A Framework of Two-level Specification-based Data Generation for Deep Neural Networks

Dissertation submitted in partial fulfillment for the
degree of Master of Informatics and Data Science

Yanzhao Xia

Under the supervision of
Professor Shaoying Liu

Dependable System Laboratory,
Department of Informatics and Data Science,
Graduate School of Advanced Science and Engineering,
Hiroshima University, Higashi-Hiroshima, Japan

January 2024

Abstract

Deep Neural Networks (DNNs) have garnered increased attention in domain-specific supervised learning applications. However, two major challenges persist in contemporary DNNs: the difficulty of obtaining well-labeled training data for supervised learning and the inefficiency of training due to the lack of characterization of objects in the training process. This research addresses these challenges through the development of a comprehensive framework for formal specification-based data generation in DNNs.

The framework relies on formal specifications to precisely define object characteristics, serving as the foundation for effective training and testing data generation. Our initial work established this framework, delving into all activities involved and presenting a detailed approach to writing formal specifications. However, identified limitations, such as insufficient formality in specification descriptions and a gap between specifications and generated data, prompted a refined approach.

To address these limitations, we introduce a two-level specification method within the framework. The first level focuses on detailing object characteristics, while the second level defines parameters and values for data generation. This novel approach enhances both human comprehension and machine handling, significantly reducing the gap between specifications and generated data.

This dissertation offers a detailed exploration of the entire framework and the two-level specification method. Through a case study on traffic sign recognition, we demonstrate the performance of the specification in describing object characteristics and facilitating data generation.

Keywords:

Formal methods, Specification, SOFL, Traffic sign recognition

Acknowledgments

First and foremost, I extend my deepest gratitude to my supervisor, Professor Shaoying Liu, for his invaluable guidance, patience, and continuous support throughout my research journey. His insights and expertise have been pivotal in shaping both the direction and success of my work.

Besides, I am grateful to my co-supervisors, Professor Hiroyuki Okamura and Professor Hiroaki Mukaidani, whose expertise and constructive feedback have significantly contributed to my research and personal development. Their dedication and willingness to impart knowledge have been immensely beneficial.

In addition, I would like to acknowledge the past and present members of the Dependable System Laboratory, Department of Informatics and Data Science. Working alongside such a talented and supportive group has been an enriching experience. Their camaraderie, collaborative spirit, and shared wisdom have greatly enhanced my time at the laboratory.

A special thanks goes to my family, whose unwavering love and support have been my constant source of strength and motivation. Their sacrifices, encouragement, and belief in my abilities have been the bedrock of my Master's graduation.

Lastly, I am thankful for all those who have contributed to my journey, whether directly or indirectly, in the successful completion of my master's thesis. This achievement is a testament to the collective effort and encouragement of each one of you.

Contents

Chapter 1

Introduction

DNNs have emerged as vital tools for implementing Deep Learning (DL), playing a significant role in supervised learning applications over the past decade [1–4]. Despite this progress, the current supervised DNNs still have some limits. Firstly, since the result of DNNs significantly relies on data, to ensure the accuracy of the DNNs, the training process requests a large-scale well-labeled data. However, in the field of supervised learning, well-labeled data may cost plenty of human resources and time to obtain. Secondly, due to the inherent calculation process, deep learning models treat data as vector values that may not clearly reflect the characteristics of the original data. This kind of omission might influence the effectiveness of the training process.

On the other hand, formal languages, formed by strict mathematical rules and syntax, have long been applied in conventional software development to define the user's requirements and/or the functionality of the systems to be implemented [5, 6]. Many formal specification languages have been proposed and/or applied in real development projects, such as Vienna Development Method – Specification Language (VDM-SL) [7] and Structured Object-Oriented Formal Language (SOFL) [8], but few existing studies utilize formal specification to improve the performance of deep learning systems. Our experience suggests that formal specifications can facilitate precise characteristics descriptions of objects, enabling effective data generation for DL systems.

Our initial research [9] endeavors a comprehensive framework that harnesses the advantages inherent in formal specifications to autonomously generate labeled data for DNNs. This framework works across a sequence of five steps. Firstly, we articulate the characteristics of the original object data through the utilization of SOFL specifications. These characteristics are described by defining attributes and their interrelationships, serving as the discriminating factors that distinguish the target object from others. For instance, in the case of recognizing a bicycle, the specification highlights two wheels as pivotal attributes. Secondly, we integrate the formal specifications into the data generation process. This procedure is designed to ensure that the generated data faithfully mirrors the characteristics of the object described in the formal specifications. Thirdly, we utilize the labeled data generated in the second step to train a DNN model. This strategy via the generated data empowers the model to effectively learn and recognize the specified characteristics, facilitating the training process. Fourthly, the next step encompasses a critical validation process, evaluating the accuracy of the trained model. This is achieved by subjecting the model to a separate set of real-life data, such as photographs, representing the original data. This step serves as a comprehensive validation mechanism, thoroughly assessing the model's performance in accurately identifying the target object under real-world conditions. Fifthly, besides the fundamental four procedures outlined above, the framework introduces an innovative concept to generate adversarial data through precise modifications to the formal specification. This novel addition broadens the framework's utility by enabling the generation of data specifically crafted to challenge and fortify the model against adversarial examples. Through these five cohesive steps, the framework is expected to not only establish a robust foundation for labeled data generation but also contribute novel insights into improving the accuracy and reliability of DNNs.

However, this existing framework exhibits limitations, particularly in the lack of formality in specification descriptions and a gap between the specification and the generated data. This is because the specifications prioritize the accuracy of the description of the specifications, rather than providing exact

mathematical parameters to direct the transformation of characteristics into visual images or precise numerical values. The lack of detailed parameters makes the specifications easy for stakeholders to understand the content, but increases the difficulty for the data generation processes. To address this, we propose a two-level specification method employing different purposes for two levels of formal specifications.

The first level focuses on presenting object characteristics in a human-readable manner, utilizing natural language and descriptive terms. However, this level lacks the precise values required for accurate data generation. To bridge this gap, the second level specification provides detailed mathematical values and parameters for precise data generation. For instance, instead of describing a shape as a "Triangle," the second level assigns precise dimensions and angles for accurate data representation.

This two-level specification approach aims to reduce the gap between specification and generated data. The second level specification offers the machine with the necessary information to generate data aligned with the formal specifications, ensuring an accurate representation of intended object characteristics for DNN training.

In this dissertation, we undertake a comprehensive discussion of all activities involved in the framework. Focusing on the foundation of data generation through formal specifications, we explore how the two-level formal specification can be written to reflect distinct object characteristics and facilitate the data generation process. To validate our method's effectiveness, we conduct a case study on traffic sign recognition, evaluating the workflow of specification generation. This method provides a novel perspective on the data source for DNNs, particularly in the context of object recognition.

The subsequent sections of this dissertation are shown as follows. Section 2 serves as background information by introducing key terms and definitions integral to both deep learning and formal methods. This section aims to establish a comprehensive background for readers, ensuring a clear understanding of the terminology used throughout the paper. Section 3 provides a meticulous in-

troduction to our framework, offering an intricate exploration of the individual activities of our proposed framework. Section 4 discusses the two-level specification approach, laying out a systematic method for crafting specifications for the purpose of enhancing both human comprehension and machine handling. In Section 5, we concentrate on a real-world application through the presentation of a case study of traffic sign recognition. Section 6 takes a dive into the analysis and discussion of the strengths and limitations of our proposed method. Section 7 introduces some existing related works. Lastly, Section 8 presents a comprehensive summary of the contributions made throughout this dissertation and potential future research.

Chapter 2

Background

2.1 DNNs

Deep Neural Networks (DNNs) [10] constitute a powerful paradigm within the domain of machine learning. At their core, DNNs mimic the human brain's structure by employing multiple layers of interconnected nodes, enabling the model to learn intricate hierarchical representations from data. This architectural depth allows DNNs to discern complex patterns and relationships in the input data.

Mathematically, a DNN $M(x) = y$ is a function from input data $x \in R^n$ and the predicted probability vector $y \in R^m$. The vector y would be used to determine the final predicted decisions for some DNN tasks. The behavior of a neural network can be expressed as a series of transformations. In specific, the output of a single layer can be denoted as:

$$y = \sigma(Wx + b), \tag{2.1}$$

where x represents the input to the network, W the weight matrix, b the bias vector, and σ the activation function.

For a DNN with multiple layers, the output of each layer serves as the input to the next:

$$y = \sigma(W_L\sigma(W_{L-1}...\sigma(W_1x + b_1)... + b_{L-1}) + b_L), \tag{2.2}$$

where W_i and b_i represent the weights and biases of the i-th layer, and σ is the activation function applied element-wise.

Typically, as depicted in Figure 2.1, Deep Neural Networks (DNNs) exhibit hierarchical structures [11]. In the computational sequence, DNNs systematically process input data layer by layer, relying on the internal parameters of the model to generate the prediction outcome. This contrasts with conventional software development, where developers explicitly code decision logic. DNNs adhere to a data-driven programming paradigm, where developers concentrate on gathering pertinent data, defining the architecture, and creating training programs. The decision logic of a DNN is autonomously acquired through the training process.

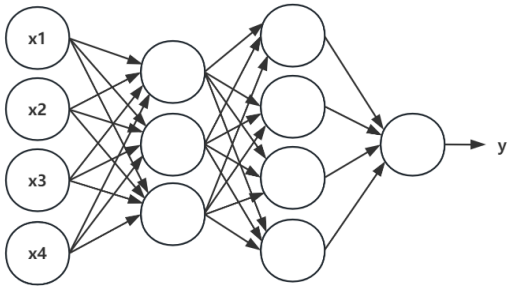

Figure 2.1: Hierarchical structures of DNNs

2.2 Deep learning

Deep Learning [12] represents a subset of machine learning methodologies that specifically harnesses the capabilities of DNNs to learn intricate representations of data.

The relationships between Deep Learning and DNNs are shown as follows. Deep Learning, as a broader concept, encompasses the use of neural networks with multiple layers, i.e., Deep Neural Networks (DNNs). DNNs, with their hierarchical architecture, enable deep learning models to automatically learn intricate representations of data through successive layers [11]. In other words, Deep Learning involves the application of DNNs, utilizing their capacity for

hierarchical feature extraction and representation learning. DNNs, in turn, are the foundational models within the broader field of Deep Learning.

2.3 Supervised learning

Supervised Learning [1] is a foundational paradigm in machine learning where the model is trained on a labeled dataset. In this setting, the algorithm learns to map input data to corresponding output labels by generalizing from the provided examples. This learning approach is particularly relevant for tasks where the goal is to predict or classify based on historical data with known outcomes.

Supervised Learning involves minimizing a predefined loss function L by adjusting the model parameters to optimize performance on the training set. Mathematically, this optimization can be expressed as:

$$min_\theta \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i; \theta)), \tag{2.3}$$

where N is the size of the training set, y_i is the ground truth label for the i-th example, $f(x_i; \theta)$ is the model's prediction, and θ represents the model parameters.

While Supervised Learning relies on labeled datasets, there exist other learning paradigms. Unsupervised Learning, in contrast, deals with unlabeled data, aiming to discover inherent patterns or structures within the data. Semi-supervised learning lies between these two, incorporating both labeled and unlabeled data for training. Each paradigm serves distinct purposes: Supervised Learning for tasks with labeled data, Unsupervised Learning [13] for exploring hidden structures, and Semi-Supervised Learning [14] for scenarios with limited labeled instances.

2.4 Data set

A Dataset forms the foundational bedrock for any machine learning endeavor, serving as the raw material upon which models are trained, tested, and evaluated. It is a structured collection of data instances, typically organized into

input-output pairs in the context of supervised learning. A well-constructed dataset is crucial for ensuring the generalization and robustness of machine learning models [15].

A dataset comprises two fundamental components:

- Input Data (X): This represents the features or attributes of the instances in the dataset. For instance, in an image classification task, each image's pixel values, size, and color channels constitute the input data.
- Output Labels (Y): In supervised learning, each instance in the dataset is associated with a corresponding label or output. For the image classification example, the labels might indicate the category of the depicted object (e.g., cat, dog).

Datasets can be categorized based on several criteria:

- Training Dataset: Used to train the machine learning model.
- Testing Dataset: Independent dataset used to evaluate the model's performance and generalization.
- Validation Dataset: An additional subset used during the training phase to fine-tune model parameters and avoid overfitting.
- Unlabeled Dataset: Used in unsupervised learning, where the data lacks predefined labels.

The quality and representativeness of a dataset significantly impact the performance of machine learning models. Biases, insufficient diversity, or data anomalies can lead to models that perform poorly in real-world scenarios.

2.5 Training process

The training process of DNN models is a fundamental stage in the field of machine learning, where the model learns to recognize patterns and make predictions based on input data [16, 17]. Understanding this process is crucial for developing effective and accurate models.

Key Components of DNN Training:

- Neural Network Architecture: DNNs consist of layers of interconnected neurons, forming a complex architecture. This architecture determines the model's capacity to learn hierarchical features from the input data.
- Loss Function: The loss function quantifies the difference between the model's predictions and the actual labels in the training data. It serves as the optimization objective during training, guiding the model to minimize prediction errors.
- Backpropagation: Backpropagation [18] is a key optimization algorithm in DNN training. It involves iteratively adjusting the model's weights and biases based on the gradients of the loss function with respect to these parameters. This process enables the model to learn and improve its performance over time.
- Activation Functions: Activation functions introduce non-linearities to the model, allowing it to capture complex relationships in the data. Common activation functions include ReLU (Rectified Linear Unit) [19] and Sigmoid [20].
- Optimization Algorithms: Various optimization algorithms, such as Stochastic Gradient Descent (SGD) [21] and Adam [22], are employed to efficiently update the model parameters during training, facilitating convergence to a minimum of the loss function.

2.6 Formal language

Formal Language is a structured and precise system of symbols and rules used for the unambiguous representation of information. It serves as a means of communication between humans and, crucially, between humans and machines. In conventional software development, formal Languages are well used for illustrating the user's requirements and/or the functionality of the program or

systems [5, 6]. Many formal specification languages have been proposed and/or applied in real development projects, such as Vienna Development Method – Specification Language (VDM-SL) [7] and Structured Object-Oriented Formal Language (SOFL) [8].

2.7 Formal specification

Formal specification is one of the three techniques of formal methods, the other two are refinement, and formal verification. Some formal methods are already used in the field of software design and development, such as VDM [23], Z [24], Event-B [25], SCADE [26], and SOFL [8]. In contrast to other design systems, formal specifications perform appropriate mathematical analysis to ensure correct behavior. The utilization of formal specifications can contribute to the reliability and robustness of a system design. The use of formal specifications offers several advantages: 1). Precision: Formal specifications leave little room for ambiguity, providing precise guidelines for the generation of labeled data. 2). Automation: Automated processes can be employed to generate data based on formal specifications, enhancing efficiency and consistency. 3). Verification: The formal nature of specifications allows for rigorous verification of the generated data against the defined characteristics.

2.8 SOFL

Structured Object-Oriented Formal Language (SOFL), developed by Liu, is a formal method integrating VDM-SL with data flow diagrams, commonly used in software engineering. It offers a systematic approach combining object-oriented modeling with formal specification techniques to precisely define system behaviors and structures. SOFL employs a three-step formal specification approach that assists in designing software systems and their evolution into object-oriented implementations. The usage of SOFL also facilitates formal verification through inspection and testing based on specifications.

As depicted in Figure 2.2, key components of SOFL include Conditional

Data Flow Diagrams (CDFD), data flows, modules, types, and processes.

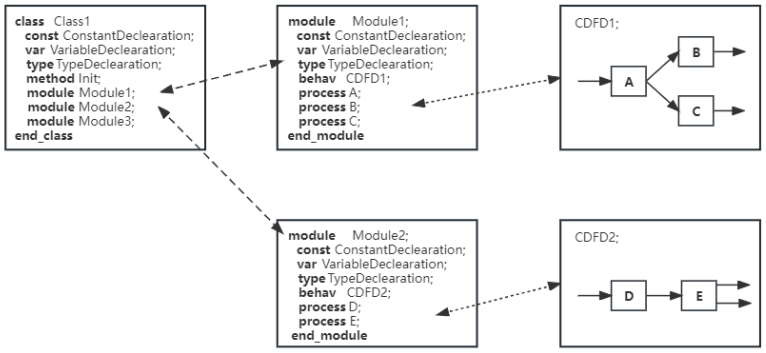

Figure 2.2: The structure of a SOFL specification

2.9 Adversarial data

Adversarial data refers to intentionally crafted input instances designed to deceive machine learning models, causing them to make incorrect predictions or classifications. The existence of adversarial data poses a significant challenge to the robustness and reliability of DNNs.

The key factor of adversarial data is the small perturbations. Adversarial samples often involve minimal, imperceptible changes to the input data. These subtle modifications can cause significant shifts in the model's predictions, while still keeping the same result to the sense of human beings.

In mathematical terms, let x represent the original input with a ground truth label y, and assume the prediction of DNN, M, aligns with the ground truth label, i.e., $M(x) = y$. An adversarial example $x' = x + \delta$ is crafted by introducing a slight perturbation δ, based on specific criteria. Notably, in certain scenarios, the DNN's prediction of the adversarial example x' may deviate from the original prediction y (i.e., $M(x') = M(x + \delta) \neq M(x)$). Consequently, this discrepancy can breach the decision boundary, leading the model astray and resulting in incorrect decisions.

Chapter 3

The Proposed Framework

Our framework of the method for data generation provides a procedure that is divided into five steps: *Preparation*, *Data generation*, *Model training*, *Evaluation*, and *Adversarial data generation*. Figure 3.1 illustrates the procedure and below we discuss each step involved in the procedure, respectively. In this section, we present the procedure by discussing the issues involved in each step.

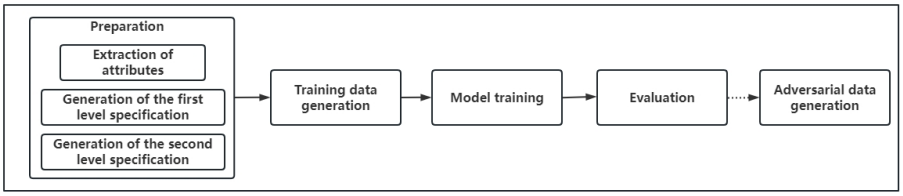

Figure 3.1: The workflow of the method

3.1 Preparation

The first step in our methodology is the preparation stage, focusing on developing formal specifications for DNNs. Due to the inherent difference between the logic of DNNs and conventional software, the process of writing specifications differs significantly from traditional ones as well. In conventional systems, developers explicitly code the decision logic, whereas DNNs operate on a data-driven programming paradigm. Hence, the developer's primary role shifts to gathering effective data, selecting appropriate DNN architectures, and oversee-

ing the training process, where the decision logic is formed implicitly through data.

Recent studies highlight the complexities of writing formal specifications for DNNs [27, 28], especially for tasks that imitate human perception. The inherent complexity of DNNs, characterized by multiple layers and numerous parameters, complicates the task of defining their behavior through traditional formal methods. Unlike conventional software, the behavior of DNNs is largely dictated by the training data, making it challenging to encapsulate their functionality in explicit rules and logical constraints inherent in formal specifications.

However, formal specifications are still important for outlining the characteristics of training data. By defining desired properties and constraints, they guide the generation and selection of training samples. In tasks like image classification, formal specifications can delineate critical object attributes like shape, color, and texture, which the DNN is expected to recognize. Thus, formal specifications are used to indirectly shape the behavior of DNNs by influencing the nature of their training data.

The preparation step encompasses three sub-stages: 1) extraction of attributes, 2) generation of the first-level specification, and 3) generation of the second-level specification, each of which will be elaborated in Section 4. These stages collectively facilitate the creation of formal specifications that describe the essential characteristics of the original data, setting the foundation for our framework.

3.2 Data generation

After establishing formal specifications in the preparation stage, the subsequent crucial step is data generation. This process, integral to training the DNN effectively, involves creating a dataset that accurately mirrors the established specifications.

When embarking on this process, the specific task addressed by the DNN is a critical consideration. Different tasks necessitate distinct types of input data,

such as images for object recognition or text for natural language processing. The choice of techniques, drawn from computer graphics or other relevant fields, hinges on the nature required of data for these tasks. The complexity of types of data makes it a challenge to propose a common method for data generation.

However, despite the difficulty of generating different kinds of data generally, we suppose the Component-based Data Generation, aligned with formal specifications, would be a promising approach. This methodology involves constructing attribute repertoires, each mapping valid values pertinent to specific domains. By progressively assigning values to attributes in accordance with the second-level specifications, we can select corresponding components from these repertoires. Consequently, the generated data comprises components that collectively reflect all attribute values, adhering to the formal specifications to form a comprehensive dataset.

The generated data should undergo rigorous scrutiny against both levels of specifications. This step ensures that the data is not only pertinent or relevant but also of superior quality, fitting the intended DNN training. This rigorous selection and filtering process aligns the synthetic data with our training objectives.

3.3 Model training

The next step delves into the process of training the DNN using the dataset prepared in the earlier stages. This phase is crucial as it translates theoretical specifications and generated data into practical machine learning applications.

First, we should discuss the criteria for selecting an appropriate DNN architecture for the task. This choice is related to various factors, including the complexity of the network, the specific nature of the task, and the characteristics of the dataset. A well-suited architecture is instrumental in harnessing the full potential of the generated data [11].

Following this, we explore the training process in detail. Key parameters such as learning rate and batch size are carefully configured to optimize the

learning process. Additionally, strategies like cross-validation are employed to bolster the model's performance and reliability. In this phase, the generated data, along with their corresponding ground truth labels, form the training set that is instrumental in training the DNN. This approach ensures that the model is not only trained on diverse and representative data but also fine-tuned to achieve high accuracy and efficiency.

3.4 Evaluation

The evaluation step focuses on evaluating the effectiveness of the DNN model trained using our formally generated dataset. The evaluation process is critical to assess the model's performance and the viability of our data generation method.

Considering different DNN applications, the criteria for evaluating the model's performance may include accuracy, precision, recall, and F1 score. These metrics provide a comprehensive understanding of how well the model recognizes and interprets the data based on our specifications.

The test dataset, distinct from the training set, should comprise real-world data that the model has not previously encountered. This ensures that our evaluation genuinely reflects the model's capability to generalize and perform in practical scenarios.

Moreover, a comparative analysis against other models trained on conventional datasets is expected to be conducted. This comparison is intended to highlight the strengths and potential limitations of our data generation method.

3.5 Adversarial data generation

In addition to the four common procedures previously discussed, our framework introduces a novel approach to adversarial data generation. As we discussed above, adversarial data or adversarial examples, crafted by introducing subtle modifications to original input data, are designed to deceive DNNs. Their generation and use are crucial for developing DNNs that are accurate, secure, and

reliable in real-world scenarios [29].

The prevailing approach in adversarial data generation involves incrementally altering original input data to progressively approach and ultimately breach the model's decision boundary, resulting in a different prediction [30]. Our data generation methodology, as outlined in the data generation section, is adaptable for this purpose. Contrasting with standard training data generation, adversarial data generation necessitates slight modifications to the attributes defined in the formal specifications. For instance, adjusting the RGB values of a color attribute or altering the value of a length attribute. Subsequently, adversarial data is generated based on these revised specifications, challenging the DNN's decision-making capabilities.

Chapter 4

Two-level specification approach

As discussed in Section 3, the proposed framework is structured into five distinct steps. Among them, the foundation of the whole workflow is the preparation step, including the extraction of attributes and the two-level specifications. Hence, in this separate section, we will present the two-level specification approach by discussing the purpose of each level of specification and introducing a novel methodology for their formulation. Both levels of specification are integral to generating satisfactory data, which is essential for the effective training of DNNs.

4.1 Extraction of attributes

Before delving into the two-level specification method, we have to illustrate the first sub-step in the preparation step, the extraction of attributes. This process involves sufficient observations and meticulous analysis of the original data to identify features crucial for the DNN application. Besides, the identification of relevant attributes requires a deep understanding of the domain and the specific problem being addressed.

For example, for some object detection tasks, key attributes may include shape, color, volume, texture, symbols, and their spatial relationships. Illus-

trating this with a small example in traffic sign recognition, the traffic sign of Pedestrians in Figure 4.1, important attributes may include the main shape, background color, and the graphic of the pedestrian in the middle of the sign.

Figure 4.1: The traffic sign of Pedestrians

On the other hand, in Natural Language Processing (NLP), attribute extraction involves identifying text-related features like the length of the text, vocabulary usage frequency, sentiment polarity, and linguistic patterns, essential for tasks like sentiment analysis or text classification.

Moreover, for some kinds of mathematical tasks where direct observation is challenging, exploratory data analysis techniques become crucial for gaining insights. Techniques like data visualization through scatter plots, histograms, and box plots are instrumental in unveiling data relationships and distributions. Statistical measures, including mean, median, and standard deviation, are key to understanding the central tendencies and variations of numerical attributes. Correlation analysis is used to unearth linkages between various attributes, and methods to determine feature importance help in identifying attributes that significantly influence model predictions. Additionally, dimension reduction techniques, such as Principal Component Analysis (PCA) [31] are valuable for visualizing complex, high-dimensional data, assisting in discerning its structure and identifying potential patterns or clusters.

Collaborative brainstorming sessions with subject matter experts, data analysts, and domain specialists are pivotal in the attribute extraction process as well. These discussions, blending varied expertise and insights, play a crucial role in uncovering the intrinsic characteristics essential for manual attribute extraction. By engaging in these interdisciplinary exchanges, a more compre-

hensive and nuanced understanding of relevant attributes is achieved, guiding the extraction of the attribute process effectively.

In sum, the process of extracting attributes is an iterative one, demanding a comprehensive understanding of both the data and its contextual problem. It involves a blend of domain expertise, insightful analytical discussions, and the application of exploratory data analysis techniques. This careful and repeated refinement and validation of extracted attributes ensure their relevance and importance in the successive stages of the framework.

4.2 Two-level specifications

Following attribute extraction, we utilize formal specifications to delineate object characteristics. As we briefly discussed in Section 3.1, due to the distinct logic between traditional software and DNNs, formal specifications are crafted to indirectly influence DNN behavior by defining data characteristics.

During employing SOFL for writing these specifications, both the accurate description of original objects and the facilitation of automatic training data generation should be paid attention to. Hence, we provide the two-level specification method for this circumstance.

Since the specifications are crafted based on SOFL, we have to adhere to SOFL's syntax and rules, including focusing on key components such as Conditional Data Flow Diagrams (CDFDs), data flows, modules, types, and processes. This ensures that our formal specifications, free from the ambiguities of informal descriptions, accurately guide the generation of training data, with particular emphasis on utilizing SOFL to describe target data characteristics. Specifically, the following components of SOFL are significant in describing the characteristics of the target data in the formal specifications:

(1) **Module**: In complex systems, functionality is typically divided into modules in a top-down approach. This kind of division aids a systematic definition and a better contraction on different functions. However, instead of a complex system, our method requires just one specific module to describe

the data. In SOFL, a module, demarcated by the keywords "module" and "end-module," encapsulates data and procedures. It encompasses elements like constants ('const'), variables ('var'), types ('type'), behavior identifiers ('behav'), and processes ('process'). The structure of a module, inclusive of these keywords, is depicted in Figure 4.2.

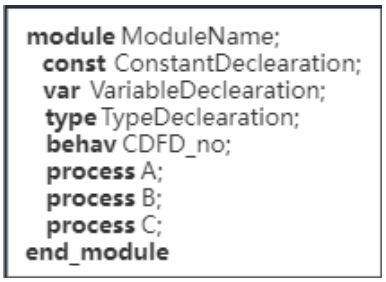

Figure 4.2: The structure of a module

The items involved in a module are explained as follows:

- The keyword **module** is the symbol of the start of the module, followed by the name of the module;
- The keyword **const** defines constants within the module;
- The keyword **var** c.specifies variable values in the module;
- The keyword **type** outlines data types used in the module;
- The keyword **behav** links to the CDFD that connect to the module;
- The keyword **process** details specific operations and functions involved in the module;
- The keyword **end_module** represents the end of the whole module.

(2) **Type**: SOFL supports a variety of data types, ranging from basic ones like integers, floats, and characters to more complex structures such as sets and sequences. Composed types, which are amalgamations of basic data types, can be tailored to match the form and complexity of the data involved in our supervised learning tasks.

(3) **Process**: This component describes specific functions and operations, depicting the transformation from input to output. For example, in a supervised learning task like object recognition, a process might evaluate if an input image correctly identifies the target object, yielding a 'yes' or 'no' outcome.

(4) **Pre-condition**: Pre-conditions in SOFL set necessary criteria for inputs prior to the execution of a process. They ensure that the input data meets certain conditions.

(5) **Post-condition**: Conversely, post-conditions define the expected state of data or results following the completion of a process. They can be treated to the outcomes or consequences of executing the process, guaranteeing that the process achieves its intended results.

To concisely summarize, we utilize some components of SOFL for precise and thorough formal specifications. This focus on particular components aligns with our method, enabling us to accurately depict the attributes of target data in supervised learning scenarios.

4.2.1 First level specification

The first-level specification is designed to accurately represent the original objects' characteristics, incorporating all extracted attributes while being comprehensible to human readers. It balances mathematical precision with accessible language. For example, in traffic sign recognition, simple descriptive terms like "Triangle", "Square", and "Circle" for shapes, and "White", "Red", and "Yellow" for colors are used. The emphasis at this stage is on the accuracy of these descriptions, reflecting the real attributes of the objects, instead of its direct transformation to visual images. For instance, in the traffic sign of Stop in Figure 4.3, it is expected to be described with a "Red" background attribute and "Right octagon" shape attribute.

Figure 4.3: The traffic sign of Stop

This level of specification lays the foundation for the framework's subsequent steps, including data generation and DNN model training. It provides clear and concise object characteristics descriptions, facilitating the creation of representative training data and effective DNN models.

To achieve better accuracy of the first level specification, collaboration with domain experts and stakeholders to validate and refine the selection of attributes and descriptions is essential. Their insights are critical in accurately capturing the key characteristics of the objects relevant to the task.

Moreover, documenting the rationale behind attribute selection is vital, offering clarity and building confidence in the framework, ensuring its applicability and reliability across diverse tasks and domains. This process ultimately ensures that the first-level specification effectively formalizes object characteristics, balancing mathematical detail with readability, and setting the foundation for subsequent stages in the framework.

In summary, the first level specification plays a pivotal role in formalizing the characteristics of the original objects. By combining mathematical rigor and intelligible natural language, it sets the foundation for generating accurate and relevant training data for DNNs. Through collaboration with domain experts and proper documentation, the first level specification ensures the effectiveness and reliability of the entire framework.

4.2.2 Second level specification

The second-level specification's role is to transform the human-readable object characteristics in the first level specification into specific, quantifiable attributes

for data synthesis. It focuses on detailed mathematical values and parameters, converting natural language descriptions into precise numerical values and constraints. This ensures that machines can interpret and implement these specifications effectively, facilitating efficient and accurate data generation that mirrors the desired object characteristics.

For example, in the traffic sign recognition task, the second level specification assigns precise dimensions to the "Triangle", "Square", and "Circle" shapes, specifying the exact ratios and angles to ensure faithful representation in the generated data. Additionally, the color descriptions "White", "Red", and "Yellow" are quantified in terms of RGB values or color codes, enabling the machine to produce realistic color variations for the traffic signs. Still, for the traffic sign of Stop, as shown in Figure 4.3, the color attribute should be described as "(190, 40, 40)" for RGB value. On the other hand, for a right polygon, we can use cos and sin functions to calculate the location of points as shown in Algorithm 1. Hence, we can use "sides = 8" to describe the shape of the right octagon for the second level specification.

Algorithm 1 Calculation of points in a right polygon

Require: Number of sides of the polygon (sides), center coordinates (center), radius of the polygon (radius)
Ensure: All points of the polygon
1: Initialize angle to 0
2: **for** $i = 1$ to sides **do**
3: $x \leftarrow \text{center}[0] + \text{radius} \times \cos(\text{angle})$
4: $y \leftarrow \text{center}[1] + \text{radius} \times \sin(\text{angle})$
5: Append point (x, y) to points
6: angle $\leftarrow$ angle $+ \frac{360}{\text{sides}}$
7: **end for**

By providing explicit mathematical values and parameters, the second level specification will facilitate automatic data generation on a lager scale. It streamlines the machine's understanding of the desired object characteristics, reducing ambiguity and potential errors during data synthesis.

Collaboration between domain experts and data scientists remains pivotal in refining the second level specification. Domain experts validate the relevance

and accuracy of the numerical values, ensuring that they align with real-world object properties. Data scientists verify the feasibility and implementation aspects of the specification, guaranteeing that the machine can seamlessly generate data based on the provided mathematical parameters.

In summary, the second level specification complements the first level by translating natural language descriptions into quantifiable and machine-interpretable values. This precision-driven approach empowers the machine to generate data with greater fidelity, contributing to the effectiveness and robustness of the entire framework for formal specification-based data generation for DNNs.

The adoption of a two-level specification approach arises from the need to strike a balance between human comprehension and machine processing efficiency. If we solely rely on a one-level specification, it may become overly complex for human readers to interpret, hindering their ability to understand the content. Simultaneously, directly writing the specification at a too fine level might introduce an overwhelming amount of details, increasing the risk of errors.

By employing a two-level specification strategy, we achieve a harmonious combination of simplicity and precision. The first level specification offers an easily understandable representation, enabling human readers to comprehend the object characteristics effortlessly. In contrast, the second level specification provides formal, detailed parameters for the machine to handle during data generation. This division ensures that the specification remains manageable, yet comprehensive enough to yield accurate and realistic data.

Overall, the two-level specification facilitates effective communication between human readers and the machine, fostering the successful application of the formal specification-based data generation framework in deep neural networks.

Chapter 5

Case study

We conduct a case study to validate our formal specification method, focusing on traffic sign recognition, a crucial aspect of autonomous driving systems [32]. Recent attention has been devoted to this field, with numerous researchers proposing methods for traffic sign detection and recognition [33–36]. However, the development of deep learning methods for this task is hindered by the scarcity of diverse and accurately labeled training data.

The dependence on large-scale, carefully labeled datasets for training accurate models is a significant challenge. Complicating matters further, different countries and regions employ distinct traffic signs, necessitating varied sets of well-labeled data for research [34]. Figure 5.1 illustrates examples of Pedestrian and Stop signs from different countries, highlighting the variations.

Acquiring a substantial number of traffic sign photos and accurately labeling them is a time-consuming and labor-intensive process, impeding researchers' ability to promptly access a comprehensive dataset. This issue underscores the importance of exploring alternative approaches to data generation, making our formal specification-based method a valuable solution for traffic sign recognition tasks.

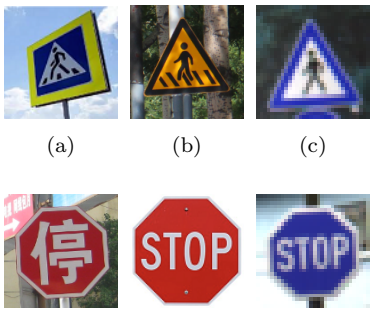

Figure 5.1: Pedestrians: (a), (b), (c); Stop: (d), (e), (f)

In this section, we use the details of the example of traffic sign recognition to demonstrate our methodology, including the introduction of the data set we used.

Data set - We utilize the Traffic Sign Recognition Database (TSRD)[1] sourced from the Chinese Traffic Sign Database, published by Beijing Jiaotong University, as our original data set. This data set comprises 6,164 traffic sign images, encompassing 58 sign categories such as Speed limits, Pedestrians, and General caution. The dataset encompasses images captured under diverse weather and lighting conditions, including instances of partial occlusion.

We have finished both levels of specifications of all the 58 categories of traffic signs. However, due to the limitation of pages, we only take two specific traffic signs, the Speed limit of 40 km/h and No left turn, as examples to demonstrate the case study.

5.1 Extraction of attributes

Different from some other general items or objects, traffic signs have some common characteristics. These signs often feature simple shapes, such as circles and triangles, alongside regular graphics and numbers. Therefore, we abstract several key attributes to describe them, including shape, background color, content, content color, border color, content number, font, arrow number, arrow turn

[1] http://www.nlpr.ia.ac.cn/pal/trafficdata/recognition.html

smooth sign, arrow start point, arrow turn points, arrow end points, graphic description, location, ratio, border ratio, and forbid type. In Figure 5.2, we present two examples, the traffic signs of the speed limit of 40 km/h and no left turn. We mainly use these two particular signs as the basis to elucidate the significance of all the aforementioned attributes.

(a) (b)

Figure 5.2: The speed limit for 40 km/h: (a); No left turn: (b)

- **Shape**: The shape attribute characterizes the primary geometric form of the traffic sign. Within our dataset of 58 distinct traffic sign classes as indicated in Figure 5.3, only four shapes are present: Circle, Triangle, Square, and Octagon. In our case study, the shape attribute representing the speed limit of 40 km/h is identified as a "Circle".

(a) (b) (c) (d)

Figure 5.3: Shape attribute: Circle: (a), Triangle: (b), Square: (c), and Octagon: (d)

- **Background color**: This attribute refers to the color of the main blank areas or the background space between the content and the border of the traffic sign. The data set contains four predominant background colors: White, Blue, Yellow, and Red. In the case study, the background color of the speed limit of 40 km/h is "White".

- **Content**: The content attribute refers to the core information displayed on the traffic sign, conveying its primary purpose. Typically, traffic signs feature numbers, graphics, or arrows as their core content. For our case

study, the content attribute of the speed limit of 40 km/h corresponds to "Number", while the attribute of no left turn corresponds to "Arrows".

- **Content color**: In contrast to the background color, this attribute specifies the color of the content parts within the traffic sign. Most traffic signs have black or white content color. In the case study, the content attribute of the speed limit of 40 km/h is denoted as "black".

- **Border color**: The border color represents the color of the exterior part of the traffic sign. Commonly, the border color is red, blue, or black. For some sign categories in the data set without a border, we consider their Border color attribute identical to "null". In the case study, the border color attribute of the speed limit of 40 km/h is "Red".

- **Content number**: This attribute exclusively applies to traffic signs where the content attribute is "Number." It directly describes the numeric value displayed on the sign, such as "40" in our case study of the speed limit of 40 km/h.

- **Font**: The font attribute, similar to the content number, is also relevant to specific traffic sign types. For those signs that have numbers or strings, the font attribute refers to the typeface used. To maintain consistency, only formal fonts like "Arial" and "Calibri" are considered. As illustrated in Figure 5.4, the font used in these two pictures are the "Arial" and the "Chaparral Pro Light". Apparently, the choice of font can significantly impact the authenticity of a traffic sign.

Figure 5.4: Example of different fonts

- **Arrow number**: As we mentioned in the content attribute, some traffic signs will contain arrows as their displayed core information. For those

types of signs, we have several arrow-related attributes, including arrow number, to discuss the details of arrows. The arrow number attribute represents the total number of arrows in the traffic sign. Based on our data set, the possible results for the arrow number attribute are 0, 1, 2, and 3. For instance, Figure 5.5 shows examples of traffic signs consisting of two or three arrows. For the case study, the arrow number attribute of the traffic sign of no left turn is 1.

Figure 5.5: Examples of traffic signs having 2 or 3 arrows

- **Arrow turn smooth sign**: Similar to the arrow number attribute, this arrow turn smooth sign attribute is only meaningful for the traffic signs whose content attribute is "Arrows". It is used to discuss whether the arrow turns smoothly or not. Figure 5.6 shows an example of the traffic sign with a smooth turn. In contrast, our case study of the traffic sign of no left turn has a "false" as its' arrow turn smooth sign attribute.

Figure 5.6: Example of traffic sign having a smooth-turn arrow

- **Arrow start point**: This attribute and the following two are expected to describe the structure of an arrow. The arrow start point is used to demonstrate the position of the starting point of an arrow. For instance, we can roughly divide the traffic sign into several parts to represent different locations to show the position of points. In our case study, the arrow start point attribute of the traffic sign of no left turn is "downright".

- **Arrow turn points**: The arrow turn points attribute is used to refer to the position of points where an arrow turns. Different from the starting point, an arrow might turn several times in a traffic sign, as the example shown in Figure 5.7. Hence, we use the plural for this attribute. The arrow turn points attribute for this case study of the traffic sign of no left turn is "upright".

Figure 5.7: Examples of a traffic sign having multiple turns

- **Arrow end points**: This attribute is used to represent all the positions of the ending points of an arrow. Similar to the turning points, for some divergent arrows, multiple ending points exist. For example, Figure 5.8 shows a divergent arrow of the traffic sign of turn left or right. However, in our case study, the traffic sign of no left turn has only one ending point. The corresponding result of the attribute is "upleft".

Figure 5.8: Examples of a traffic sign having divergent arrow

- **Graphic description**: The graphic description attribute is used only for some specific graphics that are difficult to describe in formal language. For example, the traffic sign shown in Figure 5.9 has a car inside, which can not be directly described. However, for the both two traffic signs of our case study, the graphic description attribute should be "null".

Figure 5.9: Traffic sign of Motor vehicles only

- **Location**: The location attribute illustrates the position of the main content within the sign. While most traffic signs place the main content at the center, there are still some exceptions. In the case study, the location attribute of the speed limit of 40 km/h is labeled as "center".

- **Ratio**: The ratio attribute quantifies the size of the main content relative to the overall size or diameter of the traffic sign. For the case study, the ratio attribute of the speed limit of 40 km/h is calculated as "0.55".

- **Border ratio**: Similar to the ratio attribute, the border ratio describes the relative size of the border. However, it is often disregarded or omitted. In Figure 5.10, the importance of the border ratio is depicted. For circular signs, the Border ratio is derived from the inner and outer diameters of concentric circles. In the case study, the border ratio attribute of the traffic sign of the speed limit of 40 km/h is "0.22".

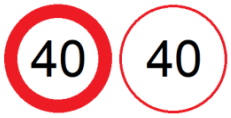

Figure 5.10: Example of different border ratios

- **Forbid type**: In our data set, some traffic signs are used to forbid specific behaviors during driving. For instance, the traffic sign of no left turn forbids the behavior of turning left at the place. Generally, there are different kinds of representing methods of forbidden type in our data set, including red oblique line, red cross, red vertical line, and black oblique lines, as illustrated in Figure 5.11. In our case study, the forbid type

attribute should be "not forbidden" for the traffic sign with the speed limit of 40 km/h, and "red oblique line" for the traffic sign of no left turn.

Figure 5.11: Example of traffic signs with different forbid type

In sum, all attributes and corresponding values of the case study are summarized in Table 5.1.

Table 5.1: The attributes of the case study

Attribute	speed limit of 40 km/h	no left turn
Shape	Circle	Circle
Background color	White	White
Content	Number	Arrows
Content color	Black	Black
Border color	Red	Red
Content number	40	Null
Font	Arial, Calibri	Null
Arrow number	0	1
Arrow turn smooth sign	False	False
Arrow start point	Null	Downright
Arrow turn points	Null	Upright
Arrow end points	Null	Upleft
Graphic description	Null	Null
Location	Center	Center
Ratio	0.55	0.55
Border ratio	0.22	0.2
Forbid type	Not forbidden	Red oblique line

Based on this case study, we demonstrate the process of selecting attributes

for a specific data set. However, considering the variety of DNN tasks we are facing, it is difficult to provide a standard to define the number of required attributes to describe the targeted data. For different kinds of tasks, the attribute selection would be different as well.

5.2 First level specification

Once the attributes have been extracted from all 58 different traffic sign classes, the subsequent step involves writing the first level formal specification strictly in SOFL principles. It is essential to acknowledge that real pictures of traffic signs may differ from ideal representations due to various factors such as weather conditions, sunlight, angles, and camera devices. Figure 5.12 illustrates examples of such changes, where the color changes to a darker version in the first picture and the shape transforms from a circle to an ellipse in the second picture. Therefore, when formulating the specification, it is crucial to accommodate this variety. For instance, the ratio of the content and the border should be treated as ranges of values instead of fixed ones.

Figure 5.12: Changes in color and shape of the real pictures

In our case study, for a specific traffic sign, only one function is required. Thus, there is no need to address a complex system comprising multiple modules. Instead, a single module, named "Traffic_sign_recognition," is created. All the essential non-basic types required in the module are declared in the **type** section. Specifically, for the traffic sign recognition task, we require self-defined types based on the extracted attributes. Among these data types, the "TraffiaSign" serves as a composite type, encompassing fields that describe all attributes belonging to a traffic sign. As we mentioned in the previous subsection, the maximum number of arrows in the data set is 3. Hence, the at-

tributes related to the points of an arrow repeat three times in the composite type declaration.

The **process** keyword denotes the primary operation responsible for determining if a set of input data satisfies the characteristics of the traffic signs in our case study. Unlike image-based recognition, the input to this process consists of data representing all attributes of a traffic sign. The objective is not to automatically recognize whether the input image corresponds to a specific traffic sign type, but rather to manually define the standards for traffic sign characteristics. In essence, when a set of input data fulfills the requirements of this process, it serves to describe the characteristics of the traffic sign of the speed limit of 40 km/h or no left turn. The **pre-condition** and **post-condition** establish the conditions that must be satisfied before and after the process, respectively. For the first level specification, no specific pre-condition is required. Hence, the pre-condition can be "true", representing the process will be directly run without any constraints.

Besides, as we mentioned previously, in the first level specification, the main purpose is ensuring the correctness of the specification to represent the characteristics of the objects, rather than its direct transformation to visual images. Therefore, we use "Circle" to describe the shape of the sign and "Red" or "White" to describe the involved color.

As a result of the considerations outlined above, the first level formal specification is constructed as following sub-sections. Since the whole specification for the case study is too long and complex, we omit some parts of the specification to illustrate the important information of the process.

5.2.1 The traffic sign of the speed limit of 40 km/h

```
module Traffic_sign_recognition
  type
    shape = set of {<Circle>, <Right triangle>, <Right
        octagon>, <Square>, <Inverted triangle>,<Right
        decagon>, <Right hendecagon>, <Right dodecagon
```

```
   >, <Right triskaidecagon>,<Right tetradecagon>,
    <Right pentadecagon>, <Right hexadecagon>, <
   Right heptadecagon>, <Ellipse>};
color = seq of nat0/*Three nums for R, G, and B*/
colortext = set of {<red>, <blue>, <white>, <black
   >, <yellow>, <green>}
content = set of {<Number>, <Character>, <Chinese
   character>, <Graphics>, <Exclamation mark>, <
   Arrows>}
location = set of {<center>, <left>, <right>, <up
   >, <down>}
points = set of {<center>, <left>, <right>, <up>,
   <down>, <upleft>, <upright>, <downleft>, <
   downright>}
font = set of {<Chaparral Pro Light>, <Arial>, <
   Calibri>, <Franklin Gothic Book>, <Microsoft
   JhengHei>, <Times New Roman>}
TrafficSign = composed of
            sign_name: string
            sign_id: nat0
            sign_shape: shape
            background_color: colortext
            content_color: colortext
            border color: colortext
            sign_content: content
            content_number: int
            content_font: font
            arrow_number: int
            arrow_turn_smooth: boolean /*used for
               u-turn or roundabout*/
            arrow1_start_point: points
            arrow1_turn_points: points
            arrow1_end_points: points
            arrow2_start_point: points
            arrow2_turn_points: points
            arrow2_end_points: points
            arrow3_start_point: points
            arrow3_turn_points: points
            arrow3_end_points: points
            graphic_des: string /*used for some
               specific graphics*/
            content_location: location
            ratio: real /*used to describe the
               size of the content*/
            border_ratio: real /*used to indicate
               the width of a border*/
```

```
                forbid: int /*0 for not forbidden, 1
                   oblique line, 2 cross, 3 vertical
                   line, 4 black oblique line */
    end
  process Recognition_speed_limit_40 (this_shape:
     shape, this_backc: colortext, this_contc:
     colortext, this_bordc: colortext, this_cont:
     content, this_number: int, this_location:location
     , this_font: font, this_ratio: real, this_bordr:
     real, this_forbid: int, this_arrown: int,
     this_arrow_ts: boolean, this_arrow1_sp: points,
     this_arrow1_tp: points, this_arrow1_ep: points,
     ..., this_graphicd: string) correct_sign:boolean
  pre true
  post this_shape inset {<Circle>, <Right tetradecagon
     >, <Right pentadecagon>, <Right hexadecagon>,
     ...} and this_backc = {<white>} and this_contc =
     {<black>} and this_bordc = {<red>} and this_cont
     = {<Number>} and this_number = 40 and
     this_location = {<center>} and 0.45 <= this_ratio
      <= 0.65 and  0.18 <= this_bordr <= 0.26 and
     this_font inset {<Arial>, ....} and this_forbid =
      0 and this_arrown = 0 and this_arrow_ts = false
     and this_arrow1_sp = null and this_arrow1_tp=
     null and this_arrow1_ep = null and ...
     this_graphicd = null and let sign_name = "
     Speed_limit_40" and correct_sign = true
    or ...
  end_process
end_module
```

5.2.2 The traffic sign of no left turn

As we mentioned before, the traffic sign of no left turn has an arrow inside the sign, which is not included in the sign of the speed limit of 40 km/h. Hence, we will omit other parts and only maintain the arrow-related information in both levels of specifications of this sign.

```
module Traffic_sign_recognition
  type
    ...
  process Recognition_no_left_turn (...) correct_sign:
     boolean
```

```
  pre true
  post ... and this_forbid = 1 and this_arrown = 1 and
       this_arrow_ts = false and this_arrow1_sp = {<
      downright>} and this_arrow1_tp = {<upright>} and
      this_arrow1_ep = {<upleft>} and this_arrow2_sp =
      null and ... and this_graphicd = null and let
      sign_name = "No_left_turn" and correct_sign =
      true
  or ...
  end_process
end_module
```

5.3 Second level specification

For the second level specification, we should provide detailed mathematical values and parameters. Hence, the natural language description of colors and shapes should be changed to a more formal style. Specifically, the color should be demonstrated as RGB values, while the shape should demonstrated as the number of sides of a right polygon. Similar to the first level one, when formulating the second level specification, it is also important to accommodate this variety. For instance, the RGB value of a specific color or the ratio may need to be defined within a range. Besides, with the increasing of sides, a right polygon will look more like a circle. Therefore, in our second level specification, we use $sides = 0$ or $sides > 14$ to represent a circle shape.

In addition to the shape and color, we also have other different descriptions between the two level specifications. For example, in the first level specification, we use location words like "left" or "right" to indicate the position of points of an arrow. But in the second level one, we involve polar coordinates to precisely manifest the location. Polar coordinates provide a two-dimensional representation of points in a plane by using two parameters: radial distance (r) and angular displacement Θ. Unlike Cartesian coordinates that rely on the X and Y axes, polar coordinates express the distance from the point to the origin as the radial distance (r), and the angle between the reference direction and a line connecting the point and the origin as angular displacement (Θ). The following

equations are used to convert between Cartesian and polar coordinates:

$$x = r \cdot \cos(\Theta)$$

$$y = r \cdot \sin(\Theta)$$

where $(x, y)$are the Cartesian coordinates, r is the radial distance, and Θ is the angle in radians.

Due to the limitation of pages, we omit some parts of the specification as well. The following part demonstrates the second level specification of the case study.

5.3.1 The traffic sign of the speed limit of 40 km/h

```
module Traffic_sign_generation
  type
    ...
    SignOutput = composed of
                sign_name: string
                sign_id: nat0
                sign_sides: int
                background_color: color
                content_color: color
                border_color: color
                text: string
                content_font: int /*1 for formal fonts
                   , 2 for informal fonts*/
                arrow_number: int
                arrow_turn_smooth: boolean
                arrow1_start_point: points_polar
                arrow1_turn_points: points_polar
                arrow1_end_points: points_polar
                ...
                graphic_des: string
                content_location: location
                size: real /*used to describe the size
                    of the content*/
                border_size: real /*used to indicate
                   the width of a border*/
                forbid: int
    end
    var radius = 250
```

```
  process Generation_speed_limit_40 (correct_sign:
     boolean, traffic_sign_name: string, this_shape:
     shape, backcolor: colortext, contcolor: colortext
     , bordcolor: colortext,this_backc: color,
     this_contc: color, this_bordc: color, this_cont:
     content,this_number: int, this_location:location,
      this_ratio: real, contfont: font, this_bordr:
     real, this_forbid: int, this_arrown: int,
     this_arrow_ts: boolean, this_arrow1_sp: points,
     ...,this_graphicd: string) this_sign: SignOutput
  pre correct_sign = true and traffic_sign_name = "
     Speed_limit_40"
  post this_shape inset {<Circle>, ...} and let sides
     = 0 or sides >= 14 and backcolor = {<white>} and
     190 <= this_backc(1) <= 255 and 190 <= this_backc
     (2) <= 255 and 190 <= this_backc(3) <= 255 and
     contcolor = {<black>} and 0 <= this_contc(1) <=
     70 and 0 <= this_contc(2) <= 70 and 0 <=
     this_contc(3) <= 70 and bordcolor = {<red>} and
     130 <= this_bordc(1) <= 255 and 0 <= this_bordc
     (2) <= 80 and 0 <= this_bordc(3) <= 80 and
     this_cont = {<Number>} and this_number = 40 and
     let text = "40" and this_location = {<center>}
     and 0.45 <= this_ratio <= 0.65 and let int(0.45 *
      2 * radius) <= size <= int(0.65 * 2 * radius)
     and 0.18 <= this_bordr <= 0.26 and let int(0.18 *
       radius) <= border_size <= int(0.26 * 2 * radius
     ) and contfont inset {<Arial>, ...} and this_font
      = 1 and this_sign = mk_TrafficSign(
     traffic_sign_name, 3, sides,this_backc,
     this_contc, this_bordc, text, this_font,
     this_arrown, this_arrow_ts, this_arrow1_sp,
     this_arrow1_tp, ..., this_graphicd, this_location
     , size, border_size, this_forbid)
  end_process
end_module
```

5.3.2 The traffic sign of no left turn

```
module Traffic_sign_generation
  type
    ...
  process Generation_no_left_turn (...) this_sign:
     SignOutput
```

```
  pre correct_sign = true and traffic_sign_name = "
     No_left_turn"
  post ... and this_arrown = 1 and this_arrow_ts =
     false and this_arrow1_sp= {<downright>} and
     arrow1_sp = {[int(0.5 * size), 315]} and
     this_arrow1_tp = {<upright>} and arrow1_tp = {[
     int(0.5 * size), 45]} and this_arrow1_ep = {<
     upleft>} and arrow1_ep = {[int(0.5 * size), 135]}
      and this_sign = mk_TrafficSign(traffic_sign_name
     , 11, sides, this_backc, this_contc, this_bordc,
     null, null, this_arrown, this_arrow_ts,arrow1_sp,
      arrow1_tp, arrow1_ep, null, null, null, null,
     null, null, null, this_location, size,
     border_size, this_forbid)
  end_process
end_module
```

Chapter 6

Discussion

This research primarily concentrates on the method of drafting two level specifications. Hence, it is crucial to evaluate the performance of the method. In this section, we aim to explore two key aspects: 1). the ability of the drafted specifications to precisely describe the characteristics of objects, and 2). the effectiveness of the two-level specification approach in facilitating the automation of data generation. We still use the case study in Section 5 to demonstrate the performance.

6.1 Precision of description

In order to evaluate whether our SOFL specification can clearly define the characteristics of the traffic sign of the speed limit of 40 km/h, we conducted a survey. This survey comprises 20 pictures that were manually generated based on selected sets of data according to the formal specification.

We prepared 20 different sets of data, 10 of them satisfy the definition of the specification while the other 10 sets do not. As we discussed previously, considering the influence of weather, sunlight, angle, and camera devices, the actual picture of a traffic sign may be different from the ideal picture. That is why when we generate the SOFL formal specification, we use ranges or sets to the attributes to consider more variety. The following four cases can serve as typical examples of all the 20 sets of selected data.

a. Case a:

```
side = 0 and this_backc(1) = 255 and this_backc(2) =
   255 and this_backc(3) = 255 and this_contc(1) = 0
   and this_contc(2) = 0 and this_contc(3) = 0 and
   this_bordc(1) = 237 and this_bordc(2) = 28 and
   this_bordc(3) = 36 and this_cont = {<Number>} and
   this_number = 40 and this_location = {<Center>} and
    this_ratio = 0.5 and this_bordr = 0.23 and
   this_font = {<Arial>} and this_forbid = 0 and
   this_arrown = 0 and this_arrow_ts = false and
   this_arrow1_sp = null and this_arrow1_tp= null and
   this_arrow1_ep = null and this_arrow2_sp = null and
    this_arrow2_tp = null and this_arrow2_ep = null
   and this_arrow3_sp = null and this_arrow3_tp = null
    and this_arrow3_ep = null and this_graphicd = null
```

b. Case b:

```
side = 0 and this_backc(1) = 190 and this_backc(2) =
   190 and this_backc(3) = 190 and this_contc(1) = 30
   and this_contc(2) = 30 and this_contc(3) = 30 and
   this_bordc(1) = 150 and this_bordc(2) = 40 and
   this_bordc(3) = 40 and this_cont = {<Number>} and
   this_number = 40 and this_location = {<Center>} and
    this_ratio = 0.5 and this_bordr = 0.23 and
   this_font = {<Arial>} and this_forbid = 0 and
   this_arrown = 0 and this_arrow_ts = false and
   this_arrow1_sp = null and this_arrow1_tp= null and
   this_arrow1_ep = null and this_arrow2_sp = null and
    this_arrow2_tp = null and this_arrow2_ep = null
   and this_arrow3_sp = null and this_arrow3_tp = null
    and this_arrow3_ep = null and this_graphicd = null
```

c. Case c:

```
side = 0 and this_backc(1) = 255 and this_backc(2) =
   255 and this_backc(3) = 255 and this_contc(1) = 0
   and this_contc(2) = 0 and this_contc(3) = 0 and
   this_bordc(1) = 237 and this_bordc(2) = 28 and
   this_bordc(3) = 36 and this_cont = {<Number>} and
   this_number = 40 and this_location = {<Center>} and
    this_ratio = 0.5 and this_bordr = 0.05 and
   this_font = {<Chaparral Pro Light>} and this_forbid
    = 0 and this_arrown = 0 and this_arrow_ts = false
   and this_arrow1_sp = null and this_arrow1_tp= null
   and this_arrow1_ep = null and this_arrow2_sp = null
    and this_arrow2_tp = null and this_arrow2_ep =
   null and this_arrow3_sp = null and this_arrow3_tp =
    null and this_arrow3_ep = null and this_graphicd =
    null
```

d. Case d:

```
side = 4 and  this_backc(1) = 255 and this_backc(2) =
    255 and this_backc(3) = 255 and this_contc(1) = 0
    and this_contc(2) = 0 and this_contc(3) = 0 and
    this_bordc(1) = 63 and this_bordc(2) = 72 and
    this_bordc(3) = 204 and this_cont = {<Number>} and
    this_number = 40 and this_location = {<Center>} and
     this_ratio = 0.25 and this_bordr = 0.25 and
    this_font = {<Calibri>} and this_forbid = 0 and
    this_arrown = 0 and this_arrow_ts = false and
    this_arrow1_sp = null and this_arrow1_tp= null and
    this_arrow1_ep = null and this_arrow2_sp = null and
     this_arrow2_tp = null and this_arrow2_ep = null
    and this_arrow3_sp = null and this_arrow3_tp = null
     and this_arrow3_ep = null and this_graphicd = null
```

Among these four sets of data, cases a and b satisfy the definition of the specification while cases c and d do not. Specifically, case a is the ideal situation of the case study; case b tries to simulate the traffic sign photographed in a low light environment; case c mainly violates the requirements of the font and the border ratio based on the formal specification; case d mainly violates the requirements of the number, the shape, and the color attributes. Figure 6.1 shows the visualized pictures that are manually generated based on each case.

Figure 6.1: Visualization of cases a, b, c, and d

After the selection of all 20 sets of data and the generation of all pictures based on them, shown in Figure 6.2, we conducted the survey among 13 subjects. Limited by capital, time, and human resources, we only organize the survey in our lab. However, we considered the diversity of our subjects to guarantee the impartiality of our survey. Among all the members in our lab, we have males and females; Chinese, Japanese, and Bengali; Ph.D. students, master's students, and research students.

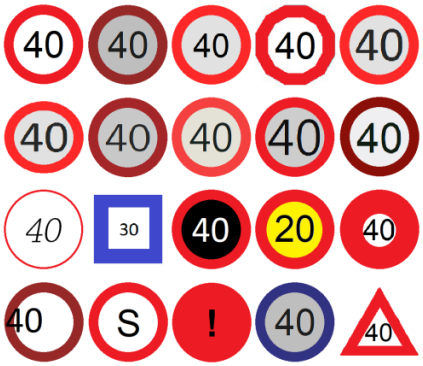

Figure 6.2: Generated pictures based on the selected sets

According to the result of the survey, the pictures of the cases that satisfy the requirements of the formal specification are treated as pictures of the traffic signs of the speed limit of 40 km/h. In the meantime, the pictures of cases that do not satisfy the formal specification are not likely to be recognized as specific traffic signs. This result indicates that the formal specification can be used to describe the characteristics of the traffic signs that are likely to help generate effective training data for DNNs.

6.2 Facilitation of automatic data generation

The primary objective of the two-level specification method is to reduce the gap between formal specification and data generation, enabling more efficient and accurate training of deep neural networks (DNNs), as it facilitates the clear definition of object characteristics while providing precise values and parameters for data generation.

To validate the effectiveness of our approach, we present an example demonstrating how our two-level specification method enables the generation of labeled training data for traffic sign recognition. In this example, we carefully select a specific set of data that satisfies the requirements of the first-level specification for a traffic sign depicting a speed limit of 40 km/h. The selected set of data is shown as follows:

```
this_shape = {<Circle>} and  this_backc = {<White>}
   and this_contc = {<Black>} and this_bordc = {<Red
   >} and this_cont = {<Number>} and this_number = 40
    andthis_location = {<Center>} and this_ratio=
   0.55 and this_bordr = 0.22 and this_font= {<Arial
   >} and this_forbid = 0 and this_arrown = 0 and
   this_arrow_ts = false and this_arrow1_sp = null
   and this_arrow1_tp = null and this_arrow1_ep= null
    and this_arrow2_sp = null and this_arrow2_tp =
   null and this_arrow2_ep = null and this_arrow3_sp
   = null and this_arrow3_tp = null and
   this_arrow3_ep = null and this_graphicd = null
```

These data servers as the input data of the first level specification. As we discussed, these data can be mapped to each one of the extracted attributes. Table 6.1 manifests the relationship between the selected data and the corresponding attributes.

Next, we leverage the second level specification to transform these selected data into precise values, ensuring that each detail is accurately represented. This meticulous process guarantees that the generated training data aligns with the formal specifications and meets the desired criteria for robust DNN training. Specifically, the RGB values for white, black, and red in our example are (245, 245, 245), (20, 20, 20), and (237, 28, 36), respectively. Based on the second level specification, the transformed parameters are illustrated as follows:

```
this_sign = mk_TrafficSign(traffic_sign_name, 3,
   0, (245, 245, 245), (20, 20, 20), (237, 28,
   36), 40, 1, 0, false, null, null, null, null,
   null, null, null, null, null, {<Center
   >}, int(0.55 * 2 * radius),  int(0.22 * 2 *
   radius), 0)
```

To implement the data generation process, we employ Python programming language along with the PIL (Python Imaging Library) package. This combination allows us to efficiently create high-quality images of the traffic signs based on the detailed attribute values derived from the second-level specification. The generated result is shown in Figure 6.3.

Table 6.1: Input data of first level specification and corresponding attributes

Attribute	Input data	Corresponding values
Shape	this_shape	{Circle}
Background color	this_backc	{White}
Content	this_cont	{Number}
Content color	this_contc	{Black}
Border color	this_bordc	{Red}
Content number	this_number	40
Font	this_font	{Arial}
Arrow number	this_arrown	0
Arrow turn smooth sign	this_arrow_ts	False
Arrow start point	this_arrow1_sp, ...	Null
Arrow turn points	this_arrow1_tp, ...	Null
Arrow end points	this_arrow1_ep, ...	Null
Graphic description	this_graphicd	Null
Location	this_location	Center
Ratio	this_ratio	0.55
Border ratio	this_bordr	0.22
Forbid type	this_forbid	0

Figure 6.3: Example of generated result

By comparing the resulting images with the original traffic sign requirements, we assess the performance of our two-level specification approach. This example serves as a practical demonstration of our method's potential to bridge the gap between formal specifications and data generation, paving the way for more

effective and precise DNNs training in various domain-specific applications.

However, our method still exhibits certain limitations. Particularly when confronted with intricate graphics akin to the one illustrated in Figure 5.9, we apply the *graphic_description* attribute as a means to encapsulate the complex image. It is because of the granularity of current specifications, which struggles to comprehensively depict such intricate graphics. On the other hand, striking a balance between granularity and precision of the specification is critical. Introducing more intricate attributes tailored to such complex graphics could lead to unwieldy composed type definitions, rendering the specification more intricate and less intuitive. Moreover, these specific attributes might not hold relevance for the majority of other traffic signs within the data set, potentially inundating the specification with needless verbosity.

To solve this problem, we must find a delicate equilibrium or a balance between the granularity of specification, which outlines the attributes, and the degree of detail required for an accurate description. A refined balance will not only optimize the precision of the specification but also streamline its usability and applicability across various traffic sign instances. This refinement process holds the key to addressing the challenge of accommodating complex visuals within the confines of our specification framework.

Chapter 7

Related work

While research on Deep Neural Networks (DNNs) has garnered significant attention, only a scant amount of literature has delved into the potential of formal specifications and methods to enhance the training and testing of neural networks. Sanjit A. Seshia and Ankush Desai et al. [27] approached the DNN scenario as a constituent within an extensive application-specific system. Recognizing the intricacies of formulating formal specifications for DNN tasks, their focus lay on an approach that comprehensively encapsulated the entire system's specifications. Similarly, Sanjit A. Seshia and Dorsa Sadigh et al. [28] scrutinized the challenges of AI verification within the purview of formal methods and their underlying principles. Sumathi Gokulanathan et al. [37] championed formal verification to streamline neural networks, reduce the size of DNNs without compromising precision. In addition, Arvid Jakobsson et al. [38] conducted a comprehensive survey spanning pivotal publications on formal methods and software engineering pertaining to DNNs. Russel, Dewey, and Tegmark [39], in their white paper, reiterated the criticality of formal verification and security in all AI systems. Their proposition underscores the importance of AI systems, including deep learning systems, being amenable to behavior, design, and specification verification.

In our research, the objective of the two-level specification method is to apply the specifications for both data generation and human understanding. On the

other hand, some researchers also focus on the research to analyze and understand specifications. For instance, Li and Liu et al. [40, 41] constructed knowledge graphs to SOFL specification and proposed a method for requirements-related fault prevention.

Turning to the domain of traffic sign recognition, a myriad of studies exist. For instance, F. Zaklouta and B. Stanciulescu et al. [33] evaluated k-d trees, random forests, and support vector machines (SVMs) for traffic-sign classification performance. J. Stallkamp et al. [42] benchmarked traffic sign recognition algorithms by juxtaposing human and state-of-the-art machine learning algorithm performances. However, a crucial gap persists in these studies: the absence of a discourse around enhancing DNN efficiency via formal methods.

Chapter 8

Conclusion & Future work

In this dissertation, we propose a framework for generating training data for supervised learning based on SOFL formal specifications, trying to provide a solution to the difficulty of obtaining well-labeled training data. In particular, we propose a two-level specification method to describe the characteristics of objects and facilitate automatic data generation. Specifically, we discuss the purpose of both the two levels of specifications and the method of writing them, respectively. In addition, we provide a case study of the traffic sign recognition task to demonstrate the process of extracting attributes and writing both the two level specifications. Around the case study, we discuss the approaches of manually extracting the attributes of the objects to be identified and defining the characteristics by SOFL specifications. Besides, we apply Python and the PIL package for a data generation example to manifest the feasibility of data generation based on the two-level specifications. This example illustrates that our method facilitates the clear definition of object characteristics while providing precise values and parameters for data generation.

Our research offers several future directions for progression. Firstly, having successfully generated formal specifications characterizing object attributes, our subsequent step involves the automated production of training data for DNN models. Hence, we anticipate formulating an effective approach for data generation based on our specifications. In the meantime, even though our efforts

to encompass variability through the inclusion of ranges and sets have yielded promising results, the results still seem too ideal. Our data generation process will delve into additional considerations, such as image skewing and blurring, to better reflect real-world scenarios.

Secondly, our current case study exclusively spotlights traffic signs, which consist of relatively simple images with a few components. Therefore, our focus will evolve toward exploring the feasibility of extending our method to intricate data types. Besides, as we discussed the challenge of accommodating complex graphics within current specification granularity, we are determined to strike a refined balance between specification granularity and precise characterization.

Moreover, we expect to conduct an extensive experiment to comprehensively validate the performance of our method. Limited by the progress of our research so far, such work cannot be done. Bur future strides related to automatic data generation will pave the way for a systematic evaluation. Our plan involves comparing our framework against existing benchmarks, thus ensuring a robust and comparative assessment.

Bibliography

[1] Singh, A., Thakur, N. & Sharma, A. A review of supervised machine learning algorithms. *2016 3rd International Conference On Computing For Sustainable Global Development (INDIACom)*. pp. 1310-1315 (2016)

[2] Hirschberg, J. & Manning, C. Advances in natural language processing. *Science*. **349**, 261-266 (2015)

[3] Jiao, L. & Zhao, J. A survey on the new generation of deep learning in image processing. *IEEE Access*. **7** pp. 172231-172263 (2019)

[4] Liang, M. & Hu, X. Recurrent convolutional neural network for object recognition. *Proceedings Of The IEEE Conference On Computer Vision And Pattern Recognition*. pp. 3367-3375 (2015)

[5] Woodcock, J., Larsen, P., Bicarregui, J. & Fitzgerald, J. Formal methods: Practice and experience. *ACM Computing Surveys (CSUR)*. **41**, 1-36 (2009)

[6] Lamsweerde, A. Formal specification: a roadmap. *Proceedings Of The Conference On The Future Of Software Engineering*. pp. 147-159 (2000)

[7] Riaz, S., Afzaal, H., Imran, M., Zafar, N. & Aksoy, M. Formalizing mobile ad hoc and sensor networks Using VDM-SL. *Procedia Computer Science*. **63** pp. 148-153 (2015)

[8] Liu, S., Offutt, A., Ho-Stuart, C., Sun, Y. & Ohba, M. SOFL: A formal engineering methodology for industrial applications. *IEEE Transactions On Software Engineering*. **24**, 24-45 (1998)

[9] Xia, Y. & Liu, S. A Framework of Formal Specification-Based Data Generation for Deep Neural Networks. *Proceedings Of The 2023 12th International Conference On Software And Computer Applications*. pp. 273-282 (2023), https://doi.org/10.1145/3587828.3587869

[10] Samek, W., Montavon, G., Lapuschkin, S., Anders, C. & Müller, K. Explaining deep neural networks and beyond: A review of methods and applications. *Proceedings Of The IEEE*. **109**, 247-278 (2021)

[11] Alzubaidi, L., Zhang, J., Humaidi, A., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamarıa, J., Fadhel, M., Al-Amidie, M. & Farhan, L. Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions. *Journal Of Big Data*. **8** pp. 1-74 (2021)

[12] Hinton, G. & Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *Science*. **313**, 504-507 (2006)

[13] Usama, M., Qadir, J., Raza, A., Arif, H., Yau, K., Elkhatib, Y., Hussain, A. & Al-Fuqaha, A. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access*. **7** pp. 65579-65615 (2019)

[14] Zhu, X. Semi-supervised learning literature survey. (University of Wisconsin-Madison Department of Computer Sciences,2005)

[15] Jain, A., Patel, H., Nagalapatti, L., Gupta, N., Mehta, S., Guttula, S., Mujumdar, S., Afzal, S., Sharma Mittal, R. & Munigala, V. Overview and importance of data quality for machine learning tasks. *Proceedings Of The 26th ACM SIGKDD International Conference On Knowledge Discovery & Data Mining*. pp. 3561-3562 (2020)

[16] Strom, N. Scalable distributed DNN training using commodity GPU cloud computing. *Sixteenth Annual Conference Of The International Speech Communication Association*. (2015)

[17] Ruder, S. An overview of gradient descent optimization algorithms. *ArXiv Preprint ArXiv:1609.04747*. (2016)

[18] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature*. **323**, 533-536 (1986)

[19] Agarap, A. Deep learning using rectified linear units (relu). *ArXiv Preprint ArXiv:1803.08375*. (2018)

[20] Han, J. & Moraga, C. The influence of the sigmoid function parameters on the speed of backpropagation learning. *International Workshop On Artificial Neural Networks*. pp. 195-201 (1995)

[21] Bottou, L. Large-scale machine learning with stochastic gradient descent. *Proceedings Of COMPSTAT'2010: 19th International Conference On Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited And Contributed Papers*. pp. 177-186 (2010)

[22] Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*. (2014)

[23] Jones, C. Systematic software development using VDM. *Prentice Hall International Series In Computer Science*. (1990)

[24] Spivey, J. Understanding Z: a specification language and its formal semantics. (Cambridge University Press,1988)

[25] Abrial, J., Butler, M., Hallerstede, S., Hoang, T., Mehta, F. & Voisin, L. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal On Software Tools For Technology Transfer*. **12**, 447-466 (2010)

[26] Berry, G. SCADE: Synchronous design and validation of embedded control software. *Next Generation Design And Verification Methodologies For Distributed Embedded Control Systems*. pp. 19-33 (2007)

[27] Seshia, S., Desai, A., Dreossi, T., Fremont, D., Ghosh, S., Kim, E., Shivakumar, S., Vazquez-Chanlatte, M. & Yue, X. Formal specification for deep neural networks. *International Symposium On Automated Technology For Verification And Analysis*. pp. 20-34 (2018)

[28] Seshia, S., Sadigh, D. & Sastry, S. Toward verified artificial intelligence. *Communications Of The ACM*. **65**, 46-55 (2022)

[29] Aldahdooh, A., Hamidouche, W., Fezza, S. & Déforges, O. Adversarial example detection for DNN models: A review and experimental comparison. *Artificial Intelligence Review*. **55**, 4403-4462 (2022)

[30] Yuan, X., He, P., Zhu, Q. & Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE Transactions On Neural Networks And Learning Systems*. **30**, 2805-2824 (2019)

[31] Jolliffe, I. & Cadima, J. Principal component analysis: a review and recent developments. *Philosophical Transactions Of The Royal Society A: Mathematical, Physical And Engineering Sciences*. **374**, 20150202 (2016)

[32] Møgelmose, A. Visual analysis in traffic & re-identification. (Aalborg Universitetsforlag,2015)

[33] Zaklouta, F. & Stanciulescu, B. Real-time traffic-sign recognition using tree classifiers. *IEEE Transactions On Intelligent Transportation Systems*. **13**, 1507-1514 (2012)

[34] Houben, S., Stallkamp, J., Salmen, J., Schlipsing, M. & Igel, C. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. *The 2013 International Joint Conference On Neural Networks (IJCNN)*. pp. 1-8 (2013)

[35] Lillo-Castellano, J., Mora-Jiménez, I., Figuera-Pozuelo, C. & Rojo-Álvarez, J. Traffic sign segmentation and classification using statistical learning methods. *Neurocomputing*. **153** pp. 286-299 (2015)

[36] Zhu, Y., Zhang, C., Zhou, D., Wang, X., Bai, X. & Liu, W. Traffic sign detection and recognition using fully convolutional network guided proposals. *Neurocomputing*. **214** pp. 758-766 (2016)

[37] Gokulanathan, S., Feldsher, A., Malca, A., Barrett, C. & Katz, G. Simplifying neural networks using formal verification. *NASA Formal Methods Symposium*. pp. 85-93 (2020)

[38] Hains, G., Jakobsson, A. & Khmelevsky, Y. Formal methods and software engineering for DL. Security, safety and productivity for DL systems development. *ArXiv Preprint ArXiv:1901.11334*. (2019)

[39] Russell, S., Dewey, D. & Tegmark, M. Research priorities for robust and beneficial artificial intelligence. *Ai Magazine*. **36**, 105-114 (2015)

[40] Li, J., Liu, S., Liu, A. & Huang, R. Knowledge graph construction for SOFL formal specifications. *International Journal Of Software Engineering And Knowledge Engineering*. **32**, 605-644 (2022)

[41] Li, J. & Liu, S. Requirements-related fault prevention during the transformation from formal specifications to programs. *IET Software*. (2023)

[42] Stallkamp, J., Schlipsing, M., Salmen, J. & Igel, C. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*. **32** pp. 323-332 (2012)