, and b_y is the bias for the output.

2.2 Adversarial Perturbation

Deep neural networks have been widely applied as an effective approach to handle complex and practical problems. However, a slight perturbation in the input of a neural network may lead to an error behavior in output (106).

Adversarial perturbations (or attacks) present a significant challenge to the integrity and reliability of the DNN models. These attacks are particularly concerning as they exploit the inherent vulnerabilities of DNNs, even those with high accuracy in standard testing environments. An adversarial attack aims to subtly modify the input data, e.g., an image, often imperceptible to human understanding, leading to misclassification.

Let's consider a DNN model f, which maps an input x to an output y. An adversarial example x_{adv} is crafted by adding a perturbation δ to the original input x. The equation for an adversarial example is:

$$x_{adv} = x + \delta$$

Where: x represents the original input. δ signifies the adversarial perturbation. x_{adv} is the input after the adversarial perturbation.

The perturbation δ is typically subtle to ensure that x_{adv} remains largely indistinguishable from x in terms of perceptual similarity, yet it is effective enough to cause misclassification by the DNN or to push a regression DNN's output beyond safe limits.

2.2.1 L_∞ Norm

In evaluating adversarial perturbations within neural networks, measuring the perturbation's extent is crucial. These perturbations can be measured using various norm types. In the context of this thesis work, the L_∞ norm is utilized, capturing the maximum perturbation-magnitude across all input elements. The L_∞ norm is mathematically defined as:

$$L_\infty : \|x - x_{adv}\|_\infty = \max \|x_i - x_{adv_i}\| \tag{2.8}$$

Here, $\|x - x_{adv}\|_\infty$ represents the L_∞ norm, indicating the maximum deviation between the original input x and its perturbed counterpart x_{adv}.

2.3 Formal Verification of Deep Neural Networks

In designing DNNs, it is critical to balance functional accuracy with dependability. The susceptibility of DNNs to adversarial attacks underscores the need for formal verification of these models.

Formal verification, in the context of DNNs, can be defined as the process of mathematically proving if a DNN f adheres to a given set of specifications or properties P. In general, the formal verification problem can be represented by the following logic:

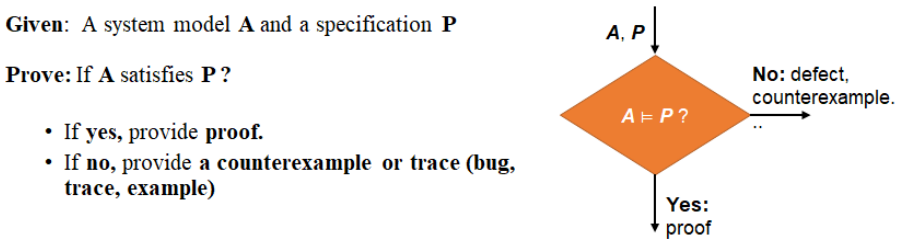

Figure 2.2: Formal Verification Logic w.r.t a system A and properties P

In the context of system models, three formal properties are typically essential for consideration:

1. **Safety**: This property ensures that something bad never happens.
2. **Liveness**: This property guarantees something good finally happens.
3. **Robustness**: This property indicates that the system is robust under adversarial attacks.

While conducting formal verification of a DNN against any adversarial perturbation, the concepts of safety and robustness are often employed interchangeably. In this scenario, the DNN, denoted as f, is treated as the system model A, and the robustness or safety characteristics of the DNN are then regarded as the formal property P, as depicted in Fig. 2.2.

2.3.1 Robustness Property

Robustness is the characteristic of a system or model to preserve its operational integrity and deliver consistent performance despite facing challenging conditions, uncertainties, or intentional perturbations. It encapsulates the system's reliability, resilience, and adaptability under diverse and potentially adverse situations. Specifically, for a DNN, robustness is measured by its capacity to resist adversarial attacks and maintain its predicted output without significant deviation from the expected outcome.

For an input perturbation measured by δ and admissible output deviation ε, the 'delta-epsilon' formulation for the desired robustness property at the softmax layer of a classification network, or at the output layer of a regression network can be written as:

$$||x - x_{adv}||_\infty < \delta \implies ||f(x) - f(x_{adv})||_\infty < \varepsilon \tag{2.9}$$

Where x is the original input belonging to the input space, x_{adv} is the noisy input, $f(x_{adv})$ and $f(x)$ are NN model outputs for, respectively, x_{adv} and x, δ is the max measure of the noise added, ε is the max deviation in the output because of the presence of perturbation ($\delta, \varepsilon \in \mathbf{R} > 0$).

2.3.2 Local Vs. Global Robustness

Verification properties can be categorized into two types: local properties and global properties. A local property is defined for a specific input x_i or a set of inputs $\mathbf{X}$ in the input space. In other words, a local property must hold for certain specific inputs. On the other hand, a global property (177) is defined over the entire input space of the network model and must hold for all inputs without any exceptions.

2.3.2.1 Local Robustness

A DNN f, when provided with an input x, is deemed to be locally robust against a perturbation δ if it consistently satisfies the robustness criterion, as specified in Eq. 2.9, regardless of the presence of the perturbation.

2.3.2.2 Global Robustness

Global robustness for a DNN f, in contrast to local robustness, is the network's ability to maintain its performance across all possible inputs within a defined set (177). A network f is considered globally robust

to perturbations δ if, for every input x in the domain of f, robustness criterion specified in Eq. 2.9 holds true, irrespective of the perturbation δ. This implies that the network's predictions remain consistent and unaffected by the perturbations for all inputs it may encounter.

2.4 Robustness Verification of Deep Neural Networks

Recently, there has been a significant effort to develop methods to establish robustness and formal guarantees of DNN based learning-enabled components (LECs) (120; 72; 55; 45; 77; 181; 175; 182; 190; 139; 141; 165).

2.4.1 Reachability-based Verification Approaches

Demonstrating a system's adherence to pertinent safety properties often employs Reachability Analysis, a technique central to numerous model-checking algorithms (8). This analytical process computes the collection of all possible states a system can reach within a finite period (11). Using reachability analysis, the output domain of a DNN is calculated for a given input space, which in turn can be used to verify the DNN's specifications against particular input conditions.

The primary challenge in conducting reachability analysis of DNNs stems from the piece-wise linear or non-linear functions, such as ReLU neurons. Based on the treatment of non-linear or piece-wise linear layers, reachability analysis can be classified into two types: (i) exact analysis and (ii) over-approximation analysis. While the exact computation can guarantee soundness and completeness, the over-approximation methods can only provide sound analysis.

Definition 2.4.1 (Soundness)**.** Let $f : R^m \rightarrow R^n$ be a NN with an input set R_i and output reachable set R_o. The computed R_o given f and R_i is sound iff $\forall x \in R_i, |y = f(x), y \in R_o$.

Definition 2.4.2 (Completeness)**.** Let $f : R^m \rightarrow R^n$ be a NN with an input set R_i and output reachable set R_o . The computed R_o given f and R_i is complete iff $\forall x \in R_i, \exists y = f(x) \mid y \in R_o$ and $\forall y \in R_o, \exists x \in R_i \mid y = f(x)$.

2.4.1.1 Approximate Methods

The approximation methods are mainly based on mixed-integer linear programs (MILPs) (95; 82; 46), zonotopes, abstract domain, global optimization, regressive polynomial, network conversion, unified framework and linearization methods can only guarantee the soundness of the analysis, but not completeness. The MILPs-based work can guarantee both aspects when neural networks have only one output, but fails when they have multiple outputs. Because it estimates the range of outputs independently and eventually generates a box domain that over approximates exact reachable sets.Most of the over-approximation methods are capable of efficiently analyzing large scales neural networks based ReLU activation functions. Their main strategy is replacing each ReLU function with a more conservative domain so that the number of output reachable sets

of each layer can be largely reduced. But these approaches are only limited to point-wise inputs with a very little perturbation, and their conservativeness of the estimated reachable sets will exponentially grow as the input domain increases.

2.4.1.2 Exact Methods

While the exact analysis is mainly based on the satisfiability modulo theory (SMT), polytopes and Star set in a tool named NNV, these approaches, both soundness and completeness of the verification can be guaranteed. The strategy of Reluplex is extending the simplex method to handle the piece-wise linear ReLU activation function by allowing variables of defined ReLU pairs to violate their semantics temporarily. The Marabou is an improved version of Reluplex. It supports arbitrary piecewise-linear activation function, parallel computation, etc. Another work ReluVal is based on interval arithmetic to over approximate the bounds on the outputs. The influence of each input variable on the output is analyzed so that it can repeatedly split the input intervals and efficiently refine the output range. All these works focus on the satisfiability problem that is to determine whether there exists output that locates in the unsafe domain. While the works are conducting the reachability analysis of a neural network. Given an input set, they can compute the exact output sets of a neural network. This reachability analysis is also associated with the quantification of linear regions of neural networks. A linear region of a piecewise linear functions $F : R^m \rightarrow R^n$ refers to a maximum convex subset of an input set in R^m, on which the function F is linear. Accordingly, the input set is split by the ReLU function in each neuron into pieces which are linear regions. Each output set of a network computed by corresponds to the output with respect to a linear region. Therefore, the number of output sets computed is equal to the number of linear regions. These reachability-based methods can provide a full understanding of the neural network's behavior and are promising directions towards safe networks.

2.5 The Neural Network Verification (NNV) Tool

The NNV (170; 97) tool serves as a comprehensive framework for verifying the safety and robustness of neural networks, which is crucial in safety-critical applications like autonomous vehicles. It employs reachability algorithms, notably exact and over-approximates methods (169; 164), to determine the reachable states of a neural network for given inputs, ensuring adherence to safety properties.

2.5.1 NNV's Main Reachability Algorithms

Initially, NNV implemented techniques using polyhedra for reach set representation, later advancing to zonotope and star set reachability analysis for the safety verification and robustness analysis of different types of DNNs.

2.5.1.1 Polyhedron

The exact polyhedron reachability algorithm (166), in NNV, computes the output reachable sets in a layer-by-layer manner for an NN.

Definition 2.5.1. A polyhedron set can be defined as a set of points in a space that satisfy a finite number of linear inequalities. Mathematically, a polyhedron P in an n-dimensional space can be described using a series of linear inequalities. These inequalities are often represented in matrix form as follows:

$$P = \{\mathbf{x} \in \mathbb{R}^n \mid A\mathbf{x} \leq \mathbf{b}\} \tag{2.10}$$

Where: $\mathbf{x}$ is a vector in n-dimensional space ($\mathbb{R}^n$). A is a matrix where each row represents the coefficients of a linear inequality. $\mathbf{b}$ is a vector where each element corresponds to the upper bound of the respective inequality. $A\mathbf{x} \leq \mathbf{b}$ represents a series of linear inequalities, and each inequality bounds the polyhedron in the n-dimensional space.

In this definition, P includes all points $\mathbf{x}$ that satisfy these inequalities, forming a convex polyhedron. This polyhedron can be bounded (a polytope) or unbounded, depending on the nature of the inequalities.

However, reachability set calculation with the polyhedron method is very computationally expensive, and it becomes the main drawback limiting the polyhedron approach's scalability (161).

2.5.1.2 Star

Compared to the polyhedron approach, star-based reachability is less expensive and, hence, more efficient. In NNV, several adaptations and optimizations of the star-based approach (164; 167) are implemented. While the exact method is computationally expensive compared to its over-approximate counterpart, it is still less expensive and more scalable than the polyhedron method (161). The over-approximate star set is generated by over-approximating the exact reachable set after applying an activation function, e.g., ReLU, Tanh, Sigmoid.

Definition 2.5.2. *A generalized star set* (or simply star (16)) Θ is a tuple $\langle c, V, P\rangle$ where $c \in \mathbb{R}^n$ is the center, $V = \{v_1, v_2, \cdots, v_m\}$ is a set of m vectors in $\mathbb{R}^n$ called basis vectors, and $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ is a predicate. The basis vectors are arranged to form the star's $n \times m$ basis matrix. The set of states represented by the star is given as:

$$[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i) \text{ such that } P(\alpha_1, \cdots, \alpha_m) = \top\}. \tag{2.11}$$

In this work, here the predicates are restricted to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p \times m}$, α is the vector of m-variables, i.e., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p \times 1}$. A star is an empty set, i.e., $\Theta = \emptyset$ if and only if the predicate $P(\alpha)$ is infeasible.

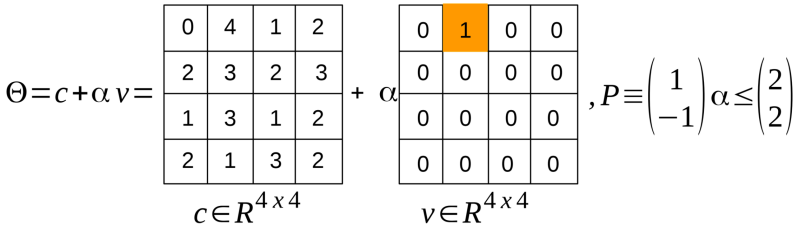

Figure 2.3: Illustration of a Star set, with input x $\in \mathbb{R}^2$

2.5.1.3 Zonotope

Zonotopes, like star sets, offer an over-approximation method for computing reachable sets in the NNV tool. This approach, known for its speed and scalability, tends to yield more conservative estimates of the reachable sets than those obtained using star sets. The implementation of zonotopes within NNV is based on the algorithm described in the paper (139), which provides an efficient framework for this approximation technique.

Definition 2.5.3. A zonotope Z is defined as a tuple $\langle l, G \rangle$ where $l \in \mathbb{R}^n$ is the center, and $G = \{g_1, g_2, \ldots, g_m\}$ is a set of m generators in $\mathbb{R}^n$. The set of states represented by a zonotope is given by:

$$[\![Z]\!] = \left\{ x \mid x = l + \sum_{i=1}^{m} (a_i g_i) \text{ such that } -1 \leq a_i \leq 1 \right\}. \tag{III.2}$$

A zonotope is essentially a star set in which all predicate variables range within $[-1, 1]$. The affine mapping of a zonotope yields another zonotope. However, the intersection of a zonotope and a half-space is generally not a zonotope. An advantage of a zonotope over a star set is that it allows for quick computation of the ranges of a state without solving LP optimization. For instance, the range of the state $x(j)$ in a zonotope can be expressed as:

$$l(j) - \sum_{i=1}^{m} |g_i(j)| \leq x(j) \leq l(j) + \sum_{i=1}^{m} |g_i(j)|. \tag{2.12}$$

2.5.1.4 Imagestar

The Imagestar representation, introduced in the NNV tool and detailed in (169), is a novel set representation primarily tailored for image-based CNN examples. This representation extends the concept of star sets by using multichannel images in place of central and generator vectors. Like its star set counterpart, the Imagestar approach also incorporates both exact and over-approximation methods in its implementation.

Definition 2.5.4. An *ImageStar (169)* Θ is a tuple $\langle c, V, P, l, u \rangle$ where $c \in \mathbb{R}^{h \times w \times nc}$ is the *anchor image*,

$V = \{v_1, v_2, \cdots, v_m\}$ is a set of m images in $\mathbb{R}^{h\times w\times nc}$ called *generator images*, $P : \mathbb{R}^m \to \{\top, \bot\}$ is a *predicate*, l and u are the *lower bound* and *upper bound* vectors of the *predicate variables*, and h, w, nc are the *height*, *width*, and *number of channels* of the images, respectively. The generator images are arranged to form the ImageStar's $h \times w \times nc \times m$ *basis array*. The set of images represented by the ImageStar is given as:

$$[\![\Theta]\!] = \{x \mid x = c + \Sigma_{i=1}^{m}(\alpha_i v_i) \text{ such that } P(\alpha_1, \cdots, \alpha_m) = \top, l_i \leq \alpha_i \leq u_i\}.$$

For imagestar, as well, the predicates are restricted to be a conjunction of linear constraints, $P(\alpha) \triangleq C\alpha \leq d$ where, for p linear constraints, $C \in \mathbb{R}^{p\times m}$, α is the vector of m-variables, i.e., $\alpha = [\alpha_1, \cdots, \alpha_m]^T$, and $d \in \mathbb{R}^{p\times 1}$. An imageStar is the empty set if and only if $P(\alpha)$ subject to $l \leq \alpha \leq u$ is empty.

2.5.2 Reachability of a Neural Network using NNV

Reachability analysis (or shortly, reach) of any NN f on an input set I is done in a layer-by-layer manner i.e., the output reachable set obtained, can be calculated as a step-by-step process of constructing the reachable sets for each layer of the network.

Let $h : \ u \in \mathbb{R}^j \to v \in \mathbb{R}^p$, be a NN layer expressed by the equation v = h(u). The **reachable set** R_h, with input, $I \in R^m$ is defined as

$$R_h \triangleq \{v \mid v = h(u),\ u \in I\} \tag{2.13}$$

Therefore, the output reach-set R_f of a NN, f, with input, $I \in R^m$ can be expressed as

$$Reach(f, I) : \ I \to R_f \tag{2.14}$$

Now, if the network has k layers $f \triangleq [l_1, l_2, ..., l_k]$ then, the output reachable set R_f can be calculated as a step-by-step process of constructing the reachable sets for each network layer:

$$\begin{aligned}
&R_{l_1} \triangleq \{v_1 \mid v_1 = h_1(x),\ x \in I\},\\
&R_{l_2} \triangleq \{v_2 \mid v_2 = h_2(v_1),\ v_1 \in R_{l_1}\},\\
&\vdots\\
&R_f = R_{l_k} \triangleq \{v_k \mid v_k = h_k(v_{k-1}),\ v_{k-1} \in R_{l_{k-1}}\}
\end{aligned} \tag{2.15}$$

where h_k is the function represented by the k^{th} layer l_k. The reachable set R_{l_k} contains all neural network outputs corresponding to all input vectors X in the input set I.

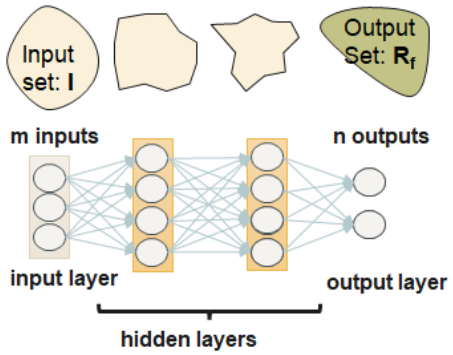

Figure 2.4: Illustration of layer-by-layer reachability analysis.

2.5.3 Robustness Verification using NNV

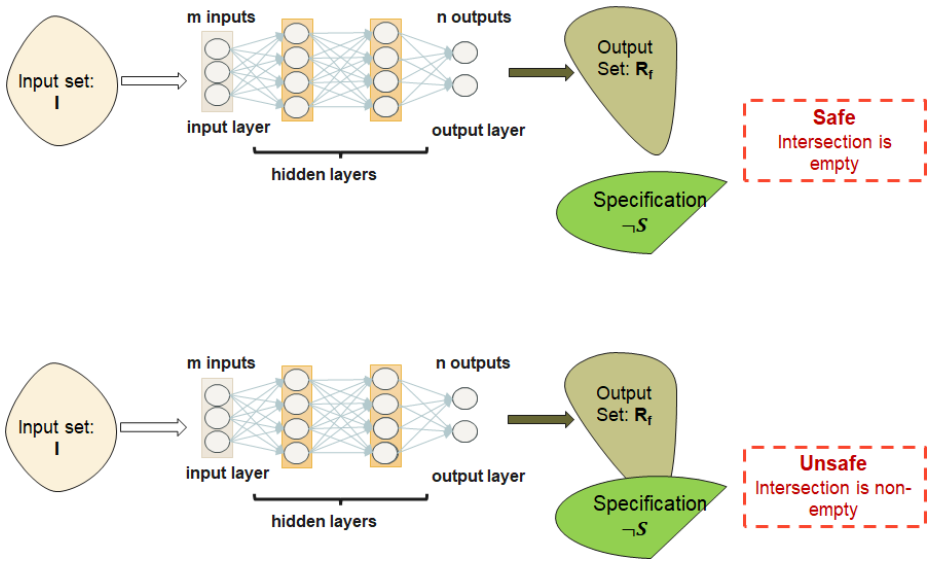

Figure 2.5: Illustration of formal verification using reachability analysis.

Given a k-layers neural network f, and a safety specification S defined as a set of linear constraints on the neural network outputs $S \triangleq \{y_k \mid Cy_k \leq d\}$, the neural network f is called to be safe corresponding to the input set I, i.e. $f(I) \vDash S$, if and only if $R_{l_k} \cap \neg S = \emptyset$, where R_{l_k} is the reachable set of the neural network with the input set I, and $\neg S$ is the complement of the set S. Otherwise, the neural network is called to be unsafe $f(I) \nvDash S$.

2.6 Other State-of-the-art Tools & Neural Network Verification Competition

2.6.1 ERAN

ERAN (140; 142; 143; 138; 109; 134) is a neural network verifier that uses abstract interpretation for network encoding as LP or MILP problems, suitable for fully-connected, convolutional, and residual architectures

with various non-linearities. It combines GPUPoly-derived (134) neuron relaxations with MILP encoding for precise analysis. ERAN, developed in Python, uses ELINA (144) for abstractions and Gurobi for solving LP and MILP, requiring significant GPU and CPU resources to function optimally. It offers complete and incomplete verification modes, generating concrete counterexamples in the former.

2.6.2 alpha-beta CROWN

Alpha-beta-CROWN, an advanced verifier for neural networks, builds on linear bound propagation techniques from research like CROWN (190; 184), α-CROWN (185), β-CROWN (176) and GCP-CROWN (189). It uses the `auto_LiRPA` (184) library for broad architecture support and GPU-optimized performance. Key methods include gradient ascent for intermediate layer bounds, and final layer bounds optimization (α-CROWN (185)), efficient branch and bound integration (β-CROWN (33)), and cutting-plane methods for tighter bounds (GCP-CROWN (189)). For smaller networks, it combines mixed integer programming (MIP) (156) with α-CROWN (referred to as α-CROWN + MIP (189)) for enhanced verification, making Alpha-beta-CROWN a powerful, scalable tool in neural network verification.

2.6.3 nnenum

The nnenum tool (15) utilizes a sophisticated multi-layered abstraction approach to ensure high performance and complete verification of neural networks (14). It effectively combines three distinct zonotope types with star set (triangle) over-approximations (167) while employing efficient and parallelized ReLU case-splitting methodologies (18). This approach enables swift propagation of the ImageStar set (160) across various neural network layers, each characterized by its own set of parameters. Developed using Python 3, nnenum also integrates GLPK for solving linear programming challenges, further enhancing its analytical capabilities.

2.6.4 Neural Network Verification Competitions

The proliferation of neural networks (NNs) in safety-critical applications has brought attention to their susceptibility to adversarial examples (151), where even minor input perturbations can significantly alter their outputs. Such perturbations, whether occurring randomly or due to malicious intent, emphasize the crucial need to rigorously analyze the robustness of deep learning systems before deploying them in safety-critical domains. Consequently, numerous methods and software tools (48; 56; 72; 77) have been developed for this purpose. However, the increasing number and specialization of these tools have made it challenging for practitioners to choose the most suitable one for their needs.

In response to this dilemma, in 2020, a friendly International Competition on Verification of Neural Networks (VNN-Comp 2020) (30; 17; 108) was conducted to address the issue and allow researchers to compare

their neural network verifiers across a wide range of benchmarks. Originally designed as a friendly competition with minimal standardization, the event evolved to introduce more standardization and automation. The goal was to ensure a fair comparison among verifiers on cost-equivalent hardware, utilizing standardized formats for properties and networks. This evolution aimed to facilitate informed decision-making by researchers and practitioners when selecting verification tools for their specific requirements. The VNN-Comp celebrates its 4th iteration this year and successfully presented the results at the Computer Aided Verification 2023 (CAV) conference.

CHAPTER 3

Safety and Robustness Verification of Autoencoder-Based Regression Models using the NNV Tool

This chapter is adapted from the material presented in (112).

3.1 Introduction

The burgeoning field of neural network (NN) research has unveiled a critical vulnerability: even state-of-the-art, meticulously trained networks are susceptible to minute input perturbations, resulting in substantial deviations in output accuracy (106; 152; 60). This vulnerability transcends image-based networks, extending to diverse input forms such as time-series data and input signals. The implications of this fragility are profound, particularly in terms of information integrity, privacy, and security, posing potential catastrophic risks in safety-critical applications (132; 49). While, the most researched domain for NN verification majorly involves image inputs, particularly safety and robustness checking of various classification neural networks (168; 9; 26; 78; 104; 169), the area of autoencoder verification remains relatively underexplored. Previous research has also analyzed feed-forward neural networks (FFNN(164)), convolutional neural networks (CNN(169)), and semantic segmentation networks (SSN(168)) using different set-based reachability tools, such as Neural Network Verification (NNV(170; 97)) and JuliaReach (24), among others. Notably, classification models employing autoencoders mirror the operational dynamics of conventional classifiers. However, regression-based autoencoders, which aim to recreate their input in their output, present a unique challenge. They necessitate verification techniques to ascertain whether the regenerated output falls within an acceptable range of the original, unperturbed input, particularly in scenarios where the input might be compromised by faults or adversarial attacks.

3.2 Preliminaries

3.2.1 Autoencoder

The concept of autoencoder (AE) was originally proposed by LeCun in his PhD thesis (88). An autoencoder's operation can be mathematically described by two main functions: one for the encoder and one for the decoder, which generally are implemented by neural networks. The encoder part compresses the input into a latent space representation, and the decoder recreates the input from the encoder output. In other terms, the encoder reduces the feature dimensions of the input (similar to Principal Component Analysis(124)) and can thus be used for the data preparation steps for other machine learning models.

The encoder and decoder in an autoencoder can be conceptualized as: $en(x)$ and $de(z)$, respectively. Here,

$en(x)$ functions to map a data point x from the data space to the feature space, effectively compressing the data. Conversely, $de(z)$ serves to reconstruct the original data point x by mapping z from the feature space back to the data space, thereby attempting to reverse the encoding process.

The objective of training an autoencoder is often to minimize the difference between the input x and the reconstructed output $\hat{x}$, typically measured by a loss function like mean squared error (MSE) for continuous input data or cross-entropy for binary input data.

The complete operation of an autoencoder can thus be summarized as:

$$\hat{x} = de(en(x))$$

Here, en represents the encoder function, de represents the decoder function, and the goal is to make $\hat{x}$ as close to x as possible.

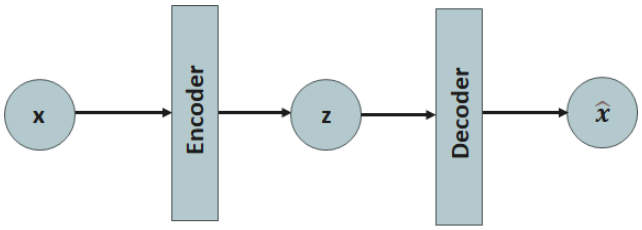

Figure 3.1: The Structure of an Autoencoder

Following a DNN model, the mathematical equation of an autoencoder can also be given as:

$$f: \mathbf{x} \in \mathbb{R}^m \rightarrow \mathbf{y} \in \mathbb{R}^m$$

Autoencoder applications, depending on their learning objectives, can be broadly divided into two primary categories:

1. **Regression Task:** In this category, the autoencoder's goal is to reproduce the input data at its output. The model learns to capture the most relevant features in the input data and then uses these features to reconstruct the input as accurately as possible at the output.

2. **Classification Task:** For classification purposes, an autoencoder is initially trained as a regression model to capture the essential features of the input data. After this initial training phase, modifications are made to the model: the decoder part is removed, and a softmax layer along with a classification layer are appended after the bottleneck (the encoder's output). Additionally, the input labels are transformed into a 'one hot-coded' format and are fed into the model along with the input data. This modified model

is then retrained, transforming it into an autoencoder-based classification model, where the encoder's latent-space representation of the input data is used for classification tasks.

As noted in Sec.3.1, classification models that utilize autoencoders operate in a manner akin to standard classifiers. Therefore, the exploration of classification tasks using autoencoders falls outside the purview of this chapter.

3.2.2 Modified Star for One Dimensional Input Data

The input for the regression-based autoencoder model considered in this chapter is a set of time-instanced signals. Therefore, for the reachability analysis using NNV, an approach with a similar underlying concept of generalized Star is used, but the dimension of input is changed to a 1D equivalent vector, and as the specific input used here will be a signal, the name 'Signalstar' can be used for the reference.

Similar to Imagestar, Signalstar is also a tuple of three variables $\langle c, V, P \rangle$, and the set of signals represented by the Signalstar is given as:

$$[\![\Theta]\!] = \{x \mid x = c + \sum_{i=1}^{m} (\alpha_i v_i) \mid P(\alpha_1, \alpha_2, ..., \alpha_m) = \top\} \tag{3.1}$$

where $c \in \mathbb{R}^n$ is the anchor signal or central signal with n time instances. Similar to Imagestar $V = \{v_1, v_2, ...v_m\}$ is generator signal instances, which is a set of m signals in $\mathbb{R}^n$. The generator signals are arranged to form the basis array of Signalstar $(n \times m)$. $P : \mathbb{R}^m \rightarrow \{\top, \bot\}$ is a predicate.

$$\Theta = c + \alpha v = \begin{bmatrix} 0 \\ 2 \\ 1 \\ 2 \end{bmatrix} + \alpha \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, P \equiv \begin{pmatrix} 1 \\ -1 \end{pmatrix} \alpha \leq \begin{pmatrix} 2 \\ 2 \end{pmatrix}$$

$$c \in \mathbb{R}^{4\,X\,1\,X\,1} \quad v \in \mathbb{R}^{4\,X\,1\,X\,1}$$

Figure 3.2: Signalstar

Another way of defining the Imagestar or Signalstar set is to use the upper and lower bounds of the attack, centering the actual input. These bounds on each input parameter along with the predicates create the complete set of constraints the optimizer will solve to generate the initial set of states. An example of Signalstar is shown in Fig. 3.2.

3.3 Problem Formulation: Verification of Autoencoder Models

3.4 Experimental Setup

The problem statement, developed in collaboration with TU Munich [1], focuses on the verification of autoencoders within a closed-loop system.

The process involves a sensor capturing a signal over a set time range, followed by the transmission of this sampled data to an autoencoder. The autoencoder's role is to reconstruct the time signal to its optimal form. Once reconstructed, the signal is forwarded to a controller for subsequent applications.

The central concern here is to scrutinize the autoencoder's reconstructed output using reachability methods, while establishing certain criteria to safeguard the controller against faulty inputs. This involves assessing the maximum fault tolerance of the controller, considering factors like uninterrupted signal, fault location, and a defined range of acceptable upper and lower limits. The range is typically determined by the tolerances of downstream devices that receive the output signal, especially in terms of their minimum and maximum permissible current or voltage values. The essential components of this complete loop are illustrated in 3.3.

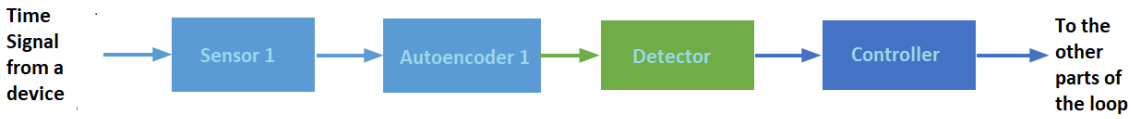

Figure 3.3: System Model

While the data is passed from the sensor to the autoencoder, sensor noises may get added to the original data. As the controller will be designed to work with a specific range of input values, external errors can cause the controller to malfunction and thus can cause the entire closed-loop system to collapse. So it is necessary to detect (in the Detector part of 3.3) if the reformed signal is within the acceptable range of the controller or not.

3.4.1 Dataset Selection and Autoencoder Model Training

The dataset was devised using the system model shown in Fig. 3.3. Each signal segment consists of 100 time samples and is then normalized using Min-Max technique. The entire dataset is divided into, 2057 training samples and 363 test samples. For generating the fault signals, the same test samples are used with spike faults at random time instances, with only one fault per signal.

Autoencoder architecture used for this paper is a five-layer NN, shown in 3.4. The first two layers comprise the encoder part of the architecture, and the last two, the decoder part. The third layer creates the latent

[1] Hongpeng Cao, Mirco Theile, Bingzhuo Zhong, Dr.Marco Caccamo of **Chair of Cyber-Physical Systems in Production Engineering, Technical University of Munich (TUM)**

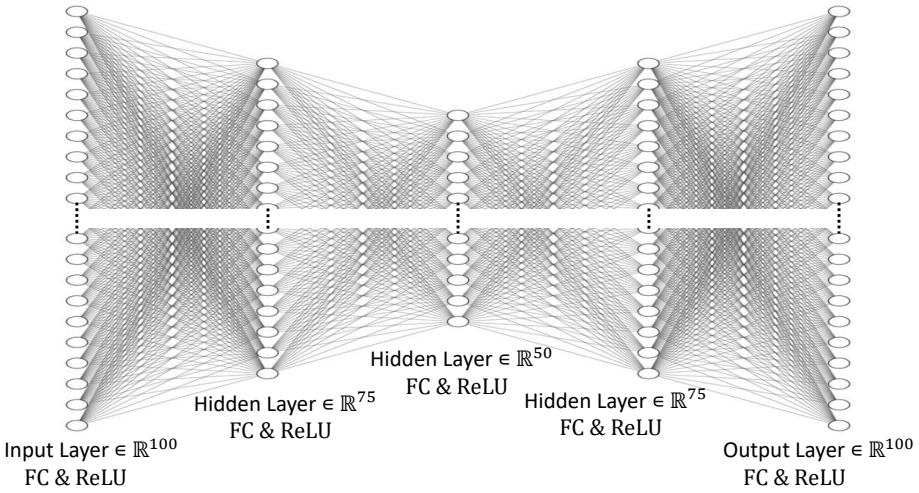

Figure 3.4: Autoencoder Model used in the paper [*FC: Fully connected]

space for the model. Here, the 'Adam' optimizer and 'Mean square error' loss function are used for training purposes.

3.4.2 Attacks/Perturbations on the Inputs for Verification:

In the verification of autoencoders, the process involves subjecting inputs to certain noises or perturbations to determine whether the model can accurately reconstruct the outputs. The goal is to verify that the autoencoder can effectively regenerate the original, unperturbed inputs despite these introduced disturbances. Here, in this chapter, a simpler noise, called spike faults, is considered for the experiment.

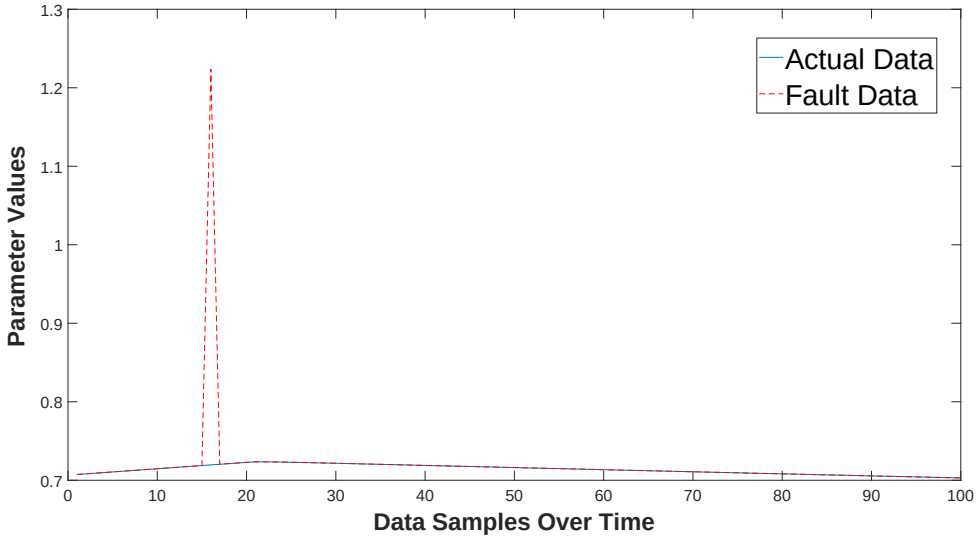

Figure 3.5: Sample Fault Signal and the actual signal

3.4.3 Spike Faults (Impulse Noise)

It is also known as impulse noises and is characterized by random occurrences of spikes that have a very short duration and relatively high amplitude. These faults are often encountered in sensor data, commonly stemming from voltage spikes in devices. Due to their erratic and transient nature, spike faults can be challenging to detect and analyze.

Mathematically, a spike fault can be represented in the context of a signal or sensor reading as follows:

$$x_{\text{faulty}}(t) = x(t) + sp(t) \tag{3.2}$$

Here: $x_{\text{faulty}}(t)$ represents the signal with the spike fault at time t. $x(t)$ is the original, unperturbed signal at time t. $sp(t)$ denotes the spike fault, which is a function that has significant amplitude for a very short duration, and is zero or negligible at other times.

In this model, the spike fault $s(t)$ is added to the original signal, causing a sudden and brief distortion. The challenge in handling spike faults lies in their sporadic occurrence and short duration, which makes standard noise filtering or prediction methods less effective.

A sample fault signal (in red) overlapping the actual signal (in blue) is shown below in Fig. 3.5

3.4.4 Reachability Analysis Using NNV

The input set w.r.t an attack is generated by establishing a pair of upper and lower limits around the central signal, utilizing a modified star (or Signalstar) method. This set is then propagated through the various layers of the network, as detailed in Sec. 2.5.2, culminating at the output layer, which produces the output set(s).

3.4.5 Robustness Measures

Once the output reachable sets are calculated using the NNV tool, these sets are evaluated to determine if they intersect with predefined unsafe regions, which are characterized in terms of the faulty signal, which in this context means calculating the upper and lower bounds of the output sets and checking if these bounds are well within a certain permissible limit of the input signal. If the reachable bounds fall within the permissible limits, the model is said to be robust.

In this context, the concept of Percentage Robustness, as generally used in the literature, is adapted with minor modifications to suit the specific use case at hand. Additionally, a new metric, referred to as 'Unrobustness Grade,' is defined specifically for this example.

3.4.5.1 Percentage Robustness (PR)

For calculating the robustness and safety measure for this application, the percentage robustness measure can be defined as the number of instances where the output bound is within the threshold values (i.e., the permissible upper and lower limit), divided by the total number of time instances. It measures the relative safety of the network w.r.t the input signal, and for a given fault, a value of 100 is perfectly safe, and 0 is completely unsafe.

$$PR = \frac{N_{robust}}{N_{total}} \times 100\%, \tag{3.3}$$

where N_{robust} is the total number of robust time instances, and N_{total} = the total number of time steps in the time series signal.

3.4.5.2 Un-robustness Grade (UrG)

This failure grade can be defined as the maximum difference between the actual signal and corresponding output bound (upper or lower) divided by the threshold value (acceptable deviation). The higher it is from value 1, the more it deviates from the permissible range, i.e., the grade of Un-robustness is higher. Clearly, this measure is only meaningful when the output reachable bounds exceed the permissible bounds.

If the reconstructed signal is deviated maximum at time-instant t, the reachable bounds at t are l_t and u_t; and the permissible limits are l_{allow} and u_{allow}, then UrG can be expressed as the given equation:

$$UrG = \frac{max\ deviation\ at\ the\ upper/lower\ side}{permissible\ deviation}, \tag{3.4}$$

where N_{robust} is the total number of robust time instances, and N_{total} = the total number of time steps in the time series signal.

3.5 Experiment Results

3.5.1 Observations and Analysis

Figures 3.7 and 3.8 illustrate the output bounds to the sample signals shown in Fig. 3.5, both for the actual and its faulty counterpart. The reachable output bounds of the faulty signals are marked in red, while the actual signals with permissible limits are indicated in blue, covering two different acceptable ranges (e.g., +/- 0.0389 and +/-0.0233). These plots demonstrate that the acceptability of the reconstructed output signal for the controller also hinges on these ranges. In Fig. 3.7, the output bounds are well within the acceptable range, indicating robustness. Conversely, in Fig. 3.8, the bounds for some time instances exceed the accepted range, rendering them unsafe for further processing. For this particular example, the previously defined evaluation

Figure 3.6: Output bounds (red) of the fault signal and bounds (blue) for the input signal with two different thresholds.

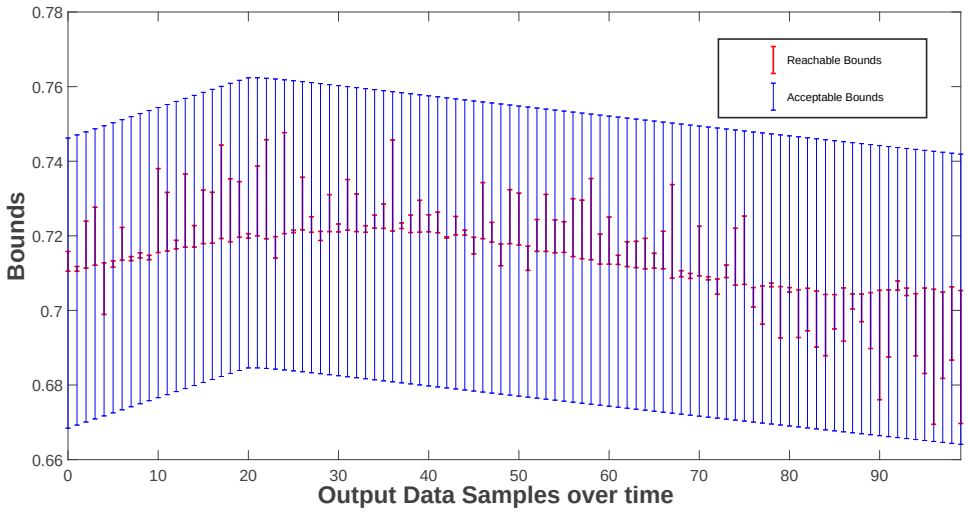

Figure 3.7: Threshold = 0.0389

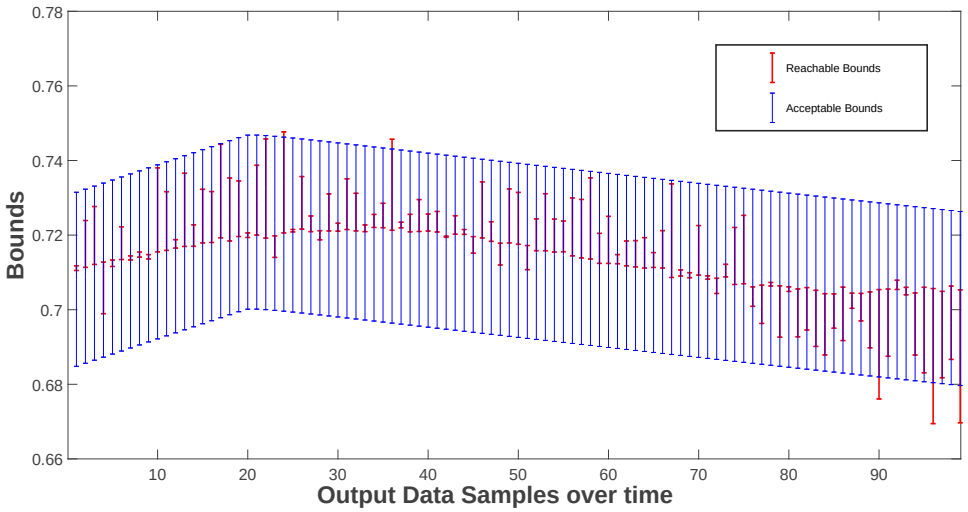

Figure 3.8: Threshold = 0.0233

metrics for the scenario in Fig. 3.8 can be calculated as follows:

3.5.2 Percentage Robustness

total time instance = 100; instances, where output bound is within the permissible limit = 95; therefore percentage Robustness = .95.

3.5.3 Un-robustness Grade

deviation is maximum at time instance 96; here lower bound is deviated maximum from the actual signal and the corresponding value is 0.6695; the lower limit of permissible range at t = 96 is 0.7057; hence maximum deviation = 0.7057-0.6695 = 0.0363; finally the Grade of Un-robustness = 0.0363/0.0233 = 1.5536.

3.6 Conclusion

This study introduces the inaugural formal verification approach for autoencoders as well as the first for any kind of time-series applications, utilizing reachability analysis. The research centers on a case study involving a regression-based autoencoder model that processes time-instance signals from devices potentially afflicted with spike faults. The analysis focuses on comparing the reconstructed output with an uninterrupted input, verifying whether the reconstructed signal falls within pre-defined upper and lower output limits. This process was assessed using two different threshold values.

As an initial exploration into autoencoder-based models, future endeavors aim to develop more nuanced evaluation metrics, tailored to scenarios where reachability-based analysis proves most beneficial. Future work includes adapting other verification tools, originally designed for classification models, to gauge their comparative effectiveness. Plans also involve expanding the scope to include various autoencoder applications as case studies. Another interesting research direction is examining how an autoencoder's denoising capability can mitigate the impact of input faults, thereby enhancing output robustness. Additionally, the robustness verification of variational autoencoder (VAE) models presents a promising area for future investigation.

CHAPTER 4

Robustness Verification of Deep Neural Networks using Star-Based Reachability Analysis in PHM Time Series Applications

This chapter is adapted from the material presented in (115).

4.1 Introduction

As mentioned earlier, the presence of intentional or unintentional input perturbations is not only confined to image-based networks but also has been extended to other input types, including time series data or input signals with different noises in predictive maintenance applications (40; 171). One such use case is in the manufacturing industry, where data from process systems, such as IoT sensors and industrial machines, are stored for future analysis (133; 53). Data analytics in this context provide insights and statistical information and can be used to diagnose past behavior (192; 98), and predict future behavior (150; 25; 93), maximizing industry production. This application is not only limited to manufacturing but is also relevant in fields like healthcare digitalization (187; 158) and smart cities (149; 146). Noisy input data, here, refers to data containing errors, uncertainties, or disturbances caused by factors like sensor measurement errors, environmental variations, or other noise sources.

While NN applications with image data have received significant attention, little work has been done in the domain of regression-type model verification, particularly with time series data in predictive maintenance applications. Regression-based models with noisy data are crucial for learning data representations and predicting future values, enabling fault prediction and anomaly detection in high-confidence, safety-critical systems (39; 79). This motivated us to use verification techniques to validate the output of regression networks and ensure that the output(s) fall within a specific safe and acceptable range.

4.2 Preliminaries

This section introduces some basic definitions and descriptions necessary to understand the progression of this study and the necessary evaluations on time series data.

Definition 4.2.1. The definition of a 'signal' varies depending on the applicable fields. In the area of signal processing, a ***signal*** S can be defined as some physical quantity that varies with respect to (w.r.t.) some independent dimension (e.g., space or time) (118). In other words, a signal can also be thought of as a

function that carries information about the behavior of a system or properties of some physical process (119).

$$S = g(q) \tag{4.1}$$

where q is space, time, etc. Depending on the nature of the spaces signals are defined over; they can be categorized as discrete or continuous. Discrete-time signals are also known as time series data.

The next section defines the specific class of signals considered in this paper, focusing on time series data.

Definition 4.2.2. A **time series signal** S_T is defined as an ordered sequence of values of a variable (or variables) at different time steps. In other words, a time series signal is an ordered sequence of discrete-time data of one or multiple features[1].

$$\begin{aligned} S_T &= s_{t_1}, s_{t_2}, s_{t_3}, ... \\ T &= t_1, t_2, t_3, ... \end{aligned} \tag{4.2}$$

where, $t_1, t_2, t_3, \ldots$ is an ordered sequence of instances in time T and $S_T = s_{t_1}, s_{t_2}, s_{t_3}, \ldots$ are the signal values at those time instances for each $t = t_i$.

Here, sometimes 'time series signal' is also referred to as 'signal.'

The following section defines the specific type of neural network focused on in this study: regression neural networks, with a particular emphasis on those used for time series regression.

Definition 4.2.3. A **time series regression neural network (TSRegNN)** f is a nonlinear/partially-linear function that maps each time-stamped value $x(i, j)$ (for i^{th} feature and j^{th} timestamp) of a single or multifeatured time series input $\mathbf{x}$ to the output $\mathbf{y}$.

$$f: \ \mathbf{x} \in \mathbb{R}^{n_f \times t_s} \rightarrow \mathbf{y} \in \mathbb{R}^{p \times q} \tag{4.3}$$

where t_s, n_f are the time-sequence length and the number of features of the input data, respectively, $(j, i) \in \{1, \ldots, t_s\} \times \{1, \ldots, n_f\}$ are the time steps and corresponding feature indices, respectively, and p is the number of values present in the output, while q is the length of each of the output values; it can either be equal to t_s or not, depending on the network design.

Here, each row of $\mathbf{x}$ represents a timestamped feature variable.

[1] Each **feature** is a measurable piece of data that is used for analysis.

4.2.1 Reachability of a Time Series Regression Network

Here, the alternative approach of defining a Star set is employed for time series data. It involves using the upper and lower bounds of the noisy input, centering the actual input. These bounds on each input parameter, along with the predicates, create the complete set of constraints the optimizer will solve to generate the initial set of states.

Reachability analysis (or shortly, reach) of a TSRegNN f on Star input set I is similar to the reachable set calculations as detailed in Sec. 2.5.2, following a layer-by-layer propagation of the input set.

$$Reach(f, I) : \ I \rightarrow R_{ts}$$

$R_{ts}(I)$ are called the *output reachable set* of the TSRegNN corresponding to the input set I.

4.2.2 Adversarial Noises Considered

For an input sequence with t_s number of time instances and n_f number of features, there can be four types of noises (l_∞ norm). They can be categorized as below:

1. **Single Feature Single-instance Noise (SFSI)** i.e., perturbing a feature value only at a particular instance (t) by a certain percentage around the actual value.

$$s^{noise} = g_{\varepsilon, s^{noise}}(s) = s + \varepsilon_t \cdot s_t^{noise} \tag{4.4}$$

2. **Single Feature All-instances Noise (SFAI)** i.e., perturbing a specific feature throughout all the time instances by a certain percentage around the actual values of a particular feature.

$$s^{noise} = g_{\varepsilon, s^{noise}}(s) = s + \Sigma_{i=1}^{n} \varepsilon_i \cdot s_i^{noise} \tag{4.5}$$

3. **Multifeature Single-instance Noise (MFSI)** i.e., perturbing all feature values but only at a particular instance (t), following Eq. 4.4 for all features.

4. **Multifeature All-instance Noise (MFAI)** i.e., perturbing all feature values throughout all the instances, following Eq. 4.5 for all features.

4.2.3 Verification Properties

Similar to the existing concepts, as detailed in Sec. 2.3.1, verification properties are categorized into two types: local properties and global properties. A local property is defined for a specific input x at time-instance

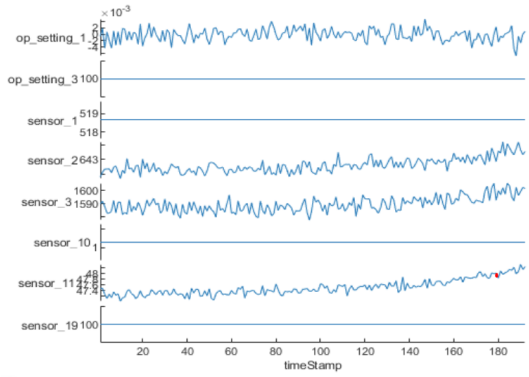

Figure 4.1: SFAI Noise

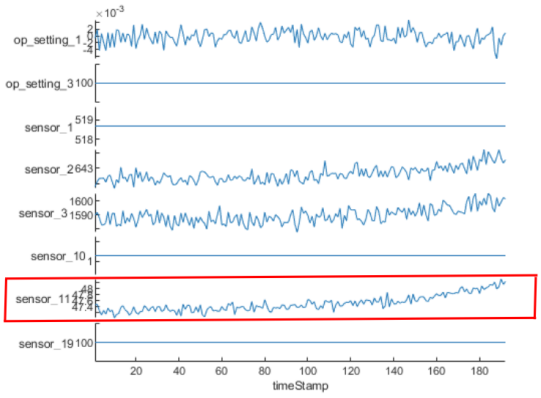

Figure 4.2: SFAI Noise

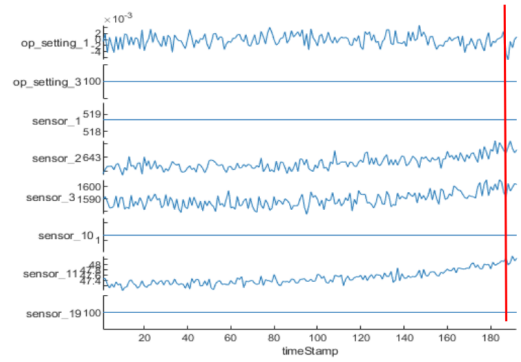

Figure 4.3: MFSI Noise

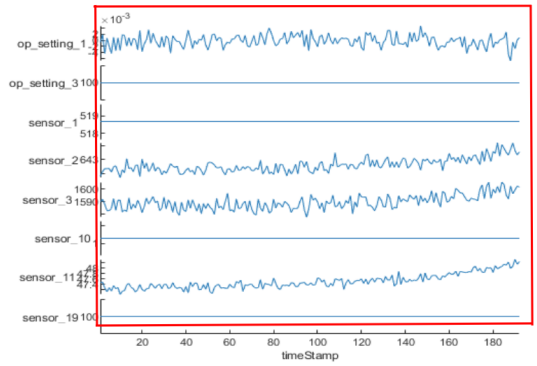

Figure 4.4: MFAI Noise

Figure 4.5: Different Noises Scenarios.

t or a set of points X in the input space $R^{n_f \times t_s}$. In other words, a local property must hold for certain specific inputs. On the other hand, a global property (177) is defined over the entire input space $R^{n_f \times t_s}$ of the network model and must hold for all inputs without any exceptions.

4.2.3.1 Robustness.

A similar concept of local and global robustness as well as robustness measures, as detailed in Sec. **??** is employed for this chapter.

Local Robustness of a TSRegNN

Given a TSRegNN f and an input time series signal S, the network is called **locally robust** to any noise δ if and only if: the estimated output reachable bounds for a particular time-step corresponding to the noisy input lie between predefined allowable bounds w.r.t to the actual signal.

Robustness Value (RV) of a time series signal S is a binary variable, which indicates the local robustness of the system. RV is 1 when the estimated output range for a particular time instance (t) lies within the

allowable range, making it locally robust at t; otherwise, RV is 0.

$RV = 1 \iff {LB_t}^{est} \geq {LB_t}^{allow} \wedge {UB_t}^{est} \leq {UB_t}^{allow}$ else, RV = 0

where ${LB_t}^{est}$ and ${UB_t}^{est}$ are estimated bounds and ${LB_t}^{allow}$ and ${UB_t}^{allow}$ are allowable bounds.

Definition 4.2.4. Percentage Sample Robustness (PR) of a TSRegNN corresponding to any noisy input is defined as

$$PR = \frac{N_{robust}}{N_{total}} \times 100\%, \tag{4.6}$$

where N_{robust} is the total number of robust time instances, and N_{total} = the total number of time steps in the time series signal. Percentage robustness can be used as a measure of **global robustness (177)** of a TSRegNN w.r.t any noise.

For use with time-series input data, this research adapts the concept of Percentage Robustness (PR), originally applied to image-based classification or segmentation neural networks (168). PR, in those cases, assessed the network's ability to correctly classify/segment inputs even with input perturbations for a given number of images/pixels. The concept is similarly employed to analyze the robustness of time-series inputs in the context of this research.

Definition 4.2.5. Percentage Overlap Robustness (POR) of a TSRegNN corresponding to any noisy input is defined as

$$POR = \frac{\Sigma_{i=1}^{N_{total}}(PO_i)}{N_{total}} \times 100\%, \tag{4.7}$$

where N_{total} = total number of time instances in the time series signal, and PO_i is the percentage overlap between estimated and allowed ranges at each time step w.r.t the estimated range

$$PO = \frac{Overlapped\ Range}{Estimated\ Range} \tag{4.8}$$

Here *Overlapped Range* is the overlap between the estimated range and the allowable range for a particular time step. The *Allowable Range* indicates the allowable upper and lower bounds, whereas *Estimated Range* is the output reachable bounds given by the TSRegNN for that time step. Percentage overlap robustness can also be used as a measure of **global robustness (177)** of TSRegNN.

When selecting robustness properties, it is crucial to consider the specific application area. If the application allows for some flexibility in terms of performance, POR can be utilized. On the other hand, if the application requires a more conservative approach, PR should be considered.

Example of Robustness Calculations:

The left picture of Fig. 4.6 depicts an example plot of output estimations (red) vs. the allowable bounds (blue). Here, it can be seen that the network is locally robust for time instances t_1 and t_5; in other instances, it is non-robust w.r.t the noise added. So the RV is 1 for both t_1 and t_5 and 0 for others.

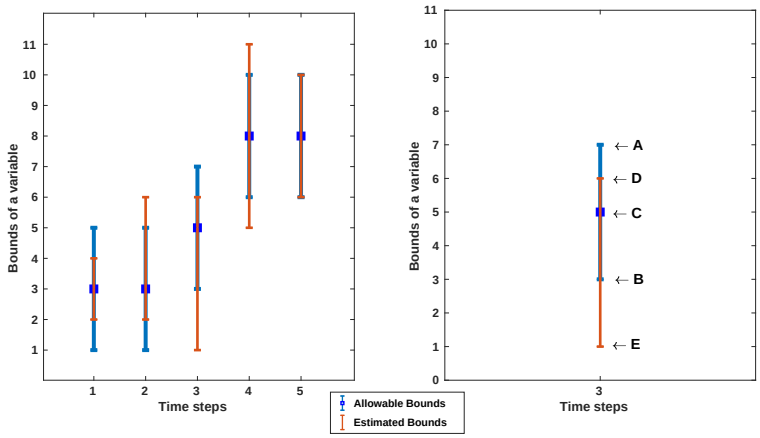

Figure 4.6: Example: (left)Output estimation bounds (red) of a TSRegNN and Allowable (blue) over five consecutive time step and (right) for a particular time step t_3

To better understand the concept of POR and PO, let's refer to the right picture of Fig. 4.6. Here for a time instance t_3, C denotes the actual signal value, AB is the allowable output range, and DE is the estimated reachable bounds. Here, N_{robust} = 3 and N_{total} = 5, so the PR for this particular example is 60%. The PO will be calculated as:

$$PO = \frac{DB}{DE} = \frac{3}{5} = 0.6$$

To calculate POR, the PO for each of the time instances needs to be calculated:

$$POR = \frac{\frac{2}{2} + \frac{3}{4} + \frac{3}{5} + \frac{4}{6} + \frac{4}{4}}{5} \times 100\% = 80.33\%(approx)$$

4.2.3.2 Monotonicity.

In PHM applications, the monotonicity property refers to the system's health indicator, i.e., the degradation parameter exhibiting a consistent increase or decrease as the system approaches failure. PHM involves monitoring a system's health condition and predicting its Remaining Useful Life (RUL) to enable informed maintenance decisions and prevent unforeseen failures. For detailed mathematical modeling of the monotonicity property, please refer to (145) and the latest report on formal methods at (47). In general, for a TSRegNN $f: \mathbf{x} \in \mathbb{R} \rightarrow \mathbf{y} \in \mathbb{R}$ with single-featured input and output spaces, at any time instance t, the property for

monotonically decreasing output can be written as:

$$\begin{aligned} \forall x' \exists \delta : x \leq x' \leq x + \delta \implies f(x') \leq f(x) \\ \forall x' \exists \delta : x - \delta \leq x' \leq x \implies f(x') \geq f(x) \end{aligned} \tag{4.9}$$

This is a local monotonicity property. If this holds true for the entire time range, then the property can be considered as a global property (177). In this work, the monotonicity property is only valid for the PHM examples for RUL estimation.

4.3 Robustness Verification Problem Formulation

For the verification of the robustness and the monotonicity properties, the following problems are considered:

1. **Local Robustness Property:** Given a TSRegNN f, a time series signal S, and a noise δ, prove if the network is locally robust or non-robust [Sec. 5.2.4.2] w.r.t the noise δ; i.e., if the estimated bounds obtained through the reachability calculations lie within the allowable range of the actual output for the particular time instance.

2. **Global Robustness Property:** Given a TSRegNN f, a set of N consecutive time-series signal $\mathbf{S} = \{S_1, \ldots, S_N\}$, and a noise δ, compute the percentage robustness values (PR [Def. 4.2.4] and POR [Def. 4.2.5]) corresponding to δ.

3. **Local Monotonicity Property:** Given a TSRegNN f, a set of N consecutive time-series signal $\mathbf{S} = \{S_1, \ldots, S_N\}$, and a noise δ, show that both the estimated RUL bounds of the network [Eq. 4.9] corresponding to noisy input S'_t at any time instance t are monotonically decreasing.

To get an idea of the global performance (177) of the network, local stability properties have been formulated and verified for each point in the test dataset for 100 consecutive time steps. The core step in solving these problems is to solve the local properties of a TSRegNN f w.r.t a noise δ. It can be done using over-approximate reachability analysis, computing the 'output reachable set' $R_ts = Reach(f, I)$ that provides an upper and lower bound estimation corresponding to the noisy input set I.

This work uses percentage values as robustness measures for verifying neural networks (NN) and performs reachability analysis on the output set to ensure it stays within predefined safe bounds specified by permissible upper-lower bounds. The calculated percentage overlap or sample robustness represents the NN's robustness achieved through the verification process under different noise conditions. The proposed solution takes a sound and incomplete approach to verify the robustness of regression neural networks with time series data. The approach over-approximates the reachable set, ensuring that any input point within the

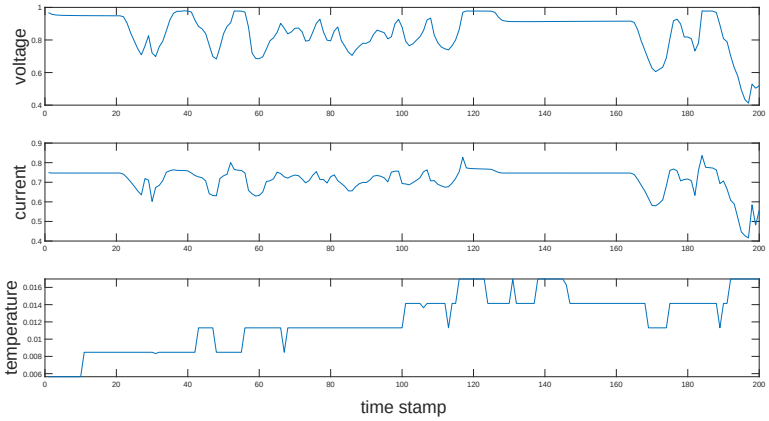

Figure 4.7: Sample feature value plot for Battery SOC Dataset.

set will always have an output point contained within the reachable output set (sound, [Def. 2.4.1]). However, due to the complexities of neural networks and the over-approximation nature of the approach, certain output points within the reachable output set may not directly correspond to specific input points (incomplete, [Def. 2.4.2]).

4.4 Experimental Setup

4.4.1 Dataset Description

For evaluation, two different time series datasets are considered for the PHM of a Li battery and a turbine.

4.4.1.1 Battery State-of-Charge Dataset (BSOC)

Battery state-of-charge is a measurement of the amount of energy available in a battery at a specific point in time, expressed as a percentage. This term is often used in various applications involving battery-powered systems, e.g., electric vehicles, renewable energy storage systems, portable electronics etc. Accurate estimation of the State of Charge (SOC) of a battery is crucial for efficient battery management and ensuring the longevity of the battery. The SOC is expressed as a percentage of the full capacity of the battery.

This dataset is derived from a new 3Ah LG HG2 cell tested in an 8 cu.ft. thermal chamber using a 75amp, 5-volt Digatron Firing Circuits Universal Battery Tester with high accuracy (0.1 of full scale) for voltage and current measurements. The main focus is to determine the State of Charge (SOC) of the battery, measured as a percentage, which indicates the charge level relative to its capacity. SOC for a Li-ion battery depends on various features, including voltage, current, temperature, and average voltage and current. The data is obtained from the 'LG_HG2_Prepared_Dataset_McMasterUniversity_Jan_2020', readily available in the dataset folder (81). The training data consists of a single sequence of experimental data collected while

the battery-powered electric vehicle during a driving cycle at an external temperature of 25 degrees Celsius. The test dataset contains experimental data with an external temperature of -10 degrees Celsius.

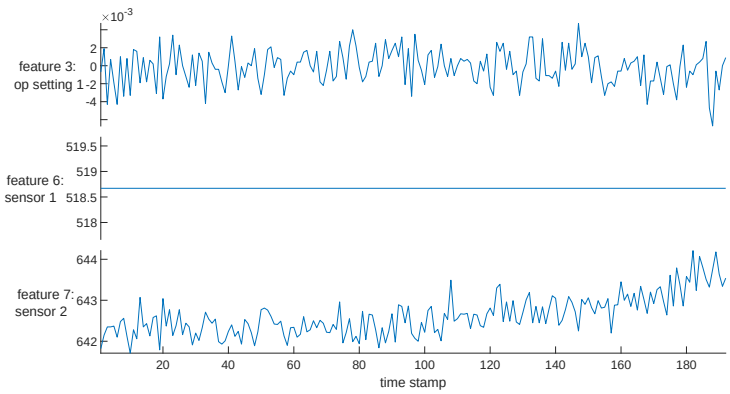

Figure 4.8: Sample feature value plot for Turbofan Engine Degradation Simulation Dataset.

4.4.1.2 Turbofan Engine Degradation Simulation Data Set (TEDS)

The Remaining Useful Life (RUL) is a subjective estimate of the lifespan of any equipment before it requires repair or replacement. This important concept is often used in various fields, including maintenance, reliability engineering, and prognostics and health management (PHM). RUL estimation is typically based on the analysis of historical data, such as sensor measurements, degradation patterns, maintenance records, and operational conditions. Various techniques and models, including statistical methods, machine learning algorithms, and physics-based approaches, are generally used to predict the RUL.

This dataset is widely used for predicting the Remaining Useful Life (RUL) of turbofan jet engines (Pro; 131). Engine degradation simulations are conducted using C-MAPSS (Commercial Modular Aero-Propulsion System Simulation) with four different sets, simulating various operational conditions and fault modes. Each engine has 26 different feature values recorded at different time instances.

1. Feature 1: Unit number
2. Feature 2: Time-stamp
3. Feature 3–5: Operational settings
4. Feature 6–26: Sensor measurements 1–21

To streamline computation, features with low variability (similar to Principal Component Analysis (117)) are removed using ‘prognosability’ to avoid negative impacts on the training process. Feature reduction is done using the ‘prognosability’ MATLAB command. **Prognosability** is actually a property relative to the

prediction of the future state of the system, and this term is mainly used for lifetime data. In MATLAB, 'prognosability' is used as a function to measure the variability of the features in a dataset at failure. The equation for the prognosability calculation is given as below:

$$prognosability = Y = \exp \frac{std_j(x_j(N_j))}{mean_j|(x_j(1) - x_j(N_j)|} \tag{4.10}$$

The output has 3 different outcomes:

1. Y = 0 means the feature values are constant, i.e., no variability in the data.
2. Y = NaN indicates the prognosability could not be calculated.
3. Y = 1 means the feature values are perfectly prognosable i.e., there is variability in the data.

After analyzing the dataset using 'prognosability', number of features considered for NN training reduced to 17 from 26, and they are

1. Feature 3–4 : Operational settings 1-2
2. Feature 7–9 : Sensor measurements 2-4
3. Feature 11–14 : Sensor measurements 6–9
4. Feature 16–20 : Sensor measurements 11–15
5. Feature 22 : Sensor measurements 17
6. Feature 25-26 : Sensor measurements 20-21

The remaining 17 features are then normalized using z-score (mean-standard deviation) for training. The training subset comprises time series data for 100 engines, but for this study, data from only one engine (FD001) are focused. For evaluation, engine 52 is randomly selected from the test dataset.

4.4.2 Network Description

The network architecture used for training the BSOC dataset, partially adopted from (4), is a regression CNN. The network has five input features which correspond to one SOC value. Therefore, the TSRegNN for the BSOC dataset can be represented as:

$$\begin{aligned} f:\ x \in \mathbb{R}^{5\times t_s} &\rightarrow y \in \mathbb{R}^{1\times t_s} \\ \hat{SOC}_{t_s} &= f(t_s) \end{aligned} \tag{4.11}$$

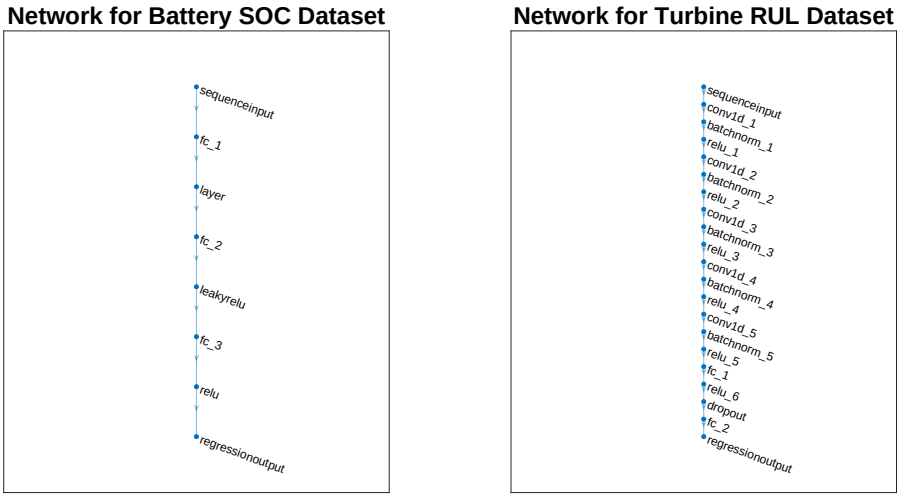

Figure 4.9: The architectures of the regression networks for both the datasets.

The network architecture used for training the TEDS dataset is also a regression CNN, adopted from (6). The input data is preprocessed to focus on 17 features, corresponding to one RUL value for the engine. Therefore, the TSRegNN for the TEDS dataset can be represented as:

$$\begin{aligned} f:\ x \in \mathbb{R}^{17\times t_s} &\rightarrow y \in \mathbb{R}^{1\times t_s} \\ \hat{RUL}_{t_s+1} &= f(t_s) \end{aligned} \tag{4.12}$$

The output's t_s^{th} value represents the desired estimation of SOC or RUL, with the given series of past t_s values for each feature variable.

4.5 Experimental Results and Evaluation

The actual experimental results shown in this chapter are conducted in a Windows-10 computer with the 64-bit operating system, Intel(R) Core(TM) i7-8850H processor, and 16 GB RAM.

For all four noise scenarios [Sec. 4.2.2], local and global (for 100 consecutive time steps) robustness properties are considered for both datasets. The local monotonicity property is only considered for the turbine RUL estimation example.

4.5.1 Battery State-of-Charge Dataset (BSOC)

In this dataset, the output value (SOC) is supposed to be any value between 0 and 1 (or 0 and 100%). But, for the instances where the lower bound is negative, the SOC is instead treated as 0 because a negative SOC

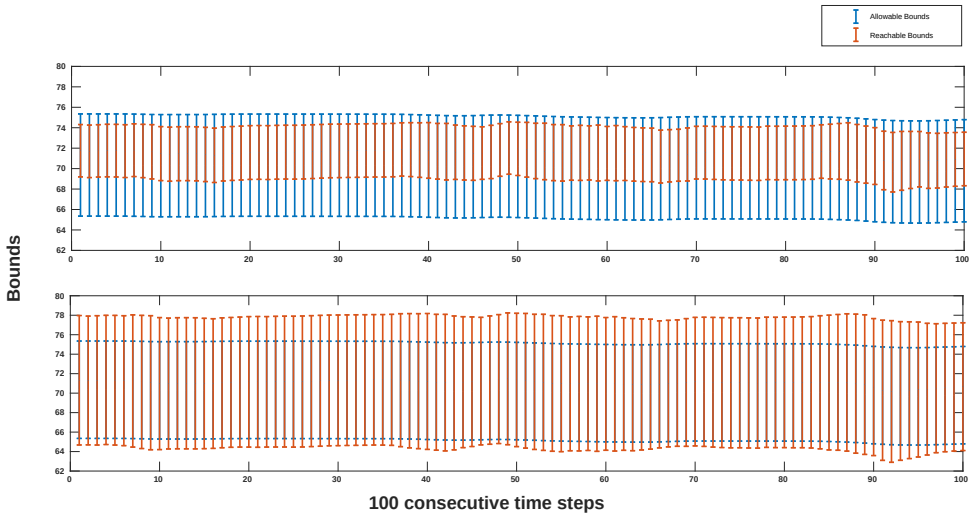

Figure 4.10: Allowable (blue) and reachable (red) bounds for battery SOC dataset for 100 consecutive time steps and two different SFAI noise values 1% (upper), and 2.5% (lower) respectively

does not provide any meaningful implications.

For SFSI, for a random (here feature 3) input feature-signal, the noise is added only at the last time step (t_{30}) of the 3rd feature, whereas for SFAI, noise is added throughout all the time instances of the input signal. The effect of four different noise values, 1%, 2.5%, 5% and 10% of the mean(μ), are then evaluated using over-approximate star reachability analysis [Sec. 4.2.1] on 100 consecutive input signal, each with 30 time instances. Allowable bounds were set at ±5% around the actual State of Charge (SOC) value. For all the noises, two different robustness values, PR [Def. 4.2.4] and POR [Def. 4.2.5], are then calculated, and comparative tables are shown below in Table 4.1.

Table 4.1: Global Robustness: Percentage Robustness(PR) for noises for 100 consecutive time steps

noise	PR_{SFSI}	POR_{SFSI}	$avgRT_{SFSI}(s)$	PR_{SFAI}	POR_{SFAI}	$avgRT_{SFAI}(s)$
1	100	100	0.7080	100	100	20.9268
2.5	100	100	0.7080	100	100	20.9991
5	100	100	0.7116	100	100	21.0729
10	100	100	0.7027	100	100	21.0780
noise	PR_{MFSI}	POR_{MFSI}	$avgRT_{MFSI}(s)$	PR_{MFAI}	POR_{MFAI}	$avgRT_{MFAI}(s)$
1	100	100	0.7653	100	100	36.1723
2.5	0	73.87	0.8251	0	73.87	59.0588
5	0	35.95	0.9026	0	35.95	91.6481
10	0	17.89	1.1051	0	17.89	163.7568

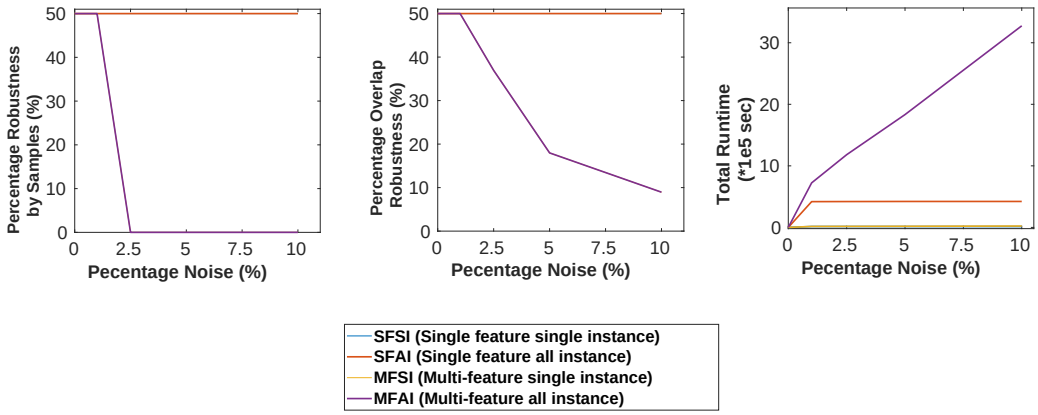

Figure 4.11: Percentage Robustness and Runtime plots w.r.t increasing noise

4.5.1.1 Observation and Analysis

Fig. 4.10 shows a sample plot for gradually increasing estimation bounds with increasing MFSI noise. It can be seen from the figure that for each time instance, the system becomes locally non-robust as the noise value increases.

Table. 8.1 presents the network's overall performance, i.e., the percentage robustness measures, PR [Def. 4.2.4], POR [Def. 4.2.5] and average verification runtime (avgRT), with respect to each noise. The percentage robustness values start decreasing and the average (as well as total) runtime starts increasing as the measure of noise increases for MFAI and MFSI, but for SFSI and SFAI it remains the same for these noise perturbations considered. This is because in the first case, the noise is added to all the features, resulting in increasing the cumulative effect of disturbance on the output estimation. However, in the other case, the noise is attached only to a single feature, assuming that not all features will get polluted by noise simultaneously; and that the reachable bounds are in the acceptable range. A plot of robustness values and the total runtime is shown in Fig 4.11.

It is observed that the decrease in POR values for MFSI and MFAI is less pronounced compared to the PR values as the noise level increases. This is because the PR calculation considers only those time steps where the estimated range is entirely within the allowable range. In contrast, the POR calculation takes into account the fractional contribution of the estimated range, even when part of it extends beyond the allowable limits.

Another interesting observation here is the robustness matrices for both SFSI and SFAI are the same; however, the computations for SFAI take almost three times longer than the computations for SFSI. The

same analogy is observed for MFSI and MFAI datasets but with an even higher time taken for MFAI. The possible reason for this observation could be that, while the data is subjected to perturbations across all time instances, the noise added to the final time step has the most significant impact on the output.

4.5.2 Turbofan Engine Degradation Simulation Data Set (TEDS)

In this dataset, the acceptable RUL bounds are considered to be ± 10 of the actual RUL. For instances where the lower bound is negative, those values are assumed to be 0. Then the percentage robustness measures, PR [Def.4.2.4], POR [Def.4.2.5], and average verification runtime (avgRT) are calculated, for an input set with all 100 consecutive data points, each having 30 time instances. The results for three different noise values, 0.1%, 0.5%, and 1% of the mean (μ), are presented in Table 4.2. For SFSI and SFAI noises, a feature (feature 7, representing sensor 2) is randomly chosen for noise addition. The noise is added to the last time step (t_{30}) of each data sample for SFSI and SFAI noises.

The MFAI noise, i.e., adding the L_∞ norm to all feature values across all time instances, significantly increases the input-set size compared to other noise types. This leads to computationally expensive calculations for layer-wise reachability, resulting in longer run times. Moreover, noise in an industrial setting affecting all features over an extended period is unlikely. Considering these factors, the results of the MFAI noise for the TEDS dataset are excluded from the analysis.

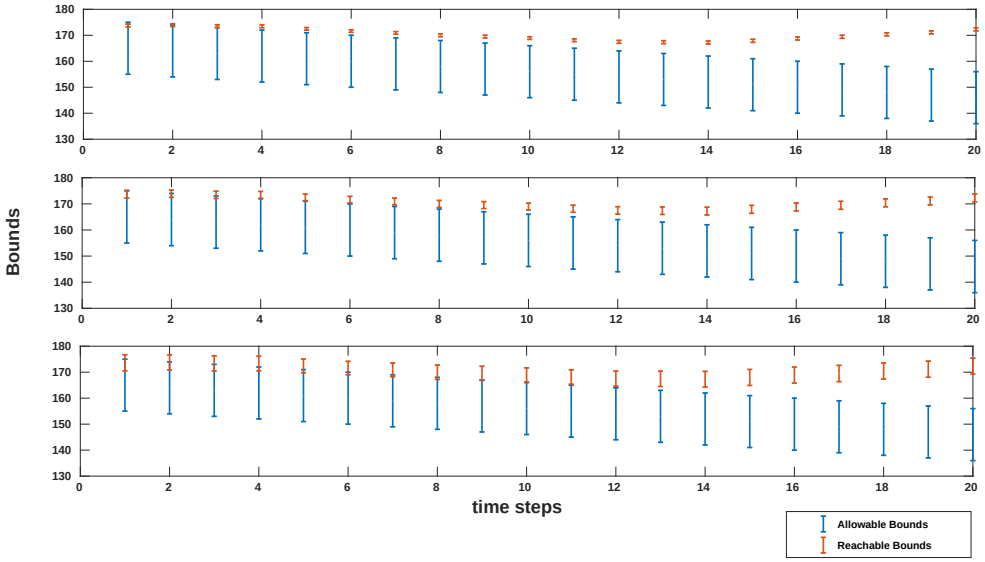

Figure 4.12: Allowable (blue) and reachable (red) bounds for battery SOC dataset for 100 consecutive time steps and three different SFAI noise values 1% (upper), 2.5% (middle) and 5% (lower) respectively

For verifying the local monotonicity of the estimated output RUL bounds at a particular time instance, the previous RUL bounds along with the estimated one have been fitted in a linear equation as shown in Fig.

4.13. This guarantees the monotonically decreasing nature of the estimated RUL at any time instance.

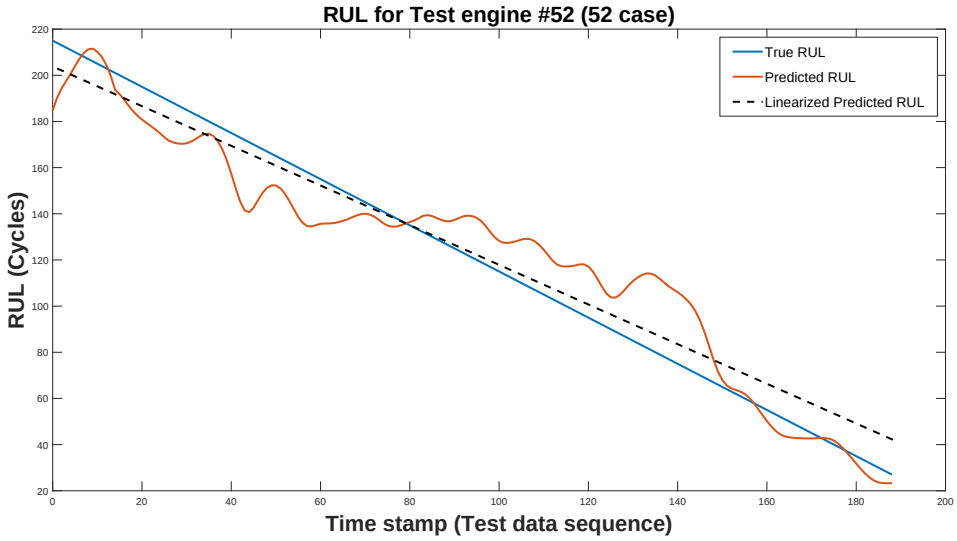

Figure 4.13: Percentage Robustness and Runtime plots w.r.t increasing noise

Table 4.2: Global Robustness: Percentage Robustness(PR) for noises for 100 consecutive time steps

noise	PR_{SFSI}	POR_{SFSI}	$avgRT_{SFSI}(s)$	PR_{SFAI}	POR_{SFAI}	$avgRT_{SFAI}(s)$
1	13	13	1.0796	13	13.31	32.8670
2.5	13	13	1.1755	12	13.13	62.1483
5	13	13	1.2908	8	12.64	108.0736

noise	PR_{MFSI}	POR_{MFSI}	$avgRT_{MFSI}(s)$
1	13	13	9.6567
2.5	13	13	10.2540
5	13	13	11.2100

4.5.2.1 Observation and Analysis

Fig. 4.12 shows a sample plot for gradually increasing estimation bounds with increasing SFAI noise. It is important to note that the network's performance in accurately tracking the actual RUL value is not optimal. However, Table. 4.2 presents the network's overall performance with respect to each noise. Contrary to the other dataset, here the percentage robustness measures corresponding to SFAI and SFSI noises differ. Interestingly, while the noise value increases, the PR, and POR for SFSI remain the same, whereas the robustness measures for SFAI decrease. However, the performance matrices for MFSI are the same as the SFSI except for the time. This might be because, for both SFSI and MFSI, the noise is added only at a single time instance, whereas for SFAI, the noise is added to the entire time instances, resulting in an increased cumulative effect of disturbance on the output.

The results consistently show higher POR values than PR values in Table. [4.1-4.2]. Considering that the

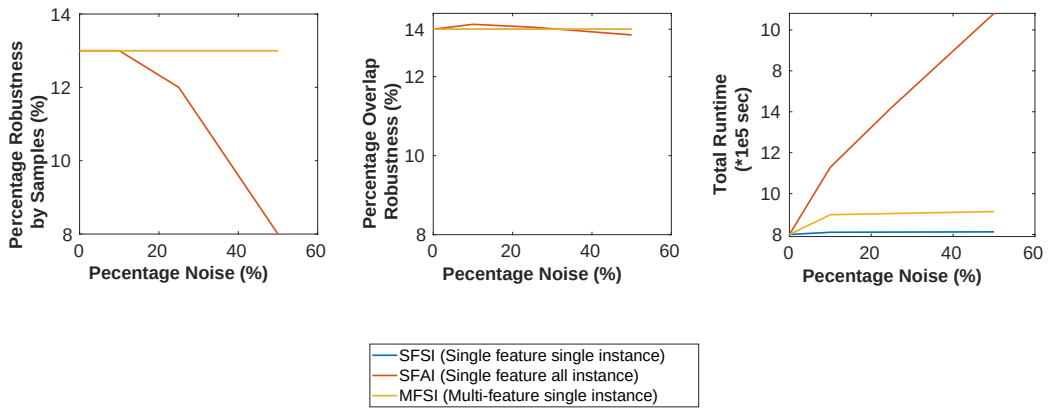

Figure 4.14: Percentage Robustness and Runtime plots w.r.t increasing noise

assessment of output reachable bounds is conducted using L_∞ perturbations in the input, the importance of instances where reachable sets overlap with permissible bounds, yet do not completely fall within them, is also acknowledged. In summary, PR measures adopt a more conservative approach, while POR captures the relationship between output reachable bounds and permissible bounds more accurately.

4.6 Conclusion and Future Work

This chapter explores formal method-based reachability analysis of variable-length time series regression neural networks (NNs) using approximate Star methods in the context of predictive maintenance, which is crucial with the rise of Industry 4.0 and the Internet of Things. The analysis considers sensor noise introduced in the data. Evaluation is conducted on two datasets, employing a unified reachability analysis that handles varying features and variable time sequence lengths while analyzing the output with acceptable upper and lower bounds. Robustness and monotonicity properties are verified for the TEDS dataset.

Real-world datasets are used here, but further research is needed to establish stronger connections between practical industrial problems and performance metrics. The study opens new avenues for exploring perturbation contributions to the output and extending reachability analysis to 3-dimensional time series data like videos. Future work involves verifying global monotonicity properties as well, and including more predictive maintenance and anomaly detection applications as case studies. The study focuses solely on offline data analysis and lacks considerations for real-time stream processing and memory constraints, which present fascinating avenues for future research.

CHAPTER 5

Formal Verification of Long Short-Term Memory based Audio Classifiers: A Star based Approach

This chapter is adapted from the material presented in (113).

5.1 Introduction

Models utilizing NNs for audio classification have found application in diverse tasks, ranging from Music Genre Classification (43; 35; 51) and Environmental Sound Classification (63; 13; 42) to Audio Generation (110; 125). Therefore, formal verification of audio classification systems holds paramount importance in ensuring their reliability and safety, particularly in safety-critical applications such as autonomous vehicles (173; 121), medical diagnosis (68; 103), and industrial monitoring (174).

This study introduces an extension, building upon the foundations laid by two recent studies (163; 115) in the domain of formal verification. The objective is to leverage set-based reachability techniques to verify audio classification models based on the Long Short Term Memory (LSTM) and CNN-LSTM architectures. Drawing inspiration from (163), which highlights the star-based verification of basic vanilla RNNs, and from (115), which demonstrates the formal verification of convolutional neural networks operating on time series data, work shown in this paper amalgamates both concepts. Specifically, it employs two LSTM models and one CNN-LSTM model for these classifications, following the ones depicted in (Mathworks Classify Sound; Mathworks Sequence Classification; Mathworks Sequence Classification1d).

5.2 Preliminaries

This section introduces some basic definitions and descriptions necessary to understand the progression of this paper and the necessary evaluations on audio classification models.

5.2.1 Network Architecture Specifics

5.2.1.1 Long Short Term Memory based Feedforward Architecture

Long Short-Term Memory Feedforward Neural Networks (LSTM FFNNs) combine the sequential data processing strengths of LSTMs with the structural efficiency of FFNNs. In this hybrid architecture, the LSTM layers effectively capture and retain temporal information, addressing the vanishing gradient problem often seen in traditional Recurrent Neural Networks. Subsequently, this processed data is fed into the FFNN component, where it undergoes further transformation. By integrating the temporal acuity of LSTMs with the direct processing power of FFNNs, LSTM FFNNs offer a versatile solution for a range of applications,

particularly those involving sequential and complex input data.

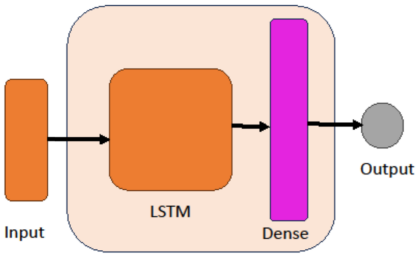

Figure 5.1: Layers of a demo LSTM FFNN Architecture model

5.2.1.2 Convolutional Neural Network + Long Short Term Memory (CNN+LSTM) Architecture

When processing sequences, a CNN uses sliding convolutional filters over the input, extracting information from spatial and temporal dimensions. Conversely, an LSTM network progresses through time steps, capturing lasting connections between them. The synergy of CNN and LSTM layers, as seen in CNN+LSTM architectures (193), harnesses the strengths of both convolutional and LSTM units for insightful data analysis.

The convolutional component forms the foundation for acquiring local feature modules that grasp both local and hierarchical correlations. This fusion enables the identification of intricate data relationships. Additionally, the inclusion of an LSTM layer enhances the network's capacity to capture prolonged dependencies by leveraging information from these localized features.

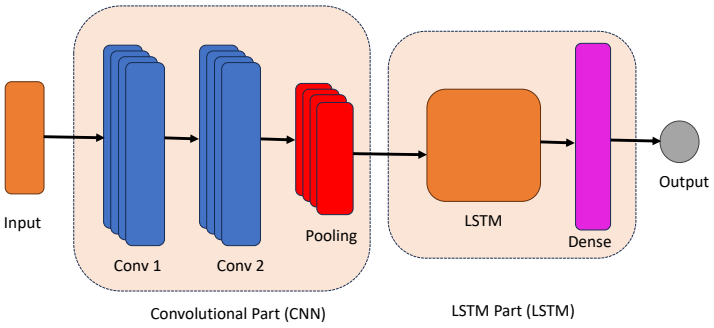

Figure 5.2: Layers of a demo CNN+LSTM Architecture model

5.2.2 Reachability Analysis Computation

This section describes how the reachability of an NN layer and the NN as a whole is computed for this study.

In this context, an alternative technique is adopted for defining a Star set. This method involves utilizing the input's upper and lower bounds with noise, subsequently aligning them around the original input. a

comprehensive array of constraints is established by incorporating these bounds for each input parameter alongside predicates. These constraints are then presented to the optimizer for a solution, ultimately yielding the initial set of states.

For an NN, the output reachable set can be calculated as a step-by-step process of constructing the reachable sets for each network layer.

$$
\begin{aligned}
&R_{l_1} \triangleq \{v_1 \mid v_1 = h_1(x),\ x \in I\},\\
&R_{l_2} \triangleq \{v_2 \mid v_2 = h_2(v_1),\ v_1 \in R_{l_1}\},\\
&\vdots\\
&R_f = R_{l_k} \triangleq \{v_k \mid v_k = h_k(v_{k-1}),\ v_{k-1} \in R_{l_{k-1}}\}
\end{aligned}
$$

where h_k is the function represented by the k^{th} layer L_k. The reachable set R_{L_k} contains all outputs of the neural network corresponding to all input vectors x in the input set I.

5.2.3 Adversarial Perturbation

An audio classification system may face real-world scenarios involving elements like background noise, interference, or distortions. While potentially perceptible, these factors remain within the scope of challenges that practical systems are designed to address. However, this paper exclusively used l-infinity perturbations, focusing on assessing how audio classification models respond to variations within specific constraints.

Considering an input sequence characterized by t_s time instances and n_f features, various perturbation types (l_∞ norm) (115) arise based on their distribution across the sequence. These adversarial perturbation categories (as detailed in Sec. 4.2.2) considered for this work are: (i) **Single Feature Single-instance (SFSI)**, (ii) **Single Feature All-instances (SFAI)**, (iii) **Multifeature Single-instance (MFSI)** and (iv) **Multifeature All-instance (MFAI)**.

5.2.4 Robustness Verification Properties

5.2.4.1 Robustness

To formally articulate the concept of robustness for quantifying the desired classification task, the following formulation can be employed:

$$||x_{adv} - x||\infty < \delta \implies f(x_{adv}) == f(x) \tag{5.1}$$

Here, x signifies the original input from the input space $R^{n_f \times t_s}$, x_{adv} represents the perturbed input, $f(x_{adv})$ and $f(x)$ correspond to the classifiers' outputs for x_{adv} and x, respectively. δ stands for the maximum magnitude

of the introduced perturbation ($\delta \in \mathbf{R} > 0$). By disregarding the softmax and classification layers within the models and focusing on the output of the layer immediately preceding the softmax, the formulation for robustness simplifies as follows:

$$||x_{adv} - x||_\infty < \delta \implies \text{maxID}(g(x_{adv})) == \text{maxID}(g(x)) \tag{5.2}$$

In this context, the function g symbolizes the operation performed by the neural network classifier model until the softmax layer, and maxID denotes the function responsible for identifying the class with the highest value in the output.

5.2.4.2 Verification Properties.

Local Robustness.

Given a sequence classifier f and an input sequence S, the network is called **locally robust** to any perturbation δ if and only if: reachable bounds of the desired class will be max compared to the bounds of the other classes, even in the presence of any perturbation.

Robustness Value (RV) of a sequence S is a binary variable, which indicates the local robustness of the system. RV is 1 when the reachable output range of the desired class is greater than the reachable bounds of other classes, making it locally robust; otherwise, RV is 0.

$RV = 1 \iff LB_{desired} \geq UB_{other}$ else, RV = 0

where $LB_{desired}$ and UB_{other} are the lower reachable bound of the desired class and UB_{other} are the upper bounds of all other classes.

Percentage Robustness (PR).

In this case, the concept of Percentage Robustness (PR) mirrors the one previously applied in image-based classification or segmentation neural networks (168), but it is adapted for sequence audio inputs. The PR for a sequence classifier, corresponding to any adversarial perturbation, is defined as:

$$PR = \frac{N_{robust}}{N_{total}} \times 100 \tag{5.3}$$

where N_{robust} represents the total number of robust sequences, and N_{total} is the overall count of sequences in the test dataset. Percentage robustness can be used as an indicator of **global robustness (177)** with respect to various types of perturbations.

5.3 Experimental Setup

5.3.1 Hardware Used:

The actual experimental results shown in this paper are conducted in a Windows-10 computer with the 64-bit operating system, Intel(R) Core(TM) i7-8850H processor, and 16 GB RAM.

5.3.2 Dataset Description

For evaluation, two different audio datasets are considered for noise classification and Japanese vowel classification.

5.3.2.0.1 Audio Noise Data

To curate this dataset, a collection of 1000 white noise signals, 1000 brown noise signals, and 1000 pink noise signals are generated using MATLAB. Each signal corresponds to a 0.5-second duration and adheres to a 44.1 kHz sample rate. From this pool of 1000 signals, a training set is fashioned, comprising 800 white noise signals, 800 brown noise signals, and 800 pink noise signals. Given the multidimensionality inherent in audio data, often containing redundant information, a dimensionality reduction strategy is employed. The process starts with feature extraction, followed by training the model using only two of the extracted features. These features are generated from the centroid and slope of the mel spectrum over time.

5.3.2.0.2 Japanese Vowel

This dataset (85) is collected from (12) from the University of Irvine Machine Learning Repository. Two Japanese vowels were sequentially pronounced by nine male speakers. A 12-degree linear prediction analysis was subjected to each instance of utterances. Each speaker's utterance constitutes a time series ranging from 7 to 29 points in length, with each point featuring 12 coefficients. For nine classes (i.e., vowels), the dataset has a total of 640 time series. Among these, 270 time series were designated for training purposes, while the remaining 370 were allocated for testing.

5.3.3 Network Description

5.3.3.0.1 Audio Noise Data

The network architecture used for training the audio noise dataset is an LSTM network, partially adopted from (Mathworks Classify Sound). The network has two input features which correspond to one noise type

at the output. Following 5.2, the network for this dataset can be represented as:

$$\begin{aligned} f:\ x \in \mathbb{R}^{2\times l_s} \rightarrow y \in \mathbb{R}^3 \\ \hat{noiseClass} = maxID(g(x)) \end{aligned} \tag{5.4}$$

5.3.3.0.2 Japanese Vowel

Here two different classifiers are trained for the Japanese Vowel dataset. The LSTM architecture is partially adopted from (Mathworks Sequence Classification) and the CNN+LSTM is partially adopted from (Mathworks Sequence Classification1d). Both the networks have twelve input features which correspond to one vowel at the output. Therefore, the networks for this dataset can be represented as:

$$\begin{aligned} f:\ x \in \mathbb{R}^{12\times l_s} \rightarrow y \in \mathbb{R}^9 \\ \hat{vowelClass} = maxID(g(x)) \end{aligned} \tag{5.5}$$

Here l_s is the audio sequence length and the function *maxID* provides the class with the maximum value.

Table 5.1: Performances of different networks used in this paper

Networks	*Accuracy*(%)
audio_noise_lstm	100
japanese_vowel_lstm	93.51
japanese_vowel_cnnlstm	96.49

5.4 Evaluation

5.4.1 Robustness Verification of Audio Noise Classifier

Robustness verification on the audio noise dataset incorporates all four categories of perturbations as outlined in (115). After curating 100 sequences of each white, brown, and pink noise to form test datasets, adversarial sequences are then generated around the original ones by applying l_∞ norms, employing five different perturbation percentage values (ε): 50%, 60%, 70%, 80%, and 90% of the mean (μ) value. These adversarial inputs are then analyzed using an over-approximate star set reachability analysis to evaluate their robustness. For Single Feature Single-instance Noise (SFSI) and Single Feature All-instances Noise (SFAI) perturbations, the strategy includes a random selection of feature 1 for input perturbation.

5.4.1.0.1 Observations and Analysis

Table 8.1 and Fig. 5.3 present the network's overall performance, i.e., the percentage robustness measures, PR [Sec. 5.2.4.2], and total verification runtime (sumRT) in seconds, w.r.t each adversarial perturbation. The observations derived from both the table and the figure provide the following insights:

Table 5.2: Global Robustness: percentage robustness (PR) and total verification runtime (sumRT in seconds) for 100 test audio noise sequences

noise	PR_{SFSI}	PR_{SFAI}	PR_{MFSI}	PR_{MFAI}	$sumRT_{SFSI}$	$sumRT_{SFAI}$	$sumRT_{MFSI}$	$sumRT_{MFAI}$
50	98	80.33	98	80.33	0.3071	0.2626	0.3018	0.2625
60	96	71.67	96	71.67	0.3034	0.2571	0.3018	0.2578
70	94	25.67	94	25.67	0.3039	0.2637	0.3070	0.2663
80	85.33	12.33	85.33	12.33	0.3073	0.2559	0.3111	0.2556
90	63.33	8.33	63.33	8.33	0.3060	0.2504	0.3093	0.2537

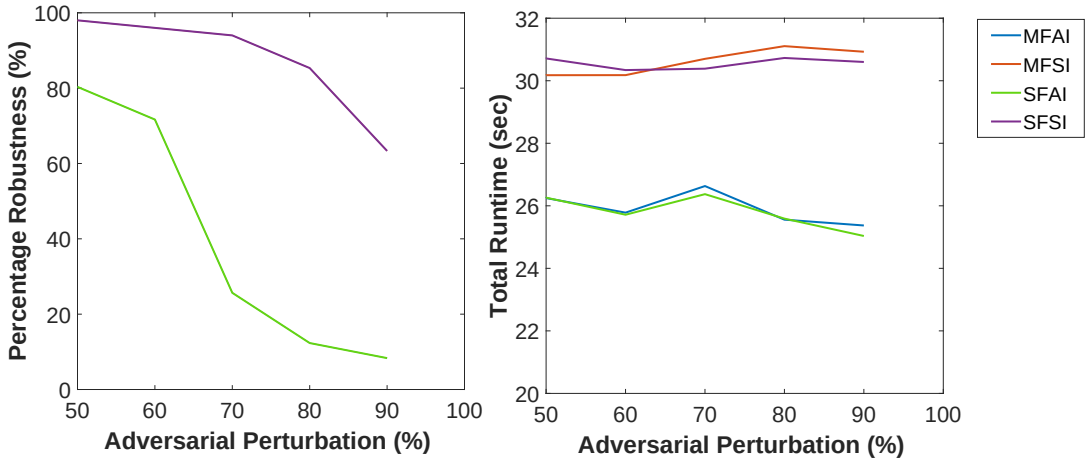

Figure 5.3: Percentage Robustness and Runtime plots w.r.t increasing perturbations

1. **Trend of Percentage Robustness (*PR*).** As the adversary level increases from 50 to 90, a consistent decrease is observed in PR values for all perturbation scenarios (SFSI, SFAI, MFSI, MFAI), which aligns with the concept of the robustness verification property. This decrease in PR signifies a reduction in the system's ability to maintain its classification accuracy in the presence of higher adversary levels.

2. **Comparative Analysis of Perturbation Scenarios.** Within each noise level, comparing PR values across different perturbation scenarios (SFSI, SFAI, MFSI, MFAI), it's evident that PR values for SFSI and MFSI are generally higher than those for SFAI and MFAI. This finding indicates that perturbing features at a single instance or all features at a single instance generally leads to better robustness against varying noise levels.

3. **Similar PR Values for Different Perturbation Scenarios.** Another notable observation is the similarity in robustness matrices between SFSI and MFSI scenarios, accompanied by closely comparable computation times for their respective verification processes. This parallelism is also evident for SFAI and MFAI perturbations as well. This pattern could be ascribed to the dataset's limited feature set of

only two dimensions, where the foremost feature likely holds paramount importance in influencing the class determination in the presence of noise. Consequently, when single-instance perturbations target the first feature, perturbing both features results in an effect akin to perturbing the first feature alone. This interpretation is applicable to both MFAI and SFAI scenarios as well.

5.4.2 Robustness Verification of Japanese Vowel Classifiers

The robustness of LSTM and CNN+LSTM models, within the framework of the Japanese vowel classifier, is verified by extending the evaluation across all four perturbation categories, following the methodology used for the audio noise classifier. The focus is on the entirety of correctly classified test sequences. Adversarial inputs, designed around these original sequences, are generated using l_∞ norms to assess robustness. This involves the application of five distinct perturbation levels (ε), specifically 50%, 60%, 70%, 80%, and 90% of the mean (μ) value. These adversarial inputs are then subjected to over-approximate star reachability analysis for both classifiers to determine their robustness. Similar to the previous case, feature 1 is selected for perturbation in both SFSI and SFAI scenarios.

Table 5.3: Global Robustness: percentage robustness (PR) and total verification runtime (sumRT in seconds) for all test Japanese Vowel audio sequences

noise	PR_{SFSI}	PR_{SFAI}	PR_{MFSI}	PR_{MFAI}	$sumRT_{SFSI}$	$sumRT_{SFAI}$	$sumRT_{MFSI}$	$sumRT_{MFAI}$
50	100	68.21	100	74.86	1.1502	1.0398	1.0392	1.0442
60	100	60.40	100	50.29	0.9989	0.9992	0.9970	0.9966
70	100	50.29	100	27.17	0.9981	0.9968	0.9965	0.9952
80	100	43.93	100	13.01	0.9920	0.9930	0.9978	0.9882
90	100	39.02	100	8.38	1.0044	1.0083	1.004	0.9985

5.4.2.0.1 Observations and Analysis: LSTM Model

Table 5.3 and Fig. 5.4 present the LSTM network's overall performance, i.e., the percentage robustness measures, PR [Sec. 5.2.4.2], and total verification runtime (sumRT), with respect to each adversarial perturbation. The notable findings are outlined as follows:

1. **Trend of Percentage Robustness (*PR*).** Similar to the audio noise classifier, the trends in *PR* values here also suggest that as noise levels increase, the percentage robustness tends to decrease across all scenarios. This aligns with the intuitive expectation that higher adversary levels lead to increased challenges in maintaining robustness.

   The PR_{SFSI} and PR_{MFSI} values remain consistently at 100% across all noise levels, indicating that perturbing either a single feature or all features at a specific instance does not significantly affect

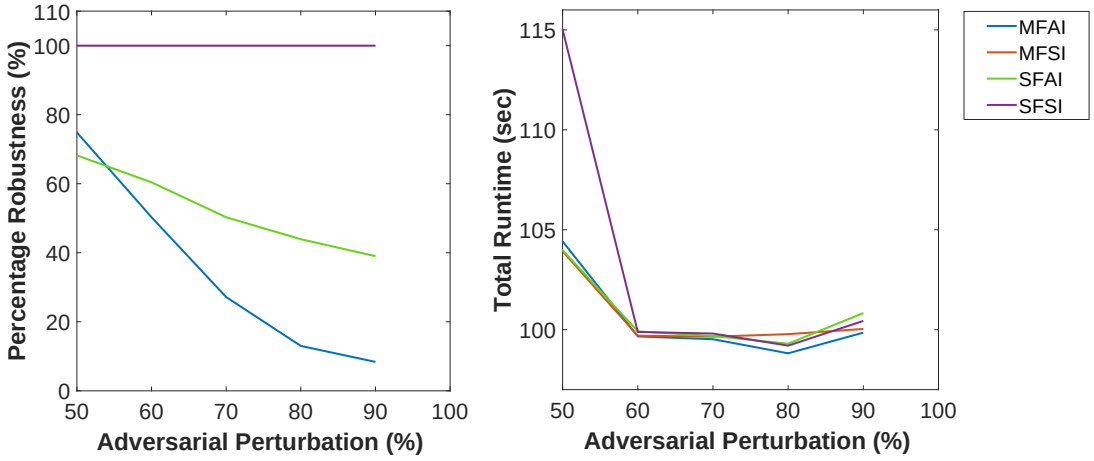

Figure 5.4: Percentage Robustness and Runtime plots w.r.t increasing perturbations, for LSTM architecture

the robustness of the audio sequences. On the other hand, PR_{SFAI} and PR_{MFAI} show distinct trends. As adversary levels increase, PR_{SFAI} gradually decreases, suggesting that perturbing all instances but only a single feature starts impacting the robustness. Similarly, PR_{MFAI} also experiences a decline with increasing noise levels, reflecting that perturbing all instances and features has an impact on the sequences' robustness.

2. **Comparative Analysis of Perturbation Scenarios.** The comparison between single-instance perturbation scenarios ($SFSI$ and $SFAI$) and multifeature perturbation scenarios ($MFSI$ and $MFAI$) reveals a pattern. The former scenarios (single-instance) generally maintain higher robustness compared to the latter (multifeature) scenarios. This suggests that perturbing all features has a larger impact on robustness than perturbing just a single feature.

   The interrelation between PR_{SFSI} and PR_{MFSI} is also notable. Both scenarios exhibit identical trends, regardless of the noise level. Similarly, PR_{SFAI} and PR_{MFAI} also demonstrate similar behaviors, with both scenarios showing a decline in robustness as noise increases.

5.4.2.0.2 Observations and Analysis: CNN+LSTM Model

Table 5.4 and Fig. 5.5 present the CNN+LSTM network's overall performance.

Key insights gleaned from both the table and the plot include:

1. **Trend of Percentage Robustness (PR).** Across all perturbation levels, the PR_{SFSI} remain consistently

Table 5.4: Global Robustness: percentage robustness (PR) and total verification runtime (sumRT in seconds) for all correctly-classified test Japanese Vowel audio sequences

noise	PR_{SFSI}	PR_{SFAI}	PR_{MFSI}	PR_{MFAI}	$sumRT_{SFSI}$	$sumRT_{SFAI}$	$sumRT_{MFSI}$	$sumRT_{MFAI}$
50	96.82	49.13	97.39	65.89	5.5019	4.2007	4.2197	4.1148
60	96.82	40.75	97.39	43.64	4.5148	3.9238	3.9123	3.9200
70	96.82	34.68	97.10	18.78	4.6223	4.0966	4.0778	4.0626
80	96.82	30.63	97.10	3.17	4.5658	4.0998	4.0550	4.0714
90	96.82	26.58	97.10	0	4.5773	4.0785	4.0715	4.0605

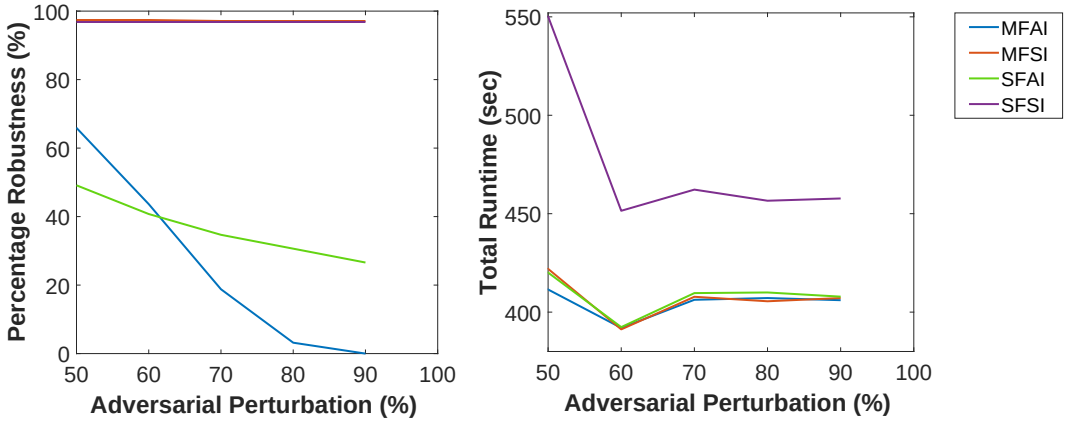

Figure 5.5: Percentage Robustness and Runtime plots w.r.t increasing perturbations, for LSTM architecture

at around 96% and the PR_{MFSI} at around 97%, indicating that the perturbations applied in these scenarios do not significantly affect the robustness of the audio sequences. For SFAI and MFAI perturbations, PR also decreases with rising noise levels, although the decline is more pronounced. PR values for SFSI and MFSI perturbations are significantly higher compared to SFAI and MFAI perturbations at all noise levels, indicating that sequences with perturbations at a single instance are more robust to noise.

2. **Trend of Verification Runtimes (*sumRT*).** Verification runtimes tend to rise with elevated noise levels across all perturbation scenarios. However, in the case of the Japanese Vowel dataset, an initial decrease is observed in the runtime trend, followed by an increase at perturbation level 70% and then again decreases at 80%, followed by another increase at 90%. It's also worth noting that contrary to the expected trend, $sumRT_{SFSI}$ exhibits a higher runtime value in comparison to $sumRT_{SFAI}$ and $sumRT_{MFAI}$.

Overall, the above tables demonstrate how different perturbation scenarios and adversary levels impact the percentage robustness of the audio noise and Japanese Vowel audio classifiers. The trends and interrelations provide insights into the varying effects of perturbations on different scenarios and noise levels, helping to

understand the robustness behavior of the neural network models under different conditions.

5.5 Conclusion and Future Directions

This study delves into formal method-based over-approximate reachability analysis for various LSTM-based neural networks (NNs), specifically in the context of audio sequence classification – a critical aspect for safety-critical applications. The investigation encompasses four distinct adversarial perturbation types, as introduced in the existing literature. The evaluation occurs across two audio sequence datasets: audio noise sequences and Japanese vowel audio sequences. The unified reachability analysis accommodates shifting features within time sequences while scrutinizing the output against the desired audio class. Robustness properties are verified for both datasets.

Exploring real-world scenarios encompassing a wider array of perturbation types and magnitudes will be fascinating to explore, potentially yielding diverse effects on system behavior. Another interesting direction will be to utilize the exact star methods for the reachability analysis, which will capture the outcomes more correctly.

This study paves the way for exploring the impact of perturbations on the output and expanding reachability analysis to three-dimensional sequence data like videos. An intriguing direction for exploration can involve analyzing the peculiar runtime patterns observed in the plots for the Japanese Vowel audio dataset. Potential future applications can also encompass medical video analysis. Notably, this work concentrates on offline data analysis, omitting considerations for real-time stream processing and memory limitations, which offers intriguing prospects for future investigation.

CHAPTER 6

Extending the usability of the Neural Network Verification (NNV) tool

6.1 Introduction

This chapter focuses on the advancement of the Neural Network Verification (NNV) tool (170; 97), addressing the challenges posed by the rapid evolution of neural network architectures and the diversification of their layers. The primary objective is to enhance NNV's capability to support an expanded range of neural network structures and layer types, in response to the increasing complexities and innovations within the field.

The significance of this endeavor is further highlighted by the Neural Network Verification Competition (VNN Comp, as detailed in Sec. 2.6.4, (30; 17; 108)), which sets benchmarks and drives advancements in neural network verification. NNV's participation in VNN Comp underlines the imperative to update and integrate new functionalities for handling sophisticated neural network architectures.

By extending NNV's functionalities, this chapter contributes to maintaining its relevance as a tool for the verification of neural networks, catering to both current and future technological demands. It outlines the methodologies adopted for these enhancements, the obstacles faced, and the strategies implemented to bolster the tool's robustness and adaptability.

6.2 Preliminaries

6.2.1 NNV Structure

The NNV toolbox has two main component: a computation engine and an analyzer. The computation engine has several sub-parts, e.g., several types of NN constructors, NNCS constructor, reachability solver and the evaluator. When a supported network constructor is obtained from the corresponding network file, it is then fed into the reachability solver to get the output reachable set from given upper and lower bounds of the input variables. For reachability analysis of a NN, the output reachable sets are calculated in a layer-by-layer manner and the reachable sets at the final layer are the collection of all possible states of the DNN at the output. The obtained reachable sets are then passed to the analyzer module.

The analyzer section has three sub-parts, visualizer, safety checker and falsifier. While the visualizer is used to plot any reachable set, the safety checker is called to reason about the robustness of any supported NN models w.r.t some given specifications. The safety checker returns either "safe" or "unsafe" along with a set of counterexamples if using exact methods, whereas it returns either "safe" or "uncertain" if using over-approximate methods. Any DNN is considered 'safe' if and only if (iff) there is no intersection between the output sets and the unsafe region, defined by the safety properties (170).

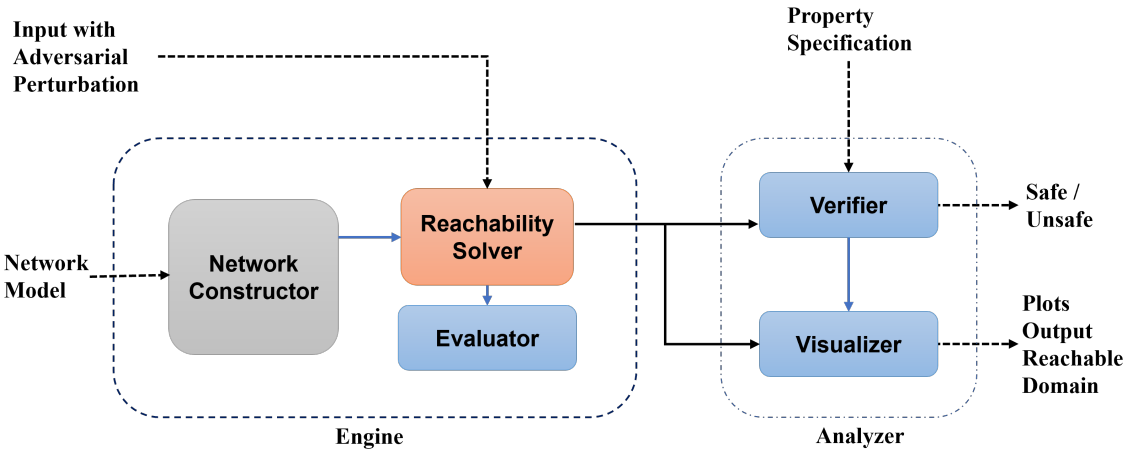

Figure 6.1: Illustration of the internal structure of NNV

6.2.2 Chapter Organization

This chapter adopts a distinct structure compared to other chapters of this thesis. It systematically explores each major contribution, beginning with the necessity for specific layer types and culminating in the proposed solutions as extensions to the NNV tool. This approach ensures a comprehensive and structured presentation of the advancements made within the NNV tool.

6.3 Problem 1: Support for Sequence Input Layer

The evolution of neural networks in MATLAB has been significantly influenced by the increasing need to process sequential data, particularly in fields like natural language processing, time-series analysis, and speech recognition. Traditional neural networks, primarily designed for image processing tasks, utilized the ‘imageInputLayer‘ for handling input data. This layer is adept at processing fixed-size image data, essential for recognizing spatial relationships within images, such as in image classification, object detection, and segmentation. However, the ‘imageInputLayer‘ was not inherently suited for sequential data, which exhibits temporal dependencies and often varies in length.

6.3.1 Primary Differences Between ImageInputLayer and SequenceInputLayer

The fundamental difference between the ‘imageInputLayer‘ and the ‘sequenceInputLayer‘ are give in the following Table. 6.1

6.3.2 Limitations of ImageInputLayer used for Sequential Data

Nevertheless, for managing long sequential data, the same ‘imageinputlayer’ was employed with necessary input modifications, a practice that naturally presented its own set of challenges:

Feature	ImageInputLayer	SequenceInputLayer
Data Type	Optimized for fixed-size image data.	Designed for sequential data of varying lengths.
Primary Use	Image processing tasks such as classification, object detection, and segmentation.	Time-series analysis, natural language processing, and other tasks requiring analysis of sequential data.
Data Handling	Processes data by normalizing images and preparing them for further processing.	Handles sequences, maintaining the integrity of temporal relationships within the data.
Application	Suitable for tasks where spatial relationships within the image are key.	Ideal for applications where temporal dependencies are crucial.

Table 6.1: Comparison of ImageInputLayer and SequenceInputLayer

- This process required segmenting the sequence into fixed-size windows, similar to image processing, and then treating these segments as image inputs.
- Although this method allowed for some processing of sequential data, it had fundamental limitations.
- The fixed-size window approach could not fully capture the dynamic nature of sequential data.
- Additionally, it was ineffective in representing the temporal dependencies and variations in sequence length inherent to this type of data.

Not only for the training of NNs, but even for the formal verification testing, a similar trend is observed. Even in the latest report (47), the time series RUL (remaining useful life) estimation benchmark was designed by the fixed-window approach.

6.3.3 Functionality of Sequence Input Layer

This layer is specifically designed to process data sequences of varying lengths, maintaining the integrity of temporal relationships within the data. Mathematically, the 'sequenceInputLayer' works by accepting input sequences $x_1, x_2, \ldots, x_n$, where each x_i represents a timestep in the sequence, and processing them through a network designed to handle such data (e.g., LSTM or GRU networks). The layer effectively maps these sequences into a feature space suitable for the neural network to analyze temporal patterns and dependencies.

6.3.4 Designing the 'SequeneceInputLayer' for NNV

To enhance user flexibility, support for this layer has been integrated into the NNV framework. For reachability-based verification, an input star set is propagated through this layer, resulting in another star set. This process incorporates normalization within the set dimensions, effectively adapting the layer to function seamlessly within the NNV's verification paradigm.

Subsequent to the 'sequenceinputlayer', the layers typically encountered are the 'fullyconnectedlayer', '1dconvolutionallayer', and similar architectures. Therefore, it is imperative to ascertain whether the sequence length influences the reachability analysis of these layers, contrasting with the impact observed in other configurations.

6.3.4.1 Reachability of 'FullyConnectedLayer' to Allow Variable-Length Time Series Input

A 'fullyconnected' layer can be considered with the following parameters: the weights $W_{fc} \in R^{op \times ip}$ and the bias $b_{fc} \in R^{op \times 1}$, where op and ip are, respectively, the output and input sizes of the layer. The output of this fully connected layer w.r.t an input $i \in R^{ip \times T_s}$ will be

$$o = W_{fc} \times i + b_{fc}$$
$$where\ output\ o \in R^{op \times T_s}$$

Thus, it can be seen that the layer functionality does not alter the output size for a variable length of time sequence, making the functionality of this layer independent of the time series length.
The reachability of a fully-connected layer will be given by the following lemma.

Lemma 6.3.1. *The reachable set of a fully-connected layer with a Star input set $I = \langle c, V, P \rangle$ is another Star $I' = \langle c', V', P' \rangle$ where $c' = W_{fc} \times c + b_{fc}$, the matrix multiplication of c with Weight matrix W_{fc},$V' = \{v'_1, ..., v'_m\}$, where $v'_i = W_{fc} \times v_i$, the matrix multiplication of the weight matrix and the i^{th} basis vector, and $P' = P$.*

6.3.4.2 Reachability of '1dConvolutionalLayer' to Allow Variable-Length Time Series Input

A 1d convolution layer can be considered with the following parameters: the weights $W_{conv1d} \in R^{w_f \times nc \times fl}$ and the bias $b_{conv1d} \in R^{1 \times fl}$ where $w_f, nc\ and\ fl$ are the filter size, number of channels and number of filters, respectively.

The output of this 1d convolution layer w.r.t an input $i \in R^{ip \times T_s}$ will be

$$o = W'_{conv1d} \cdot i' + b_{conv1d}\ \ dot\ product\ along\ time\ dimesion\ for\ each\ filter$$
$$where\ output\ o \in R^{fl \times T'_s}$$

where $T'_s = T_s + T_d - T_{fl}$ is the new time series length at the output and T_d, T_{fl} are the time lengths contributed by the dilation factor and the 1d convolution function, respectively. w'_{conv1d} is the modified weight matrix after adding dilation, and i' is the modified input after padding. When T_d becomes equal to T_{fl} for any convolution layer, the layer functionality becomes independent of the length of the time series.

The reachability of a 1dconvolution layer will be given by the following lemma.

Lemma 6.3.2. *The reachable set of a 1d convolution layer with a Star input set $I = \langle c, V, P \rangle$ is another Star $I' = \langle c', V', P' \rangle$ where $c' = W_{conv1d} \cdot c$, 1d convolution applied to the basis vector c with Weight matrix W_{conv1d},$V' = \{v'_1, ..., v'_m\}$, where $v'_i = W_{conv1d} \cdot v_i$, is the 1d convolution operation with zero bias applied to the generator vectors, i.e., only using the weights of the layer, and $P' = P$.*

6.3.5 Summary

This research guarantees that the sequential characteristics of inputs do not negatively impact the reachability and efficacy of layers following the 'sequenceinputlayer' in neural network models.

The inclusion of this layer type in NNV introduces the capacity to handle variable-length inputs in neural networks, streamlining input handling and augmenting the adaptability of network designs. Contrasting with previous studies that depended on fixed-size windows (47; 108), a method requiring additional preprocessing and size experimentation, this development offers the versatility to accommodate sequences of any length. Such adaptability significantly enhances the applicability and scope of reachability analysis in diverse neural network contexts.

6.4 Problem 2: Support for Long Short Term Memory (LSTM) Layer

Long Short-Term Memory (LSTM) layers, a fundamental advancement in neural network architecture, were developed to address the limitations of traditional Recurrent Neural Networks (RNNs) in handling long-term dependencies. RNNs were adept at processing sequences and capturing temporal information but struggled with the vanishing gradient problem, where the ability to learn long-range dependencies diminishes as the sequence length increases. This led to the development of LSTM layers, which introduced mechanisms to selectively remember patterns over long intervals, thus significantly enhancing the capability to learn from long data sequences.

LSTMs were designed to overcome the shortcomings of vanishing gradients by incorporating a more complex internal structure. Each LSTM unit contains a cell state and three types of gates: input, forget, and output gates. These components work in unison to regulate the flow of information, allowing the network to retain important long-term data and discard irrelevant information, thereby maintaining a stable learning process over time.

6.4.1 Long Short Term Memory (LSTM) Layer

An LSTM layer, a subtype of the RNN layer, excels at capturing long-term dependencies in time series and sequential data (69). It comprises two critical elements: the hidden state (h_t, also called the output state) and

the cell state (c_t). At each time step 't,' the hidden state captures the layer's output for that instance, while the cell state accumulates insights from preceding time steps.

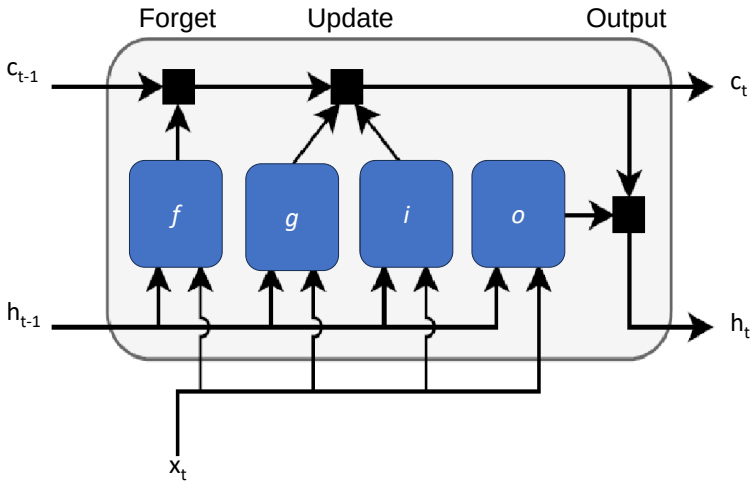

Figure 6.2: The flow of data at time step t in an LSTM layer

$$\begin{aligned} c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\ h_t &= o_t \odot \sigma_c(c_t) \end{aligned} \tag{6.1}$$

During each time step, the layer refines the cell state by incorporating or omitting information. This process is steered by distinct gates that control these adjustments, as shown in Fig. 6.3.

$$\begin{aligned} i_t &= \sigma_g(W_i x_t + R_i h_{t-1} + b_i) \\ f_t &= \sigma_g(W_f x_t + R_f h_{t-1} + b_f) \\ g_t &= \sigma_c(W_g x_t + R_g h_{t-1} + b_g) \\ o_t &= \sigma_g(W_o x_t + R_o h_{t-1} + b_o) \end{aligned} \tag{6.2}$$

In these equations, $\odot$ represents the Hadamard product (element-wise multiplication), σ_c denotes the activation function applied element-wise to the cell state c_t and to the cell state gate g_t; σ_g denotes the activation function applied element-wise to the hidden state gates. Here, W, R, and b are, respectively, hidden state weights, recurrent weights, and biases for each of the gates.

6.4.2 Vanilla Neural Network vs. LSTM Layer

The fundamental difference between Vanilla NN and LSTM Layer are give in the following Table. 6.2

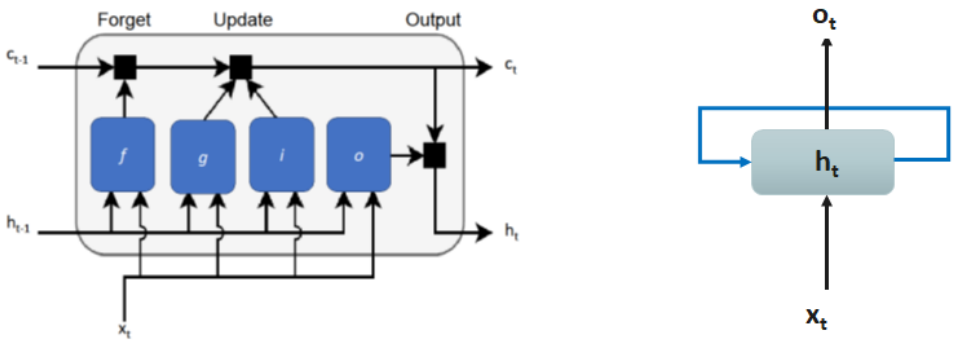

Figure 6.3: The flow of data at time step t in (left) an LSTM layer vs. (right) at Vanilla Network

Feature	**Vanilla RNN**	**LSTM Layer**
Handling Long-Term Dependencies	Struggles due to the vanishing gradient problem, making it less effective for long sequences.	Excellently handles long-term dependencies due to its gating mechanisms.
Architecture Complexity	Relatively simple architecture with fewer parameters.	More complex architecture with a larger number of parameters due to additional gates.
Susceptibility to Vanishing Gradient Problem	Highly susceptible, leading to difficulties in learning long-range dependencies.	Significantly less susceptible due to its ability to maintain gradients over longer sequences.
Ability to Regulate Information Flow	Limited capability to selectively retain or forget information across timesteps.	High capability to regulate information flow through input, output, and forget gates.
Typical Applications	Suitable for shorter sequence tasks, like simple text processing or speech recognition.	Ideal for complex tasks requiring understanding of long-term context, such as advanced NLP, speech recognition, and time-series analysis.

Table 6.2: Comparison of Vanilla RNN and LSTM Layer

6.4.3 Reachability of a Long Short Term Memory Layer

To compute the reachability of an LSTM layer in relation to a star input set S_t, a series of stepwise reachability computations are necessary to ultimately determine the reachable set of the LSTM layer's output, as depicted in Eq. 6.1-6.2. Ensuring accurate results relies on verifying the validity of specific conditions, which are crucial for this process to be sound and accurate:

1. **Affine Mapping Validity.** The transformation of a star set through an affine mapping using a given weight and bias must result in another valid star set (164).

2. **Star Set Summation.** Combining two star sets through Minkowski summation should lead to the formation of yet another valid star set (16).

3. **Activation Function Application.** Upon applying the activation function to a star set, the output

should also result in a star set(s). The outcome could manifest as a single star set or a composition of multiple star sets, contingent on factors such as the activation functions employed and the specific reachability technique utilized (164; 170; 168; 169).

4. **Hadamard Product Validity.** The Hadamard Product of two star sets should yield another valid star set.

While the validity of the first three conditions for star sets has been established in prior research, this current study aims to extend that validation to include the fourth condition as well.

Theorem 6.4.1 (Hadamard product of two star sets)**.** *Given two sets* $\Theta_1 = \langle c_1, V_1, P_1 \rangle$ *and* $\Theta_2 = \langle c_2, V_2, P_2 \rangle$*, the Hadamard product of them* $\bar{\Theta} = \Theta_1 \odot \Theta_2 = \{y \mid y = x_1 \odot x_2,\ x_1 \in \Theta_1,\ x_2 \in \Theta_2\}$ *is calculated by taking the Hadamard product of their generating vectors.*

Characteristics:

1. *The Hadamard product preserves the convexity of star-sets.*
2. *Geometrically, it combines the directions of the generating vectors from both star sets.*
3. *Depending on the linear constraints for each star set, the constraints for the resultant star set is decided. If all the predicates of both the star set have same constraints, then the resultant star set becomes:*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle,\ \bar{c} = c_1 \odot c_2,\ \bar{V} = V_1 \odot V_2,\ \bar{P} \equiv P_1 = P_2$$

If they are not same, then instead of linear constraints, it becomes a quadratic constraint optimization to generate the resultant star set. Then the resultant star set becomes:

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle,\ \bar{c} = c_1 \odot c_2,\ \bar{V} = V_1 \odot V_2,\ \bar{P} \equiv P_1 \odot P_2$$

Therefore, it can be concluded that for a given input set S_t and an LSTM layer, the output is also a star set.

6.4.4 Over-approximate Reachability Analysis for LSTM Layer

Addressing the constraints of quadratic optimization, which is both time-consuming and memory-intensive, especially when followed by the application of non-linear activation functions, necessitates an efficient approach for reachability analysis in terms of both time and memory. An over-approximation method is employed to enhance efficiency.

Rather than precisely computing the star-set after activation functions and quadratic optimization, this method approximates by applying operations directly to the input star-set's upper and lower limits. The equations below illustrate this approach:

$$lb_{out} = lstm(lb_{in})$$

$$ub_{out} = lstm(ub_{in})$$

In these equations, lb_{in} and ub_{in} denote the lower and upper bounds of the input star-set for the LSTM layer, respectively, while lb_{out} and ub_{out} represent the corresponding output bounds. The transformation applied to these bounds is facilitated by the 'lstm' function, streamlining the reachability analysis by focusing on the boundary conditions of the input star-set.

6.4.5 Summary

This work significantly broadens the applicability of NNV, making it more versatile in handling various DNN-based applications that rely on LSTM layers, even if using over-approximate analysis. Furthermore, this development lays the groundwork for future extensions to accommodate other types of RNN layers, such as BiLSTM (Bidirectional Long Short-Term Memory) layers. This expansion not only increases the functionality of the NNV tool but also marks a substantial step forward in the field of neural network verification, especially for models that require sequential data processing.

6.5 Problem 3: Support for Different Skip Connections

Deep neural networks can learn complex functions more efficiently than their shallow counterparts. However, one obstacle for deep learning might be disappearing gradients and/or exploding gradient issues (21; 58), which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization (58; 36; 130; 65) and intermediate normalization layers (73), which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with backpropagation (90).

While training deep neural nets, the performance of the model drops with the increase in depth of the architecture. This is known as the degradation problem. The major reason for the degradation problem is overfitting. Unexpectedly, here, degradation is not caused by overfitting, and adding more layers to a suitably deep model leads to higher training error, as reported in (64; 147) and thoroughly verified in (66). In (66), the authors introduced a residual block to tackle this degradation problem, which is not only easy to optimize compared to its original counterpart, but it also does not add extra parameters or computation complexity. The residual block added here is a type of shortcut connection that skips a layer or more and feeds back to a

future layer. These are also called skip connections.

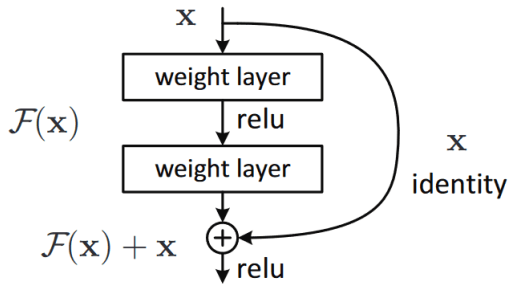

Figure 6.4: Skip Connection: Residual Block (66)

6.5.1 Skip Connections

Skip connections in deep architectures, as the name suggests, skip one or more layers in the neural network and feed the output of one layer as the input to the next layers (instead of only the next one).

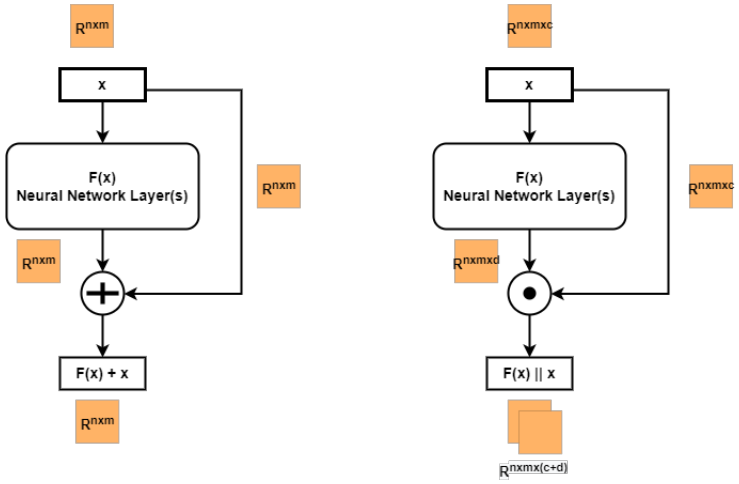

Figure 6.5: General Skip Connection, **left**: signifies addition (no dimensionality change) and **right**: concatenation operation (dimension only changes in the third dimension, i.e., channel-wise concatenation)

Generally, skip connections can be used through the non-sequential layer in two ways: addition, as in residual architectures, and concatenation, as in densely connected architectures.

6.5.1.1 Residual Networks (Res-Nets): Short Skip Connections using Addition

Res-net was first introduced in (66) and the architectures present in the paper has won the 1st place on the ILSVRC 2015 classification task. In a Res-net (Fig. 6.6), multiple residual blocks (Fig. 6.4) are added to construct a complete network. In a residual block, $H(\mathrm{x})$ is assumed to be an underlying mapping to be fit by a few stacked layers (not necessarily the entire net), with x denoting the inputs to the first of these layers,

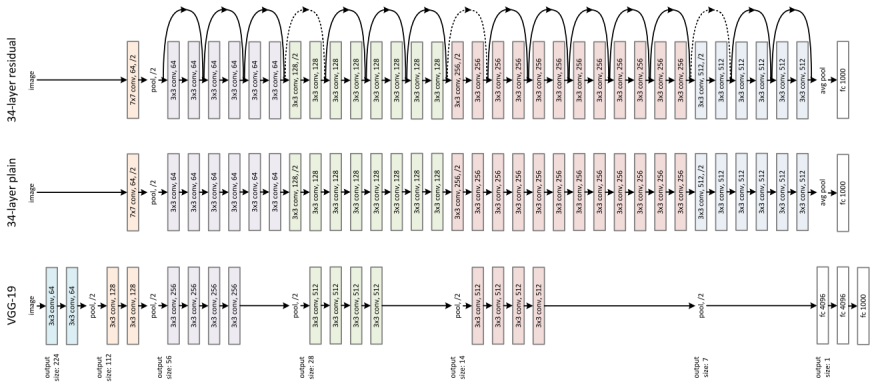

Figure 6.6: Res-net Architecture: short Skip Connections using addition (66)

where $H(x) = F(x)–x$, and $F(x)$ is the residual function.

6.5.1.2 U Net: Long Skip Connections using Concatenation

Due to the limited size of available dataset, sometimes the success of the CNNs get limited, specially in the biomedical image processing. The main use of CNNs are also in classification tasks, but the main focus for biomedical tasks are localization or semantic segmentation. Following the work of (96), the authors of (126) modified and extended the architecture so that it can work with very few training images and yields more precise segmentations. Here, high resolution features from the contracting path are combined with the upsampled output. A successive convolution layer can then learn to assemble a more precise output based on this information, as shown in Fig. 7.1. As very little training data available, excessive data augmentation is done by applying elastic deformations to the available training images.

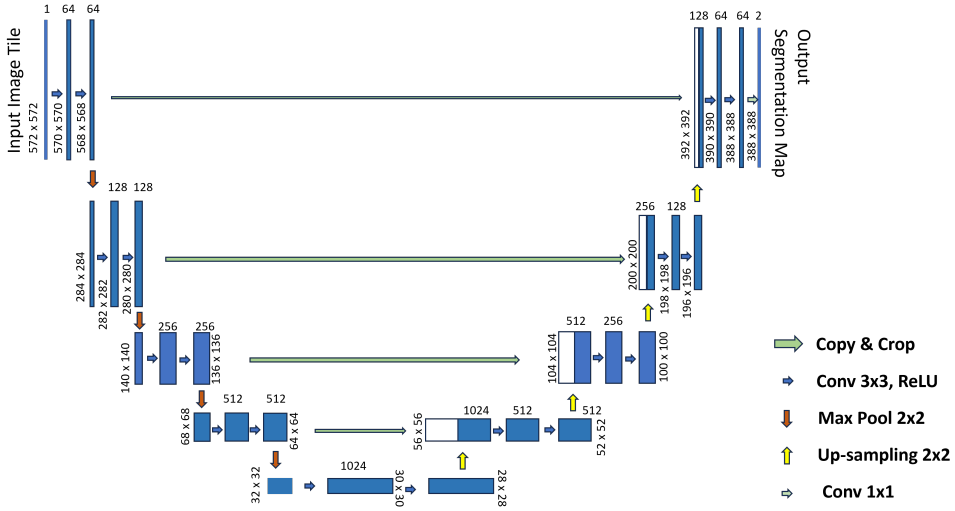

Figure 6.7: U-net Architecture: long Skip Connections using concatenation (126)

6.5.1.3 Densely Connected Convolutional Networks (DenseNets): Skip Connections using Concatenation

Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer, the network has L(L+1)/2 direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. The proposed architecture is usually evaluated on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance.

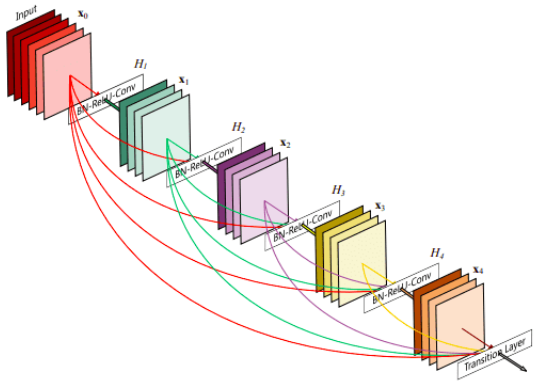

Figure 6.8: DenseNet Architecture Skip Connections using concatenation (71)

6.5.2 Problem Statement

The Neural Network Verification (NNV) tool currently offers support for a variety of deep neural network architectures, including feedforward, convolutional, semantic segmentation, and recurrent networks. A key area for enhancement is the integration of support for skip connections. This involves modifying the fundamental set-based representations within the existing framework of the NNV tool to accommodate these connections.

The motivation behind the support of skip connections is that they have an uninterrupted gradient flow from the first layer to the last layer, which tackles the vanishing gradient problem. Concatenative skip connections ensure feature reusability of the same dimensionality from the earlier layers and are also helpful when there is a limited dataset available. On the other hand, long skip connections are used to pass features from the encoder path to the decoder path in order to recover spatial information lost during downsampling.

Short skip connections appear to stabilize gradient updates in deep architectures. Finally, skip connections enable feature reusability and stabilize training and convergence.

As detailed in Section 2.5.2, reachability analysis in neural networks, given specific input bounds, is conducted on a step-by-step basis. When the network input is a star or imagestar set, it leads to the generation of corresponding sets in each layer, culminating in the final reachable set at the output layer.

Furthermore, as depicted in Fig. 6.5, skip connections primarily function to add or concatenate the output from one layer to a subsequent layer. In the case of addition, the dimensions of both layers are identical. However, in concatenation scenarios in Dense-Net architectures, while two dimensions (height and width) remain constant, the output from a previous layer is concatenated channel-wise, altering the third dimension.

So, the primary problem in incorporating skip connections within NNV lies in facilitating the addition and concatenation operations for two imagestar/star sets and proving that the result of these operations also forms a valid set. Successfully addressing this issue will lead to the incorporation of 'additionlayer' and 'concatenationlayer' within NNV. This enhancement is crucial for facilitating the robustness verification of various DNNs that utilize skip connections, thereby expanding the scope and applicability of the NNV tool in verifying complex network architectures.

6.5.3 Reachability of 'additionlayer' in NNV

The support of 'additionlayer' is equivalent to prove that the summation of two star sets is also a star set. Therefore the concept of Minkowski sum can be directly applied here (16) for this specific layer.

Theorem 6.5.1 (Addition of two Stars)**.** *Given two star-sets* $\Theta_1 = \langle c_1, V_1, P_1 \rangle$ *and* $\Theta_2 = \langle c_2, V_2, P_2 \rangle$*, the addition of them* $\bar{\Theta} = \Theta_1 + \Theta_2 = \{y \mid y = x_1 + x_2,\ x_1 \in \Theta,\ x_2 \in \Theta_2\}$*is another star with the following characteristics.*

$$\bar{\Theta} = \langle \bar{c}, \bar{V}, \bar{P} \rangle,\ \bar{c} = c_1 + c_2,\ \bar{V} = V_1 + V_2,\ \bar{P} \equiv P_1 + P_2.$$

6.5.4 Reachability of 'concatenationlayer' in NNV

The support of 'concatenationlayer' is equivalent to prove that the concatenation of two imagestar sets is also a imagestar set. For the concatenation function, the imagestar set is used as an example, as the dense-net mainly focuses on image segmentation data.

Theorem 6.5.2 (Concatenation of two Imagestars)**.** *Given two ImageStar-sets* $\Theta_1 = \langle c_1, V_1, P_1, l_1, u_1 \rangle$ *and* $\Theta_2 = \langle c_2, V_2, P_2, l_2, u_2 \rangle$*, the channel-wise concatenation of them,* $\bar{\Theta} = \Theta_1 || \Theta_2 = \{y \mid y = x_1 || x_2,\ x_1 \in \Theta,\ x_2 \in \Theta_2\}$ *is another ImageStar with the following characteristics.*

$$c' = [c_1; c_2],\ V' = [V_1, 0; 0, V_2],\ \ P' \equiv [P_1; P_2],\ \ l' \equiv [l_1; l_2],\ \ u'. \equiv [u_1; u_2].$$

6.5.5 Summary

The work enhances NNV's functionality by adding support for a wide range of DNNs that incorporate various types of skip connections. The inclusion of 'additionlayer' and 'concatenationlayer' within NNV is a key aspect of this enhancement, enabling the tool to effectively handle the unique structural complexities introduced by skip connections. This expanded capability ensures that NNV remains at the forefront of neural network verification tools, capable of addressing the evolving needs of complex DNN architectures.

CHAPTER 7

Experiment on Robustness Verification of CNN Models against Multiple Attacks using Imagestar Approach

7.1 Introduction

The advent of deep neural networks (DNNs) in the late 1900s, and their integration into Artificial Intelligence in the early 2000s, marked a transformative era in AI. Their widespread application in critical areas like biometric authentication and malware detection underscores the importance of safety and security in their design. The discovery in 2013 by Szegedy et al. (152) that well-trained CNNs could be deceived by minor, humanly imperceptible perturbations, leading to significant output errors, brought to light the concept of adversarial examples and attacks, posing substantial risks to information integrity and safety. This vulnerability of DNNs to adversarial manipulations has amplified the need to study different attack models and assess their impact on network robustness. The high sensitivity of DNNs to specific adversarial attacks can have a large impact on the actual performance of the model, even if the model accuracy is pretty high. Thus, a detailed study of these attacks can provide better insights regarding the robustness of a DNN model.

Apart from the l_∞ norm, NNV-based robustness verification has also put some focus on the Fast Gradient Sign Method (FGSM) attack (164). This work extends this analysis to include ten specific adversarial attacks executed on VGG16 and another CIFAR-10 (83) based CNN using the same NNV tool, and the attacks are formulated with the help of the Foolbox tool (123).

7.2 Preliminaries

7.2.1 Adversarial Attacks & Foolbox

There are many attack strategies available in the literature, and in this work, the focus is on ten different approaches. The ten attack types are selected to cover a variety of effects on the network parameters, as well as on the availability of open-source software libraries that can implement them. For the experiments in this study, a Python-based adversarial attack generating tool called **Foolbox**(123) was used. These attacks were then used to check the robustness of **VGG16** and **another CIFAR-10 based CNN** model using **NNV** tool.

The ten chosen adversarial attacks can be broadly classified into 3 categories, depending on how they are calculated:

1. **Gradient-based attacks:** This attack perturbs the image with the gradient of the loss function w.r.t the image, gradually increasing the magnitude unless the image is misclassified.

2. **Score based attacks:** Instead of gradients, this attack focuses on some scores (e.g., probabilities, logits etc.) to approximate the gradients

3. **Decision-based attacks:** This one relies only on the class decision of the model. It tries to find the minimum extent of perturbation that affects the model decision.

Adversarial Attack Category	Attacks Considered for This Chapter
Gradient-based	Fast Gradient Signed Method (59), Basic Iterative Method (86), Limited-memory BFGS, Deep Fool Attack (107), Jacobian-Based Saliency Map (116), Carlini Wagner Attack (34).
Score-based	Single Pixel Attack.
Decision-based	Boundary Attack, Pointwise Attack, Gaussian Blur Attack.

Table 7.1: List of Attacks Considered for This Work

Following Table 7.1, the attack descriptions are given as below:

7.2.1.1 Fast Gradient Signed Method (FGSM)

Goodfellow et al. introduced the concept of **Fast Gradient Signed Method** (59) as an attack generation technique. The basic idea was to calculate the cost $J_\theta(x,t)$ used to train the neural network and linearize it around the current value of θ to get a perturbation of for input x and target class t:

$$\eta = \varepsilon sign(\triangle_x J_\theta(x,t)) \tag{7.1}$$

Where ε is the extent of the perturbation. FGSM changes the pixel to the opposite direction, which minimizes the loss function.

7.2.1.2 Basic Iterative Method (BIM)

The Basic Iterative Method is an extension of the FGSM attack. Kurakin et al. (86) ran a finer optimization for multiple iterations, and in each iteration, the change in pixel values is bounded by a clipping function

$$Clip_{x,\varepsilon}\{x'\} = min(255, x+\varepsilon, max(0, x-\varepsilon, x')) \tag{7.2}$$

In each iteration, the generated adversarial examples are given as

$$x_{n+1} = Clip_{x,\varepsilon}\{x_n - \varepsilon sign(\triangle_x J_\theta(x_n,t))\} \tag{7.3}$$

7.2.1.3 Limited-memory BFGS (LBFGS)

Limited memory BFGS (Broyden, Fletcher, Goldfarb, Shanno) was the first of any adversarial attacks ever mentioned in the literature. Szegedy et al. (152) used L-BFGS-B, a second-order optimizer, to find the minimum distance between the input and the adversarial as well as the cross-entropy between the predictions for the adversarial and the one-hot encoded target class(Foo).

7.2.1.4 Deep Fool Attack

Moosavi-Dezfooli et al. (107) presented DeepFool to compute the minimum distance to reach the class boundary by approximating the model classifier with a linear classifier to overcome the non-linearity in high dimension. The target class for the adversarial sample becomes the one with the minimum distance, except for the real class. In the Foolbox there are two types of DeepFool attack available; one calculates the l_2 norm and the other l_∞.

7.2.1.5 Jacobian-Based Saliency Map Attack (JBSMA)

The authors Papernot et al. introduced an efficient saliency-based adversarial map, JBSMA, in their paper (116). They proposed a systematic approach emanating from the (forward derivative, defined as the) Jacobian matrix of the function F, which is learned by the neural network during training.

The computed Jacobian matrix is given as

$$J_F(x) = \frac{\partial F(x)}{\partial x} = \left[\frac{\partial F_j(x)}{\partial x_i}\right]_{i \times j} \tag{7.4}$$

In this study, the gradients are used to get a saliency score, which helps to identify the input features that impact output classification the most.

7.2.1.6 Carlini Wagner Attack

Carlini and Wanger (34) modified the JBSMA approach by using the output of the softmax layer as F and introduced a targeted attack with a motive to defense Defensive distillation. They define a modified objective function to better optimize the penalty and the distance.

The authors introduced 3 variations of attacks - l_0, l_2 and l_∞.

7.2.1.7 Single Pixel Attack

This type of attack utilizes the inherent features of Differential Evaluation, a population-based optimization algorithm for solving complex multi-modal optimization (148; 37). Su et al. showed that under the extremely

limited scenario perturbing only a single pixel can generate attacks and it requires the only information of the probability labels and using that it can perturb most of the inputs to at least one of the target classes.

7.2.1.8 Boundary Attack

A boundary attack is the first effective decision-based adversarial attack, which minimizes the L2-norm of adversarial perturbations to target the class that has the most number of predictions. In case of a boundary attack, the algorithm is initialized from an adversarial point, and then it takes a random walk along the boundary between the adversarial and non-adversarial region such that the distance between the target is reduced but remaining in the adversarial zone (29).

7.2.1.9 Pointwise Attack

Pointwise Attack is also a decision-based adversarial attack. The concept for this attack is similar to the boundary attack, the only difference being that the Pointwise attack minimizes the L0 norm of adversarial perturbations.

7.2.1.10 Gaussian Blur Attack

In Gaussian Blur attack, a line search is performed to find the minimal blur required to misclassify the input image.

7.2.2 Representation of Adversarial Attacks as Imagestars

The Foolbox takes the network model and an input image and generates the adversarial image depending on the attack chosen. Then, the disturbed image is rendered by the following expression (159):

$$inpImage = oriImage + (l + \delta) \times (advImage - oriImage) \tag{7.5}$$

where

- oriImage = the original image
- advImage = the adversarial image from Foolbox
- inpImage = the original image with l percent of the attack
- l = percentage attack on the original image
- δ = small perturbation around l always bounded by $0 \leq \delta \leq \delta_{max}$

7.3 Experimental Setup

7.3.1 Dataset Details

7.3.1.1 CIFAR-10

The data used for training and testing the classification algorithm has been derived from one of the most abundantly used and publicly available **CIFAR-10 (Canadian Institute For Advanced Research)** dataset (83). CIFAR-10 is a labeled subset of the 80 million tiny image datasets. This dataset contains a total of 60000 color (RGB) images of 10 different classes. Out of 60000 images, 50000 are used for the training and 10000 for validation. The classes available in CIFAR-10 are as follows-

- Class 0: airplane
- Class 1: automobile
- Class 2: bird
- Class 3: cat
- Class 4: deer
- Class 5: dog
- Class 6: frog
- Class 7: horse
- Class 8: ship
- Class 9: truck

The classes are completely mutually exclusive. Also, here is no overlap between automobiles and trucks. "Truck" includes only big trucks and "Automobile" includes four-wheelers other than trucks. Neither includes pickup trucks (CIF).

7.3.1.2 ImageNet

This dataset, (129; Ima), used for the VGG16 network, provides a large-scale, diverse collection of images for object recognition research. It is an open-source database consisting of over 14 million labeled images segregated into over 100,000 categories arranged in a hierarchical structure. This rich diversity of images makes it ideal for training and benchmarking advanced image recognition algorithms, especially in developing CNNs. ImageNet gained prominence through the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a competition that significantly advanced deep learning methodologies. It has been instrumental in the success of renowned models like AlexNet (84), VGG (137), and ResNet (66), firmly establishing its role as an essential benchmark in evaluating and driving progress in artificial intelligence and machine learning technologies.

7.3.2 Network Architectures

7.3.2.1 CNN for CIFAR-10

The CNN architecture used for this image classification task is described in this section. It comprises the following details:

1. image input layer with dimension = image size of Cifar-10 dataset (32 X 32X 3)
2. 3 sets of [2D convolution-2D convolution-max pool] layers with output dimension 32 X 32 X3 for 1st set, 64X64X3 for 2nd set and 128X128X3 for the 3rd one; activation = 'relu'; filter dimension = 3 X 3; padding="same"; pool size = 2 X 2 and strides= 2 X 2
3. 2 fully connected layers with dimensions 128 and 64 respectively and activation = 'relu'
4. dropout layer with keep probability = 0.50
5. 1 fully connected layer with dimension = number of classes(i.e. 10) and activation = 'softmax'
6. final classification layer

7.3.2.2 VGG16

The VGG16 (137) network is a prominent CNN architecture developed by the Visual Graphics Group (VGG) at the University of Oxford and introduced in the 2014 ILSVRC competition, and it has been proven highly effective in image recognition tasks. VGG16 consists of 16 layers, including 13 convolutional layers followed by three fully connected layers. Its architecture is characterized by its use of small, 3x3 convolutional filters throughout, which allows it to capture fine details from the input images. VGG16 also employs max pooling layers to reduce the spatial dimensions of the feature maps, leading to a significant reduction in the number of parameters. This design has set a precedent for the construction of deep CNNs. Over the years it a popular choice for various applications, including image classification and object detection, and it continues to serve as a benchmark model for testing new computer vision techniques.

7.4 Experimental Results

7.4.1 Hardware Used

The experiments are performed on a PC with Intel Core i7 CPU and 8 GB RAM.

7.4.2 Evaluation

Performing the tests involved in generating the attack-specific adversarial inputs in Python, followed by the reachability and robustness checking against particular attacks in Matlab.

As explained previously, **the purpose of this chapter is to go beyond the actual results obtained from the experimental examples and draw out some of the broader platform-specific considerations**. These areas can then be investigated to enhance the verification framework to enable a more involved analysis of any deep learning model.

Thus, the results obtained from running the NNV tool on both networks are reported first, followed by a wider overview of the challenges encountered while putting together the various aspects of the study using NNV. They form the principal takeaways from the whole enterprise, leading to a possible road map for future improvements.

7.4.2.1 Robustness Analysis of the VGG16 Model

Ten different attacks, readily implementable from Foolbox are applied to the 'bell pepper' image in VGG16.

Figure 7.1: Example of FGSM Attack on 'Bell Pepper' image from Imagenet

Two tables are shown for two different values of δ with the convention of '1', meaning the network is robust to the corresponding *'l'*, and '$\sim$' indicates robustness analysis could not be performed. Some special cases have been encoded with a '$*$.' Two different analyses (Exact and approximate) are considered for robustness checking. If a system is found to be robust in exact analysis, it ensures the network's robustness. However, being robust in Approx analysis does not always ensure the robustness of the network.

Table 7.2: Robustness checking of the VGG16 network with $\delta = 0.0000001$ using NNV

Attacks	**l(%)**	δ	**Exact Analysis Robust**	**Approximate Analysis Robust**
Fast Gradient Sign Method	≤ 98	0.0000001	1	1
DeepFool	≤ 99	0.0000001	1	1
Limited-memory BFGS	0	0.0000001	0	2
Jacobian-Based Saliency Map Attack	100	0.0000001	1	1
Basic Iterative Method	≤ 99	0.0000001	1	1
Carlini Wagner Attack	< 97	0.0000001	1	1
Single Pixel	$\sim$	0.0000001	$\sim$	$\sim$
Boundary Attack	$\sim$	0.0000001	$\sim$	$\sim$
Pointwise Attack	> 0	0.0000001	0	2
Gaussian Blur	*oom*	0.0000001	out of memory	out of memory

Table 7.3: Robustness checking of VGG16 network with $\delta = 0.0000002$ using NNV

Attacks	l(%)	δ	Exact Analysis Robust	Approximate Analysis Robust
Fast Gradient Sign Method	≤ 98	0.0000002	1	1
DeepFool	≤ 99	0.0000002	1	1
Limited-memory BFGS	> 0	0.0000002	0	2
Jacobian-Based Saliency Map Attack	100	0.0000002	1	1
Basic Iterative Method	≤ 99	0.0000002	1	1
Carlini Wagner Attack	< 97	0.0000002	1	1
Single Pixel	$\sim$	0.0000002	$\sim$	$\sim$
Boundary Attack	$\sim$	0.0000002	$\sim$	$\sim$
Pointwise Attack	> 0	0.0000002	0	2
Gaussian Blur	*oom*	0.0000002	out of memory	out of memory

7.4.2.1.1 Observation and Analysis

1. The VGG16 model is not 100% robust to any attacks except JBSMA. However, for both values of δ, the Fast Gradient Sign Method, DeepFool, Jacobian-Based Saliency Map Attack, Basic Iterative Method, and Carlini Wagner Attack consistently show a range of robustness in both exact and approximate analyses.

2. Single Pixel Attack and Boundary Attack could not be generated for VGG16 as shown by ($\sim$), possibly due to their complex interaction with the network.

3. The Gaussian Blur attack could not be tested due to potential memory constraints. This highlights the limitations in handling certain types of attacks, particularly those that cause exponential increases in memory requirements, as seen in the Imagestar approach.

4. The network demonstrates a lack of robustness against the LBFGS and Pointwise attacks, as denoted by the 0 in the 'Exact' column, signifying vulnerability. Moreover, the presence of an 2 in the 'Approximate' column indicates potential uncertainty in the decision-making process regarding these attacks.

7.4.2.2 Robustness Analysis of the CIFAR-10 CNN Model

The robustness analysis of the CNN model, as described in Section 7.3.2.1, was conducted against 10 different adversarial attacks using the NNV Imagestar tool, with results detailed in Tables 7.4 and 7.5 for two δ values, 0.0000001 and 0.0000002, respectively. The analysis revealed several key trends and effects of different attacks on the CNN model, as well as insights into exact versus approximate analysis.

Table 7.4: Robustness checking of CNN (Section 7.3.2.1) with $\delta = 0.0000001$ using NNV

Attacks	l(%)	δ	Exact Analysis Robust	Approximate Analysis Robust
Fast Gradient Sign Method	100	0.0000001	1	1
DeepFool	100	0.0000001	1	1
Limited-memory BFGS	100	0.0000001	1	1
Jacobian-Based Saliency Map Attack	100	0.0000001	1	1
Basic Iterative Method	100	0.0000001	1	1
Carlini Wagner Attack	100	0.0000001	1	1
Single Pixel	$*$	0.0000001	$*$	$*$
Boundary Attack	100	0.0000001	1	1
Pointwise Attack	100	0.0000001	1	1
Gaussian Blur	80(approx)	0.0000001	1	1

Table 7.5: Robustness checking of CNN (Section 7.3.2.1) with $\delta = 0.0000002$ using NNV

Attacks	l(%)	δ	Exact Analysis Robust	Approximate Analysis Robust
Fast Gradient Sign Method	100	0.0000002	1	1
DeepFool	100	0.0000002	1	1
Limited-memory BFGS	100	0.0000002	1	1
Jacobian-Based Saliency Map Attack	100	0.0000002	1	1
Basic Iterative Method	100	0.0000002	1	1
Carlini Wagner Attack	100	0.0000002	1	1
Single Pixel	$*$	0.0000002	$*$	$*$
Boundary Attack	100	0.0000002	1	1
Pointwise Attack	100	0.0000002	1	1
Gaussian Blur	80(approx)	0.0000002	1	1

7.4.2.2.1 Observation and Analysis

1. The CNN showed 100% robustness (marked as '1') in the exact and approximate analysis against most attacks. This indicates that the CNN model is robust to these adversarial attacks, even when the perturbation level increases.

2. Another interesting observation was made with the Gaussian Blur attack, where the robustness was approximately 80% for both δ values, indicating a certain degree of vulnerability of the CNN to this type of perturbation.

3. In the case of a Single Pixel attack, the network shows an interesting result, as marked by $*$. **It becomes non-robust in the range of 30% $<l<$80% and $l >$98%.** This is a particularly intriguing finding where a higher level of robustness was observed in the neural network at a higher level of adversarial perturbation, followed by a lower level of robustness at a lower level of perturbation. This counterintuitive result suggests a complex relationship between the perturbation level and the network's robustness, indicating that the network's response to adversarial attacks is not always linear or predictable.

   Understanding this unexpected behavior would require a detailed analysis of the NN's architecture,

training methodology, the nature of the adversarial attacks, and the specifics of how the robustness is measured. However, there might be several factors that could potentially explain this phenomenon,

- **Non-Linearity of Neural Networks:** NNs are highly non-linear systems. This non-linearity can lead to unexpected behavior where a network might learn to resist larger perturbations but fail to handle subtler ones effectively.
- **Differences in the Nature of Perturbations:** Not all adversarial attacks are created equal. The nature and methodology of the attack play a significant role. A higher magnitude attack could be more straightforward and less sophisticated, making it easier for the NN to identify and mitigate, whereas a lower magnitude attack might be more complex and insidious, slipping past the network's defenses.
- **Model Complexity and Capacity:** The architecture and capacity of the model can also impact its robustness. A more complex model might better capture and resist extensive perturbations but could miss finer details, leading to lower robustness against more subtle attacks.

7.5 Conclusion and Future Scope

The study presented here elucidates the multifaceted nature of evaluating the robustness of CNN models against a range of adversarial attacks. The findings emphasize the intricacies involved in such assessments, particularly highlighting the necessity of incorporating both exact and approximate analyses to gauge the resilience of neural networks comprehensively. The comprehensive analysis conducted offers valuable insights into how different adversarial attacks can influence the robustness of CNN models. It reinforces the significance of including diverse attack models in robustness verification and affirms the effectiveness of employing both exact and approximate methods in determining neural networks' resistance to adversarial perturbations.

The process of implementing this workflow unearthed several intriguing challenges, particularly in areas that are not typically encountered when working with well-established pre-trained networks. This exercise not only enhanced the understanding of the Neural Network Verification (NNV) tool's functionality but also identified potential areas for improvement.

CV of Neelanjana Pal

CHAPTER 8

Reachability Based Formal Verification and Benchmarking of Semantic Segmentation Applications

This chapter is adapted from the material presented in AIsoLA 2023 (114).

8.1 Introduction

The Significant Role of Semantic Segmentation. Over the past three decades, image segmentation (153) has been one of the most challenging problems in computer vision. In contrast to tasks like image classification or object recognition, image segmentation operates differently, as it does not rely on prior knowledge of visual concepts or objects within the image. Instead, it assigns a specific category label to every individual pixel in the image. This approach allows the model to accurately delineate distinct regions or objects present in the image, effectively dividing it into meaningful segments. The model creates a comprehensive and detailed representation of the image's content by associating each pixel with its corresponding category label. This capability to provide pixel-level category information has significant real-world applications (155), such as self-driving vehicles (172; 92), pedestrian detection (28; 54), defect detection (154), therapy planning (178; 61), and computer-aided diagnosis (195; 194). The segmentation task empowers intelligent systems to grasp spatial positions and make critical judgments by offering detailed semantic information at the pixel level, setting it apart from other common computer vision tasks.

Figure 8.1: The Original image

Figure 8.2: Semantically Segmented Image

Figure 8.3: An example of semantic segmentation vision tasks from CamVid dataset (31).

Deep Neural Networks (DNN) and Adversarial Attacks. Research has shown that even well-trained neural networks (NNs) are vulnerable to minor input modifications (i.e., adversarial attacks) that can cause signif-

icant changes in the output (106). Similar to image classification neural networks, SSNs are also known to be vulnerable to adversarial perturbations (186). While DNN verification is evolving as a well-established research area with numerous tools and techniques proposed to ensure the safety and robustness specifications of DNNs (94; 180) and neural network-controlled systems (74; 162; 70; 169) most state-of-the-art verification techniques for robustness validation in DNNs primarily focus on variations of classification tasks, often related to images (183; 141; 78; 128; 10; 160; 191; 38; 157; 52; 105; 27). In recent years, verification of segmentation networks has also gained immense focus from researchers all over (168; 23; 76).

Neural Network Verification Competitions. The proliferation of neural networks (NNs) in safety-critical applications has brought attention to their susceptibility to adversarial examples (151), where even minor input perturbations can significantly alter their outputs. Such perturbations, whether occurring randomly or due to malicious intent, emphasize the crucial need to rigorously analyze the robustness of deep learning systems before deploying them in safety-critical domains. Consequently, numerous methods and software tools (48; 56; 72; 77) have been developed for this purpose. However, the increasing number and specialization of these tools have made it challenging for practitioners to choose the most suitable one for their needs.

In response to this dilemma, in 2020, a friendly International Competition on Verification of Neural Networks (VNN-Comp 2020) (30; 17; 108) was conducted to address the issue and allow researchers to compare their neural network verifiers across a wide range of benchmarks. Originally designed as a friendly competition with minimal standardization, the event evolved to introduce more standardization and automation. The goal was to ensure a fair comparison among verifiers on cost-equivalent hardware, utilizing standardized formats for properties and networks. This evolution aimed to facilitate informed decision-making by researchers and practitioners when selecting verification tools for their specific requirements. The VNN-Comp celebrates its 4th iteration this year and successfully presented the results at the Computer Aided Verification 2023 (CAV) conference.

Work Presented In This Chapter. Despite the growing interest and competition in robustness verification, there remains a lack of appropriate benchmarks for evaluating different verifiers on semantic segmentation tasks. This research addresses this gap by introducing segmentation networks on two widely used datasets: MNIST (89), M2NIST (which is a multi-digit variant of MNIST suitable for segmentation evaluation) . Additionally, specific properties are defined for the verification of these networks.

An essential aspect of this work is its potential utility for the VNN-Comp's upcoming iterations. By providing these well-defined benchmarks for semantic segmentation, the hope is to contribute to advancing and standardizing robustness verification techniques in this domain.

8.2 Benchmark Design

The following sections describe the benchmark as follows: (i) the overall motivations and philosophy; (ii) the Datasets and their creation; (iii) the networks proposed; (iv) the unknown-bounded adversarial attack; (v) the metrics used for evaluation; and (vi) robustness property specification.

8.2.1 Philosophy

The motivation behind benchmarking formal verification techniques for semantic segmentation networks arises from the growing significance of deploying these networks in safety-critical applications. By establishing standardized benchmarks and datasets, researchers and practitioners can assess the strengths and limitations of various verification techniques. This enables them to make well-informed decisions when selecting the most appropriate verification methods, considering factors like accuracy, computational overhead, and scalability.

Benchmarking formal verification techniques drives innovation and encourages the development of more reliable and secure deep learning models. It paves the way for integrating formal methods into the training and deployment pipeline, instilling greater confidence in the safety and robustness of semantic segmentation networks for critical applications.

8.2.2 Datasets

The MNIST and M2NIST datasets provide a solid starting point for conducting image-segmentation benchmarking. In contrast to real-world images, especially those captured from autonomous vehicles, these datasets feature well-isolated digits positioned at the center of the images. Consequently, the task of segmentation is comparatively straightforward. The digits, which are the focal points of interest, exhibit distinct clarity and encounter minimal clutter or occlusion. A significant advantage of the MNIST and M2NIST datasets lies in their provision of well-defined ground-truth annotations. This feature greatly simplifies the accurate assessment of segmentation algorithms, enabling meticulous evaluation. Furthermore, these datasets often serve as valuable tools for conveying and elucidating image segmentation's core principles.

8.2.2.1 MNIST Dataset

The MNIST (89) dataset is a well-known dataset used for training and testing machine learning models, particularly for image classification tasks. It consists of handwritten digit images, where each image is a grayscale image of size 28×28 pixels and corresponding ground-truth-labeled masks representing random digit numbers ranging from 0 to 9. To facilitate the experiments, the dataset is divided into two sets: 50,000 images for training and 10,000 images for testing. Fig. 8.4 displays sample images from the MNIST dataset.

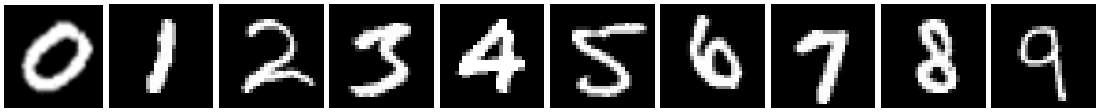

Figure 8.4: Sample images from MNIST dataset

8.2.2.2 M2NIST Dataset

The M2NIST dataset comprises images with dimensions of (64, 84, 1) and corresponding ground-truth-labeled masks depicting multiple (up to three) random digits ranging from 0 to 9. These digits are arranged in a manner that they do not overlap with each other within the images. For this experiments, the original dataset is divided into two sets: 50,000 samples for training and 10,000 for testing. Figure 8.5 displays sample images from the M2NIST dataset.

Figure 8.5: Sample images from M2NIST dataset

8.2.3 Neural Network Models

8.2.3.1 MNIST Dataset.

For the MNIST dataset, two pre-trained networks are utilized from (168), and a third network is trained with 16 layers. The networks consist of an image input layer with the input size of (28, 28, 1) followed by two-dimensional convolution layers, ReLU layers and average-pooling layers.

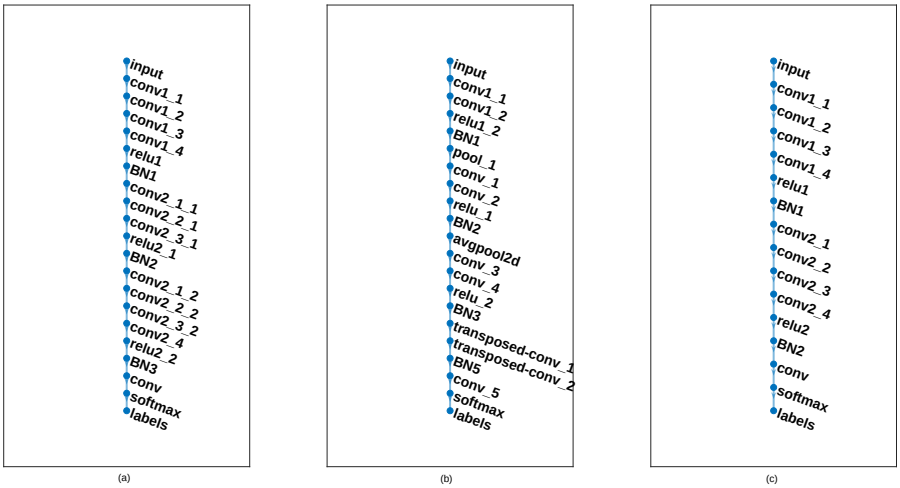

Figure 8.6: Benchmark for MNIST Networks

8.2.3.2 M2NIST Dataset.

For the M2NIST Dataset, the three pre-trained networks from (168) and two newly trained networks are used, as shown in Fig. 8.7. The input image size for these networks is changed to (64, 84, 1).

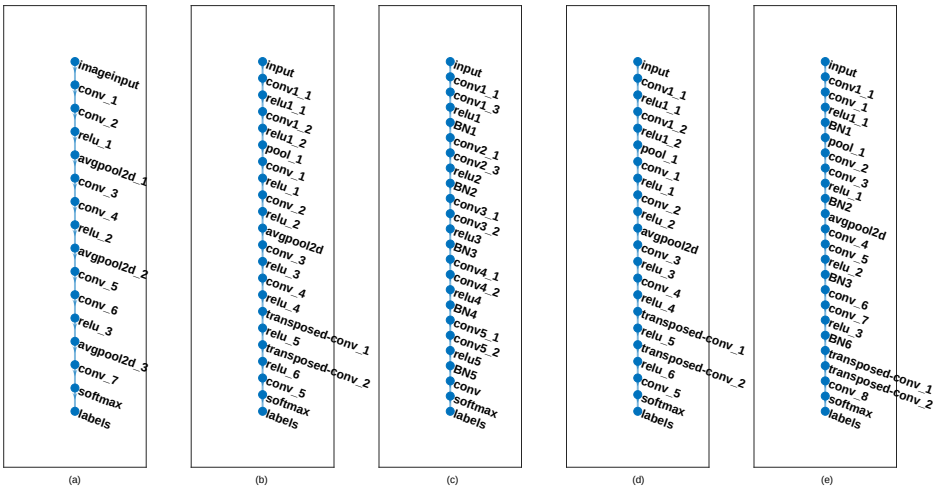

Figure 8.7: Benchmark for M2NIST Networks

Table 8.1: Performances of different networks used for MNIST and M2NIST datasets

$Network_{MNIST}$	$Accuracy_{global}(\%)$	$Accuracy_{mean}(\%)$	IoU_{mean}	$IoU_{weighted}$
mnist_21_iou83	96.88	93.70	0.8335	0.9427
mnist_avg_21	97.28	96.20	0.8675	0.9490
mnist_16	96.93	92.67	0.8376	0.9430
$Network_{M2NIST}$	$Accuracy_{global}(\%)$	$Accuracy_{mean}(\%)$	IoU_{mean}	$IoU_{weighted}$
m2nist_avg_iou62	96.61	88.30	0.6210	0.9464
m2nist_avg_iou75	98.03	97.60	0.7502	0.9660
m2nist_iou72_24	97.86	96.27	0.7271	0.9635
m2nist_avg_22	97.97	98.30	0.7495	0.9650
m2nist_avg_24	97.07	97.86	0.8321	0.9466

The performance measures of each of the proposed networks are shown in Table 8.1.

8.2.4 Segmentation in the Context of Proposed Datasets

The segmentation task in the MNIST and M2NIST datasets involves the process of precisely delineating and identifying individual digits within the given images. The goal is to assign a distinct label to each pixel or region that corresponds to a specific digit. This segmentation is vital for isolating and distinguishing the different digits present in the image, enabling accurate digit recognition and analysis.

8.2.4.1 MNIST Dataset

In the MNIST dataset, each image depicts a single handwritten digit (0-9). The segmentation task involves precisely outlining the boundaries of the digit, distinguishing it from the background. This process aims to identify the exact spatial extent of the digit, ensuring that every pixel belonging to the digit is correctly labeled while excluding pixels from the surrounding background.

8.2.4.2 M2NIST Dataset

The M2NIST dataset introduces a slightly more complex scenario. It consists of images containing two or three handwritten digits placed in non-overlapping arrangements. The segmentation task in M2NIST entails accurately segmenting each digit within the image, ensuring that the segmentation boundaries do not cross over into neighboring digits. This task becomes especially challenging when digits are in close proximity, as segmentation algorithms must correctly identify the boundaries between adjacent digits.

The segmentation task in both datasets essentially involves creating pixel-wise masks that outline the boundaries of individual digits. These masks indicate which pixels belong to each digit and which pixels constitute the background. The successful execution of the segmentation task is crucial for subsequent digit recognition, as the isolated digits can then be analyzed, classified, and identified accurately.

8.2.5 Adversarial Attacks

Inspired by the paper (168), an unknown-bounded adversarial attack (UBAA) is considered on input images in this study. The coefficient vector ε, representing the attack strength, is bounded by lower and upper bounds, denoted as $[\underline{\varepsilon}, \overline{\varepsilon}]$, with each ε_i satisfying $\underline{\varepsilon_i} \leq \varepsilon_i \leq \overline{\varepsilon_i}$. This attack concept is applied to images, where pixel values range from 0 to 255, and consider the attack's impact on either a single pixel or multiple pixels within the image. By applying this attack, a set of images is generated, each having variations in pixel values within one or multiple locations, limited by the bounds $[\underline{\varepsilon}, \overline{\varepsilon}]$.

To illustrate this, an adversarial image x^{adv} is represented as follows:

$$x_{adv} = x_{org} + \Sigma_{i=1}^{n} \varepsilon_i \cdot x_{iattack} \tag{8.1}$$

where x_{org} and x_{adv} are the original and adversarial images, respectively. The variable n denotes the total number of pixels in the image, and ε_i represents the attack coefficient for the pixel at position i.

Each image, x, within this benchmark study is subjected to a UBAA across the test datasets. Here, a pixel $x(i, j)$ is darkened by 1 unit if its value exceeds a specified threshold, denoted as d. In mathematical terms,

the adversarial darkening attack on an image x can be described as follows:

$$x_{adv} = x + \varepsilon \cdot x_{noise},\ 1 - \Delta_\delta \leq \delta \leq 1,$$
$$x_{noise}(i,j) = -1,\ \text{if } x(i,j) > d,\ \text{otherwise } x_{noise}(i,j) = 0.$$

For $\varepsilon = 1$, all pixels are darkened by 1 unit whose values exceed the threshold d (set to 150 for this work and $\delta = 1$), resulting in $x_{adv}(i,j) = x(i,j) - 1$. The size of the input set affected by the attack is determined by Δ_δ. A larger value of Δ_δ corresponds to a larger input set after applying the attack.

By varying the values of δ and d, different robustness properties are generated for verification.

8.2.6 Evaluation Metrics

For evaluation purposes, the traditional concept of Intersection-over-Union (IoU) is used for both segmentation model performances and model robustness checking. Following (168), the concepts of robustness value (RV) and robustness sensitivity (RS) are also used.

8.2.6.1 Robustness Value (RV).

An SSN's Robustness Value (RV) characterizes its resilience against an adversarial attack. Specifically, for an unknown bounded adversarial attack applied to an input image, the RV is defined as follows:

$$RV = \frac{N_{robust}}{N_{pixels}} \times 100\%,$$

where N_{robust} is the total number of robust pixels [1] under the attack, and $N_{pixels} = h \cdot w$ is the total number of input image pixels.

8.2.6.2 Robustness Sensitivity (RS)

The Robustness Sensitivity (RS) quantifies the network's susceptibility under the adversarial attack, revealing the average number of pixels in the segmentation output image that are influenced (either becoming non-robust or unknown) when a single pixel in the input image is attacked. The robustness sensitivity of an SSN corresponding to an unknown bounded adversarial attack applied to an input image is defined as

$$RS = \frac{N_{nonrobust} + N_{unknown}}{N_{attacked\,pixels}}, \tag{8.2}$$

[1] In this context, "Robust pixels" refer to those pixels that maintain their correct classification even in the presence of an adversarial attack.

where $N_{nonrobust}$ is the total number of non-robust pixels under the attack, $N_{unknown}$ is the total number of pixels whose robustness is unknown (i.e., the verifier can not guarantee on the robustness; it may or may not be robust), and $N_{attacked pixels}$ is the total number of attacked pixels of the input image.

8.2.6.3 Robust Intersection-over-Union (IoU)

The robust IoU (R_{IoU}) concept shares similarities with the traditional IoU, a fundamental metric used to evaluate accuracy during the training of semantic segmentation networks (SSNs). The robust IoU of a semantic segmentation network (SSN), when subjected to an unknown-bounded adversarial attack on an input image, is calculated as the average IoU of all labels that remain robust under the attack.

Consider a segmentation ground-truth image denoted as x, the verified segmentation image under the adversarial attack as y, and the number of classes, L. Then the IoU (IoU_p) for the p^{th} label in the label images x and y is computed as the intersection of the label images divided by their union for the i^{th} label. , then the R_{IoU} of the SSN is computed by:

$$R_{IoU} = \frac{\Sigma_{p=1}^{L} IoU_p}{L}. \tag{8.3}$$

For this chapter, the concept of robust IoU is leveraged in conjunction with the robustness value and sensitivity as core metrics to assess the robustness of an SSN when subjected to adversarial attacks within the verification framework. Instead of solely measuring accuracy, these metrics comprehensively evaluate the network's resilience against such attacks.

8.2.7 Robustness Property Specification

In semantic segmentation examples, while each pixel is assigned a class label, individual pixels do not determine the object classification. Instead, a cluster of pixels with the same class collectively contributes to the final object decision. As a result, the robustness property defined for classification models is not directly applicable to segmentation models. Therefore, the focus is on evaluating the Intersection over Union (IoU) measures of the segmentation output image w.r.t its original counterpart.

To characterize the robustness properties of a specific SSN for a given input image, its corresponding Robustness Value (RV) and Robustness Sensitivity (RS) are defined within a specified range. This range represents the maximum allowable deviation the SSN is allowed to exhibit to be within a safe region. Consequently, for an adversarial input image set denoted as X^{adv} and its output segmentation image set as Y^{adv}, the robustness property for RV is defined as follows:

$$RV_{min} \leq RV_{org} \leq RV_{max} \tag{8.4}$$

where $[RV_{min}, RV_{max}]$ is the permissible bounds for the RV and RV_{org} is the actual RV for the unperturbed image.

Similarly, the robustness property for the RS of the same example can be given as:

$$RS_{min} \leq RS_{org} \leq RS_{max} \tag{8.5}$$

where $[RS_{min}, RS_{max}]$ is the permissible bounds for the RS and RS_{org} is the actual RS for the unperturbed image.

The IoU robustness property can be given by the equation:

$$R_{IoU_{min}} \leq R_{IoU_{org}} \leq R_{IoU_{max}} \tag{8.6}$$

where $[R_{IoU_{min}}, R_{IoU_{max}}]$ is the permissible bounds for the IoU and $R_{IoU_{org}}$ is the actual IoU for the unperturbed image.

8.2.8 Verification Property Specifications in vnnlib Files: Illustrated with an Example

8.2.8.1 vnnlib file format.

Following the competition protocol, the robustness specification is proposed in a vnnlib file (Demarchi). A vnnlib file is a standard format for representing neural network verification problems. It provides details about the neural network, input constraints, and properties to be verified. The vnnlib file format is widely adopted in formal verification for neural networks (136; 57; 17; 108). The verification specification in a vnnlib file involves defining properties using a specific syntax. The structure of a vnnlib file typically includes the following components:

1. **Input Constraints:** This section defines the input bounds or constraints for the neural network.
2. **Output Behavior Specification:** This section contains the expected output or behavior of the neural network for the specified input constraints.
3. **Property Specification:** This section specifies the neural network properties to be verified. These properties include safety and robustness verification properties.

8.2.8.2 Example.

To illustrate this format, let's consider the example in Fig. 8.8.

This example focuses on an input image represented as a 4×4 2-dimensional array with pixel values from 0 to 255. The output of the Semantic Segmentation Network (SSN) pixel classification layer is also a

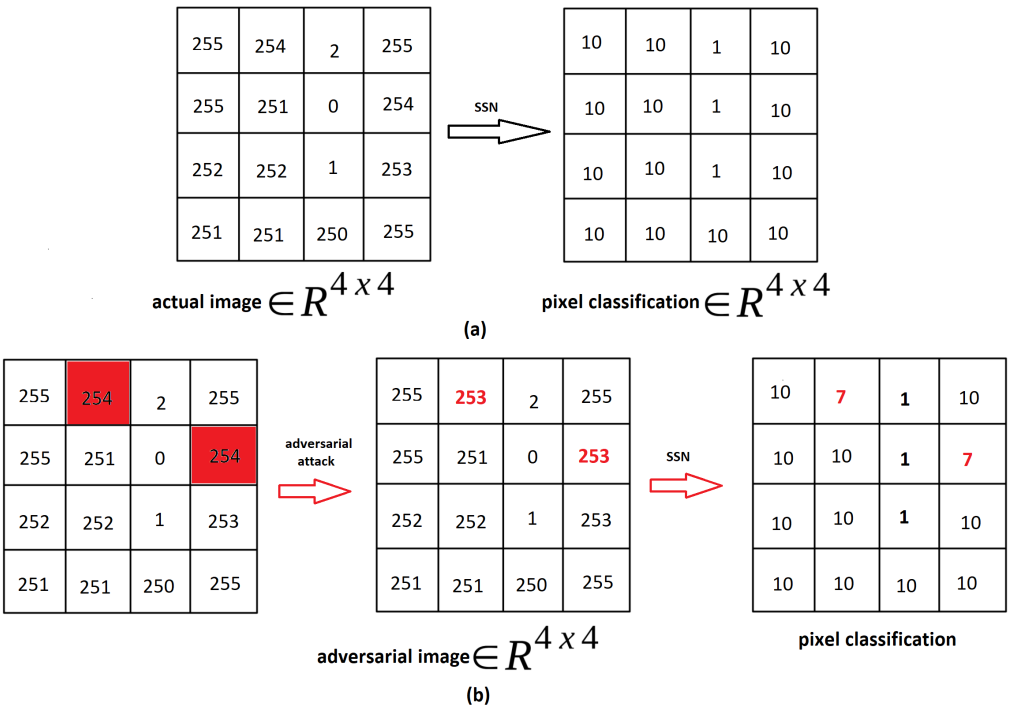

Figure 8.8: The robustness verification specification for a Semantic Segmentation Network (SSN) is illustrated as follows: (a) Example image and its corresponding pixel classification. (b) Location of the adversarial attack (red), the pixel darkened by 1 (original value 254), and the resulting pixel classification after the adversarial attack.

4×4 2-dimensional array, assigning classes to each pixel. For this image, the digit '1' is highlighted, with the first three rows in column 3 of the output classified as '1' and the remaining rows labeled as background (represented by '10').

Next, an Unbounded Adversarial Attack (UBAA) is explored on the image, targeting pixels with a value of 254 and reducing them to 253. To ensure a bounded attack, the upper bound of the attacked image is set as the original image itself [Fig. 8.8 (b) left], and the lower bound as the image with darkened pixels [Fig. 8.8 (b) middle].

To specify the input properties in a vnnlib file, first, the input image is flattened column-wise, and then the upper and lower bounds are defined for each pixel representing the attack. In the case of the provided image example, the input properties in a vnnlib file should be structured as depicted in the List. 1 (Fig. 8.9). This allows for a comprehensive representation of the image and its bounds, facilitating verification.

Similar to the input specification, the output must be flattened column-wise for each of the class/labels and then checked for the desired classes.

In the example, let's also make an assumption, as depicted in [Fig. 8.8 (b) right] that the SSN misclassifies two pixels due to the darkening effect, classifying them as '7' instead of '10'. Consequently, following

List 1 Input Constraints

```
; Verification Property Specification
(declare-const X_0 Real)
(declare-const X_1 Real)
(declare-const X_2 Real)
(declare-const X_3 Real)
(declare-const X_4 Real)
(declare-const X_5 Real)
(declare-const X_6 Real)
(declare-const X_7 Real)
(declare-const X_8 Real)
(declare-const X_9 Real)
(declare-const X_10 Real)
(declare-const X_11 Real)
(declare-const X_12 Real)
(declare-const X_13 Real)
(declare-const X_14 Real)
(declare-const X_15 Real)
; Unscaled Input 0 : (255, 255)
(assert (<= X_0 255))
(assert (>= X_0 255))

; Unscaled Input 1 : (255, 255)
(assert (<= X_1 255))
(assert (>= X_1 255))

; Unscaled Input 2 : (252, 252)
(assert (<= X_2 252))
(assert (>= X_2 252))

; Unscaled Input 3 : (251, 251)
(assert (<= X_3 251))
(assert (>= X_3 251))

; Unscaled Input 4 : (253, 254)
(assert (<= X_4 254))
(assert (>= X_4 253))

; Unscaled Input 5 : (251, 251)
(assert (<= X_5 251))
(assert (>= X_5 251))

; Unscaled Input 6 : (252, 252)
(assert (<= X_6 252))
(assert (>= X_6 252))

; Unscaled Input 7 : (251, 251)
(assert (<= X_7 251))
(assert (>= X_7 251))

; Unscaled Input 8 : (2, 2)
(assert (<= X_8 2))
(assert (>= X_8 2))

; Unscaled Input 9 : (0, 0)
(assert (<= X_9 0))
(assert (>= X_9 0))

; Unscaled Input 10 : (1, 1)
(assert (<= X_10 1))
(assert (>= X_10 1))

; Unscaled Input 11 : (250, 250)
(assert (<= X_11 250))
(assert (>= X_11 250))

; Unscaled Input 12 : (255, 255)
(assert (<= X_12 255))
(assert (>= X_12 255))

; Unscaled Input 13 : (253, 254)
(assert (<= X_13 254))
(assert (>= X_13 253))

; Unscaled Input 14 : (253, 253)
(assert (<= X_14 253))
(assert (>= X_14 253))

; Unscaled Input 15 : (255, 255)
(assert (<= X_15 255))
(assert (>= X_15 255))
```

Figure 8.9: Input Constraints

the definition of robustness measures, the values are obtained for both the unperturbed image and the one corresponding to the UBAA attack, as shown below in the Table. 8.2. Here, it needs to be emphasized that

following Sec. 8.2.6.2, the concept of robustness sensitivity is only valid for an adversarial input.

Table 8.2

RobustnessMeasures	*Unperturbed*	*Under UBAA*
RV	1	0.8750
RS	-	1
RIOU	1	0.6154

Additional output verification properties can also be introduced based on robustness measures for SSN verification against adversarial attacks. These properties are derived from the output pixel classification constraints. For the example shown in Fig. 8.8, the considerable RV is restricted in $[0.9, 1]$, RIoU in $[0.8, 1]$ and RS to be always ≤ 100, for the output reachable set to be in the safe region. The proposed properties are as follows:

8.3 Evaluation

8.3.1 Reachability Analysis

To assess the impact of the adversarial attack on each dataset, a widely used concept, reachability analysis, is employed (168; 167; 164; 179; 70; 127; 95). The perturbed input is represented as a bounded set, and the output reachable set is computed layer-by-layer for the SSN. For the final layer of an SSN, i.e., pixel-classification layer, the pixel-class reachable set at a specific pixel is denoted as $pc(i, j) = \{l_1, ..., l_m\}$. This set is obtained by determining all cross-channel max-point candidates for each pixel in the input set. Consequently, the pixel-class reachable set of the layer can be obtained, which is equivalent to the reachable set of the SSN, denoted as $R_f = [pc(i, j)]_{h \times w}$, i.e., the collection of pixel classes at every index (i; j) (168).

Subsequently, the Robustness Values (RVs), Robustness Sensitivities (RSs), and Robust Intersection-over-Union (IoU) scores are calculated for all the images in the adversarial set based on the output reachable set.

The "approx-star" method is employed for reachability analysis in this chapter. This method is preferred due to its computational efficiency, requiring less time and memory than "exact-star" methods. The readers are directed to refer to (168; 167; 164) for a more comprehensive understanding of the Star-based reachability analysis.

For calculating the output reachable set using "approx-star," make use of a NNV Tool" (170; 97). It is a comprehensive set-based framework for verifying neural networks (NNs). It supports multiple reachability algorithms, enabling safety verification and robustness analysis of various deep neural network (DNN) types.

In the context of reachability analysis, the NNV tool computes output reachable sets layer-by-layer, starting from a given input. This input is defined by upper and lower bounds, representing perturbations around

the actual input. As the analysis progresses through the layers, the reachable sets at the final layer represent the collection of all possible states of the DNN.

The primary objective of the NNV tool is to determine whether the DNN is deemed "safe." A DNN is considered safe when the specified safety properties determine no intersection between the output sets and the predefined unsafe region. By verifying safety conditions and analyzing robustness, the NNV tool aids in ensuring the reliability and trustworthiness of neural networks in various applications.

8.3.2 Results

In this section, a sample plot [Fig. 8.10]is presented, illustrating the average robustness measures of three MNIST networks, as described in Section 8.2.3.1. The analysis is conducted by subjecting 100 random digit images to the UBA attack and calculating the networks' average robustness against this attack with the following details: (1) max number of pixels attacked under UBAA: [1 2 3 4 5] and (2) $\epsilon = 1$ [Sec. 8.2.5].

When analyzing the outcomes of this experiment, several notable observations shed light on the behavior of different robustness measures under varying degrees of adversarial attacks. These observations provide valuable insights into how these measures respond and behave in the face of adversarial perturbations.

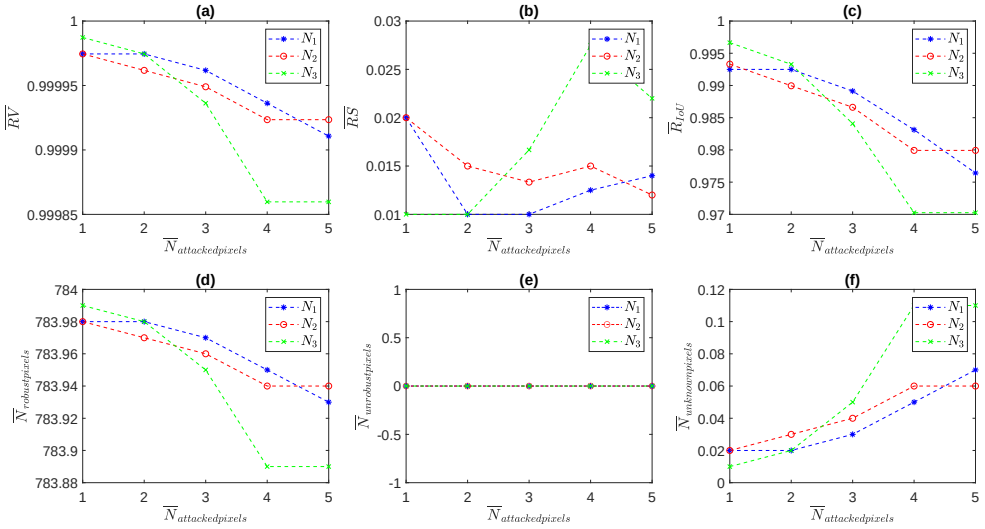

Figure 8.10: The average robustness value, sensitivity, and IoU of MNIST SSNs.

Specifically, as the number of adversarial attacks are increased, a consistent downward trend is noticed in both the robustness value and the robust IoU (Intersection-over-Union). The robustness value quantifies the extent to which the SSN's predictions remain accurate after exposure to adversarial perturbations. In this analysis, this value consistently decreased with a greater number of attacks. Similarly, the robust IoU, which measures the overlap between predicted and ground-truth segments, also demonstrated a decreasing

pattern with increased attacks. This reduction suggests that the adversarial perturbations adversely affect the network's ability to segment objects within images accurately.

Interestingly, a nuanced behavior is encountered when examining the robustness sensitivity. Unlike the robustness value and robust IoU, the trend in robustness sensitivity was not strictly uniform as the number of attacks increased. Robustness sensitivity gauges how sensitive the SSN's output segmentation is to changes in input pixels due to an attack. The observations found that this sensitivity did not consistently follow a rigid trend with escalating adversarial attacks. This variability aligns with the inherent nature of robustness sensitivity, which can be influenced by the distribution and complexity of perturbations introduced by different attacks.

Overall, these observations reaffirm the theoretical definitions and expectations of these robustness measures in the context of various adversarial attacks. The decreasing trends in robustness value and robust IoU highlight the vulnerability of the SSN's segmentation performance to increasing adversarial perturbations. The non-uniform trend in robustness sensitivity emphasizes the intricate interplay between attack characteristics and the network's responsiveness to perturbations, leading to varying degrees of sensitivity under different attack scenarios. Such insights are crucial for understanding the strengths and limitations of these robustness measures and guiding the development of more resilient semantic segmentation networks in the future.

8.4 Conclusion and Future Work

This study introduces a benchmark framework for formally verifying semantic segmentation neural networks to enhance their safety and reliability in critical applications like autonomous vehicles, medical imaging, and surveillance systems. Establishing this standardized benchmark is crucial to enabling fair comparisons of various verification methods and tools, thereby driving progress in formal verification for these networks. The framework covers a range of neural network architectures, datasets, and verification properties, focusing on MNIST and M2NIST. It also includes a detailed specification format in vnnlib files for describing verification properties and facilitating the integration and comparison of different tools and approaches.

Looking ahead, the benchmark will be continuously updated with new architectures, datasets, and verification properties to stay abreast of developments in semantic segmentation. Collaboration with the research community is key to refining the benchmark, leveraging practical insights for improvements. There's also a plan to evaluate and compare existing formal verification methods and tools using this benchmark to identify their strengths and limitations. Additionally, the integration of innovative techniques and advancements in formal verification, including machine learning-based methods and formal synthesis, will be explored to enhance the benchmark's scope and effectiveness.

Overall, this benchmark framework establishes a solid foundation for advancing the safety and reliability of semantic segmentation neural networks in crucial applications. The ongoing collaboration with the research community is anticipated to further enrich and evolve the benchmark, contributing significantly to the field of formal verification for semantic segmentation networks.

CHAPTER 9

Conclusions & Future Directions

9.1 Conclusion

In recent years, the field of neural network verification has garnered significant attention, particularly in the context of ensuring the safety and reliability of neural networks in various applications. This thesis addresses this growing concern by primarily focusing on two critical aspects: robustness verification in time-series safety-critical applications and enhancing support for complex network architectures and layer types. The motivation behind this research stems from the evolving landscape of neural network applications, where traditional methods of verification are increasingly challenged by the complexity and diversity of network models and their respective applications.

The study extends the capabilities of the Neural Network Verification (NNV) tool beyond its traditional scope, emphasizing the importance of various input data types and non-classification models like time-series classification and semantic segmentation. A pioneering formal verification approach for autoencoders is introduced, exploring the precision of autoencoder models in reconstructing signals amidst potential faults. Additionally, the research delves into the reachability analysis of variable-length time series regression neural networks, highlighting their significance in predictive maintenance in the context of Industry 4.0.

The thesis also marks a significant milestone in neural network verification with the inclusion of the 'sequenceinputlayer' in the NNV tool. This development highlights the tool's enhanced ability to adeptly handle sequential input characteristics, ensuring the reachability and efficacy of subsequent neural network layers are not compromised. This key advancement supports a wide range of DNN-based applications that rely on LSTM layers, paving the way for the future integration of additional RNN types such as BiLSTM layers. Furthermore, the integration of the 'additionlayer' and 'concatenationlayer' into NNV signifies a crucial enhancement in line with the thesis's broader objective. This integration is pivotal in augmenting the tool's capacity to effectively manage complex DNN architectures, particularly those incorporating skip connections. These collective advancements position NNV as a more robust and versatile tool in the evolving landscape of neural network verification.

9.2 Future Directions

9.2.1 Support for other RNN Layers

A promising future direction stemming from the successful integration of LSTM support in the NNV tool is the exploration and incorporation of other RNN layers. This expansion aligns with the evolving complexity

and diversity of neural network applications, especially those that process sequential data.

- **Bidirectional Long Short-Term Memory (BiLSTM) Layer Integration:** One immediate extension is the integration of BiLSTM layers into NNV. BiLSTM networks, known for their ability to process sequential data in both forward and backward directions, are crucial in tasks where the context from both past and future information is essential for accurate predictions. Supporting BiLSTM layers in NNV would significantly enhance the tool's applicability in areas such as natural language processing, time-series analysis, and audio signal processing.

- **Gated Recurrent Unit (GRU) Layer Support:** Another potential extension is the incorporation of Gated Recurrent Unit (GRU) layers. GRUs are an alternative to LSTMs and are particularly efficient in scenarios where memory and computational resources are limited. Their integration into NNV would make the tool more versatile and capable of handling a broader range of sequential data processing tasks with varying resource constraints.

- **Hybrid RNN Models:** Exploring support for hybrid RNN models that combine different types of recurrent layers or integrate recurrent layers with other neural network architectures like CNNs could also be a future direction. Such hybrid models are increasingly common in complex applications like multimodal data analysis and sophisticated sequence-to-sequence models.

These possible future directions can help evolve NNV, addressing the needs of increasingly complex and diverse neural network applications and fostering advancements in the safe and reliable deployment of RNN-based systems.

9.2.2 Explore on Video Segmentation and Classification Domains

Building on the foundation established in this thesis through the work on time series applications and audio classification, a compelling future direction is to extend the reachability-based formal verification approach to video segmentation and video classification applications. Video data, with its inherent complexity and richness, presents unique challenges and opportunities for formal verification. The following are potential avenues for future research in this domain:

- **Extending Reachability Analysis to Video Data:** Video data can be seen as a sequence of images (frames), each containing spatial information, combined with the temporal information across frames. Extending reachability analysis to this domain would involve developing methodologies that can handle the high dimensionality and temporal dependencies inherent in video data. This could include adapt-

ing existing reachability techniques used for time series and audio data to accommodate the spatial-temporal characteristics of video.

- **Video Segmentation Verification:** Video segmentation involves partitioning video frames into multiple segments or regions for analysis. Formal verification in this context would require ensuring that the segmentation algorithm correctly identifies and classifies each region across all frames under various conditions, including adversarial perturbations. Developing verification techniques that can handle the dynamic nature of video segmentation would be a key focus area.

- **Robustness Verification for Video Classification:** Video classification algorithms categorize video content into predefined classes. Future work could involve analyzing the robustness of these algorithms against adversarial attacks that introduce subtle but impactful changes in videos. This analysis would be crucial for applications like surveillance and content moderation, where accuracy and reliability are paramount.

- **Handling Real-Time Video Processing Constraints:** Many video applications, such as autonomous driving and real-time surveillance, require immediate processing and decision-making. Future research could explore verification techniques that are not only accurate but also computationally efficient to support real-time video processing requirements.

- **Dataset and Benchmark Development:** To facilitate research in this area, developing comprehensive datasets and benchmarks for video segmentation and classification verification would be essential. These resources would provide a standard for evaluating and comparing different verification techniques.

- **Integration with Advanced Neural Network Architectures in NNV:** Investigating the integration of advanced NN architectures used in video processing, such as 3D CNNs and ConvLSTM networks, into the NNV tool would also be an important area of research.

These research avenues can significantly enhance the safety, reliability, and robustness of video segmentation and classification algorithms. This advancement is particularly vital in critical applications where the accuracy and dependability of these algorithms are paramount, and any errors could lead to serious implications.

CHAPTER 10

Publications

10.1 Accepted Journal, Conference & Workshop Papers

1. Tran, Hoang-Dung, 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 Computer Aided Verification: 33rd International Conference, CAV 2021.

2. Tran, Hoang-Dung, 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." In Formal Aspects of Computing, Volume 33, Issue 4-5, pp. 519-545, 2021 August.

3. Pal, Neelanjana, and Taylor T. Johnson. "Work In Progress: Safety and Robustness Verification of Autoencoder-Based Regression Models using the NNV Tool.", In the 7th International Workshop on Symbolic-Numeric Methods for Reasoning about CPS and IoT, SNR, August 2021.

4. Pal, Neelanjana, Diego Manzanas Lopez, and Taylor T. Johnson. "Robustness verification of deep neural networks using star-based reachability analysis with variable-length time series input." In International Conference on Formal Methods for Industrial Critical Systems (FMICS), September 2023.

5. Pal, Neelanjana, and Taylor T. Johnson. "Formal Verification of Long Short-Term Memory based Audio Classifiers: A Star based Approach." In the 5th Fifth International Workshop on Formal Methods for Autonomous Systems (FMAS), November 2023.

6. Pal, Neelanjana, Seojin Lee, and Taylor T. Johnson. "Benchmark: formal verification of semantic segmentation neural networks." In International Conference on Bridging the Gap between AI and Reality, pp. 311-330. Cham: Springer Nature Switzerland, 2023.

10.2 List of Posters & Presentations

1. Pal, Neelanjana, and Taylor T. Johnson. "Work In Progress: Safety and Robustness Verification of Autoencoder-Based Regression Models using the NNV Tool." In the 7th International Workshop on Symbolic-Numeric Methods for Reasoning about CPS and IoT, SNR, August 2021.

2. Diego Manzanas Lopez, Neelanjana Pal, Taylor T Johnson. "NNV: A Verification Tool for Deep Neural Networks and Learning-Enabled Cyber-Physical Systems. (Poster)," In the 25th Anniversary of Institute for Software Integrated Systems, September 7, 2023, Vanderbilt University, Nashville, TN, USA

3. Neelanjana Pal, Taylor T Johnson. "Potential Fusion of Neural Networks and Petri Nets in the Domain of Formal Verification, (Poster)," In the 6th Advanced Course on Petri Nets, September 3-8, 2023, Torun, Poland

4. Pal, Neelanjana, Diego Manzanas Lopez, and Taylor T. Johnson. "Robustness verification of deep neural networks using star-based reachability analysis with variable-length time series input." In International Conference on Formal Methods for Industrial Critical Systems (FMICS), September 2023.

5. Pal, Neelanjana, and Taylor T. Johnson. "Formal Verification of Long Short-Term Memory based Audio Classifiers: A Star based Approach." In the 5th Fifth International Workshop on Formal Methods for Autonomous Systems (FMAS), November 2023.

6. Pal, Neelanjana, Seojin Lee, and Taylor T. Johnson. "Benchmark: formal verification of semantic segmentation neural networks." In International Conference on Bridging the Gap between AI and Reality, pp. 311-330. Cham: Springer Nature Switzerland, 2023.

References

[Foo] https://github.com/bethgelab/foolbox.

[CIF] Cifar-10 - wikipedia. https://en.wikipedia.org/wiki/CIFAR-10. (Accessed on 10/10/2019).

[Ima] Imagenet. http://www.image-net.org/. (Accessed on 10/10/2019).

[4] Predict Battery State of Charge Using Deep Learning - MATLAB &; Simulink — mathworks.com. https://www.mathworks.com/help/deeplearning/ug/predict-soc-using-deep-learning.html.

[Pro] Prognostics center of excellence - data repository. https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/#turbofan.

[6] Remaining Useful Life Estimation Using Convolutional Neural Network - MATLAB &; Simulink — mathworks.com. https://www.mathworks.com/help/predmaint/ug/remaining-useful-life-estimation-using-convolutional-neural-network.html.

[7] Alshemali, B. and Kalita, J. (2020). Improving the reliability of deep neural networks in nlp: A review. *Knowledge-Based Systems*, 191:105210.

[8] Althoff, M., Frehse, G., and Girard, A. (2021). Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:369–395.

[9] Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. (2019a). 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.

[10] Anderson, G., Pailoor, S., Dillig, I., and Chaudhuri, S. (2019b). 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*, PLDI 2019, page 731–744, New York, NY, USA. Association for Computing Machinery.

[11] Asarin, E., Dang, T., Frehse, G., Girard, A., Le Guernic, C., and Maler, O. (2006). Recent progress in continuous and hybrid reachability analysis. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 1582–1587. IEEE.

[12] Asuncion, A. and Newman, D. (2007). Uci machine learning repository.

[13] Aytar, Y., Vondrick, C., and Torralba, A. (2016). Soundnet: Learning sound representations from unlabeled video. *Advances in Neural Information Processing Systems 29: December 5-10, 2016, Barcelona, Spain*.

[14] Bak, S. (2020). Execution-guided overapproximation (ego) for improving scalability of neural network verification.

[15] Bak, S. (2021). nnenum: Verification of relu neural networks with optimized abstraction refinement. In *NASA Formal Methods Symposium*, pages 19–36. Springer.

[16] Bak, S. and Duggirala, P. S. (2017). Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer.

[17] Bak, S., Liu, C., and Johnson, T. (2021). The second international verification of neural networks competition (vnn-comp 2021): Summary and results. *arXiv preprint arXiv:2109.00498*.

[18] Bak, S., Tran, H.-D., Hobbs, K., and Johnson, T. T. (2020). Improved geometric path enumeration for verifying ReLU neural networks. In *32nd International Conference on Computer-Aided Verification (CAV)*.

[19] Balcázar, J. L., Gavalda, R., and Siegelmann, H. T. (1997). Computational power of neural networks: A characterization in terms of kolmogorov complexity. *IEEE Transactions on Information Theory*, 43(4):1175–1183.

[20] Bayraci, S., Susuz, O., et al. (2019). A deep neural network (dnn) based classification model in application to loan default prediction. *Theoretical and Applied Economics*, 4(621):75–84.

[21] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[22] Betz, J., Heilmeier, A., Wischnewski, A., Stahl, T., and Lienkamp, M. (2019). Autonomous driving—a crash explained in detail. *Applied Sciences*, 9(23):5126.

[23] Blum, H., Sarlin, P.-E., Nieto, J., Siegwart, R., and Cadena, C. (2019). Fishyscapes: A benchmark for safe semantic segmentation in autonomous driving. In *proceedings of the IEEE/CVF international conference on computer vision workshops*, pages 0–0.

[24] Bogomolov, S., Forets, M., Frehse, G., Potomkin, K., and Schilling, C. (2019). Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44.

[25] Borgi, T., Hidri, A., Neef, B., and Naceur, M. S. (2017). Data analytics for predictive maintenance of industrial robots. In *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*, pages 412–417. IEEE.

[26] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. (2020a). Efficient verification of relu-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299.

[27] Botoeva, E., Kouvaros, P., Kronqvist, J., Lomuscio, A., and Misener, R. (2020b). Efficient verification of relu-based neural networks via dependency analysis. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3291–3299.

[28] Brazil, G., Yin, X., and Liu, X. (2017). Illuminating pedestrians via simultaneous detection & segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 4950–4959.

[29] Brendel, W., Rauber, J., and Bethge, M. (2017). Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*.

[30] Brix, C., Müller, M. N., Bak, S., Johnson, T. T., and Liu, C. (2023). First three years of the international verification of neural networks competition (vnn-comp). *International Journal on Software Tools for Technology Transfer*, pages 1–11.

[31] Brostow, G. J., Fauqueur, J., and Cipolla, R. (2009). Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97.

[32] Bulsari, A. (1993). Some analytical solutions to the general approximation problem for feedforward neural networks. *Neural networks*, 6(7):991–996.

[33] Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. (2018). A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*.

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

[35] Choi, K., Fazekas, G., Sandler, M., and Cho, K. (2017). Convolutional recurrent neural networks for music classification. In *2017 IEEE International conference on acoustics, speech and signal processing (ICASSP)*, pages 2392–2396. IEEE.

[36] Cun, Y., Bottou, L., Orr, G., and Muller, K. (1998). Efficient backprop, neural networks: Tricks of the trade. *Lecture notes in computer sciences*, 1524:5–50.

[37] Das, S. and Suganthan, P. N. (2010). Differential evolution: A survey of the state-of-the-art. *IEEE transactions on evolutionary computation*, 15(1):4–31.

[38] Dathathri, S., Dvijotham, K., Kurakin, A., Raghunathan, A., Uesato, J., Bunel, R. R., Shankar, S., Steinhardt, J., Goodfellow, I., Liang, P. S., et al. (2020). Enabling certification of verification-agnostic networks via memory-efficient semidefinite programming. *Advances in Neural Information Processing Systems*, 33:5318–5331.

[39] de Riberolles, T., Zou, Y., Silvestre, G., Lochin, E., and Song, J. (2022). Anomaly detection for ics based on deep learning: a use case for aeronautical radar data. *Annals of Telecommunications*, pages 1–13.

[40] DeLillo, D. (1999). *White noise*. Penguin.

[Demarchi] Demarchi, S. VNN-LIB — vnnlib.org. https://www.vnnlib.org/. [Accessed 31-07-2023].

[42] Demir, F., Abdullah, D. A., and Sengur, A. (2020). A new deep cnn model for environmental sound classification. *IEEE Access*, 8:66529–66537.

[43] Dong, M. (2018). Convolutional neural network achieves human-level accuracy in music genre classification. *arXiv preprint arXiv:1802.09697*.

[44] Duan, J. (2019). Financial system modeling using deep neural networks (dnns) for effective risk assessment and prediction. *Journal of the Franklin Institute*, 356(8):4716–4731.

[45] Dutta, S., Jha, S., Sanakaranarayanan, S., and Tiwari, A. (2017). Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130*.

[46] Dutta, S., Jha, S., Sankaranarayanan, S., and Tiwari, A. (2018). Learning and verification of feedback control systems using feedforward neural networks. *IFAC-PapersOnLine*, 51(16):151–156.

[47] EASA and Aerospace, C. (2023). Formal methods use for learning assurance (formula). Technical report.

[48] Ehlers, R. (2017). Formal verification of piece-wise linear feed-forward neural networks. In *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*, pages 269–286. Springer.

[49] Elattar, H. M., Elminir, H. K., and Riad, A. M. (2019). Conception and implementation of a data-driven prognostics algorithm for safety–critical systems. *Soft Computing*, 23(10):3365–3382.

[50] Endsley, M. R. (2017). Autonomous driving systems: A preliminary naturalistic study of the tesla model s. *Journal of Cognitive Engineering and Decision Making*, 11(3):225–238.

[51] Falola, P. B., Alabi, E. O., Ogunajo, F. T., and Fasae, O. D. (2022). Music genre classification using machine and deep learning techniques: a review. *ResearchJet J Anal Invent*, 3(03):35–50.

[52] Fazlyab, M., Morari, M., and Pappas, G. J. (2020). Safety verification and robustness analysis of neural networks via quadratic constraints and semidefinite programming. *IEEE Transactions on Automatic Control*, pages 1–1.

[53] Ferguson, C. E. (1965). Time-series production functions and technological progress in american manufacturing industry. *Journal of Political Economy*, 73(2):135–147.

[54] Flohr, F., Gavrila, D., et al. (2013). Pedcut: an iterative framework for pedestrian segmentation combining shape models and multiple data cues. In *BMVC*.

[55] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. (2018a). Ai 2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*.

[56] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. (2018b). Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE.

[57] Girard-Satabin, J., Alberti, M., Bobot, F., Chihani, Z., and Lemesle, A. (2022). Caisar: A platform for characterizing artificial intelligence safety and robustness. *arXiv preprint arXiv:2206.03044*.

[58] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

[59] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.

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

[61] Guo, Y., Gao, Y., and Shen, D. (2015). Deformable mr prostate segmentation via deep feature learning and sparse patch matching. *IEEE transactions on medical imaging*, 35(4):1077–1089.

[62] Gupta, U., Wu, C.-J., Wang, X., Naumov, M., Reagen, B., Brooks, D., Cottel, B., Hazelwood, K., Hempstead, M., Jia, B., et al. (2020). The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. IEEE.

[63] Guzhov, A., Raue, F., Hees, J., and Dengel, A. (2021). Esresnet: Environmental sound classification based on visual domain models. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 4933–4940. IEEE.

[64] He, K. and Sun, J. (2015). Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360.

[65] He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034.

[66] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[67] He, S. (2021). Who is liable for the uber self-driving crash? analysis of the liability allocation and the regulatory model for autonomous vehicles. *Autonomous Vehicles: Business, Technology and Law*, pages 93–111.

[68] Hemdan, E. E.-D., El-Shafai, W., and Sayed, A. (2023). Cr19: A framework for preliminary detection of covid-19 in cough audio signals using machine learning algorithms for automated medical diagnosis applications. *Journal of Ambient Intelligence and Humanized Computing*, 14(9):11715–11727.

[69] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.

[70] Huang, C., Fan, J., Li, W., Chen, X., and Zhu, Q. (2019). Reachnn: Reachability analysis of neural-network controlled systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22.

[71] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017a). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

[72] Huang, X., Kwiatkowska, M., Wang, S., and Wu, M. (2017b). Safety verification of deep neural networks. In *International Conference on Computer Aided Verification*, pages 3–29. Springer.

[73] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.

[74] Ivanov, R., Weimer, J., Alur, R., Pappas, G. J., and Lee, I. (2019). Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178.

[75] Jenkins, B. and Tanguay, A. (1995). Handbook of neural computing and neural networks.

[76] Kamann, C. and Rother, C. (2020). Benchmarking the robustness of semantic segmentation models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8828–8838.

[77] Katz, G., Barrett, C., Dill, D. L., Julian, K., and Kochenderfer, M. J. (2017). Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer.

[78] Katz, G., Huang, D. A., Ibeling, D., Julian, K., Lazarus, C., Lim, R., Shah, P., Thakoor, S., Wu, H., Zeljić, A., et al. (2019). The marabou framework for verification and analysis of deep neural networks. In *International Conference on Computer Aided Verification*, pages 443–452. Springer.

[79] Kauffman, S., Dunne, M., Gracioli, G., Khan, W., Benann, N., and Fischmeister, S. (2021). Palisade: A framework for anomaly detection in embedded systems. *Journal of Systems Architecture*, 113:101876.

[80] Kollias, D., Tagaris, A., Stafylopatis, A., Kollias, S., and Tagaris, G. (2018). Deep neural architectures for prediction in healthcare. *Complex & Intelligent Systems*, 4:119–131.

[81] Kollmeyer, P., Vidal, C., Naguib, M., and Skells, M. (2020). Lg 18650hg2 li-ion battery data and example deep neural network xev soc estimator script. *Mendeley Data*, 3:2020.

[82] Kouvaros, P. and Lomuscio, A. (2018). Formal verification of cnn-based perception systems. *arXiv preprint arXiv:1811.11373*.

[83] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images. Technical report, Citeseer.

[84] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

[85] Kudo, M., Toyama, J., and Shimbo, M. (1999). Multidimensional curve classification using passing-through regions. *Pattern Recognition Letters*, 20(11-13):1103–1111.

[86] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*.

[87] Lawrence, S., Giles, C. L., Tsoi, A. C., and Back, A. D. (1997). Face recognition: A convolutional neural-network approach. *IEEE transactions on neural networks*, 8(1):98–113.

[88] LeCun, Y. (1987). Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models).

[89] LeCun, Y. (1998). The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*.

[90] LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., and Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551.

[91] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[92] Li, B., Liu, S., Xu, W., and Qiu, W. (2018). Real-time object detection and semantic segmentation for autonomous driving. In *MIPPR 2017: Automatic Target Recognition and Navigation*, volume 10608, pages 167–174. SPIE.

[93] Lin, C.-Y., Hsieh, Y.-M., Cheng, F.-T., Huang, H.-C., and Adnan, M. (2019). Time series prediction algorithm for intelligent predictive maintenance. *IEEE Robotics and Automation Letters*, 4(3):2807–2814.

[94] Liu, C., Arnon, T., Lazarus, C., Strong, C., Barrett, C., Kochenderfer, M. J., et al. (2021). Algorithms for verifying deep neural networks. *Foundations and Trends® in Optimization*, 4(3-4):244–404.

[95] Lomuscio, A. and Maganti, L. (2017). An approach to reachability analysis for feed-forward relu neural networks. *arXiv preprint arXiv:1706.07351*.

[96] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440.

[97] Lopez, D. M., Choi, S. W., Tran, H.-D., and Johnson, T. T. (2023). Nnv 2.0: the neural network verification tool. In *International Conference on Computer Aided Verification*, pages 397–412. Springer.

[98] Lv, F., Wen, C., Liu, M., and Bao, Z. (2017). Weighted time series fault diagnosis based on a stacked sparse autoencoder. *Journal of Chemometrics*, 31(9):e2912.

[Mathworks Classify Sound] Mathworks Classify Sound. Classify sound using deep learning - matlab & simulink — mathworks.com. https://www.mathworks.com/help/audio/gs/classify-sound-using-deep-learning.html.

[Mathworks Sequence Classification] Mathworks Sequence Classification. Sequence Classification Using Deep Learning - MATLAB &; Simulink — mathworks.com. https://www.mathworks.com/help/deeplearning/ug/classify-sequence-data-using-lstm-networks.html.

[Mathworks Sequence Classification1d] Mathworks Sequence Classification1d. Sequence Classification Using 1-D Convolutions - MATLAB &; Simulink — mathworks.com. https://www.mathworks.com/help/deeplearning/ug/sequence-classification-using-1-d-convolutions.html.

[102] Mittal, S. and Hasija, Y. (2020). Applications of deep learning in healthcare and biomedicine. *Deep learning techniques for biomedical and health informatics*, pages 57–77.

[103] Modegi, T. and Iisaku, S.-i. (1997). Application of midi technique for medical audio signal coding. In *Proceedings of the 19th Annual International Conference of the IEEE Engineering in Medicine and Biology Society.'Magnificent Milestones and Emerging Opportunities in Medical Engineering'(Cat. No. 97CH36136)*, volume 4, pages 1417–1420. IEEE.

[104] Mohapatra, J., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. (2020a). Towards verifying robustness of neural networks against a family of semantic perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 244–252.

[105] Mohapatra, J., Weng, T.-W., Chen, P.-Y., Liu, S., and Daniel, L. (2020b). Towards verifying robustness of neural networks against a family of semantic perturbations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[106] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016a). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.

[107] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016b). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.

[108] Müller, M. N., Brix, C., Bak, S., Liu, C., and Johnson, T. T. (2022). The third international verification of neural networks competition (vnn-comp 2022): summary and results. *arXiv preprint arXiv:2212.10376*.

[109] Müller, M. N., Makarchuk, G., Singh, G., Püschel, M., and Vechev, M. (2021). Prima: Precise and general neural network certification via multi-neuron convex relaxations. *arXiv preprint arXiv:2103.03638*.

[110] Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. (2016). Wavenet: A generative model for raw audio. *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, page 125.

[111] Ozbayoglu, A. M., Gudelek, M. U., and Sezer, O. B. (2020). Deep learning for financial applications: A survey. *Applied Soft Computing*, 93:106384.

[112] Pal, N. and Johnson, T. T. (2022). Work in progress: Safety and robustness verification of autoencoder-based regression models using the nnv tool. *arXiv preprint arXiv:2207.06759*.

[113] Pal, N. and Johnson, T. T. (2023). Formal verification of long short-term memory based audio classifiers: A star based approach. *arXiv preprint arXiv:2311.12130*.

[114] Pal, N., Lee, S., and Johnson, T. T. (2023a). Benchmark: formal verification of semantic segmentation neural networks. In *International Conference on Bridging the Gap between AI and Reality*, pages 311–330. Springer.

[115] Pal, N., Lopez, D. M., and Johnson, T. T. (2023b). Robustness verification of deep neural networks using star-based reachability analysis with variable-length time series input. In *International Conference on Formal Methods for Industrial Critical Systems*, pages 170–188. Springer.

[116] Papernot, N., McDaniel, P., Jha, S., Fredrikson, M., Celik, Z. B., and Swami, A. (2016). The limitations of deep learning in adversarial settings. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387. IEEE.

[117] Pearson, K. (1901). Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572.

[118] Priemer, R. (1991a). *Introductory signal processing*, volume 6. World Scientific.

[119] Priemer, R. (1991b). Signals and signal processing. *Introductory Signal Processing*, pages 1–9.

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

[121] Raja, G., Senthilkumar, S., Ganesan, S., Edhayachandran, R., Vijayaraghavan, G., and Bashir, A. K. (2021). Av-cps: audio visual cognitive processing system for critical intervention in autonomous vehicles. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–6. IEEE.

[122] Rakesh, K., Kumar, L. S., Mittar, R., Chakraborty, P., Ankush, P., and Gairuboina, S. K. (2020). Dnn based adaptive video streaming using combination of supervised learning and reinforcement learning. In *Computer Vision and Image Processing: 4th International Conference, CVIP 2019, Jaipur, India, September 27–29, 2019, Revised Selected Papers, Part II 4*, pages 143–154. Springer.

[123] Rauber, J., Brendel, W., and Bethge, M. (2017). Foolbox: A python toolbox to benchmark the robustness of machine learning models. *arXiv preprint arXiv:1707.04131*.

[124] Ringnér, M. (2008). What is principal component analysis? *Nature biotechnology*, 26(3):303–304.

[125] Roberts, A., Engel, J., Raffel, C., Hawthorne, C., and Eck, D. (2018). A hierarchical latent vector model for learning long-term structure in music. In *International conference on machine learning*, pages 4364–4373. PMLR.

[126] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[127] Ruan, W., Huang, X., and Kwiatkowska, M. (2018). Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*.

[128] Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., and Kwiatkowska, M. (2019). Global robustness evaluation of deep neural networks with provable guarantees for the l_0 norm. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 5944–5952.

[129] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252.

[130] Saxe, A. M., McClelland, J. L., and Ganguli, S. (2013). Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*.

[131] Saxena, A. and Goebel, K. (2008). Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository*, pages 1551–3203.

[132] Saxena, D. and Raychoudhury, V. (2017). Design and verification of an ndn-based safety-critical application: A case study with smart healthcare. *ieee transactions on systems, man, and cybernetics: systems*, 49(5):991–1005.

[133] Semenick Alam, I. M. and Sickles, R. C. (2000). Time series analysis of deregulatory dynamics and technical efficiency: the case of the us airline industry. *International Economic Review*, 41(1):203–218.

[134] Serre, F., Müller, C., Singh, G., Püschel, M., and Vechev, M. (2021). Scaling polyhedral neural network verification on GPUs. In *Proc. Machine Learning and Systems (MLSys)*.

[135] Shamshirband, S., Fathi, M., Dehzangi, A., Chronopoulos, A. T., and Alinejad-Rokny, H. (2021). A review on deep learning approaches in healthcare systems: Taxonomies, challenges, and open issues. *Journal of Biomedical Informatics*, 113:103627.

[136] Shriver, D., Elbaum, S., and Dwyer, M. B. (2021). Dnnv: A framework for deep neural network verification. In *International Conference on Computer Aided Verification*, pages 137–150. Springer.

[137] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[138] Singh, G., Ganvir, R., Püschel, M., and Vechev, M. (2019a). Beyond the single neuron convex barrier for neural network certification. In *Advances in Neural Information Processing Systems 32*, pages 15098–15109. Curran Associates, Inc.

[139] Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. (2018a). Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10825–10836.

[140] Singh, G., Gehr, T., Mirman, M., Püschel, M., and Vechev, M. (2018b). Fast and effective robustness certification. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 10802–10813. Curran Associates, Inc.

[141] Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019b). An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41.

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

[143] Singh, G., Gehr, T., Püschel, M., and Vechev, M. (2019d). Boosting robustness certification of neural networks. In *International Conference on Learning Representations (ICLR)*.

[144] Singh, G., Püschel, M., and Vechev, M. (2017). Fast polyhedra abstract domain. In *Proc. Principles of Programming Languages (POPL)*, pages 46–59.

[145] Sivaraman, A., Farnadi, G., Millstein, T., and Van den Broeck, G. (2020). Counterexample-guided learning of monotonic neural networks. *Advances in Neural Information Processing Systems*, 33:11936–11948.

[146] Soomro, K., Bhutta, M. N. M., Khan, Z., and Tahir, M. A. (2019). Smart city big data analytics: An advanced review. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(5):e1319.

[147] Srivastava, R. K., Greff, K., and Schmidhuber, J. (2015). Highway networks. *arXiv preprint arXiv:1505.00387*.

[148] Storn, R. and Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359.

[149] Stübinger, J. and Schneider, L. (2020). Understanding smart city—a data-driven literature review. *Sustainability*, 12(20):8460.

[150] Susto, G. A. and Beghi, A. (2016). Dealing with time-series data in predictive maintenance problems. In *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–4. IEEE.

[151] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.

[152] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *2nd International Conference on Learning Representations, ICLR 2014*.

[153] Szeliski, R. (2021). Computer vision: algorithms and applications 2nd edition.

[154] Tao, X., Zhang, D., Ma, W., Liu, X., and Xu, D. (2018). Automatic metallic surface defect detection and recognition with convolutional neural networks. *Applied Sciences*, 8(9):1575.

[155] Thoma, M. (2016). A survey of semantic segmentation. *arXiv preprint arXiv:1602.06541*.

[156] Tjeng, V., Xiao, K., and Tedrake, R. (2017). Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*.

[157] Tjeng, V., Xiao, K. Y., and Tedrake, R. (2019). Evaluating robustness of neural networks with mixed integer programming. In *International Conference on Learning Representations*.

[158] Touloumi, G., Atkinson, R., Tertre, A. L., Samoli, E., Schwartz, J., Schindler, C., Vonk, J. M., Rossi, G., Saez, M., Rabszenko, D., et al. (2004). Analysis of health outcome time series data in epidemiological studies. *Environmetrics: The official journal of the International Environmetrics Society*, 15(2):101–117.

[159] Tran, H. D. (2019-20). Towards verification of large convolutional neural networks using imagestars.

[160] Tran, H.-D., Bak, S., Xiang, W., and Johnson, T. T. (2020a). Verification of deep convolutional neural networks using imagestars. In *32nd International Conference on Computer-Aided Verification (CAV)*. Springer.

[161] Tran, H.-D., Cai, F., Diego, M. L., Musau, P., Johnson, T. T., and Koutsoukos, X. (2019a). Safety verification of cyber-physical systems with reinforcement learning control. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–22.

[162] Tran, H.-D., Cei, F., Lopez, D. M., Johnson, T. T., and Koutsoukos, X. (2019b). Safety verification of cyber-physical systems with reinforcement learning control. In *ACM SIGBED International Conference on Embedded Software (EMSOFT'19)*. ACM.

[163] Tran, H. D., Choi, S. W., Yang, X., Yamaguchi, T., Hoxha, B., and Prokhorov, D. (2023). Verification of recurrent neural networks with star reachability. In *Proceedings of the 26th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–13.

[164] Tran, H.-D., Lopez, D. M., Musau, P., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. (2019c). Star-based reachability analysis of deep neural networks. In *International Symposium on Formal Methods*, pages 670–686. Springer.

[165] Tran, H.-D., Musau, P., Lopez, D. M., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. (2019d). Parallelizable reachability analysis algorithms for feed-forward neural networks. In *7th International Conference on Formal Methods in Software Engineering (FormaliSE2019), Montreal, Canada*.

[166] Tran, H.-D., Musau, P., Lopez, D. M., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. (2019e). Parallelizable reachability analysis algorithms for feed-forward neural networks. In *2019 IEEE/ACM 7th International Conference on Formal Methods in Software Engineering (FormaliSE)*, pages 51–60. IEEE.

[167] Tran, H.-D., Musau, P., Lopez, D. M., Yang, X., Nguyen, L. V., Xiang, W., and Johnson, T. T. (2019f). Star-based reachability analsysis for deep neural networks. In *23rd International Symposisum on Formal Methods (FM'19)*. Springer International Publishing.

[168] Tran, H.-D., Pal, N., Musau, P., Lopez, D. M., Hamilton, N., Yang, X., Bak, S., and Johnson, T. T. (2021). Robustness verification of semantic segmentation neural networks using relaxed reachability. In *International Conference on Computer Aided Verification*, pages 263–286. Springer.

[169] Tran, H.-D., Xiang, W., and Johnson, T. T. (2020b). Verification approaches for learning-enabled autonomous cyber–physical systems. *IEEE Design & Test*, 39(1):24–34.

[170] Tran, H.-D., Yang, X., Lopez, D. M., Musau, P., Nguyen, L. V., Xiang, W., Bak, S., and Johnson, T. T. (2020c). 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.

[171] Truax, B. (1999). *Handbook for acoustic ecology*. Cambridge Street Records.

[172] Tseng, Y.-H. and Jan, S.-S. (2018). Combination of computer vision detection and segmentation for autonomous driving. In *2018 IEEE/ION Position, Location and Navigation Symposium (PLANS)*, pages 1047–1052. IEEE.

[173] Walden, F., Dasgupta, S., Rahman, M., and Islam, M. (2022). Improving the environmental perception of autonomous vehicles using deep learning-based audio classification. *CoRR*, abs/2209.04075.

[174] Wang, A. et al. (2003). An industrial strength audio search algorithm. In *ISMIR 2003, 4th International Conference on Music Information Retrieval, Baltimore, Maryland, USA, October 27-30, 2003, Proceedings*, pages 7–13. Washington, DC.

[175] Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018). Efficient formal safety analysis of neural networks. In *Advances in Neural Information Processing Systems*, pages 6369–6379.

[176] Wang, S., Zhang, H., Xu, K., Lin, X., Jana, S., Hsieh, C.-J., and Kolter, Z. (2021). Beta-CROWN: Efficient bound propagation with per-neuron split constraints for complete and incomplete neural network verification. *arXiv preprint arXiv:2103.06624*.

[177] Wang, Z., Wang, Y., Fu, F., Jiao, R., Huang, C., Li, W., and Zhu, Q. (2022). A tool for neural network global robustness certification and training. *CoRR*, abs/2208.07289.

[178] Wang, Z., Wei, L., Wang, L., Gao, Y., Chen, W., and Shen, D. (2017). Hierarchical vertex regression-based segmentation of head and neck ct images for radiotherapy planning. *IEEE Transactions on Image Processing*, 27(2):923–937.

[179] Xiang, W. and Johnson, T. T. (2018). Reachability analysis and safety verification for neural network control systems. *arXiv preprint arXiv:1805.09944*.

[180] Xiang, W., Musau, P., Wild, A. A., Lopez, D. M., Hamilton, N., Yang, X., Rosenfeld, J., and Johnson, T. T. (2018a). Verification for machine learning, autonomy, and neural networks survey. *arXiv preprint arXiv:1810.01989*.

[181] Xiang, W., Tran, H.-D., and Johnson, T. T. (2017). Reachable set computation and safety verification for neural networks with relu activations. *arXiv preprint arXiv:1712.08163*.

[182] Xiang, W., Tran, H.-D., and Johnson, T. T. (2018b). Output reachable set estimation and verification for multilayer neural networks. *IEEE transactions on neural networks and learning systems*, (99):1–7.

[183] Xie, X., Kersting, K., and Neider, D. (2022). Neuro-symbolic verification of deep neural networks. *arXiv preprint arXiv:2203.00938*.

[184] Xu, K., Shi, Z., Zhang, H., Wang, Y., Chang, K.-W., Huang, M., Kailkhura, B., Lin, X., and Hsieh, C.-J. (2020). Automatic perturbation analysis for scalable certified robustness and beyond. *Advances in Neural Information Processing Systems*, 33.

[185] Xu, K., Zhang, H., Wang, S., Wang, Y., Jana, S., Lin, X., and Hsieh, C.-J. (2021). Fast and Complete: Enabling complete neural network verification with rapid and massively parallel incomplete verifiers. In *International Conference on Learning Representations*.

[186] Yuan, X., He, P., Zhu, Q., and Li, X. (2019). Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9):2805–2824.

[187] Zeger, S. L., Irizarry, R., and Peng, R. D. (2006). On time series analysis of public health and biomedical data. *Annu. Rev. Public Health*, 27:57–79.

[188] Zhang, D., Zhang, H., Zhou, H., Bao, X., Huo, D., Chen, R., Cheng, X., Wu, M., and Zhang, Q. (2021a). Building interpretable interaction trees for deep nlp models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 14328–14337.

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

[190] Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. (2018a). Efficient neural network robustness certification with general activation functions. *Advances in Neural Information Processing Systems*, 31:4939–4948.

[191] Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., and Daniel, L. (2018b). Efficient neural network robustness certification with general activation functions. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31, pages 4939–4948. Curran Associates, Inc.

[192] Zhang, Z., Lai, X., Wu, M., Chen, L., Lu, C., and Du, S. (2021b). Fault diagnosis based on feature clustering of time series data for loss and kick of drilling process. *Journal of Process Control*, 102:24–33.

[193] Zhao, J., Mao, X., and Chen, L. (2019). Speech emotion recognition using deep 1d & 2d cnn lstm networks. *Biomedical signal processing and control*, 47:312–323.

[194] Zhu, X., Suk, H.-I., Lee, S.-W., and Shen, D. (2015). Subspace regularized sparse multitask learning for multiclass neurodegenerative disease identification. *IEEE Transactions on Biomedical Engineering*, 63(3):607–618.

[195] Zhu, X., Suk, H.-I., and Shen, D. (2014). A novel matrix-similarity based loss function for joint regression and classification in ad diagnosis. *NeuroImage*, 100:91–105.