Testing Artificial Intelligence System Towards Safety and Robustness: State of the Art

Tingting Wu, Yunwei Dong, Zhiwei Dong, Aziz Singa, Xiong Chen, and Yu Zhang

***Abstract*—With the increasing development of machine learning, conventional embedded systems cannot meet the requirement of current academic researches and industrial applications. Artificial Intelligence System (AIS) based on machine learning has been widely used in various safety-critical systems, such as machine vision, autonomous vehicles, collision avoidance system. Different from conventional embedded systems, AIS generates and updates control strategies through learning algorithms which make the control behaviors nondeterministic and bring about the test oracle problem in AIS testing procedure. There have been various testing approaches for AIS to guarantee the safety and robustness. However, few researches explain how to conduct AIS testing with a complete workflow systematically. This paper provides a comprehensive survey of existing testing techniques to detect the erroneous behaviors of AIS, and sums up the involved key steps and testing components in terms of test coverage criterion, test data generation, testing approach and common dataset. This literature review aims at organizing a standardized workflow and leading to a practicable insight and research trend towards AIS testing.**

***Index Terms*—artificial intelligence system, machine learning, neural network, testing, verification.**

I. Introduction

ARTIFICIAL Intelligence (AI) is a technique capable of making the machine think or act like a human by simulating human intelligence. With the broad applications of artificial intelligence in wearable devices, autonomous cars, smart robot and smart city [1], the traditional embedded system, including sensor, controller and actuator, has evolved into the artificial intelligence system (AIS) with the capabilities of autonomous learning, decision and controlling. For instance, autonomous cars [2] perceive the surrounding driving environment by sensors such as radar and camera. Based on the perception data, autonomous cars perform the tasks such as obstacles detection and tracking, traffic signals detection and recognition, and route planning through the intelligence component implemented by machine learning (ML). Then, autonomous cars make decisions for these tasks and drive safely on road via the autonomous interations with driving environment.

The progress of AI technology goes through three stages:- *symbolism AI*, *connectionism AI* and *actionism AI* [3] over the past decades. The *symbolism AI* is a technology based on mathematical logic, which is defined by physical symbols to imitate human thoughts intelligently with deduction rules. The heuristic algorithms and expert system are the classical technologies developed based on the symbolism AI. The *connectionism AI* is a technology based on neural network, and is defined by neurons and connections between them to mimic the human brain structure and learning function. Therefore, the neural network and deep learning (DL) are the fundamental methods to implement the connectionism AI. The *actionism AI* is a technology based on cybernetics, which aims at interacting with the external environment through intelligent and autonomous behaviors. In this case, the AI system based on actionism is regarded as an optimization and control system by the methods of evolutionary computation or reinforcement learning. With the incredible development of hardware and computing capability, the connectionism AI has become the essential technology in learning the distribution of large-scale data and predicting the output correspondingly. Thus, we mainly focus on the connectionism based AI system and discuss the state-of-the-art of testing AIS in this paper.

Since some applications of artificial intelligence are safety-critical systems, once some faults occur without any safety control behavior from system or person, there will be either error decision or fatal consequence. The traffic accident happened on March 18, 2018 that a self-driving Uber SUV "killed" a pedestrian makes the safety of self-driving system an emergency service required before the self-driving car is employed. As shown in the report [4], the perception system "sees" clearly the pedestrian through LiDAR but the decision system makes no decision, brake, slow down or alert the assistant driver, for the potential collision. Similar cases that Waymo self-driving car collides into an accident vehicle without any avoidance and Tesla Model S crashes because of the misclassification between the sky and white truck indicate that the autonomous level of self-driving system is still in Level 3, which requires the human intervention in emergency cases. Thus, it is crucial to detect the erroneous behaviors in AIS and assure its safety and robustness through certain testing techniques.

The nature and features of intelligence system[1] design and implementation, the complex requirements, large-scale network, and nondeterministic and uninterpretable learning results, bring great challenges in revealing the error behaviors

Manuscript received September 28, 2019; revised April 12, 2020.

Tingting Wu is with the School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China, email: tingtingwu@mail.nwpu.edu.cn

Yunwei Dong is with the School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China, email: yunweidong@nwpu.edu.cn

Zhiwei Dong is with the State Grid Liaoning Eletric Power Company Limited Institute, Shenyang, 110006, China, email: dzw@vip.163.com

Aziz Singa is with the School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China, email: azsinga81@gmail.com

Xiong Chen is with the NARI Group Corporation/State Grid Electric Power Research Institute, Nanjing, 210000, China, email: chenxiong@sgepri.sgcc.com.cn

Yu Zhang is with the School of Computer Science and Engineering, Northwestern Polytechnical University, Xi'an 710072, China, email: zhangyu@nwpu.edu.cn

[1]Hereafter intelligence system and artificial intelligence system (AIS) are used interchangeably.

and faults in system [5]–[7]. Various testing techniques have been investigated to test the intelligence component in intelligence system, especially machine learning based system. For example, adversarial attack [8], generative adversarial network [9] and metamorphic testing [10], [11] for detecting faults in training model, and mutation testing [12], [13] for testing adequacy of test data. Some researchers and institutions have reviewed the progress of machine learning testing from different perspectives. For example, Hains et al. [14] investigate the existing verification and simulation techniques for the safety of machine learning. Masuda et al. [15] review the error detection of machine learning and the applications of conventional software testing on machine learning. Braiek et al. [16] study the datasets, and black- and white-box based testing techniques for machine learning testing. Ma et al. [17] discuss the safety challenge in machine learning system. Huang et al. [18] introduce the testing and verification techniques for the safety of machine learning. However, the above mentioned researches mainly focus on the testing or verification techniques without a systematic testing procedure from test data generation to different testing approaches. In this paper, we outline the systematic testing workflow of artificial intelligence system in terms of test coverage metric, test data generation, testing approach and common datasets by inspecting the related publications since 2009, and discuss the dominant methods for individual aspect.

The remainder of this paper is organized as follows. Section II introduces the preliminaries of deep learning, deep neural network (DNN), the architecture of artificial intelligence system and the testing workflow. Section III introduces our review procedure by collecting related publications and analyzing the publications in terms of researcher and institution, geographical distribution, and publication venues. Section IV discusses the coverage metrics for neural network testing at three levels of different granularities. Section V introduces the dominant test data generation algorithms for training and assessing learning model. Section VI and Section VII focus on different testing and formal verification techniques. Section VIII presents the common datasets used for image classification and self-driving. Section IX summarizes the paper and discusses the future work in intelligence system testing.

II. Preliminaries

A. Deep Learning

As one of the machine learning method, deep learning [19]–[21] helps computer implement some functions in human-level by learning from enormous amounts of data and heuristic knowledge. With the purpose of designing and implementing the AIS, DL has achieved an incredible progress in human-level abilities in areas of computer vision [22], machine translation [23], speech recognition [24] and game playing [25]. The above mentioned applications have been used in various safety-critical systems such as autonomous driving system [2], collision avoidance system [26] and medical image processing system [27].

A DL system is either composed of DNN components or a combination of DNN and conventional software [28]. Different from the development of traditional software which is logically based, DNN components learn the patterns from the distribution of training data and the data features. Thus, it is crucial to detect the erroneous and insecure behaviors of DL systems through a variety of testing techniques.

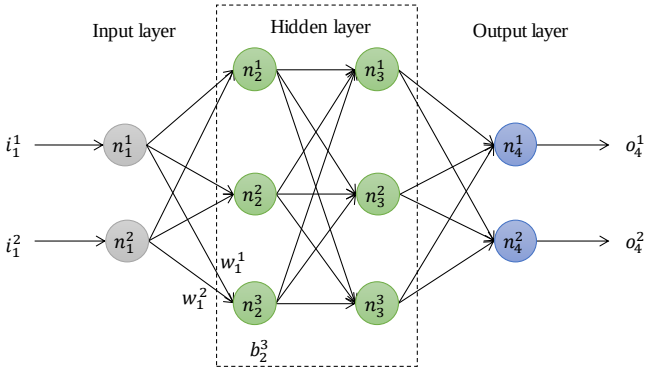

Fig. 1: Illustration of a Simple DNN

B. DNN Architecture

Deep neural network [21] is structured by layers and connections between neurons in neighbor layers. These layers are composed of one input layer, multiple hidden layers and one output layer, and each connection corresponds to a pre-trained weight. Except neurons in the input layer, each neuron in the hidden and output layers has a bias. As the basic unit, the neuron takes the outputs from neurons in last layer as inputs, and delivers the value calculated by the activation function as output. Thus, DNN is formally defined as a tuple $N = (L, T, F)$ [29], in which

1) $L = \{L_1, L_2, \cdots, L_K\}$ is a set of layers. L_1 is the input layer, $L_2 \sim L_{K-1}$ are the hidden layers, and L_K is the output layer. Assume that there are s_l neurons in the l-th layer, then $L_l = \{n_l^1, n_l^2, \cdots, n_l^k, \cdots, n_l^{s_l}\}$, where n_l^k is the k-th neuron in the l-th layer.
2) $T \subseteq L \times L$ is a set of connections between layers. Each hidden layer has incoming connections and outgoing connections, while input layer only has outgoing connections and output layer only has incoming connections.
3) $F = \{f_2, f_3, \cdots, f_K\}$ is a set of activation functions for each layer except the input layer. For the neuron n_l^k, its output value is $v_l^k = f_l(u_l^k)$, in which $u_l^k = \Sigma_{i=1}^{s_{l-1}} w_{l-1}^i v_{l-1}^i + b_l^k$, f_l is the activation function, u_l^k is the input value of n_l^k, w_{l-1}^i is the weight of connection between neuron n_{l-1}^i and n_l^k, v_{l-1}^i is the output of neuron n_{l-1}^i, and b_l^k is the bias of n_l^k.

As shown in figure 1, this is a simple example of a full connected DNN. Given the input vector $(i_1^1, i_1^2) = (0, 1)$, the trained weights $w_1^1 = -1$ and $w_1^2 = 1$, and bias $b_2^3 = 0$, take ReLU as the activation function, the output value of the third neuron in the second layer, n_2^3, will be $v_2^3 = \text{ReLU}(\Sigma_{j=1}^2 w_1^j i_1^j + b_2^3) = \text{ReLU}(1) = \max(0, 1) = 1$. Furthermore, $v_2^3 = 1$ will be regarded as the input of the neurons in the third layer.

C. Artificial Intelligence System

Since the connectionism AI is the most popular technology, we briefly discuss the architecture of artificial intelligence system based on deep learning in this section.

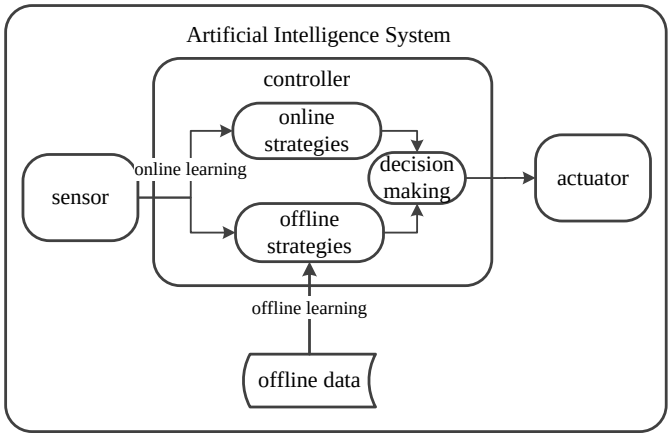

Fig. 2: Architecture of Artificial Intelligence System

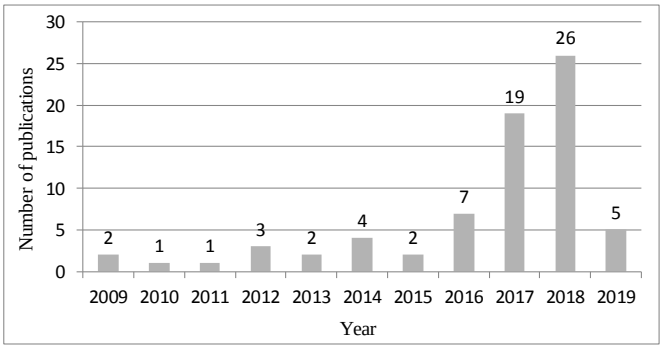

Fig. 4: Publications of Artificial Intelligence System Testing from 2009 to 2019

The conventional embedded system takes as input the data from sensors and computes the control strategies according to the program with specific task. The controllers output the corresponding decision based on the control strategies and transform this decision into a command to actuators. Then actuators consequently take some operations to control the entire embedded system. Thus, the conventional embedded system is logic deterministic with the certain program and control behaviors. Different from the conventional embedded system, the control strategies of artificial intelligence system are obtained by learning from training data under the machine learning algorithm, which will lead to less precise strategies since the intelligence system is data-driven. As shown in figure 2, the AIS learns the online or offline strategies by online learning or offline learning from the sensor data or offline data under the learning algorithms for specific application. Then make a decision according to the learned strategies and forward this decision to actuators in the format of commands. In this case, the advanced embedded system can perform the autonomous learning, decision and controlling with the intelligent component, the machine learning program. The testing of conventional embedded system follows a general testing procedure: generating test cases, executing test cases and comparing the actual outputs with the expected outputs. Nevertheless, because of the nature of machine learning algorithms, the control strategies update even for the same training data, which can generate different or interpretable control behaviors and weaken the safety and robustness of intelligence system. Therefore, there exists the oracle problem [30] in testing artificial intelligence system.

D. Artificial Intelligence System Testing

Testing intelligence system is a process of detecting system erroneous behaviors, which aims to guarantee the safety and robustness of system. The AIS is a data-driven system because system learns from the input dataset and predicts the behavior with the trained model. Therefore, the correctness of system can be determined by checking whether the output behaviors meet the requirements. As shown in figure 3, given an intelligence system, the testing activity is conducted as the following steps.

1) Generate the samples, including training data, test data and individual oracle, based on the pre-designed datasets through some samples generation algorithms to satisfy the specific requirement of test coverage.
2) When given a training model, testers would take the data generated in the first step as the training data and train this model to get a learned model.
3) After training, testers would take the test dataset generated in the first step as the evaluation data and output the predicted decision of each test data.
4) Compare the predicted output with the corresponding oracle generated in the first step to determine the correctness of the predicted output.
5) If the predicted output is not equal to the relative oracle, which indicates an incorrect behavior, then repair the intelligence system and conduct regression testing to check whether any new mistakes are introduced in the repair process; Otherwise, no error behavior is detected and terminate the testing process.

III. Review Method

To perform a comprehensive survey on artificial intelligence system testing, we follow a similar review method from the survey on metamorphic testing [31]. The review process of publication collection and results are as follows.

A. Publication Collection

We searched papers about AIS testing in the Google Scholar between 2009 to 2019, which contain either “neural network”, “machine learning”, “test”, “verification”, “robustness”, “safety”, “self-driving” or “dataset” in title, abstract or keywords. After examining all of the searched papers, we filtered out 72 publications closely related to the scope of our review. Figure 4 lists the number of publications on AIS testing we selected and shows a significant growth since 2016, because Koopman et al. [5] proposed the challenges in testing and validating autonomous vehicle in that year. Therefore, research works from 2016 to 2019 mostly focus on studying test data generation algorithms and testing approaches. While papers from 2009 to 2015 mainly argue the safety problem of intelligency system and designing datasets for new application environments.

B. Collection Results

1) Researchers and Organizations: We identified 49 coauthors from 22 different institutions in these 72 publications.

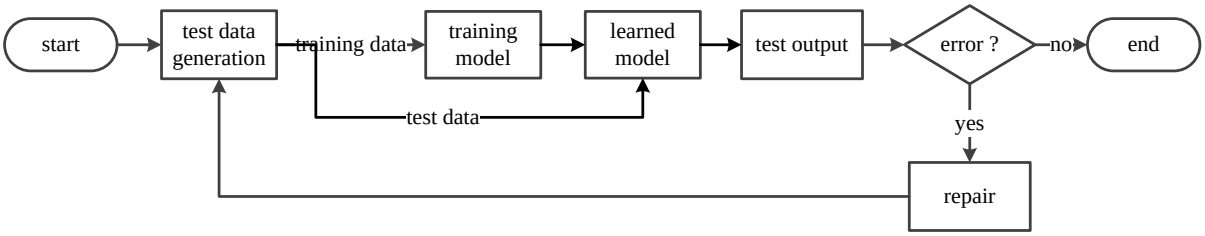

Fig. 3: Workflow of Artificial Intelligence System Testing

TABLE I: Top 10 Co-authors on AIS Testing

Author	Institution	Papers
Huang Xiaowei	University of Liverpool, UK	7
Goodfellow Ian	Google Brain, USA	7
Kwiatkowska Marta	University of Oxford, UK	5
Sun Youcheng	University of Oxford, UK	3
Papernot Nicolas	Pennsylvania State University, USA	3
Hinton Geoffrey	University of Toronto, CAN	3
Kurakin Alexey	Google Brain, USA	3
Pei Kexin	Columbia University, USA	3
Liu Yang	Nanyang Technological University, SG	3
Ma Lei	Harbin Institute of Technology, CHN	3

TABLE II: Geographical Distribution of Publication

Country	Papers
United States	30
China	10
United Kingdom	7
Switzerland	5
Australia	5
Germany	5
Canada	3
Japan	2
Italy	2
Brazil	1
France	1
India	1

Table I lists the top 10 authors with at least three papers published. Among which, both Huang Xiaowei and Goodfellow have 7 papers separately to be the most prolific authors in AIS testing, since Huang Xiaowei has proposed various testing methods including DeepCover [29] and DeepConcolic [32] and Goodfellow made a huge contribution in the area of artificial intelligence.

2) Geographical Distribution of Publications: We inspected the geographical distribution of each publication to the institution country of its first author. According to the observation of table II, all 72 papers were from only 12 countries. Among which, the USA and China take more than half of the publications. By continent, 47.2% of the papers are from America, 27.8% from Europe, 18.1% from Asia, and 6.9% from Oceania. This indicates that America, especially USA, leads the maintream in AI and AIS testing.

3) Publication Venues: The 72 papers reviewed were published in 33 different venues totally. Table III lists the top 10 venues with at least two papers about AIS tesing published. As observed, 34.7% of papers are published by arXix preprint online. 22.2% of papers are published by the journals and conferences with the scope of machine learning, such as 6.9% in CVPR, 2.8% in TNNLS and 2.8% in NIPS. 25% of papers are from the area of software engineering, such as 4.2% in SP and 2.8% in ASE.

TABLE III: Top 10 Venues on AIS Testing

Venue	Papers
arXiv preprint	25
IEEE Conference on Computer Vision and Pattern Recognition (CVPR)	5
IEEE Symposium on Security and Privacy (SP)	3
International Conference on Computer Aided Verification (CAV)	3
IEEE Transactions on Neural Networks and Learning Systems (TNNLS)	2
Advances in Neural Information Processing Systems (NIPS)	2
International Conference on Machine Learning (ICML)	2
International Conference on Automated Software Engineering (ASE)	2
Communications of the ACM	2

IV. Test Coverage

There already exists various test coverage metrics for traditional software testing adequacy: statement coverage, condition coverage, decision coverage and Modified Condition/Decision Coverage (MC/DC). There are also coverage criteria for software architecture testing, such as component path coverage with node coverage and edge coverage [33]. However, all the metrics above cannot be used directly to cover the data flow of AIS testing due to the complexity of neural network architecture and training process. Researchers take neuron as the basic coverage unit to conduct both major function behavior coverage and corner-case behavior coverage of neural network [34]. Therefore, we divide the neural network coverage into three categories from perspective of granularity:- *neuron-level coverage*, *layer-level coverage* and *neuron-pair-level coverage*. Table IV lists the fine classes for individual super class.

A. Neuron-Level Coverage

The neuron-level coverage depends on the output value of neuron. Thus, the neuron-level coverage can further be subdivided into *activated neuron coverage*, *k-multisection neuron coverage*, *neuron boundary coverage* and *strong neuron activation coverage*.

1) Activated Neuron Coverage: A neuron is considered *activated* if the neuron output is greater than the neuron activation threshold and makes contribution to neural network's behaviors including normal function and corner-case behaviors; Otherwise, the neuron is regarded *inactivated* by the input data. Therefore, as shown in Equation 1, the *activated neuron coverage* [10], [28] Cov_{ANC} is the rate of the number of neurons activated and the number of neurons in the whole DNN. It is desired to generate a test dataset to enlarge the number of activated neurons so that with the neuron coverage increased.

TABLE IV: Coverage Criteria of Testing DNN

Coverage Criterion		Description	Application
Neuron-Level Coverage	Activated Neuron Coverage	The number of neurons activated in the DNN	DeepXplore [28], DeepTest [10]
	k-multisection Neuron Coverage	The number of sections covered in k equal sections	DeepGauge [34]
	Neuron Boundary Coverage	The coverage of corner-case regions	DeepGauge [34]
	Strong Neuron Activation Coverage	The coverage of corner-case regions with hyperactive neurons	DeepGauge [34]
Layer-Level Coverage	Top-k neuron coverage	The number of the most active k neurons in each layer	DeepGauge [34]
Neuron-Pair-Level Coverage	Sign-Sign Cover	The sign change of neuron n_l^i affects the sign of neuron n_{l+1}^j	DeepCover [29]
	Distance-Sign Cover	The distance change of neurons in layer l affects the sign of neuron n_{l+1}^j	DeepCover [29]
	Sign-Value Cover	The sign change of neuron n_l^i affects the output value of neuron n_{l+1}^j	DeepCover [29]
	Distance-Value Cover	The distance change of neurons in layer l affects the output value of neuron n_{l+1}^j	DeepCover [29]

$$Cov_{ANC} = \frac{n_{activated}}{N} \tag{1}$$

DeepXplore [28] first introduced the notion of activated neuron coverage empirically to detect erroneous corner-case behaviors in deep learning system, and increased neuron coverage by generating new tests from unlabeled test inputs to activate more neurons. DeepTest [10] proposed nine image transformations to simulate real-world environment of self-driving, and each image transformation can lead to different neuron coverage with the same input data. Therefore, the neuron coverage can be increased by combining some of these image transformations. However, both DeepXplore and DeepTest require the knowlege of neural network, which limits the AIS testing a labor-intensive and time-consuming activity.

2) k-multisection Neuron Coverage: Ma et al. [34] propose using neuron output value range to distinguish the major function region and corner-case region since activated neuron coverage based on neuron output value is computationally intensive. Therefore, given a neuron n, its output range, $[low_n, high_n]$, is denoted as the major function region of the neuron. To measure how the test dataset T covers the major function region $[low_n, high_n]$, this region is divided into k equal subsections and $k > 0$. As shown in Equation 2, *k-multisection neuron coverage* for a neuron n, $Cov_{k,n}$, is the rate of the number of subsections covered by T and the total number of subsections, in which x is a test input in dataset T, $\phi(x, n)$ is the output of neuron n with test input x, and S_i^n is the set of values in the i-th subsection. The *k-multisection neuron coverage* for the neural network N, Cov_{KMN}, is based on the k-multisection neuron coverage of all neurons in network, which is defined in Equation 3.

$$Cov_{k,n} = \frac{|\{S_i^n | \exists x \in T : \phi(x, n) \in S_i^n\}|}{k} \tag{2}$$

$$Cov_{KMN} = \frac{\Sigma_{n \in N} |\{S_i^n | \exists x \in T : \phi(x, n) \in S_i^n\}|}{k \times |N|} \tag{3}$$

3) Neuron Boundary Coverage: Though $[low_n, high_n]$ is referred to approximate the major function region, there still exists some cases where neuron output $\phi(x, n) \notin [low_n, high_n]$. In other words, $\phi(x, n)$ may locate in $(-\infty, low_n)$ or $(high_n, +\infty)$. Thus, $(-\infty, low_n) \cup (high_n, +\infty)$ is referred to as the corner-case region of neuron n. In this case, to measure how many corner-case regions are covered by test dataset T, the *neuron boundary coverage* Cov_{NBC} is the rate of the number of neurons falling in corner-case region and the toal number of corner cases as in Equation 4. In which, $N_{UPPER} = \{n \in N | \exists x \in T : \phi(x, n) \in (high_n, +\infty)\}$ is the set of neurons located in the upper corner-case region, and $N_{LOWER} = \{n \in N | \exists x \in T : \phi(x, n) \in (-\infty, low_n)\}$ is the set of neurons located in the lower corner-case region. Please note that, the total number of corner cases of neuron boundary coverage is equal to $2 \times |N|$, because $(-\infty, low_n)$ and $(high_n, +\infty)$ are mutally exclusive and neuron cannot fall in two regions at the same time.

$$Cov_{NBC} = \frac{|N_{UPPER}| + |N_{LOWER}|}{2 \times |N|} \tag{4}$$

4) Strong Neuron Activation Coverage: Note that, hyperactive corner-case neurons affect the training of DNN significantly. Therefore, it is essential to measure the coverage of hyperactive corner-case neurons. That is, the coverage of upper corner-case region which is denoted as *strong neuron activation coverage*. Similar to the neuron boundary coverage, strong neuron activation coverage is the rate of the number of neurons falling in the upper corner-case region and the total number of corner cases as in Equation 5.

$$Cov_{SNA} = \frac{|N_{UPPER}|}{|N|} \tag{5}$$

B. Layer-Level Coverage

Since hyperactive neurons determine the major function behaviors of neural network, Ma et al. [34] further investigate the neuron coverage from the perspective of top hyperacive neurons in each layer. Therefore, an effective input test dataset should cover more and more hyperactive neurons. Given test input x, neuron n_1 and n_2 in the same layer, if $\phi(x, n_1) > \phi(x, n_2)$, then n_1 is more active than n_2. Here, the *top-k neuron coverage* is designed to measure how many neurons are activated as the most k neurons in each layer by

the input test dataset T. As shown in Equation 6, $top_k(x,i))$ is the set of first k neurons which are ranked descendingly with their outputs.

$$Cov_{TKN} = \frac{|\cup_{x\in T}(\cup_{1\le i\le l} top_k(x,i))|}{|N|} \quad (6)$$

C. Neuron-Pair-Level Coverage

Both neuron-level and layer-level coverage take neurons as characterization of the behaviors of neural network, but these two coverage metrics overlook the propagation of changes between neurons from adjacent layers. Inspired by the MC/DC criterion, Sun et al. [29] propose the neuron-pair-level coverage by taking $\alpha = (n_l^i, n_{l+1}^j)$ as the neuron pair, in which n_l^i is a neuron regarded as a condition in the l-th layer and n_{l+1}^j is a neuron regarded as a decision in the $(l+1)$-th layer. Hence, neuron-pair-level coverage is presented to inspect the influence of neurons in the l-th layer on neurons in the $(l+1)$-th layer, which is essentially used to measure the influence of neuron changes on the network's output.

The change of neuron n_l^k when given two test inputs x_1 and x_2 could be a *sign change* (denoted as $sc(n_l^k, x_1, x_2)$), *value change* (denoted as $vc(g, n_l^k, x_1, x_2)$, and g is a value change function), and a *distance change* (denoted as $dc(h, l, x_1, x_2)$, and h is a distance change function) for neurons in the l-th layer. Therefore, given the neuron pair $\alpha = (n_l^i, n_{l+1}^j)$, two test inputs x_1 and x_2, the *neuron-pair-level coverage* is divided into the following four categories.

1) Sign-Sign Cover: The sign change of condition neuron n_l^i and signs of other neurons in the l-th layer not changing affect the sign of decision neuron n_{l+1}^j in the next layer. That is, if $sc(n_l^i, x_1, x_2) \wedge \neg sc(n_l^k, x_1, x_2) \wedge (k \neq i) \Rightarrow sc(n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *sign-sign covered* by x_1 and x_2 which is denoted as $cov_{SS}(\alpha, x_1, x_2)$.

2) Distance-Sign Cover: The small distance change of neurons in the l-th layer can cause the sign change of decision neuron n_{l+1}^j in the next layer. Namely, if $dc(h, l, x_1, x_2) \Rightarrow sc(n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *distance-sign covered* by x_1 and x_2, denoted as $cov_{DS}^h(\alpha, x_1, x_2)$.

3) Sign-Value Cover: Similar to sign-sign cover, the sign change of condition neuron n_l^i and signs of other neurons in the l-th layer not changing affect the value of decision neuron n_{l+1}^j in the next layer. That is, if $sc(n_l^i, x_1, x_2) \wedge \neg sc(n_l^k, x_1, x_2) \wedge (k \neq i) \Rightarrow vc(g, n_{l+1}^j, x_1, x_2)$, we say that (n_l^i, n_{l+1}^j) is *sign-value covered* by x_1 and x_2, denoted as $cov_{SV}^g(\alpha, x_1, x_2)$.

4) Distance-Value Cover: Similar to distance-sign cover, the small distance change of neurons in the l-th layer leads to the value change of decision neuron n_{l+1}^j in the next layer. Namely, if $dc(h, l, x_1, x_2) \Rightarrow vc(g, n_{l+1}^j, x_1, x_2)$, then (n_l^i, n_{l+1}^j) is *distance-value covred* by x_1 and x_2, denoted as $cov_{DV}^{h,g}(\alpha, x_1, x_2)$.

V. Test Data Generation

During training procedure, some common datasets are used as the training sets to compare the training performance of different training models. During testing procedure, the testing data included in the common datasets is used to measure the training effectiveness and correctness of intelligence sytem. Thus, both training and testing procedures are data-driven. Since the AIS may output different labels for inputs with high similarity, generating *test data* that can cover not only major function behaviors but also corner-case behaviors becomes an indispensable activity in intelligence system testing. As shown in table V, there are mainly five algorithms proposed for test data generation since 2013. Below we will introduce each algorithm in detail.

A. Adversarial Examples

Adversarial examples are test data generated with small, even imperceptible, perturbation on the original test inputs, which cause the network under test to misclassify it [8], [22], [35]–[37]. As followed is the formal definition of adversarial example.

Definition 1 (Adversarial Examples)**.** *Given neural network $f: x \to y$, $x \in R^n$, $y \in R^m$, where y_i is the confidence of input x classified as label i, and $C(x) = \arg\max_i y_i$ is the label i with the largest confidence y_i for input x. Assume $C(x)$ is the correct label, and $x^{'}$ is a test input very close to x. If $C(x) \neq C(x^{'})$, then $x^{'}$ is the adversarial example.*

Following Definition 1, neural network is vulnerable to adversarial perturbation, and adversarial examples are regarded as effective means to attack network. Therefore, debuggers can detect erroneous behaviors of intelligence system by adversarial examples and enhance the robustness by retraining the intelligence system against adversarial examples.

Szegedy et al. [8] propose to generate test data by using the *box-constrained L-BFGS* to solve the following minimization problem under the condition of $f(x^{'}) = l$. Equation 7 hopes to minimize the distance between x and $x^{'}$, $L_2 = \| x - x^{'} \|_2^2$, and the loss function $loss_{f,l}(x^{'})$ for the generated test data $x^{'}$ also labeled as l.

$$\text{Minimization } c\cdot \| x - x^{'} \|_2^2 + loss_{f,l}(x^{'}) \quad (7)$$

To generate adversarial examples in a quick way, Goodfellow et al. [38] have presented the *Fast Gradient Sign Method* (FGSM) based on the norm L_∞. As shown in Equation 8, the adversarial example $x^{'}$ depends on the single-step ϵ and the gradient sign of loss function $loss_{f,t}(x)$, which determines the direction that increases the probability of the targeted class t.

$$x^{'} = x - \epsilon \cdot sign(\nabla loss_{f,t}(x)) \quad (8)$$

However, the adversarial examples generated from FGSM may not be with the smallest perturbation. Therefore, to improve the accuracy of adversarial examples, Kurakin et al. [39] proposed the *Basic Iterative Method* (BIM) by replacing single-step ϵ in FGSM with multiple smaller steps α and minimizing the loss funtion for directed at the targeted label t. Equation 9 indicates that the adversarial example generated from the i-th step depends on that from the last step iteratively.

$$x_i^{'} = x_{i-1}^{'} - clip_\epsilon(\alpha \cdot sign(\nabla loss_{f,t}(x_{i-1}^{'}))), x_0^{'} = 0 \quad (9)$$

TABLE V: Algorithms of Test Data Generation

Algorithm	Description	Feature	Year
Adversarial Examples	Generate test data with imperceptible perturbations to the input	blackbox & whitebox	2013
Generative Adversarial Examples	Generate test data using generative adversarial nets	blackbox	2014
Metamorphic Testing Based Strategy	Generate follow-up test data with the noise from the metamorphic relation	blackbox	2009
Concolic Testing Based Strategy	Generate test data with the smallest distance to the input by the actual execution and symbolic execution	whitebox	2018
Synthesis Approach	Generate test data by changing the modification space	blackbox	2017

Kurakin [40] further proposed the *Iterative Least-likely Class Method* (ILCM) by taking the target label with the least likelyhood that is the most difficult to attack rather than the label with the most possibility in BIM. DeepFool [41] also generates the smallest adversarial perturbation based on L_2 to make the adversarial image reach the other side of the classifier's boundary iteratively until misclassification. Thus, DeepFool is as fool as FGSM but with smaller perturbation.

Since image classification varies with the change of each pixel and the great influence increases the possibility of target classification, Papernot et al. [42] proposed the *Jacobian-based Saliency Map Attack* (JSMA) to obtain a saliency map, which indicates the influence of each pixel on target label classification. Therefore, we can generate adversarial example with small perturbation by changing pixels with the greatest influence.

To get an adversarial image with less distortion on the seeded image, Carlini and Wagner [43] proposed applying three distance metrics:- L_0, L_2 and L_∞ norms to have a target attack on neural network. Su et al. [44] presented an extreme method to generate adversarial image by changing only one pixel of the seeded image.

As introduced above, most of these algorithms generate specific perturbation for the specific image which leads to adversarial example generation computationally intensive. The *universal perturbation* [45]–[47] is proposed for the misclassification of entire test dataset instead of misclassifying one specific test case. That is, the same perturbation can be used for different original images even other neural networks with similar architecture (denoted as double universal) [45]. UPSET [46] generates universal perturbation according to the target label, as which any perturbed image can be recognized. In other words, the adversarial examples are generated from the same perturbation and recognized as the same target label. However, the universal perturbation is training data dependent as the universal perturbation updates according to the amount of training data. To solve this problem, Mopuri et al. [47] propose the universal perturbation in absence of training data, named *data independent universal perturbation*, which is used to fool the learned features and misclassify the majority of data samples.

Besides the approaches mentioned above, there also exist some other algorithms for adversarial example generation. Cisse et al. [48] replace the task loss function with a surrogate loss function, Houdini, which converges to the task loss eventually. Then similarly, adversarial examples are generated based on the gradient of Houdini. Baluja et al. [49] propose the *Adversarial Transformation Networks* (ATNs), which are trained to generate adversarial examples by solving a joint loss function including the input-space loss function and output-space loss function. The input-space function aims at minimizing the difference between original input and perturbed example. The output-space function aims at maximizing the probability of fooling the network with the target class. Since some adversarial examples can transfer among small scale datasets, Liu et al. [50] use an ensemble-based approach to generate non-targeted and targeted adversarial examples that transfer over different large models and large scale datasets. That is, the adversarial example $x^{'}$ from k models can fool the $(k+1)$-th model as well. Therefore, the ensemble-based approach is an optimizer-based problem as Equation 10, in which α_i is the ensemble weights, f_i is the i-th model, t is the target class. The traditional adversarial examples are considered to fool computer vision but not human visual system with subtle perturbations. To attack and fool human eyes, Elsayed et al. [51] propose to generate adversarial examples transfering from machine learning model to human visual system using ensemble-based approach as well.

$$\arg\min_{x^{'}} -log((\Sigma_{i=1}^{k}\alpha_i f_i(x^{'}))\cdot 1_t) + \lambda d(x, x^{'}) \tag{10}$$

B. Generative Adversarial Examples

The Generative Adversarial Nets (GAN) [9], including a generative model *G* and a discriminative model *D*, are proposed to generate samples by learning and try to fool the discriminative model. The purpose of generative model is producing samples by learning the training data distribution that cause the discriminative model making a mistake. And the discriminative model must determine whether the input sample is from training data or generative model *G*. As shown in Equation 11, the generator $G(z)$ learns distribution from data x by the input noise $p_z(z)$, $D(x)$ is the probability of x from the training data rather than $G(z)$. In other words, D is trained to maximize the probability of recognizing x from the training data, while G is trained to minimize $log(1 - D(G(z)))$. Thus, GAN is used as a fundamental approach to generate test data with high quality and close to the natural data as well [9], [52], [53].

$$\begin{aligned}\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] \\ +\mathbb{E}_{z\sim p_z(z)}[log(1-D(G(z)))]\end{aligned} \tag{11}$$

C. Metamorphic Testing Based Strategy

Metamorphic testing (MT) [31], [54], [55] is an approach to alleviate the test oracle problem by using the certain property of program instead of comparing the expected output with the actual output since some expected outputs are expensive to compute. Such a property is referred to as the *metamorphic relation* (MR) of the program function. Metamorphic relation is the relationship between multiple inputs and outputs that should be satisfied by the correct program. That is, given the program under test P, source test case t_s, source output o_s. Then generate the follow-up test case t_f and its corresponding output o_f. If t_s, o_s, t_f and o_f do not satisfy the relevant MR, the program P is incorrect. If they satisfy the MR, P is considered containing no faults under this testing. As described above, MT is also used as a strategy of test case generation since it generates the follow-up test cases according to the source test cases and related MRs.

MT has already been applied on machine learning classifier testing since 2009 by employing the MRs of permutation, addition, duplication and removal of attributes, samples and labels [56], [57], which are used to generate follow-up test cases. To test DNN-based self-driving system, DeepTest [10] takes the property that the output steering angle should keep unchanged under different real-world driving environments as the metamorhpic relation to generate new test cases and determine whether the self-driving system satisfies the MR. Nine transformation based MRs:- changing brightness, changing contrast, translation, scaling, horizontal shearing, rotation, blurring, fog and rain against the original natural images are used to mimic real-world weather conditions and generate follow-up images for the weather effects. In addition to the weather influence on the self-driving system, Zhou et al. [11] investigate the effect of noise outside of the drivable area on the obstacle perception inside of the drivable area. The drivable area is referred to as the region of interest (ROI). Given two frames of 3D point cloud data A_s and A_f, and two frames are identical except that A_f contains some data points outside the ROI. After executing the obstacle perception system, the O_s and O_f will be the sets of obstacles identified in A_s and A_f respectively. Then the MR of obstacle perception system is that the noise in the region outside ROI would not cause the obstacles inside ROI undetectable. That is, if $A_s \subseteq A_f$, then $O_s \subseteq O_f$. Therefore, when given A_s, the follow-up test case A_f will be obtained according to this MR.

D. Concolic Testing Based Strategy

Concolic testing is an integration of concrete execution and symbolic execution [58]. The program is executed with some concrete input values, and then solve the symbolic constraints collected for each conditional statement to cover other execution paths uncovered by concrete inputs. Thus, new test case variants from concrete inputs are generated during the constraint solving procedure. Inspired by this, Sun et al. [32] leverage concolic testing on DNN testing to generate adversarial examples with high coverage, named DeepConcolic. Given a DNN under test, a set of coverage requirements $\Re$, an unsatisfied requirement r, the initial test suite T. In the phase of *concrete execution*, a test input $t \in T$ is identified to satisfy requirement r. Then, in the phase of *symbolic execution*, a new test input $t^{'}$ close to one of the test input $t^{''}$ from T is generated to satisfy requirement r. That is, $\exists t^{''} \in T$, $\parallel t^{''} - t^{'} \parallel \leqslant d$ and $t^{'}$ satisfies r, then $t^{'}$ is added to the test suite T. Repeatedly generating $t^{'}$ until all requirements are satisfied or no more requirements in $\Re$ can be satisfied.

E. Synthesis Approach

Dreossi et al. [59] propose an image generator by synthesizing realistic images from lower dimension images to test car classification in autonomous vehicle system. They first define a *modification space* including object configurations (e.g. road backgourd, cars) and image parameters (e.g. brightness, contrast, saturation) for the original images. Then sample the *modification points* from the modification space as inputs of image generator to synthesize new images with new road scenarios. These synthetic images consequently are used to detect faults in the car classification subsytem.

VI. Testing approach

The application of machine learning on embedded systems leads to challenges on testing the safety, correctness and robustness of intelligence systems [5]–[7], [14]–[18], [60], [61]. The safety indicates the measures for ensuring system or personal safety when illegitimate attacks occur. The correctness represents the probability of correct behaviors. The robustness reflects the influence of pertubations on system's functional behaviors. Researchers have investigated various testing techniques to test the properties of intelligence system mentioned above, including adversarial attack, mutation testing, metamorphic testing and test prioritization techniques as shown in table VI. Since formal verification is also an effective approach to assure these properties of intelligence system, we also briefly discuss the relevant techniques of formal verification in this section.

A. Adversarial Attack

Neural network is found vulnerable to imperceptible perturbations [8]. In other words, the adversarial examples are misclassified with high confidence by a trained neural network. Therefore, adversarial attack can be regarded as a powerful method to detect defects in intelligence systems [62]. Adversarial attack is a procedure of adversarial example generation, which has been discussed in Section V-B. We will review the categories of adversarial attack with regard to the output label and distance metric.

1) Targeted and Non-Targeted Adversarial Attack: According to the classification results for individual adversarial example, adversarial attack is divided into two categories: *targeted* and *non-targeted adversarial attack* [22], [53]. Targeted attack indicates that the input adversarial example is misclassified as a specific label, while non-targeted attack indicates that the input adversarial example is assigned as any label not equal to the correct one. That is, following Definition 1, given the neural network f, original input x, the related adversarial example $x^{'}$, and labels $C(x)$ and $C(x^{'})$ recognized for input x and $x^{'}$ respectively. If $C(x) \neq C(x^{'})$, this attack is said to be the non-targeted attack. If $C(x) \neq C(x^{'})$ and $C(x^{'}) = l_{target}$, where l_{target}

TABLE VI: Summarization of Testing Approaches

Testing Approach	Description	Advantage	Disadvantage	Year
Adversarial Attack	Reveal the defects in DNN by executing adversarial examples	Attack the neural network easily	Computationally intensive	2013
Mutation Testing	Evaluate the testing adequacy	Generate mutants by mutating training data, training program and trained model	Computationally intensive	2018
Metamorphic Testing	Determine the system correctness by checking whether the metamorphic relation is satisfied	Blackbox	Difficult to identify the MRs thoroughly	2009
Test Prioritization	Measure the correctness of classification by the purity of test data	Reduce the cost of mannual labeling	Time cost in running and computing all of test inputs and the purity individually	2019

is the specific label, this attack is said to be the targeted attack.

2) Distance Metric: Since adversarial example is required close to the original input, the adversarial attack can also be mainly divided into three categories according to the distance metric L_p-norm: L_0-norm attack, L_2-norm attack, and L_∞-norm attack [43], [63]. Following the definition of L_p-norm in Equation 12, L_0-norm, L_2-norm and L_∞-norm are as shown in Equation 13, Equation 14 and Equation 15. L_0 indicates the number of changed feature data in the original input. As a result, L_0-norm attack is used to minimize the number of features perturbed and generate new feature data against the target label. L_2 equals the Euclidean distance between original data and adversarial example. Less L_2 indicates a smaller perturbation between the individual feature data and a high similarity between original data and adversarial example. Hence, L_2 is also used to solve the problem of overfitting in adversarial example. L_∞ represents the maximum difference among all feature data, which is utilized to control the maximum perturbation between original data and adversarial example.

$$L_p = \| x - x' \|_p = (\Sigma_{i=1}^{n} |x_i - x_i^{'}|_p)^{\frac{1}{p}} \tag{12}$$

$$L_0 = \Sigma_{i=1}^{n} |x_i - x_i^{'}|^0 \tag{13}$$

$$L_2 = \sqrt{\Sigma_{i=1}^{n} |x_i - x_i^{'}|^2} \tag{14}$$

$$L_\infty = \max(|x_1 - x_1^{'}|, |x_2 - x_2^{'}|, \cdots, |x_n - x_n^{'}|) \tag{15}$$

B. Mutation Testing

Mutation testing is a method to measure test adequacy of a test suite by injecting faults in the original program under test, namely mutants [64]. The ratio of the number of mutants detected and the total number of mutants is referred to as the mutation score. The higher the mutation score is, the stronger the fault detectability of test suite is. That is, given a test suite T, a program under test P, and m mutants of P, $P^{'} = \{P_1, P_2, \cdots, P_m\}$. Execute mutants on the test suite, if there are n mutants killed by T, the mutation score will be n/m which indicates the fault detectability of test suite T.

Motivated by mutation testing, Shen et al. [12] propose five kinds of mutation operators according to the architecture of neural network, including deleting neurons in the input layer and hidden layers, changing the bias, weights and activation functions. A mutant neural network is said to be killed once its output is distinct from the output of the original network. The more mutants networks are killed indicating a powerful test suite for DNN testing.

Since a trained model is obtained from the training program and training data, Ma et al. [13] propose a further study on mutation testing of deep learning system from two perspectives. One is generating mutant trained models based on the *source-level* mutation operators on the training data or training program. That is, given an original training dataset D, original training program P, a trained model M by training P on D, then generate mutant data $D^{'}$ with five data mutation operators and mutant program $P^{'}$ with three program mutation operators. After training program P on mutant data $D^{'}$ or training mutant program $P^{'}$ on data D, the mutant trained models $M^{'}$ are obtained. Which are evaluated against the test data T for its test adequacy. Another one is generating mutant trained models directly based on the *model-level* mutation operators on the original trained model. Given an original training dataset D, original training program P, a trained model M by training P on D, then create mutant trained models $M^{'}$ by utilizing eight model-level mutation operators on M. Similarly, analyze the test adequacy of test data T by executing $M^{'}$ on T.

C. Metamorphic Testing

As discussed in Section V-C, metamorphic testing determines the correctness of software under test by checking whether the related metamorphic relations are satisfied or not. Thus, it is a fundamental activity identifying diverse metamorphic relations to evaluate their capability of fault detection and the quality of software.

Tian et al. [10] have explored nine transformation based MRs, including changing brightness, contrast, translation, scaling, horizontal shearing, rotation, blurring, fog and rain on the images to test the robustness of autonomous vehicles. Take images from camera as the source images, and create the follow-up images by one or more transformation MRs on source images. The DNN under test takes source and follow-up images as inputs respectively, and outputs the source and follow-up steering angles under different real-world weather conditions, namely $\theta_s = \{\theta_s^1, \theta_s^2, \cdots, \theta_s^n\}$,

$\theta_f = \{\theta_f^1, \theta_f^2, \cdots, \theta_f^n\}$. Strictly speaking, the steering angles should keep unchanged under these tranformations. That is, $\theta_s = \theta_f$. However, a small variation of steering angles, $\hat{\theta} = \{\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_n\}$, in real-world driving environment would not affect the driving behaviors. Thus, the variations within the error ranges could be allowed as shown in Equation 16, in which $MSE_{orig} = \frac{1}{n}\Sigma_{i=1}^{n}(\hat{\theta}_i - \theta_s^i)^2$.

$$(\hat{\theta}_i - \theta_f^i)^2 \leqslant \lambda MSE_{orig} \tag{16}$$

D. Test Prioritization

The general procedure of testing DNN-based system is executing DNN-based system against a test dataset with manual labels and inspecting whether the each learned label and manual label are identical, during labeling each test case before testing is a labor-intensive activity.

To reduce human effort on data labeling, Shi et al. [65] propose a test prioritization techinique, DeepGini, to detect erroneous behaviors by converting the problem of misclassification to the problem of impurity of test dataset, which can determine whether the input test case is misclassified by its impurity without the need of labels. Take binary classification as example, given a test data t, the output feature vector $B = \{c_1, c_2\}$, then execute DNN to compute the probability of each feature for t. If the probability classified as c_1 is $P_{c_1} = 100\%$, and c_2 is $P_{c_2} = 0$, then the feature vector B has the highest purity and t is more likely to be classified correctly. In contrast, if $P_{c_1} = 50\%$, $P_{c_2} = 50\%$, then B has the lowest purity, and t is more likely to be misclassified. $\xi(t)$ is defined as the metric of impurity and the likelihood of t being misclassified. As shown in Equation 17, $p_{t,i}$ is the probability of test case t being classified as class i. A lower $p_{t,i}^2$ indicates a higher impurity and a higher likelihood to misclassify t. Therefore, DeepGini can reveal the incorrect behavior by only executing the test dataset without labeling them manually.

$$\xi(t) = 1 - \Sigma_{i=1}^{N} p_{t,i}^2 \tag{17}$$

VII. Formal Verification

Testing is an essential activity for detecting the erroneous behaviors of intelligence sytems, and various kinds of testing techniques have been investigated for testing the robustness, testing adequacy and decreasing test cost. Besides the above-mentioned testing techniques, there exist a small portion of formal verification studies to ensure the safety of intelligence systems by solving the satisfiability problem [26], [61], [66]–[68], non-linearity problem [60], [69]–[71], symbolic interval analysis [72], reachability analysis [73], and abstract interpretation [74], [75].

A. Satisfiability Solver

Huang et al. [61] reduce the safety verification of a image classifier to the correct behavior satisfiability which can search for adversarial examples if misclassifications exist. Given a neural network N, an input image x, a region η round x with the same class. N is said to be safe for input x and η if the classification of images in η is invariant to x. That is, $N, \eta \vDash x$. In more depth, modify the input image x with a family of manipulations Δ, N is said to be safe for input x, η and manipulations Δ if the classification of region η keeps invariant to x under manipulations Δ, namely $N, \eta, \Delta \vDash x$. If $N, \eta, \Delta \nvDash x$, then the image classifier is vulnerable to these manipulations or adversarial perturbations.

B. Non-linear Problem

The formal verification of safety is also specified to prove the counterexample not exist for the set of variable constraints to make the property always true [60]. Since some activation functions are non-linear, the formal verification is transformed into a Mixed Integer Program (MIP) with the value of binary varibales δ_a, where $\delta_a = \{0, 1\}$. The binary varibale δ_a indicates the phase of activation function ReLU. If $\delta_a = 0$, say the activation function is blocked and the output of related neuron will be 0; Otherwise, the activation function is passing and the output of related neuron will be equal to its input value.

C. Symbolic Interval Analysis

ReluVal [72] utilizes the symbolic interval arithmetic to obtain the accurate range of DNN's output according to the ranges of input variables. Given an input range X, sub-intervals of X and security property P. Say DNN is secure if no value in range X and its sub-intervals violate property P, that is, any value from range X satisfies P; Otherwise DNN is insecure if there exists one adversarial example in X violating P, that is, there exists at least one sub-interval containing an adversarial example to make property P unsatisfied.

D. Reachability Analysis

To eliminate the limitation of network scale for formal verification, Ruan et al. [73] propose to transform safety verification into reachability analysis. If all values in the output range, the lower and upper bounds $[l, u]$, correspond to an input in input subspace $X^{'} \subseteq [0, 1]^n$, then the network f is reachable and the reachability diameter is $D(X^{'}; f) = u(X^{'}) - l(X^{'})$. The network f is said to be safe *w.r.t* the input $x \in X^{'}$ if all inputs in $X^{'}$ have the same label to x, as shown in Equation 18. Furthermore, the network f is more robust than network g given the input subspace $X^{'}$ if $D(X^{'}; f) < D(X^{'}; g)$. And $X^{'}$ leads to a more robust network f than $X^{''}$ if $D(X^{'}; f) < D(X^{''}; f)$.

$$\forall x^{'} \in X^{'} : \arg\max_j c_j(x^{'}) = \arg\max_j c_j(x) \tag{18}$$

E. Abstract Interpretation

Since the scales of input data and neural network are extremely tremendous, it is infeasible to verify whether individual input satisfies the safety properties precisely. To overcome this obstacle, an abstract domain is used to approximate the concrete domain and verify the safety properties against abstract domain directly, referred to as the abstract interpretation theory [76], [77]. The computing efficiency is obtained by cutting down the precision. There are three types of abstract domains for different precision requirements, including interval domain, zonotope domain

TABLE VII: Summarization of Formal Verification Techniques

Verification Technique	Description	Advantage	Disadvantage	Year
Satisfiability solver	Transform the safety verification to satisfiability solving problem	Verify the safety based on a region around a data point	Limitation of image complexity and network scale	2017
Non-linear problem	Transform the safety verification to non-linear problem	Propose a new benchmark **PCAM-NIST**	Approximation of each sub-domain	2017
Symbolic interval analysis	Verify the security by analyzing symbolic interval	Compute rigorous bounds of DNN output	Lack of rigorous behavior analysis of hidden neurons	2018
Reachability analysis	Transform the safety verification to reachability problem	Work with large scaled networks	Expensive cost on running time	2018
Abstract interpretation	Transform the robustness verification to abstract interpretation	Trade-off between precision and scalability	More precise domains increase the running time	2018

and polyhedra domain, among which the interval domain brings the lowest precision and polyhedra domain leads to the highest precision. Therefore, considering both precision and efficiency, AI^2 [74] and DiffAI [75] propose employing the zonotope domain to represent the abstract elements in neural network and outputing the abstract elements in each layer. Finally determine the safety of network by verifying whether the label of abstract output in the output layer is consistent with that of the concrete output.

To improve the precision and efficiency of DNN safety verification, Yang et al. [78] propose a symbolic propagation method based on the abstract interpretation by representing the values of neurons symbolically and propagating them from the input layer to output layer forwardly. Given a network f, input X_0, activation function ReLU, they first transform the concrete input layer to interval domain by abstract transformer function, and then the output of each neuron will be $y = \mathrm{ReLU}(\Sigma_{i=1}^{k} w_i c_i + b)$ where c_i is a symbolic variable. Finally, compute the range of output layer by the interval abstract domain based on the symbolic representation of output layer. Note that this output range is more precise than the range from only the interval domain without the symbolic propagation.

VIII. Common Datasets

Intelligence system has been used in various safety- and mission-critical systems, and it is essential to design a dataset adaptive for diverse application scenarios to conduct the training and testing procedure. There already exist many datasets for training and testing different intelligence systems such as the Enron dataset for natural language processing [79], Speech Commands for speech recognition [80], [81], Drebin [82] and Mobilesandbox [83] for Android malware detection [84]. In this section, we primarily introduce two kinds of popular datasets for image classification and self-driving. As shown in table VIII, the first six datasets, including training data, test data and corresponding labels, are datasets common used for image classification, and the last four datasets are used for objects detection subsystem in self-driving.

A. Datasets for Image Classification

MNIST [85], [86] is a dataset for recognizing handwritten digits (0∼9) including 70,000 images originating from the NIST database [87]. The MNIST dataset is composed of a training dataset with 60,000 images and a test dataset with 10,000 images, and each of which contains half of clear digits written by government staff and half of blurred digits written by students. EMNIST [88], [89] is an extension dataset of MNIST for identifying handwritten digits (0∼9) and letters (a∼z, A∼Z). Therefore, there are totally 62 classes in EMNIST, including 10 classes of digits and 52 classes of letters. However, some uppercases and lowercases can not be distinguished easily (e.g. C and c, K and k), the letters are merged into 37 classes. Fashion-MNIST [90], [91] is a dataset with the extremely same format and size of MNIST for identifying 10 classes of fashion products, such as T-shirt, trouser, pullover, dress, coat, sandals, shirt, sneaker, bag and ankle boots. ImageNet [92], [93] is an image dataset describing the synsets in the WordNet hierarchy with on average 1000 images. CIFAR-10 and CIFAR-100 [94] are labeled image datasets consisting of 60,000 32×32 color images, 50,000 of which are training images and 10,000 are test images. CIFAR-10 is divided 10 classes, and there are 5000 training images and 1000 test images for each class. Similarly, CIFAR-100 is divided 100 fine classes, and each class contains 500 training images and 100 test images.

B. Datasets for Self-Driving

Udacity Challenge dataset [95] is a set of images for training and testing the objects detection models in the Udacity Challenge competition to measure the performance of each participated model in terms of detection capability and classification precision. The Udacity Challenge dataset contains two parts according to its classification. One consists of 9420 images in three classes:- car, truck and pedestrain; Another consists of 15,000 images in five classes: car, truck, pedestrain, traffic light and bycicle. MSCOCO 2015 [96], [97] is a dataset gathered from the daily scenes of common objects in the real-world environment, which aims at object recognition and localization. This dataset contains 165,482 training images, 81,208 verification images and 81,434 testing images in 91 classes, such as animal, vegetable and human. KITTI dataset [98]–[100] contains the realistic images captured from the real-world driving environments such as mid-size city, rural area and highway. All these images are divided into five classes, including road, city, residence, campus and human, for evaluation tasks like stero, optical flow, visual odometry, object detection and tracking. The Baidu Apollo [101] open platform provides the massive annotation data and simulation for training and testing

TABLE VIII: Common Datasets

Dataset	Description	Input Format	Class#	Training Data#/Test Data#
MNIST	Identify handwriting digits	28×28 gray image	10	60,000 / 10,000
EMNIST	Identify handwriting digits and letters	28×28 gray image	10 (by digits) / 37 (by letters)	60,000 / 10,000
Fashion-MNIST	Identify costume	28×28 gray image	10	60,000 / 10,000
ImageNet 2012	Describe synsets from WorldNet with image datasets	Images from internet	1000	1200,000 / 100,000
CIFAR-10	Identify real-world images in 10 classes	32×32 color image	10	50,000 / 10,000
CIFAR-100	Identify real-world images in 100 classes	32×32 color image	100	50,000 / 10,000
Udacity Challenge	Object identification datasets used in Udacity Challenge	1920×1920 color image	12	24,000
MSCOCO 2015	Common objects identification	image	91	328,000
KITTI	Multiple driving environments for self-driving testing	image	5	15,000
Baidu Apollo	Multiple driving environments for self-driving testing	3382×2710 image	26	116.6GB

autonomous driving tasks, such as obstacle detection and classification, traffic light detection, road hackers, obstacle trajectory prediction and scene analysis under different street view and vehicle movement images.

IX. Conclusion and Future Work

Since machine learning plays a fundamental role in safety-critical intelligence systems, it is necessary to ensure the safety, robustness of machine learning based intelligence systems. We have defined the workflow of intelligence system testing and provided an overview of intelligence system testing techniques in terms of testing coverage metric, test data generation, testing approaches, formal verification techniques and common datasets, which aims at building a standard testing framework for intelligence system testing. However, there still exist the following opportunities in intelligence system testing because of the increasing development of machine learning and system scale.

1) Test coverage. Since intelligence sytem has been embedded in more and more application areas, training and testing procedure should be enhanced to adapt to different systems which leads to more complex networks. Thus, this is the first opportunity to increase the testing coverage on different scaled intelligence systems with regard to the traditional software testing coverage metrics.
2) Test dataset. Both training and testing are data-driven, the datasets with huge size are needed to improve the testing adequacy for different environments or scenarios. Enormous test data involves powerful computing ability and expensive computing cost. Therefore, this is the second opportunity to design a dataset with relatively smaller size to test various scenarios.
3) Testing object. Intelligence system testing currently focuses on assuring the system safety and robustness. This is the third opportunity to test more properties for machine learning system including efficiency and interpretability.
4) Interpretability of AI system. After erroneous behaviors are detected in the testing process, testers can explain the occurrence by locating suspicious neurons and feature data. Therefore, this is the fourth opportunity to investigate the interpretability of AI system.

Acknowledgment

This work is supported by State Grid Technology Major Project of China under Grant No. 2019GW-12 (Security Protection Technology of Embedded Components and Control Units in Power System Terminal).

References

[1] Q. Huang, "Energy-efficient smart building driven by emerging sensing, communication, and machine learning technologies," *Engineering Letters*, vol. 26, no. 3, pp. 320–332, 2018.
[2] C. Badue, R. Guidolini, R. V. Carneiro, P. Azevedo, V. B. Cardoso, A. Forechi, L. F. R. Jesus, R. F. Berriel, T. M. Paixão, F. Mutz *et al.*, "Self-driving cars: A survey," *arXiv preprint arXiv:1901.04407*, 2019.
[3] Q. Sun and L. Yang, "From independence to interconnectiona review of ai technology applied in energy systems," *CSEE Journal of Power and Energy Systems*, vol. 5, no. 1, pp. 21–34, 2019.
[4] L. Dave, "Sensor firm velodyne baffled by uber self-driving death," https://www.bbc.com/news/technology-43523286, 23 March 2018.
[5] P. Koopman and M. Wagner, "Challenges in autonomous vehicle testing and validation," *SAE International Journal of Transportation Safety*, vol. 4, no. 1, pp. 15–24, 2016.
[6] I. Goodfellow and N. Papernot, "The challenge of verification and testing of machine learning," *Cleverhans-blog*, 2017.
[7] P. Koopman and M. Wagner, "Autonomous vehicle safety: An interdisciplinary challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.
[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," *arXiv preprint arXiv:1312.6199*, 2013.
[9] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.
[10] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*. ACM, 2018, pp. 303–314.
[11] Z. Q. Zhou and L. Sun, "Metamorphic testing of driverless cars," *Communication of the ACM*, no. 3, pp. 61–67, 2019.
[12] W. Shen, J. Wan, and Z. Chen, "Munn: Mutation analysis of neural networks," in *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 2018, pp. 108–115.
[13] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2018, pp. 100–111.
[14] G. Hains, A. Jakobsson, and Y. Khmelevsky, "Towards formal methods and software engineering for deep learning: security, safety and productivity for dl systems development," in *2018 Annual IEEE International Systems Conference (SysCon)*. IEEE, 2018, pp. 1–5.

[15] S. Masuda, K. Ono, T. Yasue, and N. Hosokawa, "A survey of software quality for machine learning applications," in *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2018, pp. 279–284.

[16] H. B. Braiek and F. Khomh, "On testing machine learning programs," *arXiv preprint arXiv:1812.02257*, 2018.

[17] L. Ma, F. Juefei-Xu, M. Xue, Q. Hu, S. Chen, B. Li, Y. Liu, J. Zhao, J. Yin, and S. See, "Secure deep learning engineering: A software quality assurance perspective," *arXiv preprint arXiv:1810.04538*, 2018.

[18] X. Huang, D. Kroening, M. Kwiatkowska, W. Ruan, Y. Sun, E. Thamo, M. Wu, and X. Yi, "Safety and trustworthiness of deep neural networks: A survey," *arXiv preprint arXiv:1812.08342*, 2018.

[19] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[21] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[22] N. Akhtar and A. Mian, "Threat of adversarial attacks on deep learning in computer vision: A survey," *IEEE Access*, vol. 6, pp. 14 410–14 430, 2018.

[23] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing," *ieee Computational intelligenCe magazine*, vol. 13, no. 3, pp. 55–75, 2018.

[24] G. Hinton, L. Deng, D. Yu, G. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, B. Kingsbury *et al.*, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal processing magazine*, vol. 29, 2012.

[25] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

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

[27] M. I. Razzak, S. Naz, and A. Zaib, "Deep learning for medical image processing: Overview, challenges and the future," in *Classification in BioApps*. Springer, 2018, pp. 323–350.

[28] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 1–18.

[29] Y. Sun, X. Huang, and D. Kroening, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[30] E. T. Barr, M. Harman, P. Mcminn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE Transactions on Software Engineering*, vol. 41, no. 5, pp. 507–525, 2015.

[31] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortés, "A survey on metamorphic testing," *IEEE Transactions on software engineering*, vol. 42, no. 9, pp. 805–824, 2016.

[32] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 109–119.

[33] L. Lun, X. Chi, and H. Xu, "Testing approach of component interaction for software architecture," *IAENG International Journal of Computer Science*, vol. 45, no. 2, pp. 353–363, 2018.

[34] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. ACM, 2018, pp. 120–131.

[35] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, 2019.

[36] B. Biggio and F. Roli, "Wild patterns: Ten years after the rise of adversarial machine learning," *Pattern Recognition*, vol. 84, pp. 317–331, 2018.

[37] S. Zhang, X. Zuo, and J. Liu, "The problem of the adversarial examples in deep learning," *Journal of Computers*, no. 102, pp. 1–21, 2018.

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

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

[40] ——, "Adversarial machine learning at scale," *arXiv preprint arXiv:1611.01236*, 2016.

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

[42] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 372–387.

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

[44] J. Su, D. V. Vargas, and K. Sakurai, "One pixel attack for fooling deep neural networks," *IEEE Transactions on Evolutionary Computation*, 2019.

[45] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, and P. Frossard, "Universal adversarial perturbations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1765–1773.

[46] S. Sarkar, A. Bansal, U. Mahbub, and R. Chellappa, "Upset and angri: breaking high performance image classifiers," *arXiv preprint arXiv:1707.01159*, 2017.

[47] K. R. Mopuri, U. Garg, and B. R. Venkatesh, "Fast feature fool: A data independent approach to universal adversarial perturbations," *arXiv preprint arXiv:1707.05572*, 2017.

[48] M. Cisse, Y. Adi, N. Neverova, and J. Keshet, "Houdini: Fooling deep structured prediction models," *arXiv preprint arXiv:1707.05373*, 2017.

[49] S. Baluja and I. Fischer, "Adversarial transformation networks: Learning to generate adversarial examples," *arXiv preprint arXiv:1703.09387*, 2017.

[50] Y. Liu, X. Chen, C. Liu, and D. Song, "Delving into transferable adversarial examples and black-box attacks," *arXiv preprint arXiv:1611.02770*, 2016.

[51] G. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. Goodfellow, and J. Sohl-Dickstein, "Adversarial examples that fool both computer vision and time-limited humans," in *Advances in Neural Information Processing Systems*, 2018, pp. 3910–3920.

[52] X. Huang, Y. Li, O. Poursaeed, J. Hopcroft, and S. Belongie, "Stacked generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 5077–5086.

[53] O. Poursaeed, I. Katsman, B. Gao, and S. Belongie, "Generative adversarial perturbations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4422–4431.

[54] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: a new approach for generating next test cases," Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong , Tech. Rep., 1998.

[55] T. Y. Chen, F.-C. Kuo, H. Liu, P.-L. Poon, D. Towey, T. Tse, and Z. Q. Zhou, "Metamorphic testing: A review of challenges and opportunities," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 4, 2018.

[56] X. Xie, J. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Application of metamorphic testing to supervised classifiers," in *2009 Ninth International Conference on Quality Software*. IEEE, 2009, pp. 135–144.

[57] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[58] C. Cadar and K. Sen, "Symbolic execution for software testing: three decades later." *Communications of the ACM*, vol. 56, no. 2, pp. 82–90, 2013.

[59] T. Dreossi, S. Ghosh, A. Sangiovanni-Vincentelli, and S. A. Seshia, "Systematic testing of convolutional neural networks for autonomous driving," *arXiv preprint arXiv:1708.03309*, 2017.

[60] R. Bunel, I. Turkaslan, P. H. Torr, P. Kohli, and M. P. Kumar, "A unified view of piecewise linear neural network verification," *arXiv preprint arXiv:1711.00455*, 2017.

[61] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *International Conference on Computer Aided Verification*. Springer, 2017, pp. 3–29.

[62] P. Yi, K. Wang, C. Huang, S. Gu, F. Zou, and J. Li, "Adversarial attacks in artificial intelligence: A survey," *Journal of Shanghai Jiao Tong University*, no. 10, pp. 1298–1306, 2018.

[63] M. Wicker, X. Huang, and M. Kwiatkowska, "Feature-guided black-box safety testing of deep neural networks," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2018, pp. 408–426.

[64] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE transactions on software engineering*, vol. 37, no. 5, pp. 649–678, 2010.

[65] Q. Shi, J. Wan, Y. Feng, C. Fang, and Z. Chen, "Deepgini: Prioritizing massive tests to reduce labeling cost," *arXiv preprint arXiv:1903.00661*, 2019.

[66] L. Pulina and A. Tacchella, "An abstraction-refinement approach to verification of artificial neural networks," in *International Conference on Computer Aided Verification*. Springer, 2010, pp. 243–257.

[67] N. Narodytska, S. Kasiviswanathan, L. Ryzhyk, M. Sagiv, and T. Walsh, "Verifying properties of binarized deep neural networks," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[68] C.-H. Cheng, G. Nührenberg, C.-H. Huang, and H. Ruess, "Verification of binarized neural networks via inter-neuron factoring," *arXiv preprint arXiv:1710.03107*, 2017.

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

[70] C.-H. Cheng, G. Nührenberg, and H. Ruess, "Maximum resilience of artificial neural networks," in *International Symposium on Automated Technology for Verification and Analysis*. Springer, 2017, pp. 251–268.

[71] W. Xiang, H.-D. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5777–5783, 2018.

[72] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Formal security analysis of neural networks using symbolic intervals," in *27th {USENIX} Security Symposium ({USENIX} Security 18)*, 2018, pp. 1599–1614.

[73] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," *arXiv preprint arXiv:1805.02242*, 2018.

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

[75] M. Mirman, T. Gehr, and M. Vechev, "Differentiable abstract interpretation for provably robust neural networks," in *International Conference on Machine Learning*, 2018, pp. 3575–3583.

[76] P. Cousot and R. Cousot, "Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints," in *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 1977, pp. 238–252.

[77] ——, "Abstract interpretation frameworks," *Journal of logic and computation*, vol. 2, no. 4, pp. 511–547, 1992.

[78] P. Yang, J. Liu, J. Li, L. Chen, and X. Huang, "Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification," *arXiv preprint arXiv:1902.09866*, 2019.

[79] "Enron," https://www.cs.cmu.edu/∼./enron/, 2015.

[80] P. Warden, "Speech commands: A public dataset for single-word speech recognition," 2017.

[81] "Speech commands," https://download.tensorflow.org/data/speech_commands_v0.01.tar.gz, 2017.

[82] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket." in *Ndss*, vol. 14, 2014, pp. 23–26.

[83] M. Spreitzenbarth, F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann, "Mobile-sandbox: having a deeper look into android applications," in *Proceedings of the 28th Annual ACM Symposium on Applied Computing*. ACM, 2013, pp. 1808–1815.

[84] S. Sachdeva, R. Jolivot, and W. Choensawat, "Android malware classification based on mobile security framework," *IAENG International Journal of Computer Science*, vol. 45, no. 4, pp. 514–522, 2018.

[85] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[86] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," http://yann.lecun.com/exdb/mnist/, 1998.

[87] P. J. Grother, "Nist special database 19 handprinted forms and characters database," Technical Report, National Institute of Standards and Technology, Tech. Rep., 1995.

[88] G. Cohen, S. Afshar, J. Tapson, and A. van Schaik, "Emnist: an extension of mnist to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.

[89] "Emnist," https://www.westernsydney.edu.au/bens/home/reproducible_research/emnist, 2017.

[90] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[91] "Fashion-mnist," https://github.com/zalandoresearch/fashion-mnist, 2017.

[92] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[93] "Imagenet," http://www.image-net.org/, 2009.

[94] "Cifar," http://www.cs.toronto.edu/∼kriz/cifar.html, 2014.

[95] "Udacity-challenge 2016. using deep learning to predict steering angles," https://github.com/udacity/self-driving-car, 2016.

[96] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[97] "Mscoco," http://cocodataset.org/, 2015.

[98] "Kitti," http://www.cvlibs.net/datasets/kitti/index.php, 2015.

[99] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[100] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2012, pp. 3354–3361.

[101] "Baidu apollo," http://apolloscape.auto/, 2017.