Computer Science Review 37 (2020) 100270

Contents lists available at ScienceDirect

Computer Science Review

journal homepage: www.elsevier.com/locate/cosrev

Survey

A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability☆

Xiaowei Huang [a,*], Daniel Kroening [b], Wenjie Ruan [c], James Sharp [d], Youcheng Sun [e], Emese Thamo [a], Min Wu [b], Xinping Yi [a]

[a] University of Liverpool, UK
[b] University of Oxford, UK
[c] Lancaster University, UK
[d] Defence Science and Technology Laboratory (Dstl), UK
[e] Queen's University Belfast, UK

ARTICLE INFO

Article history:
Received 6 September 2019
Received in revised form 22 April 2020
Accepted 18 May 2020
Available online 17 June 2020

ABSTRACT

In the past few years, significant progress has been made on deep neural networks (DNNs) in achieving human-level performance on several long-standing tasks. With the broader deployment of DNNs on various applications, the concerns over their safety and trustworthiness have been raised in public, especially after the widely reported fatal incidents involving self-driving cars. Research to address these concerns is particularly active, with a significant number of papers released in the past few years. This survey paper conducts a review of the current research effort into making DNNs safe and trustworthy, by focusing on four aspects: verification, testing, adversarial attack and defence, and interpretability. In total, we survey 202 papers, most of which were published after 2017.

Contents

☆ This work is supported by the UK EPSRC projects on Offshore Robotics for Certification of Assets (ORCA) [EP/R026173/1] and End-to-End Conceptual Guarding of Neural Architectures [EP/T026995/1], and ORCA Partnership Resource Fund (PRF) on Towards the Accountable and Explainable Learning-enabled Autonomous Robotic Systems, as well as the UK Dstl projects on Test Coverage Metrics for Artificial Intelligence.

* Corresponding author.
E-mail addresses: xiaowei.huang@liverpool.ac.uk (X. Huang), daniel.kroening@cs.ox.ac.uk (D. Kroening), wenjie.ruan@lancaster.ac.uk (W. Ruan), jsharp1@dstl.gov.uk (J. Sharp), youcheng.sun@qub.ac.uk (Y. Sun), emese.thamo@liverpool.ac.uk (E. Thamo), min.wu@cs.ox.ac.uk (M. Wu), xinping.yi@liverpool.ac.uk (X. Yi).

https://doi.org/10.1016/j.cosrev.2020.100270
1574-0137/

1. Introduction

In the past few years, significant progress has been made in the development of deep neural networks (DNNs), which now outperform humans in several difficult tasks, such as image classification [1], natural language processing [2], and two-player games [3]. Given the prospect of a broad deployment of DNNs in a wide range of applications, concerns regarding the safety and trustworthiness of this approach have been raised [4,5]. There is significant research that aims to address these concerns, with many publications appearing in the past few years.

Whilst it is difficult to cover all related research activities, we strive to survey the past and current research efforts on making the application of DNNs safe and trustworthy. Fig. 1 visualises the rapid growth of this area: it gives the number of surveyed papers per calendar year, starting from 2008 to 2018. In total, we surveyed 202 papers.

Trust, or trustworthiness, is a general term and its definition varies in different contexts. Our definition is based on the practice that has been widely adopted in established industries, e.g., automotive and avionics. Specifically, trustworthiness is addressed predominantly within two processes: a certification process and an explanation process. The **certification** process is held before the deployment of the product to make sure that it functions correctly (and safely). During the certification process, the manufacturer needs to demonstrate to the relevant certification authority, e.g., the European Aviation Safety Agency or the Vehicle Certification Agency, that the product behaves correctly with respect to a set of high-level requirements. The **explanation** process is held whenever needed during the lifetime of the product. The user manual explains a set of expected behaviours of the product that its user may frequently experience. More importantly, an investigation can be conducted, with a formal report produced, to understand any unexpected behaviour of the product. We believe that a similar practice should be carried out when working with data-driven deep learning systems. That is, in this survey, we address trustworthiness based on the following concept:

Trustworthiness = Certification + Explanation

In other words, a user is able to trust a system if the system has been certified by a certification authority and any of its behaviour can be well explained. Moreover, we will discuss briefly in Section 8.11 our view on the impact of human–machine interactions on trust.

In this survey, we will consider the advancement of enabling techniques for both the certification and the explanation processes of DNNs. Both processes are challenging, owing to the black-box nature of DNNs and the lack of rigorous foundations.

1.1. Certification

For certification, an important low-level requirement for DNNs is that they are robustness against input perturbations. DNNs have been shown to suffer from a lack of robustness because of their susceptibility to *adversarial examples* [6] such that small modifications to an input, sometimes imperceptible to humans, can make the network unstable. Significant efforts have been

Symbols and Acronyms

List of Symbols

We provide an incomplete list of symbols that will be used in the survey.

$\mathcal{N}$	A neural network
f	Function represented by a neural network
W	Weight
b	Bias
$n_{k,l}$	lth neuron on the kth layer
$v_{k,l}$	Activation value of the lth neuron on the kth layer
ℓ	Loss function
x	Input
y	Output
η	A region around a point
Δ	A set of manipulations
$\mathcal{R}$	A set of test conditions
$\mathcal{T}$	Test suite
$\mathcal{G}$	Regularisation term
$\mathcal{X}$	The ground truth distribution of the inputs
ϵ	Error tolerance bound
L_0(-norm)	L0 norm distance metric
L_1(-norm)	L1 norm distance metric
L_2(-norm)	L2 norm distance metric
L_∞(-norm)	L infinity norm distance metric
$\mathbb{E}$	Probabilistic expectation

List of Acronyms

DNN	Deep Neural Network
AI	Artificial Intelligence
DL	Deep Learning
MILP	Mixed Integer Linear Programming
SMT	Satisfiability Modulo Theory
MC/DC	Modified Condition/Decision Coverage
B&B	Branch and Bound
ReLU	Rectified Linear Unit

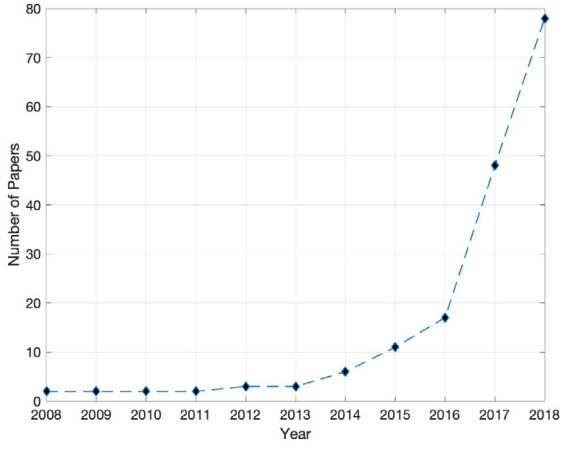


Fig. 1. Number of publications, by publication year, surveyed.

taken in order to achieve robust machine learning, including e.g., attack and defence techniques for adversarial examples. Attack techniques aim to find adversarial examples that exploit a DNN e.g., it classifies the adversarial inputs with high probability to wrong classes; meanwhile defence techniques aim to enhance the DNN so that they can identify or eliminate adversarial attack. These techniques cannot be directly applied to certify a DNN, due to their inability to provide clear assurance evidence with their results. Nevertheless, we review some prominent methods since they provide useful insights to certification techniques.

The certification techniques covered in this survey include verification and testing, both of which have been demonstrated as useful in checking the dependability of real world software and hardware systems. However, traditional techniques developed in these two areas, see e.g., [7,8], cannot be directly applied to deep learning systems, as DNNs exhibit complex internal behaviours not commonly seen within traditional verification and testing.

DNN verification techniques determine whether a property, e.g., the local robustness for a given input x, holds for a DNN; if it holds, it may be feasible to supplement the answer with a mathematical proof. Otherwise, such verification techniques will return a counterexample. If a deterministic answer is hard to achieve, an answer with certain error tolerance bounds may suffice in many practical scenarios. Whilst DNN verification techniques are promising, they suffer from a scalability problem, due to the high computational complexity and the large size of DNNs. Up to now, DNN verification techniques either work with small scale DNNs or with approximate methods with convergence guarantees on the bounds.

DNN testing techniques arise as a complement to the DNN verification techniques. Instead of providing mathematical proofs to the satisfiability of a property over the system, testing techniques aim to either find bugs (i.e., counterexamples to a property) or provide assurance cases [9], by exercising the system with a large set of test cases. These testing techniques are computationally less expensive and therefore are able to work with state-of-the-art DNNs. In particular, coverage-guided testing generates test cases according to pre-specified coverage criteria. Intuitively, a high coverage suggests that more of a DNN's behaviour has been tested and therefore the DNN has a lower chance of containing undetected bugs.

1.2. Explanation

The goal of Explainable AI [10] is to overcome the interpretability problem of AI [11]; that is, to explain why the AI outputs a specific decision, say to in determining whether to give a loan someone. The EU General Data Protection Regulation (GDPR) [12] mandates a "right to explanation", meaning that an explanation of how the model reached its decision can be requested. While this "explainability" request is definitely beneficial to the end consumer, obtaining such information from a DNN is challenging for the very developers of the DNNs.

1.3. Organisation of this survey

The structure of this survey is summarised as follows. In Section 2, we will present preliminaries on the DNNs, and a few key concepts such as verification, testing, interpretability, and distance metrics. This is followed by Section 3, which discusses safety problems and safety properties. In Sections 4 and 5, we review DNN verification and DNN testing techniques, respectively. The attack and defence techniques are reviewed in Section 6; and is followed by a review of a set of interpretability techniques

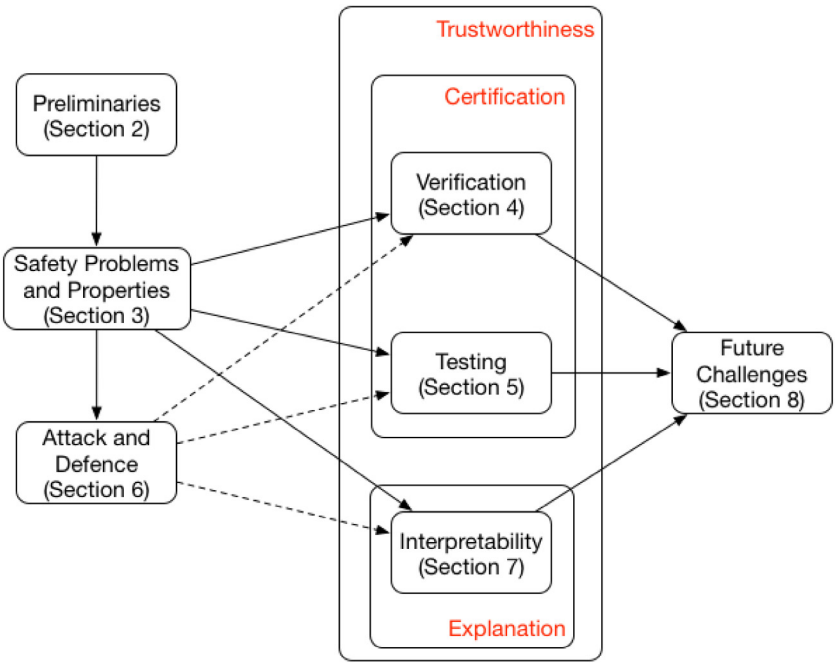


Fig. 2. Relationship between sections.

for DNNs in Section 7. Finally, we discuss future challenges in Section 8 and conclude this survey in Section 9.

Fig. 2 outlines the causality relation between the sections in this paper. We use dashed arrows from attack and defence techniques (Section 6) to several sections because of their potential application in certification and explanation techniques.

2. Preliminaries

In the following, we provide preliminaries on deep neural networks, automated verification, software testing, interpretability, and distance metrics.

2.1. Deep neural networks

A (deep and feedforward) neural network, or DNN, is a tuple $\mathcal{N} = (\mathbb{S}, \mathbb{T}, \Phi)$, where $\mathbb{S} = \{\mathbb{S}_k \mid k \in \{1..K\}\}$ is a set of layers, $\mathbb{T} \subseteq \mathbb{S} \times \mathbb{S}$ is a set of connections between layers and $\Phi = \{\phi_k \mid k \in \{2..K\}\}$ is a set of functions, one for each non-input layer. In a DNN, $\mathbb{S}_1$ is the *input* layer, $\mathbb{S}_K$ is the *output* layer, and layers other than input and output layers are called *hidden layers*. Each layer $\mathbb{S}_k$ consists of s_k *neurons* (or nodes). The lth node of layer k is denoted by $n_{k,l}$.

Each node $n_{k,l}$ for $2 \leq k \leq K$ and $1 \leq l \leq s_k$ is associated with two variables $u_{k,l}$ and $v_{k,l}$, to record its values before and after an activation function, respectively. The Rectified Linear Unit (ReLU) [13] is one of the most popular activation functions for DNNs, according to which the *activation value* of each node of hidden layers is defined as

$$v_{k,l} = ReLU(u_{k,l}) = \begin{cases} u_{k,l} & \text{if } u_{k,l} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Each input node $n_{1,l}$ for $1 \leq l \leq s_1$ is associated with a variable $v_{1,l}$ and each output node $n_{K,l}$ for $1 \leq l \leq s_K$ is associated with a variable $u_{K,l}$, because no activation function is applied on them. Other popular activation functions beside ReLU include: sigmoid, tanh, and softmax.

Except for the nodes at the input layer, every node is connected to nodes in the preceding layer by pre-trained parameters such that for all k and l with $2 \leq k \leq K$ and $1 \leq l \leq s_k$

$$u_{k,l} = b_{k,l} + \sum_{1 \leq h \leq s_{k-1}} w_{k-1,h,l} \cdot v_{k-1,h} \tag{2}$$

where $w_{k-1,h,l}$ is the weight for the connection between $n_{k-1,h}$ (i.e., the hth node of layer $k-1$) and $n_{k,l}$ (i.e., the lth node of layer k), and $b_{k,l}$ the so-called *bias* for node $n_{k,l}$. We note that this definition can express both fully-connected functions and convolutional functions.[1] The function ϕ_k is the composition of Eqs. (1) and (2) by having $u_{k,l}$ for $1 \leq l \leq s_k$ as the intermediate variables. Owing to the use of the ReLU as in (1), the behaviour of a neural network is highly non-linear.

Let $\mathbb{R}$ be the set of real numbers. We let $D_k = \mathbb{R}^{s_k}$ be the vector space associated with layer $\mathbb{S}_k$, one dimension for each variable $v_{k,l}$. Notably, every point $x \in D_1$ is an input. Without loss of generality, the dimensions of an input are normalised as real values in $[0, 1]$, i.e., $D_1 = [0, 1]^{s_1}$. A DNN $\mathcal{N}$ can alternatively be expressed as a function $f : D_1 \to D_K$ such that

$$f(x) = \phi_K(\phi_{K-1}(...\phi_2(x))) \tag{3}$$

Finally, for any input, the DNN $\mathcal{N}$ assigns a *label*, that is, the index of the node of output layer with the largest value:

$$label = \text{argmax}_{1 \leq l \leq s_K} u_{K,l} \tag{4}$$

Moreover, we let $\mathcal{L} = \{1..s_K\}$ be the set of labels.

Example 1. Fig. 3 is a simple DNN with four layers. The input space is $D_1 = [0, 1]^2$, the two hidden vector spaces are $D_2 = D_3 = \mathbb{R}^3$, and the set of labels is $\mathcal{L} = \{1, 2\}$.

Given one particular input x, the DNN $\mathcal{N}$ is *instantiated* and we use $\mathcal{N}[x]$ to denote this instance of the network. In $\mathcal{N}[x]$, for each node $n_{k,l}$, the values of the variables $u_{k,l}$ and $v_{k,l}$ are fixed and

[1] Many of the surveyed techniques can work with other types of functional layers such as max-pooling, batch-normalisation, etc. Here for simplicity, we omit their expressions.

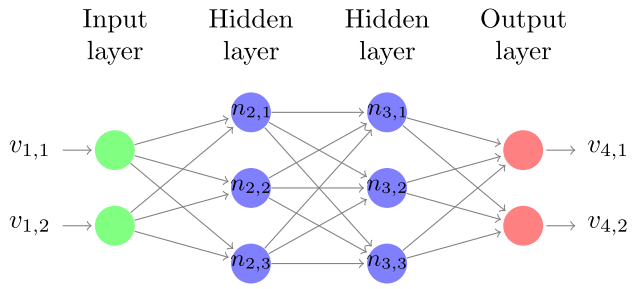


Fig. 3. A simple neural network.

denoted as $u_{k,l}[x]$ and $v_{k,l}[x]$, respectively. Thus, the activation or deactivation of each ReLU operation in the network is similarly determined. We define

$$sign_{\mathcal{N}}(n_{k,l}, x) = \begin{cases} +1 & \text{if } u_{k,l}[x] = v_{k,l}[x] \\ -1 & \text{otherwise} \end{cases} \tag{5}$$

The subscript $\mathcal{N}$ will be omitted when clear from the context. The classification label of x is denoted as $\mathcal{N}[x].label$.

Example 2. Let $\mathcal{N}$ be a DNN whose architecture is given in Fig. 3. Assume that the weights for the first three layers are as follows:

$$W_1 = \begin{bmatrix} 4 & 0 & -1 \\ 1 & -2 & 1 \end{bmatrix}, \quad W_2 = \begin{bmatrix} 2 & 3 & -1 \\ -7 & 6 & 4 \\ 1 & -5 & 9 \end{bmatrix}$$

and that all biases are 0. When given an input $x = [0, 1]$, we get $sign(n_{2,1}, x) = +1$, since $u_{2,1}[x] = v_{2,1}[x] = 1$, and $sign(n_{2,2}, x) = -1$, since $u_{2,2}[x] = -2 \neq 0 = v_{2,2}[x]$.

2.2. Verification

Given a DNN $\mathcal{N}$ and a property C, verification is considered a set of techniques to check whether the property C holds on $\mathcal{N}$. Different from other techniques such as testing and adversarial attack, a verification technique needs to provide provable guarantees for its results. A guarantee can be either a Boolean guarantee, an approximate guarantee, an anytime approximate guarantee, or a statistical guarantee. A Boolean guarantee means that the verification technique is able to affirm when the property holds, or otherwise return a counterexample. An approximate guarantee provides either an over-approximation or an under-approximation to a property. An anytime approximate guarantee has both an over-approximation and an under-approximation, and the approximations can be continuously improved until converged. All the above guarantees need to be supported with a mathematical proof. When a mathematical proof is hard to achieve, a statistical guarantee provides a quantitative error tolerance bound on the resulting claim.

We will formally define safety properties for DNNs in Section 3, together with their associated verification problems and provable guarantees.

2.3. Testing

Verification problems usually have high computational complexity, such as being NP-hard, when the properties are simple input–output constraints [14,15]. This, compounded with the high-dimensionality and the high non-linearity of DNNs, makes the existing verification techniques hard to work with for industrial scale DNNs. This computational intensity can be partially alleviated by considering testing techniques, at the price of provable guarantees. Instead, assurance cases are pursued in line with those of existing safety critical systems [9].

The goal of testing DNNs is to generate a set of test cases, that can demonstrate confidence in a DNN's performance, when passed, such that they can support an assurance case. Usually, the generation of test cases is guided by coverage metrics.

Let N be a set of DNNs, R a set of test objective sets, and T a set of test suites. We use $\mathcal{N}, \mathcal{R}, \mathcal{T}$ to range over N, R and T, respectively. Note that, normally both $\mathcal{R}$ and $\mathcal{T}$ contain a set of elements by themselves. The following is an adaption of a definition in [16] for software testing:

Definition 1 (*Test Accuracy Criterion/Test Coverage Metric*)**.** A test adequacy criterion, or a test coverage metric, is a function $M : \mathsf{N} \times \mathsf{R} \times \mathsf{T} \to [0, 1]$.

Intuitively, $M(\mathcal{N}, \mathcal{R}, \mathcal{T})$ quantifies the degree of adequacy to which a DNN $\mathcal{N}$ is tested by a test suite $\mathcal{T}$ with respect to a set $\mathcal{R}$ of test objectives. Usually, the greater the number $M(\mathcal{N}, \mathcal{R}, \mathcal{T})$, the more adequate the testing.[2] We will elaborate on the test criteria in Section 5. Moreover, a testing oracle is a mechanism that determines whether the DNN behaves correctly for a test case. It is dependent on the properties to be tested, and will be discussed further in Section 3.

2.4. Interpretability

Interpretability is an issue arising as a result of the black-box nature of DNNs. Intuitively, it should provide a human-understandable explanation for the behaviour of a DNN. An explanation procedure can be separated into two steps: an extraction step and an exhibition step. The *extraction step* obtains an intermediate representation, and the *exhibition step* presents the obtained intermediate representation in a way easy for human users to understand. Given the fact that DNNs are usually high-dimensional, and the information should be made accessible for understanding by the human users, the intermediate representation needs to be at a lower dimensionality. Since the exhibition step is closely related to the intermediate representation, and is usually conducted by e.g., visualising the representation, we will focus on the extraction step in this survey.

Depending on the requirements, the explanation can be either an instance-wise explanation, a model explanation, or an information-flow explanation. In the following, we give their general definitions, trying to cover as many techniques to be reviewed as possible.

Definition 2 (*Instance-Wise Explanation*)**.** Given a function $f : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, which represents a DNN $\mathcal{N}$, and an input $x \in \mathbb{R}^{s_1}$, an instance-wise explanation $\texttt{expl}(f, x) \in \mathbb{R}^t$ is another representation of x such that $t \leq s_1$.

Intuitively, for instance-wise explanation, the goal is to find another representation of an input x (with respect to the function f associated to the DNN $\mathcal{N}$), with the expectation that the representation carries simple, yet essential, information that can help the user understand the decision $f(x)$. Most of the techniques surveyed in Sections 7.1, 7.2, and 7.3 fit with this definition.

Definition 3 (*Model Explanation*)**.** Given a function $f : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, which represents a DNN $\mathcal{N}$, a model explanation $\texttt{expl}(f)$ includes two functions $g_1 : \mathbb{R}^{a_1} \to \mathbb{R}^{a_2}$, which is a representation of f such that $a_1 \leq s_1$ and $a_2 \leq s_K$, and $g_2 : \mathbb{R}^{s_1} \to \mathbb{R}^{a_1}$, which maps original inputs to valid inputs of the function g_1.

[2] We may use criterion and metric interchangeably.

Intuitively, the point of model explanation is to find a simpler model which can not only be used for prediction by applying $g_1(g_2(x))$ (with certain loss) but also be comprehended by the user. Most of the techniques surveyed in Section 7.5 fit with this definition. There are other model explanations such as the influence function based approach reviewed in Section 7.4, which provides explanation by comparing different learned parameters by e.g., up-weighting some training samples.

Besides the above two deterministic methods for the explanation of data and models, there is another stochastic explanation method for the explanation of information flow in the DNN training process.

Definition 4 (*Information-Flow Explanation*)**.** Given a function family $\mathcal{F}$, which represents a stochastic DNN, an information-flow explanation $\texttt{expl}(\mathcal{F})$ includes a stochastic encoder $g_1(T_k|X)$, which maps the input X to a representation T_k at layer k, and a stochastic decoder $g_2(Y|T_k)$, which maps the representation T_k to the output Y.

The aim of the information-flow explanation is to find the optimal information representation of the output at each layer when information (data) flow goes through, and understand why and how a function $f \in \mathcal{F}$ is chosen as the training outcome given the training dataset (X, Y). The information is transparent to data and models, and its representations can be described by some quantities in information theory, such as entropy and mutual information. This is an emerging research avenue for interpretability, and a few information theoretical approaches will be reviewed in Section 7.6, which aim to provide a theoretical explanation to the training procedure.

2.5. Distance metric and d-neighbourhood

Usually, a distance function is employed to compare inputs. Ideally, such a distance should reflect perceptual similarity between inputs, comparable to e.g., human perception for image classification networks. A distance metric should satisfy a few axioms which are usually needed for defining a metric space:

- $\|x\| \geq 0$ (non-negativity),
- $\|x - y\| = 0$ implies that $x = y$ (identity of indiscernibles),
- $\|x - y\| = \|y - x\|$ (symmetry),
- $\|x - y\| + \|y - z\| \geq \|x - z\|$ (triangle inequality).

In practise, L_p-norm distances are used, including

- L_1 (Manhattan distance): $\|x\|_1 = \sum_{i=1}^{n} |x_i|$
- L_2 (Euclidean distance): $\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}$
- L_∞ (Chebyshev distance): $\|x\|_\infty = \max_i |x_i|$

Moreover, we also consider L_0-norm as $\|x\|_0 = |\{x_i \mid x_i \neq 0, i = 1..n\}|$, i.e., the number of non-zero elements. Note that, L_0-norm does not satisfy the triangle inequality. In addition to these, there exist other distance metrics such as Fréchet Inception Distance [17].

Given an input x and a distance metric L_p, the *neighbourhood* of x is defined as follows.

Definition 5 (*d-Neighbourhood*)**.** Given an input x, a distance function L_p, and a distance d, we define the *d-neighbourhood* $\eta(x, L_p, d)$ of x w.r.t. L_p as

$$\eta(x, L_p, d) = \{\hat{x} \mid \|\hat{x} - x\|_p \leq d\}, \tag{6}$$

the set of inputs whose distance to x is no greater than d with respect to L_p.

3. Safety problems and safety properties

Despite the success of deep learning (DL) in many areas, serious concerns have been raised in applying DNNs to real-world safety-critical systems such as self-driving cars, automatic medical diagnosis, etc. In this section, we will discuss the key safety problem of DNNs, and present a set of safety features that analysis techniques are employing.

For any $f(x)$ whose value is a vector of scalar numbers, we use $f_j(x)$ to denote its jth element.

Definition 6 (*Erroneous Behaviour of DNNs*)**.** Given a (trained) deep neural network $f : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, a human decision oracle $\mathcal{H} : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, and a legitimate input $x \in \mathbb{R}^{s_1}$, an erroneous behaviour of the DNN is such that

$$\arg\max_j f_j(x) \neq \arg\max_j \mathcal{H}_j(x) \tag{7}$$

Intuitively, an erroneous behaviour is witnessed by the existence of an input x on which the DNN and a human user have different perception.

3.1. Adversarial examples

Erroneous behaviours include those training and test samples which are classified incorrectly by the model and have safety implications. Adversarial examples [6] represent another class of erroneous behaviours that also introduce safety implications. Here, we take the name "adversarial example" due to historical reasons. Actually, as suggested in the below definition, it represents a mis-match of the decisions made by a human and by a neural network, and does not necessarily involve an adversary.

Definition 7 (*Adversarial Example*)**.** Given a (trained) deep neural network $f : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, a human decision oracle $\mathcal{H} : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, and a legitimate input $x \in \mathbb{R}^{s_1}$ with $\arg\max_j f_j(x) = \arg\max_j \mathcal{H}_j(x)$, an adversarial example to a DNN is defined as:

$$\begin{aligned} \exists \hat{x} : \quad & \arg\max_j \mathcal{H}_j(\hat{x}) = \arg\max_j \mathcal{H}_j(x) \\ & \wedge \ \|x - \hat{x}\|_p \leq d \\ & \wedge \ \arg\max_j f_j(\hat{x}) \neq \arg\max_j f_j(x) \end{aligned} \tag{8}$$

where $p \in \mathbb{N}$, $p \geq 1$, $d \in \mathbb{R}$, and $d > 0$.

Intuitively, x is an input on which the DNN and an human user have the same classification and, based on this, an adversarial example is another input $\hat{x}$ that is classified differently than x by network f (i.e., $\arg\max_j f_j(\hat{x}) \neq \arg\max_j f_j(x)$), even when they are geographically close in the input space (i.e., $\|x - \hat{x}\|_p \leq d$) and the human user believes that they should be the same (i.e., $\arg\max_j \mathcal{H}_j(\hat{x}) = \arg\max_j \mathcal{H}_j(x)$). We do not consider the labelling error introduced by human operators, because it is part of the Bayes error which is irreducible for a given classification problem. On the other hand, the approaches we review in the paper are for the estimation error, which measures how far the learned network $\mathcal{N}$ is from the best network of the same architecture.

Fig. 4 shows three concrete examples of the safety concerns brought by the potential use of DNNs in safety-critical application scenarios including: medical diagnosis systems, self-driving cars and automated sentiment analysis of medical records.

Example 3. As shown in the top row of Fig. 4, for an fMRI image, a human-invisible perturbation will turn a DL-enabled diagnosis decision of "malignant tumour" into "benign tumour". In this case, the human oracle is the medical expert [18].

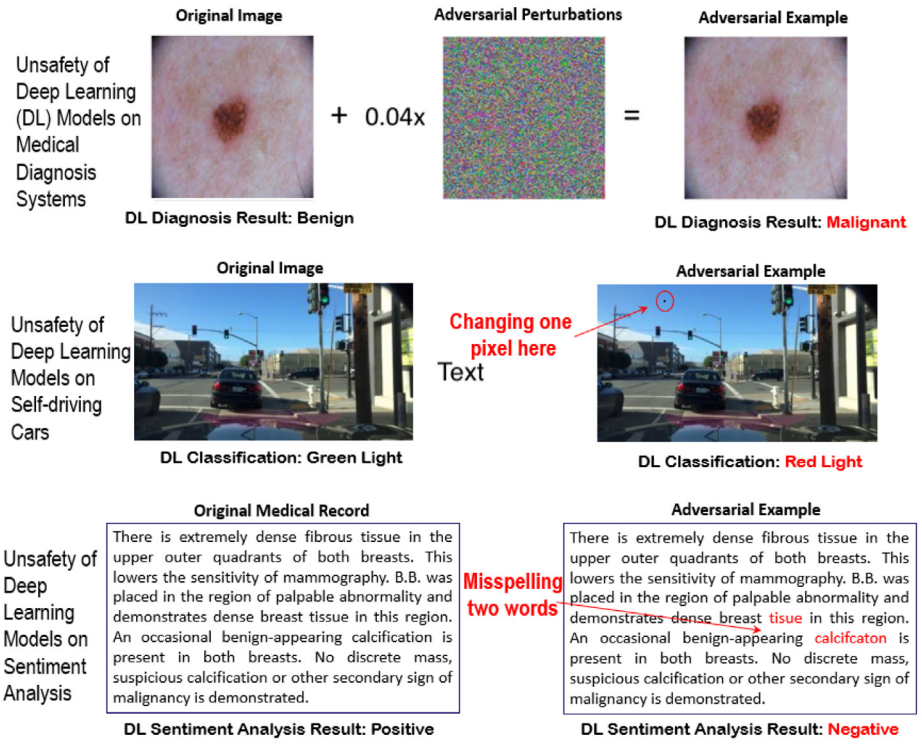


Fig. 4. Examples of erroneous behaviour on deep learning models. Top Row [18]: In a medical diagnosis system, a "Benign" tumour is misclassified as "Malignant" after adding a small amount of human-imperceptible perturbations; Second Row [19]: By just changing one pixel in a "Green-Light" image, a state-of-the-art DNN misclassifies it as "Red-Light"; Bottom Row [20]: In a sentiment analysis task for medical records, with two misspelt words, a well-trained deep learning model classifies a "Positive" medical record as "Negative".

Example 4. As shown in the second row of Fig. 4, in classification tasks, by adding a small amount of adversarial perturbation (w.r.t. L_p-norm distance), the DNNs will misclassify an image of traffic sign "red light" into "green light" [19,21]. In this case, the human decision oracle $\mathcal{H}$ is approximated by stating that two inputs within a very small L_p-norm distance are the same.

Example 5. In a DL-enabled end-to-end controller deployed in autonomous vehicles, by adding some natural transformations such as "rain", the controller will output an erroneous decision, "turning left", instead of a righteous decision, "turning right" [22]. However, it is clear that, from the human driver's point of view, adding "rain" should not change the driving decision of a car.

Example 6. As shown in the bottom row of Fig. 4, for medical record, some minor misspellings – which happen very often in the medical records – will lead to significant mis-classification on the diagnosis result, from "Positive" to "Negative".

As we can see, these unsafe, or erroneous, phenomenon acting on DNNs are essentially caused by the inconsistency of the decision boundaries from DL models (that are learned from training datasets) and human oracles. This inevitably raises significant concerns on whether DL models can be safely applied in safety-critical domains.

In the following, we review a few safety properties that have been studied in the literature.

3.2. Local robustness property

Robustness requires that the decision of a DNN is invariant against small perturbations. The following definition is adapted from that of Huang et al. [23].

Definition 8 (*Local Robustness*)**.** Given a DNN $\mathcal{N}$ with its associated function f, and an input region $\eta \subseteq [0,1]^{s_1}$, the (untargeted) local robustness of f on η is defined as

$$Robust(f,\eta) \triangleq \forall x \in \eta, \exists\, l \in [1..s_K], \forall j \in [1..s_K] : f_l(x) \geq f_j(x) \quad (9)$$

For targeted local robustness of a label j, it is defined as

$$Robust_j(f,\eta) \triangleq \forall x \in \eta, \exists\, l \in [1..s_K] : f_l(x) > f_j(x) \quad (10)$$

Intuitively, local robustness states that all inputs in the region η have the same class label. More specifically, there exists a label l such that, for all inputs x in region η, and other labels j, the DNN believes that x is more possible to be in class l than in any class j. Moreover, targeted local robustness means that a specific label j cannot be perturbed for all inputs in η; specifically, all inputs x in η have a class $l \neq j$, which the DNN believes is more possible than the class j. Usually, the region η is defined with respect to an input x and a norm L_p, as in Definition 5. If so, it means that all inputs in η have the same class as input x. For targeted local robustness, it is required that none of the inputs in the region η is classified as a given label j.

In the following, we define a test oracle for the local robustness property. Note that, all existing testing approaches surveyed relate to local robustness, and therefore we only provide the test oracle for local robustness.

Definition 9 (*Test Oracle of Local Robustness Property*)**.** Let D be a set of correctly-labelled inputs. Given a norm distance L_p and a real number d, a test case $(x_1, \ldots, x_k) \in \mathcal{T}$ passes the test oracle's local robustness property, or oracle for simplicity, if

$$\forall 1 \leq i \leq k \exists\, x_0 \in D : x_i \in \eta(x_0, L_p, d) \quad (11)$$

Intuitively, a test case $(x_1, \ldots, x_k)$ passes the oracle if all of its components x_i are close to one of the correctly-labelled inputs, with respect to L_p and d. Recall that, we define $\eta(x_0, L_p, d)$ in Definition 5.

3.3. Output reachability property

Output reachability computes the set of outputs with respect to a given set of inputs. We follow the naming convention from [15,24,25]. Formally, we have the following definition.

Definition 10 (*Output Reachability*)**.** Given a DNN $\mathcal{N}$ with its associated function f, and an input region $\eta \subseteq [0, 1]^{s_1}$, the output reachable set of f and η is a set $Reach(f, \eta)$ such that

$$Reach(f, \eta) \triangleq \{f(x) \mid x \in \eta\} \tag{12}$$

The region η includes a large, and potentially infinite, number of inputs, so that no practical algorithm that can exhaustively check their classifications. The output reachability problem is *highly non-trivial* for this reason, and that f is highly non-linear (or black-box). Based on this, we can define the following verification problem.

Definition 11 (*Verification of Output Reachability*)**.** Given a DNN $\mathcal{N}$ with its associated function f, an input region $\eta \subseteq [0, 1]^{s_1}$, and an output region $\mathcal{Y}$, the verification of output reachability on f, η, and $\mathcal{Y}$ is to determine if

$$Reach(f, \eta) = \mathcal{Y}. \tag{13}$$

Thus, the verification of reachability determines whether all inputs in η are mapped onto a given set $\mathcal{Y}$ of outputs, and whether all outputs in $\mathcal{Y}$ have a corresponding x in η.

As a simpler question, we might be interested in computing for a specific dimension of the set $Reach(f, \eta)$, its greatest or smallest value; for example, the greatest classification confidence of a specific label. We call such a problem a reachability problem.

3.4. Interval property

The Interval property computes a convex over-approximation of the output reachable set. We follow the naming convention from interval-based approaches, which are a typical class of methods for computing this property. Formally, we have the following definition.

Definition 12 (*Interval Property*)**.** Given a DNN $\mathcal{N}$ with its associated function f, and an input region $\eta \subseteq [0, 1]^{s_1}$, the interval property of f and η is a convex set $Interval(f, \eta)$ such that

$$Interval(f, \eta) \supseteq \{f(x) \mid x \in \eta\} \tag{14}$$

Ideally, we expect this set to be a convex hull of points in $\{f(x) \mid x \in \eta\}$. A convex hull of a set of points is the smallest convex set that contains the points.

While the computation of such a set can be trivial since $[0, 1]^{s_1} \supseteq \{f(x) \mid x \in \eta\}$, it is expected that $Interval(f, \eta)$ is as close as possible to $\{f(x) \mid x \in \eta\}$, i.e., ideally it is a convex hull. Intuitively, an interval is an over-approximation of the output reachability. Based on this, we can define the following verification problem.

Definition 13 (*Verification of Interval Property*)**.** Given a DNN $\mathcal{N}$ with its associated function f, an input region $\eta \subseteq [0, 1]^{s_1}$, and an output region $\mathcal{Y}$ represented as a convex set, the verification of the interval property on f, η, and $\mathcal{Y}$ is to determine if

$$\mathcal{Y} \supseteq \{f(x) \mid x \in \eta\} \tag{15}$$

In other words, it is to determine whether the given $\mathcal{Y}$ is an interval satisfying Expression (14).

Intuitively, the verification of the interval property determine whether all inputs in η are mapped onto $\mathcal{Y}$. Similar to the reachability property, we might also be interested in simpler problems e.g., determining whether a given real number d is a valid upper bound for a specific dimension of $\{f(x) \mid x \in \eta\}$.

3.5. Lipschitzian property

The Lipschitzian property, inspired by the Lipschitz continuity (see textbooks such as OSearcoid [26]), monitors the changes of the output with respect to small changes of the inputs.

Definition 14 (*Lipschitzian Property*)**.** Given a DNN $\mathcal{N}$ with its associated function f, an input region $\eta \subseteq [0, 1]^{s_1}$, and the L_p-norm,

$$Lips(f, \eta, L_p) \equiv \sup_{x_1, x_2 \in \eta} \frac{|f(x_1) - f(x_2)|}{\|x_1 - x_2\|_p} \tag{16}$$

is a Lipschitzian metric of f, η, and L_p.

Intuitively, the value of this metric is the best Lipschitz constant. Therefore, we have the following verification problem.

Definition 15 (*Verification of Lipschitzian Property*)**.** Given a Lipschitzian metric $Lips(f, \eta, L_p)$ and a real value $d \in \mathbb{R}$, it must be determined whether

$$Lips(f, \eta, L_p) \leq d. \tag{17}$$

3.6. Relationship between properties

Fig. 5 gives the relationship between the four properties discussed above. An arrow from a value A to another value B represents the existence of a simple computation to enable the computation of B based on A. For example, given a Lipschitzian metric $Lips(f, \eta, L_p)$ and $\eta = \eta(x, L_p, d)$, we can compute an interval

$$Interval(f, \eta) = [f(x) - Lips(f, \eta, L_p) \cdot d, f(x) + Lips(f, \eta, L_p) \cdot d] \tag{18}$$

It can be verified that $Interval(f, \eta) \supseteq \{f(x) \mid x \in \eta\}$. Given an interval $Interval(f, \eta)$ or a reachable set $Reach(f, \eta)$, we can check their respective robustness by determining the following expressions:

$$Interval(f, \eta) \subseteq \mathcal{Y}_l = \{y \in \mathbb{R}^{s_K} \mid \forall j \neq l : y_l \geq y_j\}, \text{ for some } l \tag{19}$$

$$Reach(f, \eta) \subseteq \mathcal{Y}_l = \{y \in \mathbb{R}^{s_K} \mid \forall j \neq l : y_l \geq y_j\}, \text{ for some } l \tag{20}$$

where y_l is the l-entry of the output vector y. The relation between $Reach(f, \eta)$ and $Interval(f, \eta)$ is an implication relation, which can be seen from their definitions. Actually, if we can compute precisely the set $Reach(f, \eta)$, the inclusion of the set in another convex set $Interval(f, \eta)$ can be computed easily.

Moreover, we use a dashed arrow between $Lips(f, \eta, L_p)$ and $Reach(f, \eta)$, as the computation is more involved by e.g., algorithms from [15,21,27].

3.7. Instancewise interpretability

First, we need to have a ranking among explanations for a given input.

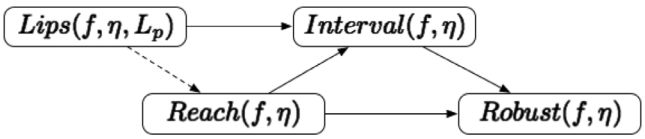


Fig. 5. Relationship between properties. An arrow from a value A to another value B represents the existence of a simple computation to enable the computation of B based on A. The dashed arrow between $Lips(f, \eta, L_p)$ and $Reach(f, \eta)$ means that the computation is more involved.

Definition 16 (*Human Ranking of Explanations*)**.** Let $\mathcal{N}$ be a network with associated function f and $\mathcal{E} \subseteq \mathbb{R}^t$ be the set of possible explanations. We define an evaluation function $eval_{\mathcal{H}} : \mathbb{R}^{s_1} \times \mathcal{E} \rightarrow [0, 1]$, which assigns for each input $x \in \mathbb{R}^{s_1}$ and each explanation $e \in \mathcal{E}$ a probability value $eval_{\mathcal{H}}(x, e)$ in $[0, 1]$ such that a higher value suggests a better explanation of e over x.

Intuitively, $eval_{\mathcal{H}}(x, e)$ is a ranking of the explanation e by human users, when given an input x. For example, given an image and an explanation algorithm which highlights part of an image, human users are able to rank all the highlighted images. While this ranking can be seen as the ground truth for the instance-wise interpretability, similar to using distance metrics to measure human perception, it is hard to approximate. Based on this, we have the following definition.

Definition 17 (*Validity of Explanation*)**.** Let f be the associated function of a DNN $\mathcal{N}$, x an input, and $\epsilon > 0$ a real number, $\texttt{expl}(f, x) \in \mathcal{E} \subseteq \mathbb{R}^t$ is a valid instance-wise explanation if

$$eval_{\mathcal{H}}(x, \texttt{expl}(f, x)) > 1 - \epsilon. \tag{21}$$

Intuitively, an explanation is valid if it should be among the set of explanations that are ranked sufficiently high by human users.

4. Verification

In this section, we review verification techniques for neural networks. Existing approaches on the verification of networks largely fall into the following categories: *constraint solving*, *search based approach*, *global optimisation*, and *over-approximation*; note, the separation between them may not be strict. Fig. 6 classifies some of the approaches surveyed in this paper into these categories.

In this survey, we take a different approach by classifying verification techniques with respect to the type of guarantees they can provide; these guarantee types can be:

- *Exact deterministic* guarantee, which states exactly whether a property holds. We will omit the word *exact* and call it deterministic guarantee in the remainder of the paper.
- *One-sided* guarantee, which provides either a lower bound or an upper bound to a variable, and thus can serve as a sufficient condition for a property to hold – the variable can denote, e.g., the greatest value of some dimension in $Reach(f, \eta)$.
- Guarantee with *converging* lower and upper bounds to a variable.
- *Statistical* guarantee, which quantifies the probability that a property holds.

Note that, algorithms with one-sided guarantee and bound-converging guarantee are used to compute the real values, e.g., output reachability property (Definition 10), interval property (Definition 12), or Lipschitzian property (Definition 14). Their respective verification problems are based on these values, see Definitions 11, 13, and 15.

4.1. Approaches with deterministic guarantees

Deterministic guarantees are achieved by transforming a verification problem into a set of constraints (with or without optimisation objectives) so that they can be solved with a constraint solver. The name "deterministic" comes from the fact that solvers often return a deterministic answer to a query, i.e., either satisfiable or unsatisfiable. This is based on the current success of various constraint solvers such as SAT solvers, linear programming (LP) solvers, mixed integer linear programming (MILP) solvers, Satisfiability Modulo Theories (SMT) solvers.

4.1.1. SMT/SAT

The Boolean satisfiability problem (SAT) determines if, given a Boolean formula, there exists an assignment to the Boolean variables such that the formula is satisfiable. Based on SAT, the satisfiability modulo theories (SMT) problem determines the satisfiability of logical formulae with respect to combinations of background theories expressed in classical first-order logic with equality. The theories we consider in the context of neural networks include the theory of real numbers and the theory of integers. For both SAT and SMT problems, there are sophisticated, open-source solvers that can automatically answer the queries about the satisfiability of the formulae.

An abstraction-refinement approach based on SMT solving. A solution to the verification of the interval property (which can be easily extended to work with the reachability property for ReLU activation functions) is proposed in [28] by abstracting a network into a set of Boolean combinations of linear arithmetic constraints. Basically, the linear transformations between layers can be encoded directly, and the non-linear activation functions such as Sigmoid are approximated – with both lower and upper bounds – by piece-wise linear functions. It is shown that whenever the abstracted model is declared to be safe, the same holds for the concrete model. Spurious counterexamples, on the other hand, trigger refinements and can be leveraged to automate the correction of misbehaviour. This approach is validated on neural networks with fewer than 10 neurons, with logistic activation function.

SMT solvers for neural networks. Two SMT solvers Reluplex [14] and Planet [29] were put forward to verify neural networks on properties expressible with SMT constraints. SMT solvers often have good performance on problems that can be represented as a Boolean combination of constraints over other variable types. Typically, an SMT solver combines a SAT solver with specialised decision procedures for other theories. In the verification of networks, they adapt linear arithmetic over real numbers, in which an atom (i.e., the most basic expression) is of the form $\sum_{i=1}^{n} a_i x_i \leq b$, where a_i and b are real numbers.

In both Reluplex and Planet, they use the architecture of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm in splitting cases and ruling out conflict clauses, while they differ slightly in dealing with the intersection. For Reluplex, the approach inherits rules in the algorithm of Simplex and adds some rules for the ReLU operation. Through the classical pivot operation, it first looks for a solution for the linear constraints, and then applies the rules for ReLU to satisfy the ReLU relation for every node. Conversely, Planet uses linear approximation to over-approximate the neural network, and manages the condition of ReLU and max-pooling nodes with a logic formula.

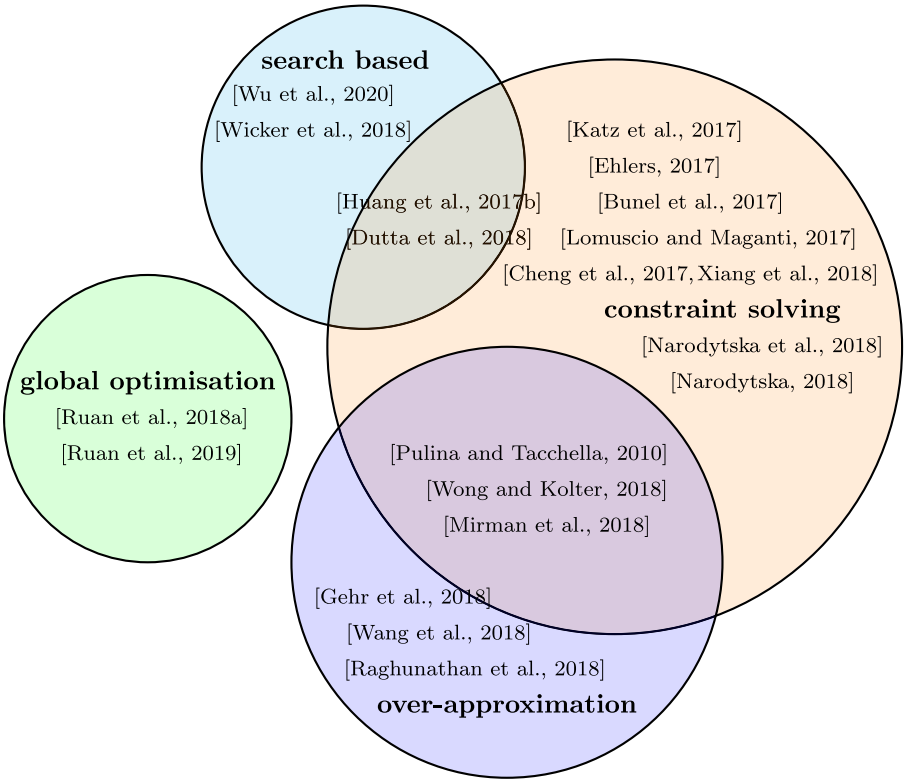


Fig. 6. A taxonomy of verification approaches for neural networks. The classification is based on the (surveyed) core algorithms for solving the verification problem. *Search based approaches* suggest that the verification algorithm is based on exhaustive search. *Constraint solving* methods suggest that the neural network is encoded into a set of constraints, with the verification problem reduced into a constraint solving problem. The *over-approximation* methods can guarantee the soundness of the result, but not the completeness. *Global optimisation* methods suggest that the core verification algorithms are inspired by global optimisation techniques.

SAT approach. Narodytska et al. [30] and Narodytska [31] propose to verify properties of a class of neural networks (i.e., binarised neural networks) in which both weights and activations are binary, by reduction to the well-known Boolean satisfiability. Using this Boolean encoding, they leverage the power of modern SAT solvers, along with a proposed counterexample-guided search procedure, to verify various properties of these networks. A particular focus is on the robustness to adversarial perturbations. The experimental results demonstrate that this approach scales to medium-size DNN used in image classification tasks.

4.1.2. Mixed integer linear programming (MILP)

Linear programming (LP) is a technique for optimising a linear objective function, subject to linear equality and linear inequality constraints. All the variables in an LP are real, if some of the variables are integers, the problem becomes a mixed integer linear programming (MILP) problem. It is noted that, while LP can be solved in polynomial time, MILP is NP-hard.

MILP formulation for neural networks. Lomuscio and Maganti [32] encodes the behaviours of fully connected neural networks with MILP. For instance, a hidden layer $z_{i+1} = \text{ReLU}(W_i z_i + b_i)$ can be described with the following MILP:

$$
\begin{aligned}
&z_{i+1} \geq W_i z_i + b_i,\\
&z_{i+1} \leq W_i z_i + b_i + M t_{i+1},\\
&z_{i+1} \geq 0,\\
&z_{i+1} \leq M(1 - t_{i+1}),
\end{aligned}
$$

where t_{i+1} has value 0 or 1 in its entries and has the same dimension as z_{i+1}, and $M > 0$ is a large constant which can be treated as ∞. Here each integer variable in t_{i+1} expresses the possibility that a neuron is activated or not. The optimisation objective can be used to express properties related to the bounds, such as $\|z_1 - x\|_\infty$, which expresses the L_∞ distance of z_1 to some given input x. This approach can work with both the reachability property and the interval property.

However, it is not efficient to simply use MILP to verify networks, or to compute the output range. In [33], a number of MILP encoding heuristics are developed to speed up the solving process, and moreover, parallelisation of MILP-solvers is used, resulting in an almost linear speed-up in the number (up to a certain limit) of computing cores in experiments. In [34], Sherlock alternately conducts a local and global search to efficiently calculate the output range. In a local search phase, Sherlock uses gradient descent method to find a local maximum (or minimum), while in a global search phase, it encodes the problem with MILP to check whether the local maximum (or minimum) is the global output range.

Additionally, Bunel et al. [35] presents a branch and bound (B&B) algorithm and claims that both SAT/SMT-based and MILP-based approaches can be regarded as its special cases.

4.2. Approaches to compute an approximate bound

The approaches surveyed in this subsection consider the computation of a *lower* (or by duality, an *upper*) bound, and are able to claim the sufficiency of achieving properties. Whilst these approaches can only have a bounded estimation to the value of some variable, they are able to work with larger models, e.g., up to 10,000 hidden neurons. Another advantage is their potential to avoid floating point issues in existing constraint solver implementations. Actually, most state-of-the-art constraint solvers implementing floating-point arithmetic only give approximate solutions, which may not be the actual optimal solution or may even lie outside the feasible space [36]. Indeed, it may happen

that a solver wrongly claims the satisfiability or un-satisfiability of a property. For example, Dutta et al. [34] reports several false positive results in Reluplex, and mentions that this may come from an unsound floating point implementation.

4.2.1. Abstract interpretation

Abstract interpretation is a theory of sound approximation of the semantics of computer programs [37]. It has been used in static analysis to verify properties of a program without it actually being run. The basic idea of abstract interpretation is to use abstract domains (represented as e.g., boxes, zonotopes, and polyhedra) to over-approximate the computation of a set of inputs; its application has been explored in a few approaches, including AI^2 [38,39] and [40].

Generally, on the input layer, a concrete domain $\mathcal{C}$ is defined such that the set of inputs η is one of its elements. To enable an efficient computation, a comparatively simple domain, i.e., abstract domain $\mathcal{A}$, which over-approximates the range and relation of variables in $\mathcal{C}$, is chosen. There is a partial order $\leq$ on $\mathcal{C}$ as well as $\mathcal{A}$, which is the subset relation $\subseteq$.

Definition 18. A pair of functions $\alpha : \mathcal{C} \to \mathcal{A}$ and $\gamma : \mathcal{A} \to \mathcal{C}$ is a *Galois connection*, if for any $a \in \mathcal{A}$ and $c \in \mathcal{C}$, we have $\alpha(c) \leq a \Leftrightarrow c \leq \gamma(a)$.

Intuitively, a Galois connection (α, γ) expresses abstraction and concretisation relations between domains, respectively. A Galois connection is chosen because it preserves the order of elements in two domains. Note that, $a \in \mathcal{A}$ is a sound abstraction of $c \in \mathcal{C}$ if and only if $\alpha(c) \leq a$.

In abstract interpretation, it is important to choose a suitable abstract domain because it determines the efficiency and precision of the abstract interpretation. In practice, a certain type of special shapes is used as the abstraction elements. Formally, an abstract domain consists of shapes expressible as a set of logical constraints. The most popular abstract domains for the Euclidean space abstraction include Interval, Zonotope, and Polyhedron; these are detailed below.

- **Interval.** An *interval* I contains logical constraints in the form of $a \leq x_i \leq b$, and for each variable x_i, I contains at most one constraint with x_i.
- **Zonotope.** A *zonotope* Z consists of constraints in the form of $z_i = a_i + \sum_{j=1}^{m} b_{ij}\epsilon_j$, where a_i, b_{ij} are real constants and $\epsilon_j \in [l_j, u_j]$. The conjunction of these constraints expresses a centre-symmetric polyhedron in the Euclidean space.
- **Polyhedron.** A *polyhedron* P has constraints in the form of linear inequalities, i.e., $\sum_{i=1}^{n} a_i x_i \leq b$, and it gives a closed convex polyhedron in the Euclidean space.

Example 7. Let $\bar{x} \in \mathbb{R}^2$. Assume that the range of $\bar{x}$ is a discrete set $X = \{(1, 0), (0, 2), (1, 2), (2, 1)\}$. We can have abstraction of the input X with Interval, Zonotope, and Polyhedron as follows.

- Interval: $[0, 2] \times [0, 2]$.
- Zonotope: $\{x_1 = 1 - \frac{1}{2}\epsilon_1 - \frac{1}{2}\epsilon_2,\ \ x_2 = 1 + \frac{1}{2}\epsilon_1 + \frac{1}{2}\epsilon_3\}$, where $\epsilon_1, \epsilon_2, \epsilon_3 \in [-1, 1]$.
- Polyhedron: $\{x_2 \leq 2,\ \ x_2 \leq -x_1 + 3,\ \ x_2 \geq x_1 - 1,\ \ x_2 \geq -2x_1 + 2\}$.

The abstract interpretation based approaches can verify interval property, but cannot verify reachability property.

4.2.2. Convex optimisation based methods

Convex optimisation is to minimise convex functions over convex sets. Most neural network functions are not convex (even if it is convex, it can be a very complicated), and hence an approximation is needed for the computation based on it. A method is proposed in [41] to learn deep ReLU-based classifiers that are provably robust against norm-bounded adversarial perturbations on the training data. The approach works with interval property, but not reachability property. It may flag some non-adversarial examples as adversarial examples. The basic idea is to consider a convex outer over-approximation of the set of activations reachable through a norm-bounded perturbation, and then develop a robust optimisation procedure that minimises the worst case loss over this outer region (via a linear programme). Crucially, it is shown that the dual problem to this linear programme can be represented itself as a deep network similar to the back-propagation network, leading to very efficient optimisation approaches that produce guaranteed bounds on the robust loss. The approach is illustrated on a number of tasks with robust adversarial guarantees. For example, for MNIST, they produce a convolutional classifier that provably has less than 5.8% test error for any adversarial attack with bounded L_∞ norm less than $\epsilon = 0.1$.

Moreover, Dvijotham et al. [42] works by taking a different formulation of the dual problem, i.e., applying Lagrangian relaxation on the optimisation. This is to avoid working with constrained non-convex optimisation problem.

4.2.3. Interval analysis

In [43], the interval arithmetic is leveraged to compute rigorous bounds on the DNN outputs, i.e., interval property. The key idea is that, given the ranges of operands, an over-estimated range of the output can be computed by using only the lower and upper bounds of the operands. Starting from the first hidden layer, this computation can be conducted through to the output layer. Beyond this explicit computation, symbolic interval analysis along with several other optimisations are also developed to minimise over-estimations of output bounds. These methods are implemented in ReluVal, a system for formally checking security properties of ReLU-based DNNs. An advantage of this approach, comparing to constraint-solving based approaches, is that it can be easily parallelisable. In general, interval analysis is close to the interval-based abstract interpretation, which we explained in Section 4.2.1.

In [44], lower bounds of adversarial perturbations needed to alter the classification of the neural networks are derived by utilising the layer functions. The proposed bounds have theoretical guarantee that no adversarial manipulation could be any smaller, and in this case, can be computed efficiently — at most linear time in the number of (hyper)parameters of a given model and any input, which makes them applicable for choosing classifiers based on robustness.

4.2.4. Output reachable set estimation

In [24], the output reachable set estimation is addressed. Given a DNN $\mathcal{N}$ with its associated function f, and a set of inputs η, the output reachable set is $Reach(f, \eta)$ as in Definition 10. The problem is to either compute a close estimation $\mathcal{Y}'$ such that $Reach(f, \eta) \subseteq \mathcal{Y}'$, or to determine whether $Reach(f, \eta) \cap \neg\mathcal{S} = \emptyset$ for a safety specification $\mathcal{S}$, where $\mathcal{S}$ is also expressed with a set similar as the one in Eq. (12). Therefore, it is actually to compute the interval property. First, a concept called maximum sensitivity is introduced and, for a class of multi-layer perceptrons whose activation functions are monotonic functions, the maximum sensitivity can be reduced to a MILP problem as that of Section 4.1.2. Then, using a simulation-based method, the output reachable set estimation problem for neural networks is formulated into a chain of optimisation problems. Finally, an automated safety verification is developed based on the output reachable set estimation result. The approach is applied to the safety verification for a robotic arm model with two joints.

4.2.5. Linear approximation of ReLU networks

FastLin/FastLip [45] analyses the ReLU networks on both interval property and Lipschitzian property. For interval property, they consider linear approximation over those ReLU neurons that are uncertain on their status of being activated or deactivated. For Lipschitzian property, they use the gradient computation for the approximation computation. Crown [46] generalises the interval property computation algorithm in FastLin/FastLip by allowing the linear expressions for the upper and lower bounds to be different and enabling its working with other activation functions such as tanh, sigmoid and arctan. The Lipschitzian property computation is improved in RecurJac [47].

4.3. Approaches to compute converging bounds

While the above approaches can work with small networks (up to a few thousands hidden neurons), state-of-the-art DNNs usually contain at least multi-million hidden neurons. It is necessary that other approaches are developed to work with real-world systems. In Sections 4.3 and 4.4, the approaches are able to work with large-scale networks, although they might have other restrictions or limitations. Since the approaches surveyed in this subsection compute converging upper and lower bounds, they can work with both output reachability property and interval property.

4.3.1. Layer-by-layer refinement

Huang et al. [23] develops an automated verification framework for feedforward multi-layer neural networks based on Satisfiability Modulo Theory (SMT). The key features of this framework are that it *guarantees* a misclassification being found if it exists, and that it propagates the analysis *layer-by-layer*, i.e., from the input layer to, in particular, the hidden layers, and to the output layer.

In this work, *safety* for an individual classification decision, i.e., pointwise (or local) robustness, is defined as the invariance of a classifier's outcome to perturbations within a small neighbourhood of an original input. Formally,

$$\mathcal{N}, \eta_k, \Delta_k \models x$$

where x denotes an input, $\mathcal{N}$ a neural network, η_k a region surrounding the corresponding activation of the input x at layer k, and Δ_k a set of manipulations at layer k. Later, in [19,21], it is shown that the minimality of the manipulations in Δ can be guaranteed with the existence of Lipschitz constant.

To be more specific, its verification algorithm uses single-/multi-path search to exhaustively explore a finite region of the vector spaces associated with the input layer or the hidden layers, and a layer-by-layer refinement is implemented using the Z3 solver to ensure that the local robustness of a deeper layer implies the robustness of a shallower layer. The methodology is implemented in the software tool DLV, and evaluated on image benchmarks such as MNIST, CIFAR10, GTSRB, and ImageNet. Though the complexity is high, it scales to work with state-of-the-art networks such as VGG16. Furthermore, in [19,21], the search problem is alleviated by Monte-Carlo tree search.

4.3.2. Reduction to a two-player turn-based game

In DeepGame [19], two variants of pointwise robustness are studied:

- the *maximum safe radius* (MSR) problem, which for a given input sample computes the minimum distance to an adversarial example, and
- the *feature robustness* (FR) problem, which aims to quantify the robustness of individual features to adversarial perturbations.

It demonstrates that, under the assumption of Lipschitz continuity, both problems can be approximated using finite optimisation by discretising the input space, and the approximation has provable guarantees, i.e., the error is bounded. It subsequently reduces the resulting optimisation problems to the solution of a two-player turn-based game, where Player I selects features and Player II perturbs the image within the feature. While Player II aims to minimise the distance to an adversarial example, depending on the optimisation objective Player I can be *cooperative* or *competitive*. An anytime approach is employed to solve the games, in the sense of approximating the value of a game by monotonically improving its upper and lower bounds. The Monte-Carlo tree search algorithm is applied to compute upper bounds for both games, and the Admissible A* and the Alpha–Beta Pruning algorithms are, respectively, used to compute lower bounds for the MSR and FR games.

4.3.3. Global optimisation based approaches

DeepGO [15] shows that most known layers of DNNs are Lipschitz continuous, and presents a verification approach based on global optimisation. For a single dimension, an algorithm is presented to always compute the lower bounds (by utilising the Lipschitz constant) and eventually converge to the optimal value. Based on this single-dimensional algorithm, the algorithm for multiple dimensions is to exhaustively search for the best combinations. The algorithm is able to work with state-of-the-art DNNs, but is restricted by the number of dimensions to be perturbed.

In DeepTRE [48], the authors focus on the Hamming distance, and study the problem of quantifying the global robustness of a trained DNN, where global robustness is defined as the expectation of the maximum safe radius over a testing dataset. They propose an approach to iteratively generate lower and upper bounds on the network's robustness. The approach is anytime, i.e., it returns intermediate bounds and robustness estimates that are gradually, but strictly, improved as the computation proceeds; tensor-based, i.e., the computation is conducted over a set of inputs simultaneously, instead of one by one, to enable efficient GPU computation; and has provable guarantees, i.e., both the bounds and the robustness estimates can converge to their optimal values.

4.4. Approaches with statistical guarantees

This subsection reviews a few approaches aiming to achieve statistical guarantees on their results, by claiming e.g., the satisfiability of a property, or a value is a lower bound of another value, etc., with certain probability.

4.4.1. Lipschitz constant estimation by extreme value theory

Weng et al. [27] proposes a metric, called CLEVER, to estimate the Lipschitz constant, i.e., the approach works with Lipschitzian property. It estimates the robustness lower bound by sampling the norm of gradients and fitting a limit distribution using extreme value theory. However, as argued by Goodfellow [49], their evaluation approach can only find statistical approximation of the lower bound, i.e., their approach has a soundness problem.

4.4.2. Robustness estimation

Bastani et al. [50] proposes two statistics of robustness to measure the frequency and the severity of adversarial examples, respectively. Both statistics are based on a parameter ϵ, which is the maximum radius within which no adversarial examples exist. The computation of these statistics is based on the local linearity assumption which holds when ϵ is small enough. Except for the application of the ReLU activation function which is piece-wise linear, this assumption can be satisfied by the existence of the Lipschitz constant as shown in [15].

4.5. Computational complexity of verification

There are two ways to measure the complexity of conducting formal verification. The first, appeared in [14], measures the complexity with respect to the number of hidden neurons. This is due to the fact that their approach is to encode the DNN into a set of constraints, and in the constraints every hidden neuron is associated with two variables. On the other hand, in [15], the complexity is measured with respect to the number of input dimensions. This is due to the fact that their approach is to manipulate the input. For both cases, the complexity is shown NP-complete, although it is understandable that the number of hidden neurons can be larger than the number of input dimensions.

4.6. Summary

We summarise some existing approaches to the verification of DNNs in Table 1, from the aspects of the type of achievable guarantees, underlying algorithms, and objective properties, i.e., robustness, reachability, interval, and Lipschitzian.

5. Testing

Similar to traditional software testing against software verification, DNN testing provides a certification methodology with a balance between completeness and efficiency. In established industries, e.g., avionics and automotive, the needs for software testing have been settled in various standards such as DO-178C and MISRA. However, due to the lack of logical structures and system specification, it is still unclear how to extend such standards to work with systems with DNN components. In the following, we survey testing techniques from three aspects: coverage criteria (Section 5.1), test case generation (Section 5.2), and model-level mutation testing (Section 5.3). The first two do not alter the structure of the DNN, while the mutation testing involves the change to the structure and parameters of the DNN.

5.1. Coverage criteria for DNNs

Research in software engineering has resulted in a broad range of approaches to testing software. Please refer to Zhu et al. [16], Jia and Harman [51] and Su et al. [52] for comprehensive reviews. In white-box testing, the structure of a programme is exploited to (perhaps automatically) generate test cases. Structural coverage criteria (or metrics) define a set of test objectives to be covered, guiding the generation of test cases and evaluating the completeness of a test suite. E.g., a test suite with 100% statement coverage exercises all statements of the programme at least once. While it is arguable whether this ensures functional correctness, high coverage is able to increase users' confidence (or trust) in the testing results [16]. Structural coverage analysis and testing are also used as a means of assessment in a number of safety-critical scenarios, and criteria such as statement and modified condition/decision coverage (MC/DC) are applicable measures with respect to different criticality levels. MC/DC was developed by NASA [53] and has been widely adopted. It is used in avionics software development guidance to ensure adequate testing of applications with the highest criticality [54].

We let F be a set of covering methods, and $\mathcal{R} = \mathsf{O}(\mathcal{N})$ be the set of test objectives to be covered. In [55] $\mathsf{O}(\mathcal{N})$ is instantiated as the set of causal relationships between feature pairs, while in [56] $\mathsf{O}(\mathcal{N})$ is instantiated as the set of statuses of hidden neurons.

Definition 19 (*Test Suite*)**.** Given a DNN $\mathcal{N}$, a test suite $\mathcal{T}$ is a finite set of input vectors, i.e., $\mathcal{T} \subseteq D_1 \times D_1 \times \cdots \times D_1$. Each vector is called a test case.

Usually, a test case is a single input $\mathcal{T} \subseteq D_1$, e.g., in [56], or a pair of inputs $\mathcal{T} \subseteq D_1 \times D_1$, e.g., in [55]. Ideally, given the set of test objectives $\mathcal{R}$ according to some covering method cov, we run a test case generation algorithm to find a test suite $\mathcal{T}$ such that

$$\forall \alpha \in \mathcal{R} \exists (x_1, x_2, \ldots, x_k) \in \mathcal{T} : cov(\alpha, (x_1, x_2, \ldots, x_k)) \tag{22}$$

where $cov(\alpha, (x_1, x_2, \ldots, x_k))$ intuitively means that the test objective α is satisfied under the test case $(x_1, x_2, \ldots, x_k)$. In practice, we might want to compute the degree to which the test objectives are satisfied by a generated test suite $\mathcal{T}$.

Definition 20 (*Test Criterion*)**.** Given a DNN $\mathcal{N}$ with its associated function f, a covering method cov, test objectives in $\mathcal{R}$, and a test suite $\mathcal{T}$, the test criterion $M_{cov}(\mathcal{R}, \mathcal{T})$ is as follows:

$$M_{cov}(\mathcal{R}, \mathcal{T}) = \frac{|\{\alpha \in \mathcal{R} | \exists (x_1, x_2, \ldots, x_k) \in \mathcal{T} : cov(\alpha, (x_1, x_2, \ldots, x_k))\}|}{|\mathcal{R}|} \tag{23}$$

Intuitively, it computes the percentage of the test objectives that are covered by test cases in $\mathcal{T}$ w.r.t. the covering method cov.

5.1.1. Neuron coverage

Neuron coverage [56] can be seen as the statement coverage variant for DNN testing. Note that, the neuron coverage is primarily designed for ReLU networks, although an easy adaptation can be applied to make it work with other activation functions.

Definition 21 (*[56]*)**.** A node $n_{k,i}$ is neuron covered by a test case x, denoted as $N(n_{k,i}, x)$, if $sign(n_{k,i}, x) = +1$.

The set of objectives to be covered is $\mathsf{O}(\mathcal{N}) = \{n_{k,i} \mid 2 \leq k \leq K-1, 1 \leq i \leq s_k\}$. Each test case is a single input, i.e., $\mathcal{T} \subseteq D_{L_1}$. The covering method is as follows: $cov(n_{k,i}, x')$ if and only if $N(n_{k,i}, x')$.

A search algorithm DeepXplore [56] is developed to generate test cases for neuron coverage. It takes multiple DNNs $\{f^k \mid k \in 1..n\}$ as the input and maximises over both the number of observed differential behaviours and the neuron coverage while preserving domain-specific constraints provided by the users. Let $f_c^k(x)$ be the class probability that f^k predicts x to be c. The optimisation objective for a neuron $n_{j,i}$ is as follows.

$$obj(x, n_{j,i}) = (\sum_{k \neq j} f_c^k(x) - \lambda_1 f_c^j(x)) + \lambda_2 v_{j,i}(x) \tag{24}$$

where λ_1 and λ_2 are a tunable parameters, $\sum_{k \neq j} f_c^k(x) - \lambda_1 f_c^j(x)$ denotes differential behaviours, and $v_{j,i}(x)$ is the activation value of neuron $n_{j,i}$ on x.

Moreover, the greedy search combined with image transformations is used in [57] to increase neuron coverage, and is applied to DNNs for autonomous driving. In [58], the conclic testing algorithm – a software analysis technique combining symbolic execution and concrete execution – can work with a set of coverage metrics including neuron coverage and some other coverage metrics to be surveyed in the following.

5.1.2. Safety coverage

In [21], the input space is discretised with a set of hyper-rectangles, and then one test case is generated for each hyper-rectangle.

Definition 22. Let each hyper-rectangle *rec* contain those inputs with the same pattern of ReLU, i.e., for all $x_1, x_2 \in rec$, $2 \leq k \leq K-1$ and $1 \leq l \leq s_k$, we have $sign(n_{k,l}, x_1) = sign(n_{k,l}, x_2)$. A hyper-rectangle *rec* is safe covered by a test case x, denoted as $S(rec, x)$, if $x \in rec$.

Table 1
Comparison between the verification approaches of deep neural networks.

	Guarantees	Algorithm		Property			
				Robustness	*Reachability*	*Interval*	*Lipschitzian*
Pulina and Tacchella [28]	Deterministic guarantees	Constraints solving	SMT	✓	✓	✓	
Katz et al. [14]				✓	✓	✓	
Ehlers [29]				✓	✓	✓	
Narodytska et al. [30] and Narodytska [31]			SAT	✓	✓	✓	
Lomuscio and Maganti [32]		MILP		✓	✓	✓	
Cheng et al. [33]				✓	✓	✓	
Dutta et al. [34]				✓	✓	✓	
Bunel et al. [35]				✓	✓	✓	
Gehr et al. [38]	Lower/Upper bound	Abstract interpretation		✓		✓	
Mirman et al. [39]				✓		✓	
Li et al. [40]				✓		✓	
Wong and Kolter [41]		Convex optimisation		✓		✓	
Wang et al. [43]		Interval analysis		✓		✓	
Peck et al. [44]				✓		✓	
Xiang et al. [24]		Set estimation		✓		✓	
Weng et al. [45]		Linear approximation		✓		✓	✓
Huang et al. [23]	Converging bounds	Search based	Layer-by-Layer refinement	✓	✓	✓	
Wicker et al. [21]			Two-player turn-based game	✓	✓	✓	✓
Wu et al. [19]				✓	✓	✓	✓
Ruan et al. [15]		Global optimisation		✓	✓	✓	✓
Ruan et al. [48]				✓	✓	✓	
Weng et al. [27]	Statistical guarantees	Extreme value theory					✓
Bastani et al. [50]		Robustness estimation		✓			

Let $Rec(\mathcal{N})$ be the set of hyper-rectangles. The set of objectives to be covered is $\mathsf{O}(\mathcal{N}) = Rec(\mathcal{N})$. Each test case is a single input, i.e., $\mathcal{T} \subseteq D_{L_1}$. The covering method is as follows: $cov(rec, x)$ if and only if $S(rec, x)$.

Moreover, there are different ways to define the set of hyper-rectangles. For example, the "boxing clever" method in [59], initially proposed for designing training datasets, divides the input space into a series of representative boxes. When the hyper-rectangle is sufficiently fine-grained with respect to Lipschitz constant of the DNN, the method in [21] becomes exhaustive search and has provable guarantee on its result. In terms of the test case generation algorithm, it uses Monte Carlo tree search to exhaustively enumerate for each hyper-rectangle a test case.

5.1.3. Extensions of neuron coverage

In [60], several coverage criteria are proposed, following similar rationale as neuron coverage and focusing on individual neurons' activation values.

Definition 23 ([60]). A node $n_{k,i}$ is neuron boundary covered by a test case x, denoted as $NB(n_{k,i}, x)$, if $v_{k,i}[x] > v^u_{k,i}$.

Let $rank(n_{k,i}, x)$ represent the rank of the value $v_{k,i}[x]$ among those values $\{v_{k,j}[x] \mid 1 \le j \le s_k\}$ of the nodes at the same layer, i.e., there are $rank(n_{k,i}, x) - 1$ elements in $\{v_{k,j}[x] \mid 1 \le j \le s_k\}$ which are greater than $v_{k,i}[x]$.

Definition 24 ([60]). For $1 \le m \le s_k$, a node $n_{k,i}$ is top-m neuron covered by x, denoted as $TN^m(n_{k,i}, x)$, if $rank(n_{k,i}, x) \le m$.

Let $v^l_{k,i} = \min_{x \in X} v_{k,i}[x]$ and $v^u_{k,i} = \max_{x \in X} v_{k,i}[x]$ for some input x. We can split the interval $I_{k,i} = [v^l_{k,i}, v^u_{k,i}]$ into m equal sections, and let $I^j_{k,i}$ be the jth section.

Definition 25 ([60]). Given $m \ge 1$, a node $n_{k,i}$ is m-multisection neuron covered by a test suite $\mathcal{T}$, denoted as $MN^m(n_{k,i}, \mathcal{T})$, if $\forall 1 \le j \le m \exists x \in \mathcal{T} : v_{k,i}[x] \in I^j_{k,i}$, i.e., all sections are covered by some test cases.

Each test case is a single input, i.e., $\mathcal{T} \subseteq D_{L_1}$. We omit the definition of test objectives O and covering methods cov, which are similar to the original neuron coverage case.

No particular algorithm is developed in [60] for generating test cases for the criteria proposed; instead, they apply adversarial attack methods (e.g., [61]) to generate an extra set of new inputs that is shown to increase the coverage. Following Ma et al. [60], an exploratory study on combinatorial testing is conducted in [62] to cover combinations of neurons' activations at the same layer.

5.1.4. Modified condition/decision coverage (MC/DC)

Modified Condition/Decision Coverage (MC/DC) [53] is a method of ensuring adequate testing for safety-critical software. At its core is the idea that if a choice can be made, all the possible factors (conditions) that contribute to that choice (decision) must be tested. For traditional software, both conditions and the decision are usually Boolean variables or Boolean expressions.

Example 8. The decision

$$d \iff ((a > 3) \vee (b = 0)) \wedge (c \neq 4) \tag{25}$$

contains the three conditions $(a > 3)$, $(b = 0)$ and $(c \neq 4)$. The following test cases provide 100% MC/DC coverage:

1. $(a > 3) = \text{false}$, $(b = 0) = \text{true}$, $(c \neq 4) = \text{false}$
2. $(a > 3) = \text{true}$, $(b = 0) = \text{false}$, $(c \neq 4) = \text{true}$
3. $(a > 3) = \text{false}$, $(b = 0) = \text{false}$, $(c \neq 4) = \text{true}$

4. $(a > 3) = \text{false}$, $(b = 0) = \text{true}$, $(c \neq 4) = \text{true}$

The first two test cases already satisfy both *condition coverage* (i.e., all possibilities of the conditions are exploited) and *decision coverage* (i.e., all possibilities of the decision are exploited). The other two cases are needed because, for MC/DC each condition should evaluate to true and false at least once, and should independently affect the decision outcome

Motivated by the MC/DC testing for traditional software, an MC/DC variant for DNNs are initially proposed in [63,64], which is further refined in [55]. Different from the criteria in [56,60] that only consider individual neurons' activations, the criteria in [55,63,64] take into account the causal relation between features in DNNs: *the core idea is to ensure that not only the presence of a feature needs to be tested but also the effects of less complex features on a more complex feature must be tested*.

We let Ψ_k be a collection of subsets of nodes at layer k. Without loss of generality, each element of Ψ_k, i.e., a subset of nodes in the kth layer, represents a *feature* learned at layer k.

At first, different from the Boolean case, where changes of conditions and decisions are straightforwardly switches of true/false values, the change observed on a feature can be either a sign change or a value change.

Definition 26 (*Sign Change*)**.** Given a feature $\psi_{k,l}$ and two test cases x_1 and x_2, the sign change of $\psi_{k,l}$ is exploited by x_1 and x_2, denoted as $sc(\psi_{k,l}, x_1, x_2)$, if

- $sign(n_{k,j}, x_1) \neq sign(n_{k,j}, x_2)$ for all $n_{k,j} \in \psi_{k,l}$.

Moreover, we write $nsc(\psi_{k,l}, x_1, x_2)$ if

- $sign(n_{k,j}, x_1) = sign(n_{k,j}, x_2)$ for all $n_{k,j} \in \psi_{k,l}$.

Note that $nsc(\psi_{k,l}, x_1, x_2) \not\equiv \neg sc(\psi_{k,l}, x_1, x_2)$. When the ReLU activation function is assumed, the sign change of a feature represents switch of the two cases, in which neuron activations of this feature are and are not propagated to the next layer.

A feature's sign change is sometimes too restrictive and its value change compensates this. We can denote a value function as $g : \Psi_k \times D_{L_1} \times D_{L_1} \rightarrow \{\text{true}, \text{false}\}$. Simply speaking, it expresses the DNN developer's intuition (or knowledge) on what contributes as a significant change on the feature $\psi_{k,l}$, by specifying the difference between two vectors $\psi_{k,l}[x_1]$ and $\psi_{k,l}[x_2]$. The following are a few examples.

Example 9. For a singleton set $\psi_{k,l} = \{n_{k,j}\}$, the function $g(\psi_{k,l}, x_1, x_2)$ can express e.g., $|u_{k,j}[x_1] - u_{k,j}[x_2]| \geq d$ (absolute change), or $\frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} > d \vee \frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} < 1/d$ (relative change), etc. It can also express the constraint on one of the values $u_{k,j}[x_2]$ such as $u_{k,j}[x_2] > d$ (upper boundary).

Example 10. For the general case, the function $g(\psi_{k,l}, x_1, x_2)$ can express the distance between two vectors $\psi_{k,l}[x_1]$ and $\psi_{k,l}[x_2]$ by e.g., norm-based distances $\|\psi_{k,l}[x_1] - \psi_{k,l}[x_2]\|_p \leq d$ for a real number d and a distance measure L^p, or structural similarity distances such as SSIM [65]. It can also express constraints between nodes of the same layer such as $\bigwedge_{j \neq i} v_{k,i}[x_1] \geq v_{k,j}[x_1]$.

Consequently, the value change of a feature is defined as follows.

Definition 27 (*Value Change*)**.** Given a feature $\psi_{k,l}$, two test cases x_1 and x_2, and a value function g, the value change of $\psi_{k,l}$ is exploited by x_1 and x_2 with respect to g, denoted as $vc(g, \psi_{k,l}, x_1, x_2)$, if

- $g(\psi_{k,l}, x_1, x_2) = \text{true}$.

Moreover, we write $\neg vc(g, \psi_{k,l}, x_1, x_2)$ when the condition is not satisfied.

Based on the concept of sign changes and value changes, a family of four coverage criteria are proposed in [55,63,64], i.e., the MC/DC variant for DNNs, to exploit the causal relationship between the changes of features at consecutive layers of the neural network.

Definition 28 (*Sign–Sign Coverage, or SS Coverage*)**.** A feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is SS-covered by two test cases x_1, x_2, denoted as $SS(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:

- $sc(\psi_{k,i}, x_1, x_2)$ and $nsc(P_k \setminus \psi_{k,i}, x_1, x_2)$;
- $sc(\psi_{k+1,j}, x_1, x_2)$.

where P_k is the set of nodes in layer k.

Definition 29 (*Sign–Value Coverage, or SV Coverage*)**.** Given a value function g, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is SV-covered by two test cases x_1, x_2, denoted as $SV^g(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:

- $sc(\psi_{k,i}, x_1, x_2)$ and $nsc(P_k \setminus \psi_{k,i}, x_1, x_2)$;
- $vc(g, \psi_{k+1,j}, x_1, x_2)$ and $nsc(\psi_{k+1,j}, x_1, x_2)$.

Definition 30 (*Value–Sign Coverage, or VS Coverage*)**.** Given a value function g, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is VS-covered by two test cases x_1, x_2, denoted as $VS^g(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:

- $vc(g, \psi_{k,i}, x_1, x_2)$ and $nsc(P_k, x_1, x_2)$;
- $sc(\psi_{k+1,j}, x_1, x_2)$.

Definition 31 (*Value–Value Coverage, or VV Coverage*)**.** Given two value functions g_1 and g_2, a feature pair $\alpha = (\psi_{k,i}, \psi_{k+1,j})$ is VV-covered by two test cases x_1, x_2, denoted as $VV^{g_1,g_2}(\alpha, x_1, x_2)$, if the following conditions are satisfied by the DNN instances $\mathcal{N}[x_1]$ and $\mathcal{N}[x_2]$:

- $vc(g_1, \psi_{k,i}, x_1, x_2)$ and $nsc(P_k, x_1, x_2)$;
- $vc(g_2, \psi_{k+1,j}, x_1, x_2)$ and $nsc(\psi_{k+1,j}, x_1, x_2)$.

For all the above, each test case is a pair of inputs, i.e., $\mathcal{T} \subseteq D_{L_1} \times D_{L_1}$. The test objectives O is a set of feature pairs, provided by the user or computed automatically according to the structure of the DNN. The covering methods cov has been defined in the above definitions.

For the test case generation, Sun et al. [63,64] develops an algorithm based on linear programming (LP). This is complemented with an adaptive gradient descent (GD) search algorithm in [55] and a concolic testing algorithm in [58].

5.1.5. Quantitative projection coverage

In [66], it is assumed that there exist a number of weighted criteria for describing the operation conditions. For example, for self-driving cars, the criteria can be based on e.g., weather, landscape, partially occluding pedestrians, etc. With these criteria one can systematically partition the input domain and weight each partitioned class based on its relative importance. Based on this, the quantitative k-projection is such that the data set, when being projected onto the k-hyperplane, needs to have (in each region) data points no less than the associated weight. While the criteria in [66] are based on self-driving scenes, Cheng et al. [67] present a few further criteria that take into account DNN internal structures, focusing on individual neurons' or neuron sets' activation values.

In terms of the test case generation, a method based on 0–1 Integer Linear Programming is developed. It has been integrated into the nn-dependability-kit [68].

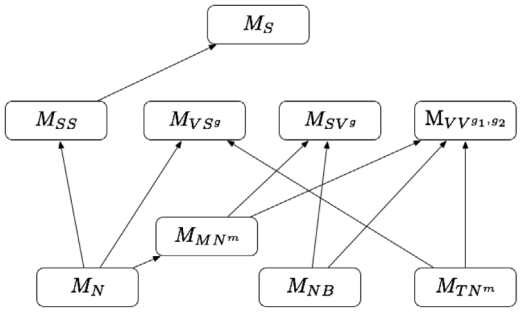


Fig. 7. Relationship between test criteria.

5.1.6. Surprise coverage

Kim et al. [69] aims to measure the relative novelty (i.e., surprise) of the test inputs with respect to the training dataset, by measuring the difference of activation patterns [63,64] between inputs. Given a training set $X \subseteq D_{L_1}$, a layer k, and a new input x, one of the measurements is to compute the following value

$$-\log(\frac{1}{|X|}\sum_{x_i \in X} \mathcal{K}_H(v_k(x) - v_k(x_i))) \tag{26}$$

where $v_k(x)$ is the vector of activation values for neurons in layer k when the input is x, and $|X|$ is the number of training samples. Moreover, $\mathcal{K}$ is a Gaussian kernel function and H is a bandwidth matrix, used in Kernel Density Estimation [70]. Based on this, the coverage is defined, similar as the m-multisection [60], to cover a few pre-specified segments within a range $(0, U]$. Intuitively, a good test input set for a DNN should be systematically diversified to include inputs ranging from those similar to training data (i.e., having lower values for Expression (26)) to those significantly different (i.e., having higher values for Expression (26)).

In terms of test case generation, Kim et al. [69] utilises a few existing algorithms for adversarial attack, including FGSM [61], Basic iterative method [71], JSMA [72], CW attack [73].

5.1.7. Comparison between existing coverage criteria

Fig. 7 gives a diagrammatic illustration of the relationship between some coverage criteria we survey above. An arrow from A to B denotes that the coverage metric A is weaker than the coverage metric B. We say that metric M_{cov_1} is weaker than another metric M_{cov_2}, if for any given test suite $\mathcal{T}$ on $\mathcal{N}$, we have that $M_{cov_1}(obj_1, \mathcal{T}) < 1$ implies $M_{cov_2}(obj_2, \mathcal{T}) < 1$, for their respective objectives obj_1 and obj_2. Particularly, as discussed in [74], when considering the use of machine models in safety critical applications like automotive software, DNN structural coverage criteria can be applied in a similar manner as their traditional software counterparts (e.g., statement coverage or MC/DC) to different Automotive Safety Integrity Levels (ASALs).

5.2. Test case generation

In the following, we survey the test case generation methods for DNNs that have not been covered in Section 5.1 and that do not employ the existing adversarial attack algorithms. We will review adversarial attacks in Section 6.

5.2.1. Input mutation

Given a set of inputs, input mutation generates new inputs (as test cases) by changing the existing input according to some predefined transformation rules or algorithms. For example, Wicker et al. [21] systematically mutates input dimensions with the goal of enumerating all hyper-rectangles in the input space. Moreover, aiming at testing the fairness (i.e., free of unintended bias) of DNNs, AEQUITAS [75] essentially employs an input mutation technique to first randomly sample a set of inputs and then explore the neighbourhood of the sampled inputs by changing a subset of input dimensions, however, it has not been applied to DNN model.

5.2.2. Fuzzing

Fuzzing, or fuzz testing, is an automated software testing technique that efficiently generates a massive amount of random input data (possibly invalid or unexpected) to a programme, which is then monitored for exceptions and failures. A fuzzer can be mutation-based that modifies existing input data. Depending on the level of awareness of the programme structure, the fuzzer can be white/grey/block-box. There are recent works that adopt fuzz testing to deep neural networks.

TensorFuzz [76] is a coverage-guided fuzzing method for DNNs. It randomly mutates the inputs, guided by a coverage metric over the goal of satisfying user-specified constraints. The coverage is measured by a fast approximate nearest neighbour algorithm. TensorFuzz is validated in finding numerical errors, generating disagreements between DNNs and their quantised versions, and surfacing undesirable behaviour in DNNs. Similar to TensorFuzz, DeepHunter [77] is another coverage-guided grey-box DNN fuzzer, which utilises these extensions of neuron coverage from [60]. Moreover, DLFuzz [78] is a differential fuzzing testing framework. It mutates the input to maximise the neuron coverage and the prediction difference between the original input and the mutated input.

5.2.3. Symbolic execution and testing

Though input mutation and fuzzing are good at generating a large amount of random data, there is no guarantee that certain test objectives will be satisfied. Symbolic execution (also symbolic evaluation) is a means of analysing a programme to determine what inputs cause each part of a programme to execute. It assumes symbolic values for inputs rather than obtaining actual inputs as normal execution of the programme would, and thus arrives at expressions in terms of those symbols for expressions and variables in the programme, and constraints in terms of those symbols for the possible outcomes of each conditional branch.

Concolic testing is a hybrid software testing technique that alternates between concrete execution, i.e., testing on particular inputs, and symbolic execution, i.e., symbolically encoding particular execution paths. This idea still holds for deep neural networks. In DeepConcolic [58,79], coverage criteria for DNNs that have been studied in the literature are first formulated using the Quantified Linear Arithmetic over Rationals, and then a coherent method for performing concolic testing to increase test coverage is provided. The concolic procedure starts from executing the DNN using concrete inputs. Then, for those test objectives that have not been satisfied, they are ranked according to some heuristic. Consequently, a top ranked pair of test objective and the corresponding concrete input are selected and symbolic analysis is thus applied to find a new input test. The experimental results show the effectiveness of the concolic testing approach in both achieving high coverage and finding adversarial examples.

The idea in [80] is to translate a DNN into an imperative programme, thereby enabling programme analysis to assist with DNN validation. It introduces novel techniques for lightweight symbolic analysis of DNNs and applies them in the context of image classification to address two challenging problems, i.e., identification of important pixels (for attribution and adversarial generation), and creation of 1-pixel and 2-pixel attacks. In [81], black-box style local explanations are first called to build a decision tree, to which the symbolic execution is then applied to

detect individual discrimination in a DNN: such a discrimination exists when two inputs, differing only in the values of some specified attributes (e.g., gender/race), get different decisions from the neural network.

5.2.4. Testing using generative adversarial networks

Generative adversarial networks (GANs) are a class of AI algorithms used in unsupervised machine learning. It is implemented by a system of two neural networks contesting with each other in a zero-sum game framework. DeepRoad [22] automatically generate large amounts of accurate driving scenes to test the consistency of DNN-based autonomous driving systems across different scenes. In particular, it synthesises driving scenes with various weather conditions (including those with rather extreme conditions) by applying the Generative Adversarial Networks (GANs) along with the corresponding real-world weather scenes.

5.2.5. Differential analysis

We have already seen differential analysis techniques in [56] and [78] that analyse the differences between multiple DNNs to maximise the neuron coverage. Differential analysis of a single DNN's internal states has been also applied to debug the neural network model by Ma et al. [82], in which a DNN is said to be buggy when its test accuracy for a specific output label is lower than the ideal accuracy. Given a buggy output label, the differential analysis in [82] builds two heat maps corresponding to its correct and wrong classifications. Intuitively, a heat map is an image whose size equals to the number of neurons and the pixel value represents the importance of a neuron (for the output). Subsequently, the difference between these two maps can be used to highlight these faulty neurons that are responsible for the output bug. Then, new inputs are generated (e.g., using GAN) to re-train the DNN so to reduce the influence of the detected faulty neurons and the buggy output.

5.3. Model-level mutation testing

Mutation testing is a white-box testing technique that performs by changing certain statements in the source code and checking if the test cases are able to find the errors. Once a test case fails on a mutant, the mutant is said to be killed. Mutation testing is not used to find the bugs in software but evaluate the quality of the test suite which is measured by the percentage of mutants that they kill.

Shen et al. [83] proposes five mutation operators, including (i) deleting one neuron in input layer, (ii) deleting one or more hidden neurons, (iii) changing one or more activation functions, (iv) changing one or more bias values, and (v) changing weight value. Ma et al. [84] considers data mutations, programme mutations, and model-level mutations. For data mutations, a few operations on training dataset are considered, including duplicating a small portion of data, injecting faults to the labels, removing some data points, and adding noises to the data. For programme mutations, a few operations are considered, including adding or deleting a layer and removing activation function. Model-level mutations include changing the weights, shuffling the weights between neurons in neighbouring layers, etc. Moreover, Cheng et al. [85] simulates programme bugs by mutating Weka implementations of several classification algorithms, including Naive Bayes, C4.5, k-NN, and SVM.

5.4. Summary

We compare the testing methods for DNNs in Table 2. Overall, search algorithms, including greedy search and gradient ascent (GA) search, are often used in the testing. For simple coverage criteria such as neuron coverage and its extensions and surprise coverage, established machine learning adversarial attack algorithms (e.g., FGSM [61] and JSMA [72]) are sufficient enough for test case generation. In the more complex cases, Linear Programming (LP) or Integer Linear Programming (ILP) approaches can be used, and the Monte Carlo Tree Search (MCTS) method is called for generating tests for the safety coverage. Advanced testing methods like concolic testing and fuzzing have been also developed for DNNs.

Sometimes, distances between test inputs need to be taken into account in DNN testing to guide the tests generation. As in the last column of Table 2, different norm distances have been applied, however, there is no conclusion on which one is the best. Works like [56,78] are in principle based on differential analysis of multiple DNNs, thus more than one DNN inputs are expected. As in [86], DNN adversarial testing can be formulated in the form of software differential testing.

Up to now, most techniques are developed by extending the existing techniques from software testing with simple adaptations. It is necessary to validate the developed techniques (as discussed in Section 8.6) and study the necessity of developing dedicated testing techniques. Meanwhile, recent effort has been also made to apply surveyed DNN testing methods from feed-forward networks to other deep learning models like recurrent neural networks [87–89].

6. Adversarial attack and defence

The goal of attack techniques is to provide evidence (i.e., adversarial examples) for the lack of robustness of a DNN without having a provable guarantee. Defence techniques are dual to attack techniques, by either improving the robustness of the DNN to immune the adversarial examples, or differentiating the adversarial examples from the correct inputs. From Sections 6.1 to 6.3, we review the attack techniques from different aspects. These techniques are compared in Section 6.5 with a few other techniques from verification. Then, in Sections 6.6 and 6.7, we review defence techniques (without a guarantee), and certified defence techniques, respectively.

6.1. Adversarial attacks

Given an input, an adversarial attack (or attacker) tries to craft a perturbation or distortion to the input to make it misclassified by a well-trained DNN. Usually, it is required that the adversarial example is misclassified with high confidence. Attack techniques can be divided roughly into two groups based on the type of misclassification they try to achieve:

- With a targeted perturbation, the attacker is able to control the resulting misclassification label.
- With an un-targeted perturbation, the attacker can enable the misclassification but cannot control its resulting misclassification label.

According to the amount of information an attacker can access, adversarial perturbations can also be classified into two categories:

- White-box perturbation — the attacker needs to access the parameters and the internal structure of the trained DNN, and may also need to access the training dataset.

Table 2
Comparison between different DNN testing methods.

	Test generation	Coverage criteria	DNN inputs	Distance metric
Pei et al. [56]	Dual-objective search	Neuron coverage	Multiple	L_1
Tian et al. [57]	Greedy search	Neuron coverage	Single	Jaccard distance
Wicker et al. [21]	MCTS	Safety coverage	Single	n/a
Ma et al. [60]	Adversarial attack	Neuron coverage extensions	Single	n/a
Sun et al. [55,63,64]	LP, adaptive GA search	MC/DC	Single	L_∞
Cheng et al. [66]	0-1 ILP	Quantitative projection coverage	Single	n/a
Kim et al. [69]	Adversarial attacks	Surprise coverage	Single	n/a
Sun et al. [58] and Sun et al. [79]	Concolic testing	MC/DC, neuron coverage and its extensions	Single	L_0, L_∞
Odena and Goodfellow [76]	Fuzzing	Fast approximate nearest neighbour	Single	n/a
Xie et al. [77]	Fuzzing	Neuron coverage extensions	Single	n/a
Guo et al. [78]	Fuzzing	Neuron coverage	Multiple	n/a
Gopinath et al. [80] and Agarwal et al. [81]	Symbolic execution	n/a	Single	n/a
Zhang et al. [22]	GAN	n/a	Single	n/a
Ma et al. [82]	GAN	n/a	Single	n/a

- Black-box perturbation — an attacker can only query the trained DNN with perturbed inputs, without the ability to access the internal structure and parameters of the DNN.

Moreover, according to the norm-distances used to evaluate the difference between a perturbed input and the original input, adversarial attacks can be generally classified as L_0, L_1, L_2 or L_∞-attacks. Please note that all perturbations can be measured with any of these norms, but an attack technique can produce adversarial examples which are better measured with a particular norm.

Currently, most of the adversarial attacks are concentrating on computer vision models. From a technical point of view, those attacks can be categorised as using cost gradients, such as in [61,90,91], or using gradients of the output w.r.t neural network's input, such as JSMA attack [72], or directly formulating into optimisation problems to produce adversarial perturbations such as DeepFool [92]and C&W attack [93].

It also has been demonstrated that adversarial examples are transferable across different neural network models [6,72]. In [71], the authors further demonstrate that adversarial examples can be transferred into real world scenarios. Namely, adversarial examples can be still misclassified after being printed in a paper physically. In the following, we will review a few notable works in details.

6.1.1. Limited-memory BFGS algorithm (L-BFGS)

Szegedy et al. [6] noticed the existence of adversarial examples, and described them as 'blind spots' in DNNs. They found that adversarial images usually appear in the neighbourhood of correctly-classified examples, which can fool the DNNs although they are human-visually similar to the natural ones. It also empirically observes that random sampling in the neighbouring area is not efficient to generate such examples due to the sparsity of adversarial images in the high-dimensional space. Thus, they proposed an optimisation solution to efficiently search the adversarial examples. Formally, assume we have a classifier $f : \mathbb{R}^{s_1} \to \{1 \dots s_K\}$ that maps inputs to one of s_K labels, and $x \in \mathbb{R}^{s_1}$ is an input, $t \in \{1 \dots s_K\}$ is a target label such that $t \neq \arg\max_l f_l(x)$. Then the adversarial perturbation r can be solved by

$$\begin{aligned} &\min \|r\|_2 \\ s.t.\ &\arg\max_l f_l(x+r) = t \\ &x + r \in \mathbb{R}^{s_1} \end{aligned} \tag{27}$$

Since the exact computation is hard, an approximate algorithm based on the limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) is used instead. Furthermore, Szegedy et al. [6] observed that adversarial perturbations are able to transfer among different model structures and training sets, i.e., an adversarial image that aims to fool one DNN classifier also potentially deceives another neural network with different architectures or training datasets.

6.1.2. Fast gradient sign method (FGSM)

Fast Gradient Sign Method [61] is able to find adversarial perturbations with a fixed L_∞-norm constraint. FGSM conducts a one-step modification to all pixel values so that the value of the loss function is increased under a certain L_∞-norm constraint. The authors claim that the linearity of the neural network classifier leads to the adversarial images because the adversarial examples are found by moving linearly along the reverse direction of the gradient of the cost function. Based on this linear explanation, Goodfellow et al. [61] proposes an efficient linear approach to generate adversarial images. Let θ represents the model parameters, x, y denote the input and the label and $J(\theta, x, y)$ is the loss function. We can calculate adversarial perturbation r by

$$r = \epsilon \ \text{sign}\ (\nabla_x J(\theta, x, y)) \tag{28}$$

A larger ϵ leads to a higher success rate of attacking, but potentially results in a bigger human visual difference. This attacking method has since been extended to a targeted and iterative version [71].

6.1.3. Jacobian saliency map based attack (JSMA)

Papernot et al. [72] present a L_0-norm based adversarial attacking method by exploring the *forward derivative* of a neural network. Specifically it utilises the Jacobian matrix of a DNN's logit output w.r.t. its input to identify those most sensitive pixels which then are perturbed to fool the neural network model effectively. Let $c \in \mathcal{L}$ denote a target class and $x \in [0, 1]^{s_1}$ represent an input image. JSMA will assign each pixel in x a salient weight based on the Jacobian matrix. Each salient value basically quantifies the sensitivity of the pixel to the predicted probability of class c. To generate the adversarial perturbation, the pixel with the highest salient weight is firstly perturbed by a *maximum distortion parameter* $\tau > 0$. If the perturbation leads to a mis-classification, then JSMA attack terminates. Otherwise, the algorithm will continue until a mis-classification is achieved. When a maximum L_0-norm distortion $d > 0$ is reached, the algorithm also terminates. This algorithm is primarily to produce adversarial images that are optimised under the L_0-norm distance. JMSA is generally slower than FGSM due to the computation of the Jacobian matrix.

6.1.4. DeepFool: A simple and accurate method to fool deep neural networks

In DeepFool, Moosavi-Dezfooli et al. [92] introduces an iterative approach to generate adversarial images on any $L_p, p \in$

Fig. 8. Rotation-Translation: Original (L) 'automobile', adversarial (R) 'dog' from [95]. The original image of an 'automobile' from the CIFAR-10 dataset is rotated (by at most 30°) and translated (by at most 3 pixels) results in an image that state-of-art classifier ResNet [96] classifies as 'dog'.

$[1, \infty)$ norm distance. In this work, the authors first show how to search adversarial images for an affine binary classifier, i.e., $g(x) = \text{sign}(w^T \cdot x + b)$. Given an input image x_0, DeepFool is able to produce an optimal adversarial image by projecting x_0 orthogonally on the hyper-plane $\mathcal{F} = \{x | w^T \cdot x + b = 0\}$. Then this approach is generalised for a multi-class classifier: $W \in \mathbb{R}^{m \times k}$ and $b \in \mathbb{R}^k$. Let W_i and b_i be the ith component of W and b, respectively. We have

$$g(x) = \underset{i \in \{1 \ldots k\}}{\text{argmax}}\, g_i(x) \text{ where } g_i(x) = W_i^T x + b_i$$

For this case, the input x_0 is projected to the nearest face of the hyper-polyhedron P to produce the optimal adversarial image, namely,

$$P(x_0) = \bigcap_{i=1}^{k} \{x | g_{k_0}(x) \geq g_i(x)\}$$

where $k_0 = g(x_0)$. We can see that P is the set of the inputs with the same label as x_0. In order to generalise DeepFool to neural networks, the authors introduce an iterative approach, namely, the adversarial perturbation is updated at each iteration by approximately linearising the neural network and then perform the projection. Please note that, DeepFool is a heuristic algorithm for a neural network classifier that provides no guarantee to find the adversarial image with the minimum distortion, but in practise it is an very effective attacking method.

6.1.5. Carlini & wagner attack

C&W Attack [93] is an optimisation based adversarial attack method which formulates finding an adversarial example as image distance minimisation problem such as L_0, L_2 and L_∞-norm. Formally, it formalises the adversarial attack as an optimisation problem:

$$\ell(\upsilon) = \|\upsilon\|_p + c \cdot F(x + \upsilon), \tag{29}$$

where $x + \upsilon$ is a valid input, and F represents a surrogate function such as $x + \upsilon$ is able to fool the neural network when it is negative. The authors directly adopt the optimiser Adam [94] to solve this optimisation problem. It is worthy to mention that C&W attack can work on three distance including L_2, L_0 and L_∞ norms. A smart trick in C&W Attack lies on that it introduces a new optimisation variable to avoid box constraint (image pixel need to within $[0, 1]$). C&W attack is shown to be a very strong attack which is more effective than JSMA [72], FGSM [61] and DeepFool [92]. It is able to find an adversarial example that has a significant smaller distortion distance, especially on L_2-norm metric.

6.2. Adversarial attacks by natural transformations

Additional to the above approaches which perform adversarial attack on pixel-level, research has been done on crafting adversarial examples by applying natural transformations.

6.2.1. Rotation and translation

Engstrom et al. [95] argues that most existing adversarial attacking techniques generate adversarial images which appear to be human-crafted and less likely to be 'natural'. It shows that DNNs are also vulnerable to some image transformations which are likely occurring in a natural setting. For example, translating or/and rotating an input image could significantly degrade the performance of a neural network classifier. Fig. 8 gives a few examples. Technically, given an allowed range of translation and rotation such as ± 3 *pixels* $\times \pm 30°$, Engstrom et al. [95] aims to find the minimum rotation and translation to cause a misclassification. To achieve such an purpose, in this paper, several ideas are explored including

- a first-order iterative method using the gradient of the DNN's loss function,
- performing an exhaustive search by discretizing the parameter space,
- a worst-of-k method by randomly sampling k possible parameter values and choosing the value that cause the DNN to perform the worst.

6.2.2. Spatially transformed adversarial examples

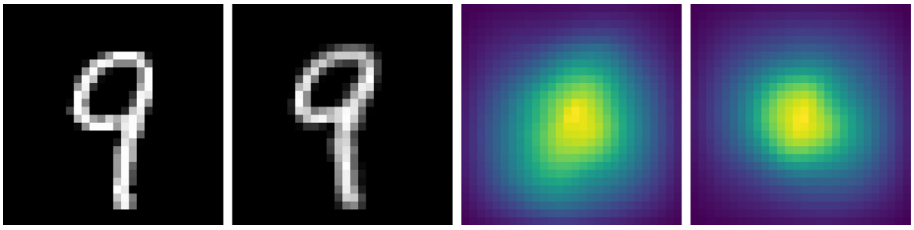
Xiao et al. [97] introduces to produce uncontrived adversarial images via mortifying the pixel's location using spatial transformations instead of directly changing its pixel value. The authors use the flow field to control the spatial transformations, which essentially quantifies the location displacement of a pixel to its new position. Fig. 9 gives a few examples. Using a bi-linear interpolation approach the generated adversarial example is differentiable w.r.t. the flow field, which then can be formulated as an optimisation problem to calculate the adversarial flow field. Technically, Xiao et al. [97] introduce a distance measure $L_{flow}(\cdot)$ (rather than the usual L_p distance) to capture the local geometric distortion. Similar to C&W attack [93], the flow field is obtained by solving an optimisation problem in which the loss function is defined to balance between the L_{flow} loss and adversarial loss. Through human study, Xiao et al. [97] demonstrate that adversarial examples based on such spatial transformation are more similar to original images based on human perception, comparing to those adversarial examples from L_p-norm based attacks such as FGSM [61] and C&W [93].

6.2.3. Towards practical verification of machine learning: The case of computer vision systems (VeriVis)

Pei et al. [99] introduce a verification framework, called VeriVis, to measure the robustness of DNNs on a set of 12 practical image transformations including reflection, translation, scale, shear, rotation, occlusion brightness, contrast, dilation, erosion, average smoothing, and median smoothing. Every transformation is controlled by a key parameter with a *polynomial-sized* domain. Those transformations are exhaustively operated on a set of input images. Then the robustness of a neural network model can be measured. VeriVis is applied to evaluate several state-of-the-art classification models, which empirically reveals that all classifiers show a significant number of safety violations.

6.3. Input-agnostic adversarial attacks

A key characteristic of the above attacks lies on that an adversarial example is generated with respect to a specific input, and therefore cannot be applied to the other input. Thus some researchers show more interests in *input-agnostic* adversarial perturbations.



(a) Original: '9' (b) Adversarial: '8' (c) x-dimension (d) y-dimension

Fig. 9. Applying spatial transformation to MNIST image of a '9' [97]. the Image (a) on the left is the original MNIST example image of a '9', and image (b) is the spatially transformed adversarial version that a simple convolutional network [98] labels as '8'. Notice how minor the difference between the two images is – the '9' digit has been very slightly 'bent' – but is sufficient for miss-classification. The flow-field that defines the spatial transformation is visualised in Image (c) (x-dimension) and Image (d) (y-dimension). The brighter areas indicate where the transformation is most intense — leftwards in the x-dimension and upwards in the y-dimension.

6.3.1. Universal adversarial perturbations

The first method on input-agnostic adversarial perturbation was proposed by Moosavi-Dezfooli et al. [90], called *universal* adversarial perturbations (UAP), since UAP is able to fool a neural network on *any* input images with high probability. Let υ the current perturbation. UAP iteratively goes through a set X of inputs sampled from the input distribution $\mathcal{X}$. At the iteration for $x_i \in X$ it updates the perturbation υ as follows. First, it finds the minimal $\Delta\upsilon_i$ w.r.t. L_2-norm distance so that $x_i + \upsilon + \Delta\upsilon_i$ is incorrectly classified by neural network $\mathcal{N}$. Then, it projects $\upsilon + \Delta\upsilon_i$ back into L_p-norm ball with a radius d to enable that the generated perturbation is sufficiently small, i.e., let

$$\begin{aligned} &\upsilon = \arg\min_{\upsilon'} \|\upsilon' - (\upsilon + \Delta\upsilon_i)\|_2 \\ &\text{s.t.} \quad \|\upsilon'\|_p \leq d. \end{aligned} \tag{30}$$

The algorithm will proceed until the empirical error of the sample set is sufficiently large, namely, no less than $1 - \delta$ for a pre-specified threshold δ.

6.3.2. Generative adversarial perturbations

By training a universal adversarial network (UAN), Hayes and Danezis [100] generalises the C&W attack in [93] to generates input-agnostic adversarial perturbations. Assume that we have a maximum perturbation distance d and an L_p norm, a UAN $\mathcal{U}_\theta$ randomly samples an input z from a normal distribution and generates a raw perturbation r. Then it is scaled through $w \in [0, \frac{d}{\|r\|_p}]$ to have $r' = w \cdot r$. Then, the new input $x + r'$ needs to be checked with DNN $\mathcal{N}$ to see if it is an adversarial example. The parameters θ is optimised by adopting a gradient descent method, similar to the one used by C&W attack [93].

Later on, Poursaeed et al. [101] introduces a similar approach in [100] to produce input-agnostic adversarial images. It first samples a random noise to input to UAN, and the output then is resized to meet an L_p constraint which is further added to input data, clipped, and then is used to train a classifier. This method is different to Hayes and Danezis [100] in two aspects. Firstly, it explores two UAN architectures including U-Net [102] and ResNet Generator [103], and find ResNet Generator works better in the majority of the cases. Secondly, this work also trained a UAN by adopting several different classifiers, thus the proposed UAP can explicitly fool multiple classifiers, which is obtained by the below loss function:

$$l_{multi-fool}(\lambda) = \lambda_1 \cdot l_{fool_1} + \cdots + \lambda_m \cdot l_{fool_m} \tag{31}$$

where l_{fool_i} denotes the adversarial loss for classifier i, m is the number of the classifiers, and λ_i is the weight to indicate the difficulty of fooling classifier i.

6.4. Other types of universal adversarial perturbations

Instead of fooling neural networks on image classification tasks, recently there are other types of universal attacking methods emerged, which target to attack DNN models in a wide range of applications including semantic image segmentation [104], image retrieval [105], speech recognition [106], and object recognition [107].

In the work of Metzen et al. [104], the authors proposed two approaches to generate universal perturbations that can adversarially fool a set of images for semantic image segmentation. The first approach in their paper is able to produce a fixed target segmentation as the output, and the second method can remove a designated target class while reserving the original segmentation outcome. This work also empirically demonstrated that UAP on semantic level has a smaller perturbation space than image classification tasks. Later on, Li et al. [105] presented a UAP method that can attack image retrieval systems, i.e., enabling an image retrieval system to put irrelevant images at the top places of a returned ranking list for a query. Since images in a retrieval system are labelled by similarity instead of labels in classification, the authors proposed to corrupt the similarity relationship of neighbourhood structures by swapping the similarity in the tuple structures. Except for those UAPs on image domain, lately [106] proposed a universal attacking method to fool speech recognition systems. In this work, the authors revealed the existence of audio-agnostic perturbations on deep speech recognition systems. They discovered that the generated universal perturbation is transferable across different DNN model architectures. Those universal adversarial perturbations further demonstrate that the venerability of DNNs to universal adversarial examples not only appears in image classification domain but also exists in a wide range of DNN-based systems.

6.5. Summary of adversarial attack techniques

Table 3 summarises the main similarities and differences of a number of surveyed adversarial attacks from five aspects: distance metric, whether the attack is targeted or un-targeted, the level of accessed information required (i.e., model structures/parameters, logits, output confidences, label), dataset tested, and core method used.

6.6. Adversarial defence

On the opposite side of adversarial attacks [6,91], researchers also show huge interests in designing various defence techniques, which are to either identify or reduce adversarial examples so that the decision of the DNN can be more robust. Until now, the developments of attack and defence techniques have been seen as

Table 3
Comparison between different adversarial attacks.

	Distance metric	Targeted/Untargeted	Accessed information	Dataset tested	Core method
L-BFGS [6]	L_2	Untarget	Model parameters	MNIST	L-BFGS
FGSM [61]	L_∞	Untarget	Model parameters	MNIST, CIFAR10	Fast linear algorithm
DeepFool [92]	$L_p, p \in [1, \infty)$	Both	Model parameters	MNIST, CIFAR10	Iterative linearisation
C & W [93]	L_0, L_2, L_∞	Both	Logits	MNIST, CIFAR10	Adam Optimiser
JSMA [72]	L_0	Both	Model parameters	MNIST, CIFAR10	Jacobian Saliency
DeepGame [19]	L_0, L_1, L_2, L_∞	Untarget	Logits	MNIST, CIFAR10	Game-based approach
L0-TRE [48]	L_0	Untarget	Logits	MNIST, CIFAR10, ImageNet	Tensor-based grid search
DLV [23]	L_1, L_2	Untarget	Model parameters	MNIST, CIFAR10, GTSRB	Layer-by-layer search
SafeCV [21]	L_0	Both	Logits	MNIST, CIFAR10	Stochastic search
Engstrom et al. [95]	N/A (Natural Transformations)	Both	Logits	MNIST, CIFAR10, ImageNet	Rotating and/or translating input images
Xiao et al. [97]	$L_{flow}(\cdot)$ (Measuring geometric distortion)	Both	Logits	MNIST, CIFAR10, ImageNet	Minimising adversarial and L_{flow} loss
VeriVis [99]	N/A (Natural Transformations)	Both	Logits	MNIST, CIFAR10, ImageNet	A set of 12 'real-world' transformations
UAP [90]	L_2 (Universal perturbation)	Both	Logits	MNIST, CIFAR10, ImageNet	Generalising DeepFool into universal adversarial attacks
UAN [100]	L_p (Universal perturbation)	Both	Logits	MNIST, CIFAR10, ImageNet	Generalising C&W into universal adversarial attacks
Poursaeed et al. [101]	L_p (Universal perturbation)	Both	Logits	MNIST, CIFAR10, ImageNet	Training a generative network
Metzen et al. [104]	Semantic Segmentation (Universal perturbation)	Target	Model parameters	Cityscapes, CamVid	Generalising UAP to segmentation tasks
Li et al. [105]	Image Retrieval (Universal perturbation)	Disturbing ranking list	Model parameters	Oxford5k, Paris6k, ROxford5k, RParis6k	Corrupting neighbourhood relationships in feature space
Neekhara et al. [106]	Speech Recognition (Universal perturbation)	Untarget	Model parameter	Mozilla Common Voice Dataset	Iterative gradient sign method to maximise the CTC-Loss

an "arm-race". For example, most defences against attacks in the white-box setting, including [108–111], have been demonstrated to be vulnerable to e.g., iterative optimisation-based attacks [73, 112].

6.6.1. Adversarial training

Adversarial training is one of the most notable defence methods, which was first proposed by Goodfellow et al. [61]. It can improve the robustness of DNNs against adversarial attacks by retraining the model on adversarial examples. Its basic idea can be expressed as below:

$$\theta^* = \arg\min_\theta \ \mathbb{E}_{(x,y)\in X} \ \ell(x; y; f_\theta). \tag{32}$$

where f_θ is a DNN parameterised by θ, $\ell(\cdot)$ is the loss function, X is the training dataset. This is improved in [113] by assuming that all neighbours within the ϵ-ball should have the same class label, i.e., local robustness. Technically, this is done by changing the optimisation problem by requiring that for a given ϵ-ball (represented as a d-Neighbourhood), to solve

$$\theta^* = \arg\min_\theta \ \mathbb{E}_{(x,y)\in X}\left[\max_{\delta\in[-\epsilon,\epsilon]^{S_1}} \ell(x+\delta; y; f_\theta)\right]. \tag{33}$$

Madry et al. [113] adopted Projected Gradient Descent (PGD) to approximately solve the inner maximisation problem.

Later on, to defeat the iterative attacks, Na et al. [114] proposed to use a cascade adversarial method which can produce adversarial images in every mini-batch. Namely, at each batch, it performs a separate adversarial training by putting the adversarial images (produced in that batch) into the training dataset. Moreover, Tramèr et al. [115] introduces ensemble adversarial training, which augments training data with perturbations transferred from other models.

6.6.2. Defensive distillation

Distillation [116] is a training procedure which trains a DNN using knowledge transferred from a different DNN. Based on this idea, Papernot et al. [117] proposes defensive distillation which keeps the same network architecture to train both the original network as well as the distilled network. It proceeds by (i) sampling a set $\{(x, f(x))\}$ of samples from the original network and training a new DNN f^1, and (ii) sampling a set $\{(x, f^1(x))\}$ of samples from the new DNN f^1 and training another new DNN f^d. It is shown that the distilled DNN f^d is more robust than the original DNN.

6.6.3. Dimensionality reduction

Some researchers propose to defense adversarial attacks by dimension reduction. For example, Bhagoji et al. [118] use Principal Component Analysis and data 'anti-whitening' reduces the high dimensional inputs for improving the resilience of various machine learning models including deep neural networks. In [119], the authors introduce a feature squeezing approach to

detect the adversarial images by "squeezing" out unnecessary input features.

6.6.4. Input transformations

A popular defence approach is to do input transformations before feeding an input into the DNN. Meng and Chen [120] suggest that an adversarial example can be either far away from existing data or close to the decision boundary. For the former, one or more separate detector networks are used to learn to differentiate between normal and adversarial examples by approximating the manifold of normal examples. For the latter, a reformer network implemented by an auto-encoder moves adversarial examples towards the manifold of normal examples so that they can be classified correctly.

On the other hand, Song et al. [121] observe that most of the adversarial images lie in the low probabilistic area of the distribution. So they train a PixelCNN generative model [122] to project the potential perturbed images into the data manifold and then greedily search an image with a highest probability, which was finally fed into DNNs. Along this line, Samangouei et al. [123] adopt a similar idea but apply Generative Adversarial Networks (GAN) [124] instead of the PixelCNN.

Guo et al. [125] exercise over five input transformations, including (i) random image cropping and re-scaling, to altering the spatial positioning of the adversarial perturbation; (ii) conducting bit-depth reduction to removes small (adversarial) variations in pixel values from an image; (iii) JPEG compression to remove small perturbations; (iv) total variance minimisation by randomly drop pixels; and (v) image quilting, which reconstructs images by replacing small (e.g., 5×5) patches with patches from clean images.

Xie et al. [126] present a defence idea through adding several randomisation layers before the neural networks. For example, for a DNN with a 299×299 input, it first uses a randomisation layer to randomly rescale the image into an image with a $[299, 331) \times [299, 331)$ size, and then takes a second randomisation layer to randomly zero-pad the image. By doing so, a number of randomised patterns can be created before an image is fed to the DNN. The paper claims that randomisation at inference time makes the network much more robust to adversarial images, especially for iterative attacks (both white-box and black box), but hardly hurts the performance on clean (non-adversarial) images.

6.6.5. Combining input discretisation with adversarial training

Input discretisation is to separate continuous-valued pixel inputs into a set of non-overlapping buckets, which are each mapped to a fixed binary vector. Similar as input transformations, input discretisation based approaches apply a non-differentiable and non-linear transformation (discretisation) to the input, before passing it into the model. In [127], the authors present an idea of using thermometer encoding in adversarial training. Because the gradient-based methods such as PGD is impossible due to the discretisation brought by the thermometer encoding in adversarial training, as an alternative, the authors introduce Logit-Space Projected Gradient Ascent (LS-PGA). This paper demonstrates that, the combination of thermometer encoding with adversarial training, can improve the robustness of neural networks against adversarial attacks.

6.6.6. Activation transformations

Dhillon et al. [128] introduces stochastic activation pruning, by adapting the activation of hidden layers on their way to propagating to the output. The idea is that, in each layer during forward propagation, it stochastically drops out nodes, retains nodes with probabilities proportional to the magnitude of their activation, and scales up the surviving nodes to preserve the dynamic range of the activations. Applying SAP to increase the robustness at the price of slightly decreasing clean classification accuracy.

6.6.7. Characterisation of adversarial region

Adversarial region is a connected region of the input domain in which all points subvert the DNN in a similar way. Summarised in [129], they are of low probability (i.e., not naturally occurring), span a contiguous multidimensional space, lie off (but are close to) the data submanifold, and have class distributions that differ from that of their closest data submanifold.

Feinman et al. [130] uses Kernel Density estimation (KDE) as a measure to identify adversarial subspaces. Given an input x and its label t, it is to compute

$$KDE(x) = \frac{1}{|X_t|} \sum_{\hat{x} \in X_t} \exp\left(\frac{|f^{K-1}(x) - f^{K-1}(\hat{x})|^2}{\sigma^2}\right) \tag{34}$$

where f^{K-1} is the function for the first $K-1$ layers, i.e., $f^{K-1}(x)$ is the logit output of the input x from f, and σ is a Gaussian standard deviation. Then based on the threshold τ, it can distinguish if x is an adversarial or natural image.

Ma et al. [129] uses Local Intrinsic Dimensionality (LID) [131] to measure the adversarial region by considering the local distance distribution from a reference point to its neighbours.

6.6.8. Defence against data poisoning attack

Instead of defencing against adversarial attacks, Steinhardt et al. [132] considers a data poisoning attack in which the attacker is to manipulate a percentage ϵ of the training dataset X to have a new dataset X_p such that $|X_p| = \epsilon|X|$. The purpose of the attacker is to mislead the defender who trains the model over the set $X \cup X_p$. The success of defence or attack is defined with respect to a loss function ℓ.

6.7. Certified adversarial defence

The approaches in Section 6.6 cannot provide guarantee over the defence results. In this subsection, we review a few principled approaches on achieving robustness. Basically, they adapt the training objective (either the loss function or the regularisation terms) to enforce some robustness constraints.

6.7.1. Robustness through regularisation in training

Attempts have been made on achieving robustness by applying dedicated regularisation terms in the training objective. Since training with an objective does not necessarily guarantee the achievement of the objective for all inputs, the robustness is approximate. For example, Hein and Andriushchenko [133] trains with a cross-entropy loss, which makes the difference $f_c(x) - f_k(x)$ as large as possible for c the class of x and all $k = 1, \ldots, s_K$, and a Cross-Lipschitz regularisation term

$$\mathcal{G}(f) = \frac{1}{nK^2} \sum_{i=1}^{n} \sum_{l,k=1}^{s_K} \|\nabla f_c(x_i) - \nabla f_k(x_i)\|_2^2 \tag{35}$$

where $\{x_i\}, i = 1..n$ is the training dataset. The goal of this regularisation term is to make the differences of the classifier functions at the data points as constant as possible. Moreover, in [134], DNNs with one hidden layer are studied. An upper bound on the worst-case loss is computed, based on a semi-definite relaxation. Since this upper bound is differentiable, it is taken as a regularisation term.

6.7.2. Robustness through training objective

Sinha et al. [135] considers the problem

$$\min_{\theta} \sup_{\mathcal{X} \in \mathcal{P}} \mathbb{E}_{\mathcal{X}}[\ell(x, y; \theta)] \tag{36}$$

where $\mathcal{P}$ is a set of distributions around the data-generating distribution $\mathcal{X}_0$, and $x \sim \mathcal{X}_0$. The set $\mathcal{P}$ includes the distributions

that are close to $\mathcal{X}_0$ in terms of the Wasserstein metric. Then, considering a Lagrangian penalty formulation of the Wasserstein metric, a training procedure is applied so that the model parameters can be updated with worst-case perturbations of training data.

Moreover, Gowal et al. [136] considers the following loss:

$$\kappa \ell(z_K, y_{true}; \theta) + (1-\kappa)\ell(\hat{z}_K(\epsilon), y_{true}; \theta) \tag{37}$$

where ℓ is the cross-entropy loss function, $\ell(z_K, y_{true}; \theta)$ is the loss of fitting data such that z_K is the output logit and y_{true} is the correct output label, and $\ell(\hat{z}_K(\epsilon), y_{true}; \theta)$ is the loss of satisfying the specification such that $\hat{z}_k(\epsilon)$ is the worst-case logit where the logit of the true class is equal to its lower bound and the other logits are equal to their upper bound. The lower and upper bounds are approximated with interval bound propagation [14]. The hyper-parameter κ is to control the relative weight of satisfying the specification versus fitting the data.

6.8. Summary of adversarial defence techniques

In summary, defence techniques are either to identify or to reduce adversarial examples. Since these do not lead to any guarantees on the robustness, the so-called "arm race" with attack techniques appears. Certified defence aims to improve on this situation by adding robustness constraints into the training procedure (either through regularisation term or training objective).

Moreover, defence techniques are closely related to the verification techniques. Every verification method can serve as a defence method, for its ability to identify the adversarial examples with guarantees. For an adversarial input, it may be classified wrongly if directly passing the DNN. To defence this, a DNN verifier can step in and determine if it is an adversarial example. If it is, the verification output can be utilised to alert that the classification from the DNN is not trustable.

7. Interpretability

Interpretability (or explainability) has been an active area of research, due to the black-box nature of DNNs. DNNs have shown the ability of achieving high precision in their predictions. However, to gain the trust from human users, it is essential to enable them to understand the decisions a DNN has made. Moreover, it is also possible that the results in interpretability can enhance other techniques, such as verification, testing, and adversarial attacks.

In the following, we review three categories of interpretability methods. First of all, instance-wise explanation, which aims to explain the decision made by a DNN on a given input, is presented in Sections 7.1, 7.2, and 7.3. Second, model explanation, which aims to provide a better understanding on the complex model, is reviewed in Sections 7.4 and 7.5. Finally, in Section 7.6, we review the recent progress of using information theoretical methods to explain the training procedure.

7.1. Instance-wise explanation by visualising a synthesised input

The first line of research aims to understand the representation learned by DNNs through visualisation over another input generated based on the current input. It focuses on convolutional neural networks (CNNs) with images as their inputs, and is mainly for instance-wise explanation. Technically, with respect to Definition 2, it is to let $t = s_1$ and $\texttt{expl}(f, x) \in \mathbb{R}^{s_1}$.

7.1.1. Optimising over a hidden neuron

Recall that $u_{k,l}(x)$ is the activation of the lth neuron on the kth layer for the input x. Erhan et al. [137] synthesises images that cause high activations for particular neurons by treating the synthesis problem as an optimisation problem as follows.

$$x^* = \arg\max_x u_{k,l}(x). \tag{38}$$

In general, the optimisation problem is non-convex, and therefore a gradient ascent based algorithm is applied to find a local optimum. Starting with some initial input $x = x_0$, the activation $u_{k,l}(x)$ is computed, and then steps are taken in the input space along the gradient direction $\partial u_{k,l}(x)/\partial x$ to synthesise inputs that cause higher and higher activations for the neuron $n_{k,l}$, and eventually the process terminates at some x^* which is deemed to be a preferred input stimulus for the neuron in question. By visualising the hidden representations, it provides insights into how the neural network processes an input.

7.1.2. Inverting representation

Mahendran and Vedaldi [138] computes an approximate inverse of an image representation. Let $rep : \mathbb{R}^{s_1} \to \mathbb{R}^{a_1}$ be a representation function, which maps an input to a representation of a_1 dimensions. Now, given a representation T_0 to be inverted, the task is to find an input x^* such that

$$x^* = \arg\min_{x \in \mathbb{R}^{s_1}} \ell(rep(x), T_0) + \lambda \mathcal{G}(x) \tag{39}$$

where ℓ is a loss function measuring the difference of two representations (e.g., Euclidean norm), $\mathcal{G}(x)$ is a regularisation term, and λ is a multiplier to balance between the two terms. For the regulariser, they use either an image prior

$$\mathcal{G}_\alpha(x) = \|x\|_\alpha^\alpha = \sum_{i=1}^{s_1} |x_i|^\alpha \tag{40}$$

where $\alpha = 6$ is taken in their experiments, or a total variation, which encourages images to consist of piece-wise constant patches. To solve the above optimisation problem (39), the classical gradient descent approach can be applied with the use of momentum to escape from saddle points. Mahendran and Vedaldi [138] argues that, while there may be no unique solution to this problem, sampling the space of possible reconstructions can be used to characterise the space of images that the representation deems to be equivalent, revealing its invariances.

Dosovitskiy and Brox [139] proposes to analyse which information is preserved by a feature representation and which information is discarded. Given a feature vector, they train a DNN to predict the expected pre-image, i.e., the weighted average of all natural images which could have produced the given feature vector. Given a training set of images and their features $\{x_i, \phi_i\}$, the task is to learn the weight w of a network $f(\phi, w)$ to minimise a Monte-Carlo estimate of the loss:

$$\hat{w} = \arg\min_w \sum_i \|x_i - f(\phi_i, w)\|_2^2 \tag{41}$$

As a side note, it has been argued in [140] that gradient-based approaches do not produce images that resemble natural images.

7.2. Instancewise explanation by ranking

The next major thread of research is to compute a ranking of a set of features. Specifically, given an input x and a set Ψ of features, we aim to find a mapping $r_x : \Psi \to \mathbb{R}$ such that each feature $\psi \in \Psi$ gets a rank $r_x(\psi)$. Intuitively, the ranking r_x provides a total order among features on their contributions to a decision made by $\mathcal{N}$ on the input x.

7.2.1. Local interpretable model-agnostic explanations (LIME)

Local Interpretable Model-agnostic Explanations (LIME) [141] interprets individual model prediction by locally approximating the model around a given prediction. Given an image x, LIME treats it as a set of superpixels. To compute the rank, it minimises the following objective function:

$$\texttt{expl}(f, x) = \arg\min_{e \in \mathcal{E}} \ell(f, e, \pi_x) + \mathcal{G}(e) \tag{42}$$

where $\ell(f, e, \pi_x)$ is a loss of taking the explanation e with respect to the original model f, under the condition of a local kernel π_x. The regularisation term $\mathcal{G}(e)$ is to penalise the complexity of the explanation e. For instance, if G is the class of linear models with weights w_e, $\mathcal{G}(e)$ can be chosen as L_0 norm $\|w_e\|_0$ to encourage sparsity, and the explainer $\texttt{expl}(f, x)$ is the sparse weights which rank the importance of dominant features.

7.2.2. Integrated gradients

Sundararajan et al. [142] suggests that a good explanation should satisfy the following two axiomatic attributes:

- sensitivity — for every input and baseline that differ in one feature but have different predictions, the differing feature is given a non-zero attribution.
- implementation invariance — the attributions are always identical for two functionally equivalent networks.

They propose the use of integrated gradients and show that it is the only method that satisfies certain desirable axioms, including sensitivity and implementation invariance. Technically, the integrated gradient along the ith dimension for an input x and a baseline x' is the following.

$$\text{IntGrad}_i(x) = (x_i - x_i') \times \int_0^1 \frac{\partial f(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \tag{43}$$

where $\frac{\partial f(x)}{\partial x_i}$ is the gradient of $f(x)$ along the ith dimension of input x. The quantity $\text{IntGrad}_i(x)$ is used to indicate the contribution of x_i to the prediction $f(x)$ relative to a baseline input x'. If $f(\cdot)$ is differentiable everywhere, then

$$\sum_{i=1}^{s_1} \text{IntGrad}_i(x) = f(x) - f(x'), \tag{44}$$

where s_1 is the input dimensions. For a given baseline x' subject to $f(x') \approx 0$, an explainer can distribute the output to the individual features of the inputs.

7.2.3. Layer-wise relevance propagation (LRP)

Given an input–output mapping (for example a DNN) $f : \mathbb{R}^{s_1} \to \mathbb{R}$, the layer-wise relevance propagation (LRP) method [143] is a concept of pixel-wise decomposition

$$f(x) \approx \sum_{i=1}^{s_1} R_i \tag{45}$$

to understand how much a single pixel i contributes to the prediction $f(x)$. By propagating backward the prediction probability of the input through DNN and calculating the relevance scores, LRP attributes the importance scores to the pixels. Intuitively, the importance score, R_i, represents the local contribution of the pixel d to the prediction function $f(x)$. It is suggested in [144] that LRP is equivalent to the Grad $\odot$ Input method (to be reviewed in Section 7.3.1) when the reference activations of all neurons are fixed to zero.

7.2.4. Deep learning important features (DeepLIFT)

Shrikumar et al. [144] is a recursive prediction explanation method. It attributes to each input neuron x_i a value $C_{\Delta x_i \Delta y}$ that represents the effect of that input neuron being set to a reference value as opposite to its original value. The reference value, chosen by the user, represents a typical uninformative background value for the feature. DeepLIFT uses a "summation-to-delta" property that states:

$$\sum_{i=1}^{s_1} C_{\Delta x_i \Delta y} = \Delta o, \tag{46}$$

where $o = f(x)$ is the method output, $\Delta o = f(x) - f(r)$, $\Delta x_i = x_i - r_i$, and r is the reference input. DeepLIFT improves over the canonical gradient-based methods by placing meaningful importance scores even if the gradient is zero, and avoiding discontinuities due to bias terms.

7.2.5. Gradient-weighted class activation mapping (GradCAM)

By flowing the gradient information into the last convolutional layer of CNNs, Gradient-weighted Class Activation Mapping (GradCAM) [145] computes a feature-importance map (i.e., a coarse localisation) highlighting regions in the image corresponding to a certain concept. Specifically, GradCAM computes the feature importance scores of a feature map k for a class c as follows.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{47}$$

where $\frac{\partial y^c}{\partial A_{ij}^k}$ is the gradients of the score via backpropagation with respect to feature maps A^k of a convolutional layer, and the sums aim at global average pooling. The weights of the feature maps are used to indicate the importance of the input. Intuitively, α_k^c represents a partial linearisation of the deep network downstream from A^k, and captures the 'importance' of feature map k for a target class c. Furthermore, Guided Grad-CAM obtains the more fine-grained feature importance scores, by multiplying the feature importance scores obtained from Grad-CAM with those from Guided Backpropagation in an elementwise manner.

7.2.6. SHapley additive explanation (SHAP)

SHapley Additive exPlanation (SHAP) [146] suggests taking an additive model

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z_i'$$

as an explanation model, where M is the number of simplified input features, $z' \in \{0, 1\}^M$, and $\phi_i \in \mathbb{R}$ is a coefficient. It shows that under three properties (local accuracy, missingness, consistency), there is only one possible explanation model as follows.

$$\texttt{expl}_i(f, x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)] \tag{48}$$

where $|z'|$ is the number of non-zero entries in z', and $z' \subseteq x'$ represents all z' vectors where the non-zero entries are a subset of the non-zero entries in x'. Intuitively, $\texttt{expl}_i(f, x)$ – the Shapley values from cooperative game theory – is an importance value to each feature i that represents the effect on the model prediction of including that feature.

7.2.7. Prediction difference analysis

Zintgraf et al. [147] presents a prediction difference analysis (PDA) method to visualise the influence corresponding to a special input change or element removal. The idea is to assign a relevance value to each input feature w.r.t. a class c, so that the influence in terms of prediction difference can be measured by evaluating the difference between two conditional probabilities $p(c|x)$ and $p(c|x_{-i})$, where

$$p(c|x_{-i}) = \sum_{x_i} p(x_i|x_{-i})p(c|x) \tag{49}$$

calculates the conditional probability if x_i is removed from the input x.

7.2.8. Testing with concept activation vector (TCAV)

Kim et al. [148] argues that, since most machine learning models operate on features, such as pixel values, that do not correspond to high-level concepts that humans understand, existing ranking-based approaches do not produce an explanation that can be easily accessible to humans. It then works on the idea of supplementing high level concepts (i.e., concept activation vector, CAV) into this explanation to make the final explanation closer to human understandable explanation. The high-level concepts are learned independently from other user-provided data. Specifically, Testing with CAV (TCAV) uses directional derivatives

$$S_{C,k,l}(x) = \nabla h_{l,k}(f_l(x)) \cdot v_C^l \tag{50}$$

to rank a user-defined concept with respect to a classification result, according to TCAV score

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|\{x \in X_k : S_{C,k,l}(x) > 0\}|}{|X_k|} \tag{51}$$

where C is a concept, l is a layer, k is a class label, $h_{l,k} : \mathbb{R}^{s_l} \to \mathbb{R}$ maps the activations at layer l to the logit output (i.e., the layer $K-1$) of a class k, $v_C^l \in \mathbb{R}^{s_l}$ is a unit CAV vector for a concept at layer l, and $f_l(x)$ is the activation for input x at layer l.

7.2.9. Learning to explain (L2X)

Learning to explain (L2X) [149] utilises an instancewise feature selection for model interpretation. Roughly speaking, for a given input x, among a set of n features, it is to choose k most informative features, which are of most relevance to the output $y = f(x)$. Let $\mathscr{P}_k = \{S \subset 2^{s_1} \mid |S| = k\}$ be the set of all subsets of size k. An explainer $\texttt{expl}(f, \cdot)$ of size k is a mapping from the feature space $\mathbb{R}^{s_1}$ to the power set $\mathscr{P}_k$. Then, by modelling the input and the output as random variables X and Y respectively, the explanation problem is to find a solution to the following optimisation problem:

$$\max_e I(X_S; Y) \text{ subject to } S \sim e(X) \tag{52}$$

where $I(X_S; Y)$ represents the mutual information between the selected features X_S and the output Y, and $e(\cdot)$ is the specific feature selection explainer. Intuitively, the explainer is to find a set of features that maximise the statistical dependence. between selected features and the output.

7.3. Instancewise explanation by saliency maps

While the ranking based methods are often used to generate saliency maps for visualisation purpose, the methods surveyed in this subsection is to compute saliency maps without computing a ranking score.

7.3.1. Gradient-based methods

Gradient [150]. Given an image classification model, let $S_c(x)$ be a score function for an image x with respect to the class c. By back-propagation method, it aims to find an input locally optimising

$$\max_x S_c(x) - \lambda \|x\|_2^2 \tag{53}$$

where $S_c(x)$ can be approximated to a linear function $S_c(x) = w^T x + b$ by first-order Taylor expansion in the neighbourhood of a reference image x_0, so that $w_c(x_0) = \frac{\partial S_c}{\partial x}\big|_{x_0}$ serves as an explanation map. Such gradient indicates the difference that a tiny change of each pixel of the input x affects the classification score c.

To sharpen the sensitive maps, **SmoothGrad** [151] randomly perturbs the input x with a small noise and averages the resulting explanation maps, i.e., $\hat{w}_c(x) = \frac{1}{n}\sum_{i=1}^n w_c(x + g_i)$, where $g_i \sim \mathcal{N}(0, \sigma^2)$ is a Gaussian noise enabling random sampling around x.

Moreover, the **Grad** $\odot$ **Input** [144] method yields another explanation $\hat{w}_c(x) = x \odot w_c(x)$ to reduce visual diffusion. **DeConvNet** [152] highlights the portions of a particular image that are responsible for the firing of each neural unit. **Guided Backpropagation** [153] builds upon DeConvNet and sets negative gradients to zero during backpropagation.

7.3.2. Perturbation-based methods

Dabkowski and Gal [154] proposes an accurate saliency detection method that manipulates the classification output by masking certain regions of the input image. Two concepts of saliency maps were considered: a smallest sufficient region (SSR) that allows for a confident classification, and a smallest destroying region (SDR) that prevents a confidence classification if removed. For instance, it applies a mask (e.g., a binary matrix) to the image data matrix by e.g., element-wise product, so as to set certain regions of the image to zero. A masking model can be trained to find such a mask for any input image in a single feed-forward pass.

Fong and Vedaldi [155] proposes a general framework that inspects how the output of a black-box model is affected by an input perturbation. Given a mask function $m : \Lambda \mapsto [0, 1]$, a meaningful perturbation by constant mapping, adding noise, and blurring over a pixel $u \in \Lambda$ can be respectively defined by

$$\Phi(u) = \begin{cases} m(u)x(u) + (1 - m(u))\mu, & \text{Constant mapping} \\ m(u)x(u) + (1 - m(u))\eta(u), & \text{Noising} \\ \int g(v - u)x(v)\mathrm{d}v, & \text{Blurring} \end{cases} \tag{54}$$

where the constant μ is the average colour, $\eta(u)$ is a random Gaussian noise i.i.d. over pixels, and $g(\cdot)$ is a Gaussian blur kernel.

7.4. Model explanation by influence functions

The empirical influence function, a classic technique from robust statistics, is a measure of the dependence of the estimator on the value of one of the points in the sample. It has been utilised to interpret the effects of training data to the model. Koh and Liang [156] considers influence functions to trace a model's prediction through the learning algorithm and back to its training data, without retraining the model. Basically, it can be done by e.g., upweighting a training point or perturbing a training input. For instance, given the training points $\{z_i = (x_i, y_i)\}_{i=1}^n$ and the loss function $\ell(z, \theta)$ w.r.t. parameters $\theta \in \Theta$, the idea of unweighting is to compute the parameter change with respect to an infinitesimal ϵ,

$$\hat{\theta}_{\epsilon,z} = \arg\min_{\theta \in \Theta} \frac{1}{n}\sum_{i=1}^n \ell(z_i, \theta) + \epsilon\ell(z, \theta) \tag{55}$$

and evaluate the influence by

$$\mathcal{I}(z) = \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\Big|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta \ell(z,\hat{\theta}) \tag{56}$$

where $H_{\hat{\theta}} = \frac{1}{n}\sum_{i=1}^{n}\nabla_\theta^2 L(z_i,\hat{\theta})$ is the Hessian. The influence of the removal of one specific training point z from the training set can be directly evaluated by calculating the parameter change

$$\hat{\theta}_{-z} - \hat{\theta} \approx -\frac{1}{n}\mathcal{I}(z) \tag{57}$$

where $\hat{\theta}_{-z} = \arg\min_{\theta\in\Theta}\frac{1}{n}\sum_{z_i\neq z} L(z_i,\theta)$ is the new parameter due to the removal of the training point z.

7.5. Model explanation by simpler models

Interpretability can be achieved by approximating the neural network (either a feedforward neural network or a recurrent neural network) with a simpler model (or a set of simpler models), on which an intuitive explanation can be obtained. We remark that, some local interpretability methods in Section 7.2 – such as LIME and SHAP – computes a simpler model, e.g., an additive linear model, for every input sample. Because the simpler models are different for different input samples, we do not consider them as model explanation methods, which intend to use a single simpler model to explain the neural network (and its decision on all input samples).

7.5.1. Rule extraction

Ribeiro et al. [157] ties a prediction locally to a decision rule (represented as a set of predicates), based on the assumption that while the model is globally too complex to be explained succinctly, "zooming in" on individual predictions makes the explanation task feasible. Let A be a rule, such that $A(x) = 1$ if all its feature predicates are true for the input x. A is an anchor (i.e., a valid explanation) if

$$\mathbb{E}_{\mathcal{D}(z|A)}[f(x) = f(z)] > 1-\epsilon \text{ and } A(x) = 1 \tag{58}$$

where $\mathcal{D}(\cdot|A)$ is the conditional distribution when the rule A applies, and $f(x) = f(z)$ represents that the inputs x and z have the same label. Intuitively, it requires that for inputs on which the anchor holds, the prediction is (almost) always the same.

7.5.2. Decision tree extraction

Extraction of decision tree from complex models is a popular thread of research. An example of this is [158], which interprets DNN by a decision tree representation, in which decision rules are extracted from each layer of DNN by decomposition. It is based on Continuous/discrete Rule Extractor via Decision tree induction (CRED) [159] and C4.5 decision tree learning algorithm.

7.5.3. Linear classifiers to approximate piece-wise linear neural networks

Chu et al. [160] interprets piecewise linear neural networks (PLNN) by a set of locally linear classifiers. In PLNN, activation functions are piecewise linear such as ReLU. Given an input x, its corresponding activation pattern, i.e., the states of all hidden neurons, is fixed and equivalent to a set of linear inequalities. Each inequality represents a decision feature, and therefore the interpretation of x includes all the decision features. Moreover, because all inputs sharing the same activation pattern form a unique convex polytope, the interpretation of an input x can also include the decision features of the polytope boundaries.

7.5.4. Automata extraction from recurrent neural networks

Weiss et al. [161] extends the L* automata learning algorithm [162] to work with recurrent neural networks (RNNs). L* algorithm requires a mechanism to handle two types of queries: membership query and equivalence query. Let A be the learning automaton. For membership query, the RNN classifier is used to check whether an input is correctly classified. For equivalence query, it takes an automaton A^p generated by Omlin and Lee Giles [163] and check if A and A^p are equivalent. If there is a disagreement between the two automata, i.e., an input has different classes, it will determine whether to return the input as a counterexample or to refine the automata A and restart the comparison. This process iterates until it converges (i.e., A and A^p are equivalent) or a specific limit has been reached.

7.6. Information-flow explanation by information theoretical methods

Information theory is increasingly believed to be one of the most promising tools to open the black-box of DNN. The key building block is an information bottleneck method originally proposed by Tishby et al. [164].

7.6.1. Information bottleneck method

Given two discrete real-valued variables X and Y with joint distribution $p(x,y)$, the objective of the information bottleneck (IB) method [164] is to figure out the stochastic encoder $p(t|x)$, i.e.,

$$\text{IB: } \min_{p(t|x)} \quad \mathcal{L}_{\text{IB}} = I(X;T) - \beta I(T;Y) \tag{59a}$$

$$\text{subject to} \quad T \leftrightarrow X \leftrightarrow Y \tag{59b}$$

for $\beta > 1$, where $T \leftrightarrow X \leftrightarrow Y$ forms a Markov chain and $I(X;T)$ represents the mutual information between variables X and T. Otherwise if $\beta \leq 1$, the optimal solution is degenerated $I(X;T) = I(T;Y) = 0$. The information-flow explanation $\texttt{expl}(\mathcal{F})$ is the solution to the above optimisation problem, stochastic encoder $p(t|x)$ and decoder $p(y|t)$, which can be given by

$$p(t|x) = \frac{p(t)}{Z(x,\beta)}\exp[-\beta D_{KL}[p(y|x) \mid p(y|t)]] \tag{60}$$

$$p(y|t) = \frac{1}{p(t)}\sum_x p(t|x)p(x,y) \tag{61}$$

where the normalisation factor $Z(x,\beta)$ is given by

$$Z(x,\beta) = \exp[\frac{\lambda(x)}{p(x)} - \beta\sum_y p(y|x)\log\frac{p(y|x)}{p(y)}] \tag{62}$$

and $\lambda(x)$ is a Lagrange multiplier. The computation of the above quantities can be obtained by applying the iterative Blahut–Arimoto algorithm, which however becomes computationally intractable when $p(x,y)$ is high-dimensional and/or non-Gaussian distributed.

7.6.2. Information plane

Given a K-layer deterministic DNN with T_k being a multivariate random variables representing the output of kth hidden layer, Shwartz-Ziv and Tishby [165] introduced the concept of information plane with coordinates $(I(X;T_k), I(T_k,Y))$.[3]

[3] According to Goldfeld et al. [166], the quantity computed and plotted in [165] is actually $I(X;\text{Bin}(T_k))$ rather than $I(X;T_k)$, where $\text{Bin}(\cdot)$ is "a per-neuron discretisation of each hidden activity of T_k into a user-selected number of bins".

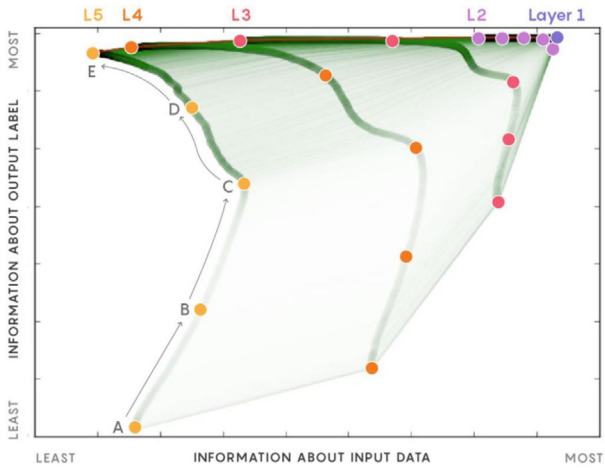


Fig. 10. The evolution of Information Plane ($I(X;T_k), I(T_k;Y)$) during the training procedure of a 5-layer DNN [167], where A-B-C indicates the fitting phase, and C-D-E is the compression phase.

By assuming the Markov chain of K-layer DNN, the information bottleneck method turns to

$$\text{IB:} \min_{p(t_k|x)} \quad \mathcal{L}_{\text{IB}} = I(X;T_k) - \beta I(T_k;Y) \tag{63a}$$

$$\text{subject to} \quad Y \leftrightarrow X \leftrightarrow T_1 \leftrightarrow \ldots \leftrightarrow T_K \leftrightarrow \hat{Y} \tag{63b}$$

where the coordinates of the information planes follow

$$I(X;Y) \geq I(T_1;Y) \geq \cdots \geq I(T_K;Y) \geq I(\hat{Y};Y) \tag{64}$$

$$H(X) \geq I(X;T_1) \geq \cdots \geq I(X;T_K) \geq I(X;\hat{Y}). \tag{65}$$

In doing so, DNN training procedure can be interpreted by visualising the information planes with respect to the SGD layer dynamics of the training. It was suggested by Shwartz-Ziv and Tishby [165] that the training procedure consists of two phases (see Fig. 10):

- Fitting Phase: The neurons in higher layers learn as much information as possible about the inputs, and try to fit it to the outputs. In this phase, both $I(X;T_k)$ and $I(T_k;Y)$ increase, indicating that neurons are learning.
- Compression Phase: Higher layer neurons try to compress their representation of the input, while keep the most relevant information to predict the output. In this phase, while $I(T_k;Y)$ still keeps increasing, $I(X;T_k)$ is decreasing, which indicates the neurons try to forget what they have learnt in the fitting phase but only retain the most relevant information for the output prediction.

It was also claimed in [165] that the compression phase is responsible for the generalisation performance of DNN. Recently, such a fresh perspective for interpreting DNN training process has attracted a lot of attention, and also caused the ongoing debate on whether or not it is possible to translate theoretical insights into DNN optimisation in practice.

7.6.3. From deterministic to stochastic DNNs

It is argued by several papers (e.g., [166,168,169]) that the IB method suffers from some issues on explaining the training process of deterministic DNNs. The main issue is that, the quantity $I(X,T_k)$, whose decrease leads to compression, will not change because it is either a constant value (when X is discrete) or infinite (when X is continuous), given a deterministic DNN.

To resolve this issue, several stochastic DNN frameworks have been proposed to transform deterministic DNNs to stochastic ones and make the compression term $I(X,T_k)$ more meaningful. For instance, Saxe et al. [168] suggested injecting an independent Gaussian noise with user-defined variance to T_k followed by quantisation. Amjad and Geiger [169] suggested including a (hard or soft) decision rule to increase the dynamicity or replacing $I(X,T_k)$ by other similar yet well-behaved cost functions. Goldfeld et al. [166] suggested relating the mapping $X \mapsto T_k$ to the input and the output of a stochastic parameterised communication channel with parameters being DNN's weights and biases.

This is still an active ongoing research to interpret how DNNs get trained from an information-theoretic perspective, in the hope to guide DNN practice.

7.7. Summary

In this section, we reviewed the main approaches for DNN interpretability from data, model, and information perspectives. They are not mutually-exclusive but complement, and some techniques can be combined to provide instance-wise, model, and/or information-flow explanation in different dimensions.

Table 4 summarises some papers according to three aspects, i.e., explanation subject, explanation scope, and whether the method is model agnostic or not. Explanation scope can be either local or global. For local explanation, it only considers the function f working on the local region close to an instance. For global explanation, it is to explain the entire input domain.

8. Future challenges

Research on enhancing the safety and trustworthiness of DNNs is still in its infancy. In the following, we discuss a few prominent challenges.

8.1. Distance metrics closer to human perception

Distance metrics are key building blocks of the techniques we surveyed, and are mainly used to measure the similarity between two inputs. Ideally, two inputs with smaller distance should be more similar with respect to human perception ability. Given the fact that it is hard to measure human perception ability, well-known metrics including L_0, L_1, L_2, and L_∞-norms, are applied. While most techniques are orthogonal to the consideration of distance metrics (under the condition that the distances can be efficiently computed), it has been argued that better metrics may be needed. For example, Lu et al. [170] argued that adversarial examples found by a few existing methods (mostly based on L_2 or L_∞) are not necessarily physical, and Yosinski et al. [140] claims that gradient-based approaches (where the computation of gradients are usually based on certain norm distance) for interpretability do not produce images that resemble natural images.

Lipton [171] states that the demand for interpretability arises when there is a mismatch between the formal objectives of supervised learning (test set predictive performance) and the real world costs in a deployment setting. Distance metrics are key components in various training objectives of supervised learning. Actually, for DNNs on perception tasks, it is the mismatch of distance metrics used in the training objective and used by human to differentiate objects that hinders an intuitive explanation of a given decision.

For pattern recognition and image classification, there are other image similarity distances proposed, such as the structure similarity measure SSIM [65], but they are restricted to the computational efficiency problem and the existence of analytical solution when computing gradients. It will be interesting to understand if the application of such metrics become more practical for the tasks we are concerned with when more efficient computational methods such as Bruni and Vitulano [172] become available.

Table 4
Comparison between different interpretability techniques.

	Explanation subject	Explanation scope	Model agnostic
Erhan et al. [137]	Instancewise	Local	No
Mahendran and Vedaldi [138]	Instancewise	Local	No
Dosovitskiy and Brox [139]	Instancewise	Local	No
Ribeiro et al. [141]	Instancewise	Local	Yes
Sundararajan et al. [142]	Instancewise	Local	No
Bach et al. [143]	Instancewise	Local	No
Shrikumar et al. [144]	Instancewise	Local	No
Selvaraju et al. [145]	Instancewise	Local	No
Lundberg and Lee [146]	Instancewise	Local	Yes
Zintgraf et al. [147]	Instancewise	Local	Yes
Kim et al. [148]	Instancewise	Local	No
Chen et al. [149]	Instancewise	Local	No
Simonyan et al. [150]	Instancewise	Local	No
Smilkov et al. [151]	Instancewise	Local	No
Shrikumar et al. [144]	Instancewise	Local	No
Zeiler and Fergus [152]	Instancewise	Local	No
Springenberg et al. [153]	Instancewise	Local	No
Dabkowski and Gal [154]	Instancewise	Local	Yes
Fong and Vedaldi [155]	Instancewise	Local	Yes
Koh and Liang [156]	Model	Global	No
Ribeiro et al. [157]	Model	Local	Yes
Zilke et al. [158]	Model	Global	No
Chu et al. [160]	Model	Local	No
Weiss et al. [161]	Model	Global	Yes
Tishby et al. [164]	Information flow	Global	Yes
Shwartz-Ziv and Tishby [165]	Information flow	Global	Yes
Saxe et al. [168]	Information flow	Global	Yes
Amjad and Geiger [169]	Information flow	Global	Yes
Goldfeld et al. [166]	Information flow	Global	Yes

8.2. Improvement to robustness

Traditional verification techniques aim to not only verify (software and hardware) systems when they are correct, but also find counterexamples whenever they are incorrect. These counterexamples can be used to either directly improve the system (e.g., counterexample-guided abstract refinement [173], etc.) or to provide useful information to the system designers for them to improve the system. This is similar for the software testing, where bugs are utilised to repair implementation errors (e.g., automatic repair techniques [174], etc.) and for the programmers to improve their implementation. While similar expectation looks reasonable since existing DNN verification and testing tools are able to find e.g., counterexamples to local robustness (i.e., adversarial examples), it is relatively unclear how these counterexamples can be utilised to improve the correctness (such as the local robustness) of the DNN, other than the straightforward method of adding the adversarial examples into the training dataset, which may lead to bias towards those input subspaces with adversarial examples. Section 6.6 reviews a few techniques such as adversarial training.

Another pragmatic way can be to design a set of quantitative metrics to compare the robustness of DNNs with different architectures (for example different numbers and types of hidden layers) so that a DNN designer is able to diagnose the DNN and figure out a good architecture for a particular problem. Relevant study has been started in e.g., [55,58], with some preliminary results reported in the experiments. A significant next step will be to automate the procedure and synthesise an architecture according to some given criteria.

8.3. Other machine learning models

Up to now, most efforts are spent on feedforward DNNs, with image classification as one of the main tasks. Research is needed to consider other types of neural networks, such as deep reinforcement learning models [175–178] and recursive neural networks, and other types of machine learning algorithms, such as SVM, k-NN, naive Bayesian classifier, etc. Most deep reinforcement learning models use feedforward DNNs to store their learned policies, and therefore for a single observation (i.e., an input) similar safety analysis techniques can be applied. However, reinforcement learning optimises over the objectives which may base on the rewards of multiple, or even an infinite number of, time steps. Therefore, other than the DNN, the subject of study includes a sequence of, instead of one, observations. A few attack techniques, such as Huang et al. [179],Pattanaik et al. [180],Lin et al. [181] and Tretschk et al. [182], have been proposed, with the key techniques based on generalising the idea of FGSM [61]. For recurrent neural networks, there are a few attack techniques such as Wei et al. [183]. Less work has been done for verification, testing, and interpretability.

8.4. Verification completeness

Additional to the properties we mentioned in Section 3, the correctness of a DNN may include other properties. More importantly, the properties in Section 3 are expressed in different ways, and for each property, a set of ad hoc analysis techniques are developed to work with them. See e.g., Table 1 for a comparison of verification techniques. Similar to the logic languages LTL (linear time logic) and CTL (computational tree logic) which can express various temporal properties related to the safety and liveness properties of a concurrent system, research is needed to develop a high-level language that can express a set of properties related to the correctness of a DNN. Such a language will enable the development of general, instead of ad hoc, formal analysis techniques to work with various properties expressible with that language. The development of a high-level specification language for DNNs is hindered by the lack of specifications for data-driven machine learning techniques, which learn the model directly from a set of data samples. A possible direction can be to obtain specifications from the training data, e.g., by studying how the data samples are scattered in the input high-dimensional space.

8.5. Scalable verification with tighter bounds

Existing verification approaches either cannot scale to work with state-of-the-art networks (e.g., for constraint-solving based approaches) or cannot achieve a tight bound (e.g., for over-approximation approaches). After the initial efforts on conducting exact computation, such as Katz et al. [14],Huang et al. [23], Lomuscio and Maganti [32] and Ehlers [29], recent efforts have been on approximate techniques to alleviate the computational problem while still achieve tangible bounds, e.g., [15,21,38]. Significant research efforts are required to achieve tight bounds with approximate techniques for state-of-the-art DNNs. It can be hard

to work with real-world models which usually contain complex structures and lots of real-valued parameters. A possible direction will be to develop an abstraction and refinement framework, as Clarke et al. [184] did for concurrent systems, and it will be interesting to see how it is related to the layer-by-layer refinement [23].

8.6. Validation of testing approaches

Up to now, testing DNNs is mainly on coverage-based testing, trying to emulate the structural coverage testing developed in software testing. However, different from traditional (sequential) software in which every input is associated with a single activated path and eventually leads to an output, in DNNs every input is associated with a large set of activated paths of neurons and the output is collectively determined by these paths, i.e., activation pattern [63,64]. Such semantic difference suggests that, a careful validation of the coverage-based testing is needed to make sure that the extended methods can work effectively in the context of DNNs.

In particular, for most proposals up to now, neurons are treated as the most basic elements in the coverage definitions and are regarded as the variables in the traditional software. However, unlike a variable which usually holds certain weight in determining the execution path, a single neuron in most cases cannot solely determine, or change, the activation path of an input. Therefore, the testing coverage based on neurons does not examine the actual functionality of DNNs. It can be reasonable to consider emulating variables in traditional software with a set of neurons (instead of a single neuron) and therefore let paths be the sequences of sets of neurons. A preliminary study appears in [55] with more sophisticated design on the coverage metrics required. The lift of the most basic element from neuron to a set of neurons will also affect the design of test case generation algorithms. Moreover, it is expected that interpretability techniques can be employed as building blocks for test case generation algorithms.

8.7. Learning-enabled systems

As suggested in e.g., [185], real-world autonomous systems contain both logic components – to handle e.g., high-level goal management, planning, etc. – and data-drive learning components – to handle e.g., perception tasks, etc. To analyse such learning-enabled systems, methods are needed to interface the analysis techniques for individual components. Compositional and assume-guarantee reasoning can be applied in this context. Significant efforts are needed to be on a few topics including e.g., how to utilise logic components to identify safety properties of DNN components (e.g., a set of constraints DNN components need to satisfy, a subspace of the input domain needed to be considered, etc.), how the safety assurance cases of DNN components can be streamlined into the assurance cases of the whole system, etc.

Verification techniques have been developed to work with learning-enabled systems. In [186], a compositional framework is developed for the falsification of temporal logic properties of cyber physical systems with ML components. Through experiments on a dynamic tracking system based on high resolution imagery input, Sun et al. [187] studies the uncertainties eliminated and introduced from the interaction between components. Based on this, Huang et al. [188] suggests a formal verification technique to work with robustness and resilience of learning-enabled state estimation systems.

For the safety assurance of learning-enabled systems, it would be useful to understand how low level evidences – obtained via e.g., verification, testing, etc. – can be utilised to reason about high level assurance goals such as the system can operate correctly in the next 100 days with probability more than 99%. Research on this has started in [189], where a Bayesian inference approach is taken.

8.8. Distributional shift, out-of-distribution detection, and run-time monitoring

DNNs are trained over a set of inputs sampled from the real distribution. However, due to its high-dimensionality, the training dataset may not be able to cover the input space. Therefore, although it is reasonable to believe that the resulting trained models can perform well on new inputs close to the training data, it is also understandable that the trained models might not perform correctly in those inputs where there is no neighbouring training data. While techniques are being requested to achieve better generalisability for DNN training algorithm including various regularisation techniques (see e.g., [190] for a comprehensive overview), as suggested in e.g., [59,191,192], it is also meaningful (particularly for the certification of safety critical systems) to be able to identify those inputs on which the trained models should not have high confidence. Technically, such inputs can be formally defined as both topologically far away from training data in the input space and being classified with high probability by the trained models. Moreover, Abbasi et al. [193] suggests having an extra class "dustbin" for such inputs.

The distributional shift has been studied from the perspective of out-of-distribution detection, which checks whether an input is on the distribution from which the training data is sampled. Research on this direction has been active, see e.g., [194,195].

The ubiquity of experiencing distributional shift in the application of deep neural networks, and the difficulty of having the verification completeness, suggest the importance of developing run-time monitoring techniques to enable the detection of safety problems on-the-fly. Research is needed to understand which quantity or property is to monitor, although some suggestions have been made including e.g., the activation patterns [196].

8.9. Unifying formulation of interpretability

There is a lack of consistent definitions of interpretability, although a number of approaches have been surveyed (e.g., in [171]) from different angles. While the existing research is able to provide various partial information about the DNNs, it is hard to compare them under a systematic framework. Recently, there were some attempts to accommodate several similar approaches to a single unified framework, to facilitate the comparison. For example, Olah et al. [197] suggested that some visualisation methods could be fitted in the optimisation problem with different regularisation terms, Lundberg and Lee [146] used Shapley value to explain a few attribute-based approaches with an additive model, and Ancona et al. [198] used a modified gradient function to accommodate a few gradient-based approaches. Although it may be difficult, if not impossible, to have a unified definition for interpretability, it is necessary to define it in a consistent way for the purpose of comparison. It is hoped that our attempt in defining interpretability from different dimensions of data, model, and information could shed light on such a unifying formulation.

8.10. Application of interpretability to other tasks

Except for the size of DNNs, the key difficulty of verifying, testing, or attacking DNNs is due to the fact that DNNs are black-box. It is therefore reasonable to expect that, the interpretation results will be able to enable better verification and testing approaches. For testing, interpretability can potentially enhance, or refine,

the design of coverage metrics and enable more efficient test case generation algorithms. For verification, it is expected that interpretability can help identify more specifications to enhance the verification completeness. The applications of interpretability in attack techniques have been seen in e.g., [72] and [48], where a ranking over the input dimensions provides heuristics to find good adversarial examples.

8.11. Human-in-the-loop

All the techniques reviewed are to improve the trust of human users on the DNNs through the angles of certification and explanation. Certification techniques improve the confidence of the users on the correctness of the DNNs, and the explanation techniques increase human users' understanding about the DNNs and thus improve the trust. These can be seen as a one-way enhancement of confidence from the DNNs to the human users. The other direction, i.e., how human users can help improve the trustworthiness of the DNNs, is less explored. There are only a few works such as [199], where a visual analytic interface is presented to enable expert user by interactively exploring a set of instance-level explanations.

Trust is a complex notion, viewed as a belief, attitude, intention or behaviour, and is most generally understood as a subjective evaluation of a truster on a trustee about something in particular, e.g., the completion of a task [200]. A classical definition from organisation theory [201] defines trust as the willingness of a party to be vulnerable to the actions of another party based on the expectation that the other will perform a particular action important to the trustor, irrespective of the ability to monitor or control that party. It is therefore reasonable to assume that the interaction between the DNNs and their human users can significantly affect the trust, and the trust is not a constant value and can be fluctuated. The formulation of the changes of trust in terms of the interaction has been started in [202] with a comprehensive reasoning framework.

9. Conclusions

In this survey, we reviewed techniques to determine and improve the safety and trustworthiness of deep neural networks, based on the assumption that trustworthiness is determined by two major concepts: certification and explanation. This is a new, and exciting, area requiring expertise and close collaborations from multidisciplinary fields including formal verification, software testing, machine learning, and logic reasoning.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. (IJCV) 115 (3) (2015) 211–252.

[2] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, P. Kuksa, Natural language processing (almost) from scratch, J. Mach. Learn. Res. 12 (2011) 2493–2537.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the game of Go without human knowledge, Nature 550 (354–359) (2017).

[4] NTSB releases preliminary report on fatal Tesla crash on autopilot, 2018, https://electrek.co/2018/06/07/tesla-fatal-crash-autopilot-ntsb-releases-preliminary-report/.

[5] Why Uber's self-driving car killed a pedestrian, 2018, https://www.economist.com/the-economist-explains/2018/05/29/why-ubers-self-driving-car-killed-a-pedestrian.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, R. Fergus, Intriguing properties of neural networks, in: In ICLR, Citeseer, 2014.

[7] P. Ammann, J. Offutt, Introduction to Software Testing, Cambridge University Press, 2008.

[8] E.M. Clarke Jr., O. Grumberg, D. Kroening, D. Peled, H. Veith, Model Checking, The MIT Press, 2018.

[9] J. Rushby, The Interpretation and Evaluation of Assurance Cases, Technical report, SRI International, 2015.

[10] Explainable artificial intelligence, 2018, https://www.darpa.mil/program/explainable-artificial-intelligence.

[11] P. Voosen, How AI detectives are cracking open the black box of deep learning, Science (2017).

[12] General data protection regulation, 2016, http://data.europa.eu/eli/reg/2016/679/oj.

[13] V. Nair, G.E. Hinton, Rectified linear units improve restricted Boltzmann machines, in: Proceedings of the 27th International Conference on Machine Learning (ICML), 2010, pp. 807–814.

[14] G. Katz, C. Barrett, D.L. Dill, K. Julian, 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.

[15] W. Ruan, X. Huang, M. Kwiatkowska, Reachability analysis of deep neural networks with provable guarantees, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, AAAI Press, 2018, pp. 2651–2659.

[16] H. Zhu, P.A. Hall, J.H. May, Software unit test coverage and adequacy, ACM Comput. Surv. 29 (4) (1997) 366–427.

[17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, S. Hochreiter, Gans trained by a two time-scale update rule converge to a local nash equilibrium, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, in: NIPS'17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 6629–6640.

[18] S.G. Finlayson, I.S. Kohane, A.L. Beam, Beam adversarial attacks against medical deep learning systems, 2018, arXiv preprint arXiv:1804.05296.

[19] M. Wu, M. Wicker, W. Ruan, X. Huang, M. Kwiatkowska, A game-based approximate verification of deep neural networks with provable guarantees, Theoret. Comput. Sci. 807 (2020) 298–329, In memory of Maurice Nivat, a founding father of Theoretical Computer Science - Part II.

[20] J. Ebrahimi, A. Rao, D. Lowd, D. Dou, Hotflip: White-box adversarial examples for text classification, in: Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Vol. 2, 2018, pp. 31–36.

[21] M. Wicker, X. Huang, 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.

[22] M. Zhang, Y. Zhang, L. Zhang, C. Liu, S. Khurshid, DeepRoad: GAN-based metamorphic autonomous driving system testing, in: Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on, 2018.

[23] X. Huang, M. Kwiatkowska, S. Wang, M. Wu, Safety verification of deep neural networks, in: R. Majumdar, V. Kunčak (Eds.), Computer Aided Verification, Springer International Publishing, Cham, 2017, pp. 3–29.

[24] W. Xiang, H.-D. Tran, T.T. Johnson, Output reachable set estimation and verification for multi-layer neural networks, IEEE Trans. Neural Netw. Learn. Syst. 29 (2018) 5777–5783.

[25] W. Ruan, X. Huang, M. Kwiatkowska, Reachability analysis of deep neural networks with provable guarantees, in: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, 2018, pp. 2651–2659.

[26] M. OSearcoid, Metric Spaces, Springer Science & Business Media, 2006.
[27] T.-W. Weng, H. Zhang, P.-Y. Chen, J. Yi, D. Su, Y. Gao, C.-J. Hsieh, L. Daniel, Evaluating the robustness of neural networks: An extreme value theory approach, in: International Conference on Learning Representations (ICLR), 2018.
[28] L. Pulina, A. Tacchella, An abstraction-refinement approach to verification of artificial neural networks, in: International Conference on Computer Aided Verification, Springer, 2010, pp. 243–257.
[29] R. Ehlers, Formal verification of piece-wise linear feed-forward neural networks, in: D. D'Souza, K. Narayan Kumar (Eds.), Automated Technology for Verification and Analysis, Springer International Publishing, Cham, 2017, pp. 269–286.
[30] N. Narodytska, S.P. Kasiviswanathan, L. Ryzhyk, M. Sagiv, T. Walsh, Verifying properties of binarized deep neural networks, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018, pp. 6615–6624.
[31] N. Narodytska, Formal analysis of deep binarized neural networks, in: Proceedings of the 27th International Joint Conference on Artificial Intelligence, AAAI Press, 2018, pp. 5692–5696.
[32] A. Lomuscio, L. Maganti, An approach to reachability analysis for feed-forward ReLU neural networks, 2017, arXiv preprint arXiv:1706.07351.
[33] C.-H. Cheng, G. Nührenberg, H. Ruess, Maximum resilience of artificial neural networks, in: D. D'Souza, K. Narayan Kumar (Eds.), Automated Technology for Verification and Analysis, Springer, 2017, pp. 251–268.
[34] S. Dutta, S. Jha, S. Sanakaranarayanan, A. Tiwari, Output range analysis for deep neural networks, 2018, arXiv preprint arXiv:1709.09130.
[35] R. Bunel, I. Turkaslan, P.H. Torr, P. Kohli, M.P. Kumar, Piecewise linear neural network verification: A comparative study, 2017, arXiv preprint arXiv:1711.00455.
[36] A. Neumaier, O. Shcherbina, Safe bounds in linear and mixed-integer linear programming, Math. Program. 99 (2) (2004) 283–296.
[37] P. Cousot, R. Cousot, Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints, in: Fourth ACM Symposium on Principles of Programming Languages (POPL), 1977, pp. 238–252.
[38] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, M. Vechev, AI2: Safety and robustness certification of neural networks with abstract interpretation, in: 2018 IEEE Symposium on Security and Privacy (S & P2018), 2018, pp. 948–963.
[39] M. Mirman, T. Gehr, M. Vechev, Differentiable abstract interpretation for provably robust neural networks, in: International Conference on Machine Learning, 2018, pp. 3575–3583.
[40] J. Li, J. Liu, P. Yang, L. Chen, X. Huang, Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification, 2018, submitted for publication.
[41] E. Wong, Z. Kolter, Provable defenses against adversarial examples via the convex outer adversarial polytope, in: International Conference on Machine Learning, 2018, pp. 5283–5292.
[42] K. Dvijotham, R. Stanforth, S. Gowal, T.A. Mann, P. Kohli, A dual approach to scalable verification of deep networks, in: Conference on Uncertainty in Artificial Intelligence (UAI), 2018, pp. 550–559.
[43] S. Wang, K. Pei, J. Whitehouse, J. Yang, S. Jana, Formal security analysis of neural networks using symbolic intervals, in: USENIX Security Symposium, USENIX Association, 2018.
[44] J. Peck, J. Roels, B. Goossens, Y. Saeys, Lower bounds on the robustness to adversarial perturbations, in: I. Guyon, U.V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett (Eds.), Advances in Neural Information Processing Systems 30, Curran Associates, Inc., 2017, pp. 804–813.
[45] T.-W. Weng, H. Zhang, H. Chen, Z. Song, C.-J. Hsieh, D. Boning, I.S. Dhillon, L. Daniel, Towards fast computation of certified robustness for relu networks, in: Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 5273–5282.
[46] H. Zhang, T.-W. Weng, P.-Y. Chen, C.-J. Hsieh, L. Daniel, Efficient neural network robustness certification with general activation functions, in: Advances in Neural Information Processing Systems, 2018, pp. 4939–4948.
[47] H. Zhang, P. Zhang, C.-J. Hsieh, Recurjac: An efficient recursive algorithm for bounding jacobian matrix of neural networks and its applications, in: AAAI Conference on Artificial Intelligence (AAAI), 2019, arXiv preprint arXiv:1810.11783.
[48] W. Ruan, M. Wu, Y. Sun, X. Huang, D. Kroening, M. Kwiatkowska, Global robustness evaluation of deep neural networks with provable guarantees for the Hamming distance, in: Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19, International Joint Conferences on Artificial Intelligence Organization, 2019, pp. 5944–5952.
[49] I.J. Goodfellow, Gradient masking causes CLEVER to overestimate adversarial perturbation size, 2018, CoRR, abs/1804.07870.
[50] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. Nori, A. Criminisi, Measuring neural net robustness with constraints, in: Advances in Neural Information Processing Systems, 2016, pp. 2613–2621.
[51] Y. Jia, M. Harman, An analysis and survey of the development of mutation testing, IEEE Trans. Softw. Eng. 37 (5) (2011) 649–678.
[52] T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen, Z. Su, A survey on data-flow testing, ACM Comput. Surv. 50 (1) (2017) 5:1–5:35.
[53] K. Hayhurst, D. Veerhusen, J. Chilenski, L. Rierson, A Practical Tutorial on Modified Condition/Decision Coverage, Technical report, NASA, 2001.
[54] RTCA, Do-178c, software considerations in airborne systems and equipment certification, 2011.
[55] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, R. Ashmore, Structural test coverage criteria for deep neural networks, in: 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2019, pp. 320–321.
[56] K. Pei, Y. Cao, J. Yang, 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.
[57] Y. Tian, K. Pei, S. Jana, B. Ray, DeepTest: Automated testing of deep-neural-network-driven autonomous cars, in: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), 2018, pp. 303–314.
[58] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, D. Kroening, Concolic testing for deep neural networks, in: Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on, 2018.
[59] R. Ashmore, M. Hill, Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift, in: First International Workshop on Artificial Intelligence Safety Engineering, 2018.
[60] L. Ma, F. Juefei-Xu, J. Sun, C. Chen, T. Su, F. Zhang, M. Xue, B. Li, L. Li, Y. Liu, J. Zhao, Y. Wang, DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems, in: Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on, 2018.
[61] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, 2014, CoRR, abs/1412.6572.
[62] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, Y. Wang, Combinatorial testing for deep learning systems, 2018, arXiv preprint arXiv:1806.07723.
[63] Y. Sun, X. Huang, D. Kroening, Testing deep neural networks, 2018, arXiv preprint arXiv:1803.04792.
[64] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, R. Ashmore, Structural test coverage criteria for deep neural networks, ACM Trans. Embed. Comput. Syst. 18 (5s) (2019).
[65] Z. Wang, E.P. Simoncelli, A.C. Bovik, Multiscale structural similarity for image quality assessment, in: Signals, Systems and Computers, Conference Record of the Thirty-Seventh Asilomar Conference on, 2003.
[66] C.-H. Cheng, C.-H. Huang, H. Yasuoka, Quantitative projection coverage for testing ML-enabled autonomous systems, in: International Symposium on Automated Technology for Verification and Analysis, Springer, 2018.
[67] C.-H. Cheng, G. Nührenberg, C.-H. Huang, H. Yasuoka, Towards dependability metrics for neural networks, in: Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design, 2018.
[68] C.-H. Cheng, C.-H. Huang, G. Nührenberg, Nn-dependability-kit: Engineering neural networks for safety-critical systems, 2018, arXiv preprint arXiv:1811.06746.
[69] J. Kim, R. Feldt, S. Yoo, Guiding deep learning system testing using surprise adequacy, 2018, arXiv preprint arXiv:1808.08444.
[70] M.P. Wand, M.C. Jones, Kernel Smoothing, Chapman and Hall/CRC, 1994.
[71] A. Kurakin, I.J. Goodfellow, S. Bengio, Adversarial examples in the physical world, 2016, CoRR, abs/1607.02533.
[72] N. Papernot, P.D. McDaniel, S. Jha, M. Fredrikson, Z. Berkay Celik, A. Swami, The limitations of deep learning in adversarial settings, in: Security and Privacy (EuroS & P), 2016 IEEE European Symposium on, IEEE, 2016, pp. 372–387.
[73] N. Carlini, D. Wagner, Adversarial examples are not easily detected: Bypassing ten detection methods, in: Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security, ACM, 2017, pp. 3–14.
[74] R. Salay, K. Czarnecki, Using machine learning safely in automotive software: An assessment and adaption of software process requirements in iso 26262, 2018, arXiv preprint arXiv:1808.01614.
[75] S. Udeshi, P. Arora, S. Chattopadhyay, Automated directed fairness testing, in: Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on, 2018.
[76] A. Odena, I. Goodfellow, TensorFuzz: Debugging neural networks with coverage-guided fuzzing, 2018, arXiv preprint arXiv:1807.10875.
[77] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, M. Xue, B. Li, Y. Liu, J. Zhao, J. Yin, S. See, Coverage-guided fuzzing for deep neural networks, 2018, arXiv preprint arXiv:1809.01266.
[78] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, J. Sun, DLFuzz: Differential fuzzing testing of deep learning systems, in: Proceedings of the 2018 12nd Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2018, 2018.

[79] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, R. Ashmore, Deepconcolic: testing and debugging deep neural networks, in: 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), IEEE, 2019, pp. 111–114.

[80] D. Gopinath, K. Wang, M. Zhang, C.S. Pasareanu, S. Khurshid, Symbolic execution for deep neural networks, 2018, arXiv preprint arXiv:1807.10439.

[81] A. Agarwal, P. Lohia, S. Nagar, K. Dey, D. Saha, Automated test generation to detect individual discrimination in ai models, 2018, arXiv preprint arXiv:1809.03260.

[82] S. Ma, Y. Liu, X. Zhang, W.-C. Lee, A. Grama, MODE: Automated neural network model debugging via state differential analysis and input selection, in: Proceedings of the 12nd Joint Meeting on Foundations of Software Engineering, ACM, 2018.

[83] W. Shen, J. Wan, Z. Chen, MuNN: Mutation analysis of neural networks, in: International Conference on Software Quality, Reliability and Security Companion, QRS-C, IEEE, 2018.

[84] 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: Software Reliability Engineering, IEEE 29th International Symposium on, 2018.

[85] D. Cheng, C. Cao, C. Xu, X. Ma, Manifesting bugs in machine learning code: An explorative study with mutation testing, in: International Conference on Software Quality, Reliability and Security (QRS), IEEE, 2018, pp. 313–324.

[86] Y. Noller, C.S. Păsăreanu, M. Böhme, Y. Sun, H.L. Nguyen, L. Grunske, HyDiff: Hybrid differential software analysis, in: Proceedings of the 42nd International Conference on Software Engineering, ICSE, 2020.

[87] W. Huang, Y. Sun, J. Sharp, X. Huang, testRNN: Coverage-guided testing on recurrent neural networks, 2019, arXiv preprint arXiv:1906.08557.

[88] W. Huang, Y. Sun, J. Sharp, X. Huang, Test metrics for recurrent neural networks, 2019, arXiv preprint arXiv:1911.01952.

[89] X. Du, X. Xie, Y. Li, L. Ma, Y. Liu, J. Zhao, Deepstellar: model-based quantitative analysis of stateful deep learning systems, in: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2019, pp. 477–487.

[90] S.-M. Moosavi-Dezfooli, A. Fawzi, O. Fawzi, P. Frossard, Universal adversarial perturbations, in: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, pp. 86–94.

[91] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, F. Roli, Evasion attacks against machine learning at test time, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2013, pp. 387–402.

[92] S.-M. Moosavi-Dezfooli, A. Fawzi, 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.

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

[94] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2014, arXiv preprint arXiv:1412.6980.

[95] L. Engstrom, D. Tsipras, L. Schmidt, A. Madry, A rotation and a translation suffice: Fooling cnns with simple transformations, 2017, arXiv preprint arXiv:1712.02779.

[96] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.

[97] C. Xiao, J.-Y. Zhu, B. Li, W. He, M. Liu, D. Song, Spatially transformed adversarial examples, 2018, arXiv preprint arXiv:1801.02612.

[98] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, A. Matyasko, V. Behzadan, K. Hambardzumyan, Z. Zhang, Y.-L. Juang, Z. Li, R. Sheatsley, A. Garg, J. Uesato, W. Gierke, Y. Dong, D. Berthelot, P. Hendricks, J. Rauber, R. Long, Technical report on the cleverhans v2.1.0 adversarial examples library, 2018, arXiv preprint arXiv:1610.00768.

[99] K. Pei, Y. Cao, J. Yang, S. Jana, Towards practical verification of machine learning: The case of computer vision systems, 2017, arXiv preprint arXiv:1712.01785.

[100] J. Hayes, G. Danezis, Learning universal adversarial perturbations with generative models, in: 2018 IEEE Security and Privacy Workshops (SPW), IEEE, 2018, pp. 43–49.

[101] O. Poursaeed, I. Katsman, B. Gao, S.J. Belongie, Generative adversarial perturbations, in: Conference on Computer Vision and Pattern Recognition, 2018.

[102] O. Ronneberger, P. Fischer, T. Brox, U-net: Convolutional networks for biomedical image segmentation, in: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, 2015, pp. 234–241.

[103] J. Johnson, A. Alahi, L. Fei-Fei, Perceptual losses for real-time style transfer and super-resolution, in: European Conference on Computer Vision, Springer, 2016, pp. 694–711.

[104] J.H. Metzen, M.C. Kumar, T. Brox, V. Fischer, Universal adversarial perturbations against semantic image segmentation, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2755–2764.

[105] J. Li, R. Ji, H. Liu, X. Hong, Y. Gao, Q. Tian, Universal perturbation attack against image retrieval, in: Proceedings of the IEEE International Conference on Computer Vision, 2019, pp. 4899–4908.

[106] P. Neekhara, S. Hussain, P. Pandey, S. Dubnov, J. McAuley, F. Koushanfar, Universal adversarial perturbations for speech recognition systems, in: Proc. Interspeech 2019, 2019, pp. 481–485.

[107] K.R. Mopuri, A. Ganeshan, R. Venkatesh Babu, Generalizable data-free objective for crafting universal adversarial perturbations, IEEE Trans. Pattern Anal. Mach. Intell. 41 (10) (2018) 2452–2465.

[108] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: Security and Privacy (SP), 2016 IEEE Symposium on, IEEE, 2016, pp. 582–597.

[109] J.H. Metzen, T. Genewein, V. Fischer, B. Bischoff, On detecting adversarial perturbations, 2017, arXiv preprint arXiv:1702.04267.

[110] D. Hendrycks, K. Gimpel, Early methods for detecting adversarial images, 2016, arXiv preprint arXiv:1608.00530.

[111] D. Meng, H. Chen, Magnet: a two-pronged defense against adversarial examples, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 135–147.

[112] N. Carlini, D. Wagner, Magnet and efficient defenses against adversarial attacks are not robust to adversarial examples, 2017, arXiv preprint arXiv:1711.08478.

[113] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, A. Vladu, Towards deep learning models resistant to adversarial attacks, 2017, arXiv preprint arXiv:1706.06083.

[114] T. Na, J.H. Ko, S. Mukhopadhyay, Cascade adversarial machine learning regularized with a unified embedding, in: International Conference on Learning Representations (ICLR), 2018.

[115] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, P. McDaniel, Ensemble adversarial training: Attacks and defenses, in: International Conference on Learning Representations, 2018.

[116] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, arXiv e-prints, page arXiv:1503.02531.

[117] N. Papernot, P. McDaniel, X. Wu, S. Jha, A. Swami, Distillation as a defense to adversarial perturbations against deep neural networks, in: 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 582–597.

[118] A.N. Bhagoji, D. Cullina, P. Mittal, Dimensionality reduction as a defense against evasion attacks on machine learning classifiers, 2017, CoRR, abs/1704.02654.

[119] W. Xu, D. Evans, Y. Qi, Feature squeezing: Detecting adversarial examples in deep neural networks, in: Network and Distributed System Security Symposium (NDSS), 2018.

[120] D. Meng, H. Chen, Magnet: a two-pronged defense against adversarial examples, in: ACM Conference on Computer and Communications Security, 2017.

[121] Y. Song, T. Kim, S. Nowozin, S. Ermon, N. Kushman, Pixeldefend: Leveraging generative models to understand and defend against adversarial examples, in: International Conference on Learning Representations, 2018.

[122] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, K. Kavukcuoglu, Conditional image generation with pixelcnn decoders, in: NIPS, 2016.

[123] P. Samangouei, M. Kabkab, R. Chellappa, Defense-gan: Protecting classifiers against adversarial attacks using generative models, in: International Conference on Learning Representations (ICLR), 2018.

[124] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio, Generative adversarial nets, in: Advances in Neural Information Processing Systems, 2014, pp. 2672–2680.

[125] C. Guo, M. Rana, M. Cisse, L. van der Maaten, Countering adversarial images using input transformations, 2017, arXiv preprint arXiv:1711.00117.

[126] C. Xie, J. Wang, Z. Zhang, Z. Ren, A. Yuille, Mitigating adversarial effects through randomization, in: International Conference on Learning Representations (ICLR), 2017.

[127] J. Buckman, A. Roy, C. Raffel, I. Goodfellow, Thermometer encoding: One hot way to resist adversarial examples, in: International Conference on Learning Representations, 2018.

[128] G.S. Dhillon, K. Azizzadenesheli, Z.C. Lipton, J. Bernstein, J. Kossaifi, A. Khanna, A. Anandkumar, Stochastic activation pruning for robust adversarial defense, in: International Conference on Learning Representations, 2018.

[129] X. Ma, B. Li, Y. Wang, S.M. Erfani, S. Wijewickrema, M.E. Houle, G. Schoenebeck, D. Song, J. Bailey, Characterizing adversarial subspaces using local intrinsic dimensionality, 2018, arXiv preprint arXiv:1801.02613.

[130] R. Feinman, R.R. Curtin, S. Shintre, A.B. Gardner, Detecting adversarial samples from artifacts, 2017, arXiv e-prints, page arXiv:1703.00410.

[131] M.E. Houle, Local intrinsic dimensionality i: An extreme-value-theoretic foundation for similarity applications, in: C. Beecks, F. Borutta, P. Kröger, T. Seidl (Eds.), Similarity Search and Applications, Springer International Publishing, Cham, 2017, pp. 64–79.

[132] J. Steinhardt, P. Liang, P.W. Koh, Certified defenses for data poisoning attacks, in: Advances in Neural Information Processing Systems 30, 2017.

[133] M. Hein, M. Andriushchenko, Formal guarantees on the robustness of a classifier against adversarial manipulation, in: NIPS, 2017.

[134] A. Raghunathan, J. Steinhardt, P. Liang, Certified defenses against adversarial examples, 2018, arXiv preprint arXiv:1801.09344.

[135] A. Sinha, H. Namkoong, J. Duchi, Certifiable distributional robustness with principled adversarial training, in: International Conference on Learning Representations, 2018.

[136] S. Gowal, K.D. Dvijotham, R. Stanforth, R. Bunel, C. Qin, J. Uesato, R. Arandjelovic, T. Mann, P. Kohli, Scalable verified training for provably robust image classification, in: The IEEE International Conference on Computer Vision (ICCV), 2019.

[137] D. Erhan, Y. Bengio, A. Courville, P. Vincent, Visualizing Higher-Layer Features of a Deep Network, Technical report, University of Montreal, 2009.

[138] A. Mahendran, A. Vedaldi, Understanding deep image representations by inverting them, in: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5188–5196.

[139] A. Dosovitskiy, T. Brox, Inverting convolutional networks with convolutional networks, 2015, CoRR, abs/1506.02753.

[140] J. Yosinski, J. Clune, A.M. Nguyen, T.J. Fuchs, H. Lipson, Understanding neural networks through deep visualization, in: International Conference on Machine Learning (ICML) Deep Learning Workshop, 2015.

[141] M.T. Ribeiro, S. Singh, C. Guestrin, Why should I trust you?: Explaining the predictions of any classifier, in: HLT-NAACL Demos, 2016.

[142] M. Sundararajan, A. Taly, Q. Yan, Axiomatic attribution for deep networks, in: International Conference on Machine Learning, 2017.

[143] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, W. Samek, On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation, PLoS ONE 10 (7) (2015).

[144] A. Shrikumar, P. Greenside, A. Kundaje, Learning important features through propagating activation differences, in: Proceedings of Machine Learning Research, Vol. 70, 2017, pp. 3145-3153.

[145] R.R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, D. Batra, Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization, in: NIPS 2016 Workshop on Interpretable Machine Learning in Complex Systems, 2016.

[146] S. Lundberg, S. Lee, A unified approach to interpreting model predictions, in: NIPS, 2017.

[147] L.M. Zintgraf, T.S. Cohen, T. Adel, M. Welling, Visualizing deep neural network decisions: Prediction difference analysis, in: International Conference on Machine Learning (ICML), 2017.

[148] B. Kim, M. Wattenberg, J. Gilmer, C. Cai, J. Wexler, F. Viegas, R. sayres, Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV), in: J. Dy, A. Krause (Eds.), Proceedings of the 35th International Conference on Machine Learning, in: Proceedings of Machine Learning Research, vol. 80, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, pp. 2668–2677.

[149] J. Chen, L. Song, M.J. Wainwright, M.I. Jordan, Learning to explain: An information-theoretic perspective on model interpretation, in: International Conference on Machine Learning, 2018.

[150] K. Simonyan, A. Vedaldi, A. Zisserman, Deep inside convolutional networks: Visualising image classification models and saliency maps, 2013, CoRR, abs/1312.6034.

[151] D. Smilkov, N. Thorat, B. Kim, F.B. Viégas, M. Wattenberg, Smoothgrad: removing noise by adding noise, 2017, CoRR, abs/1706.03825.

[152] M.D. Zeiler, R. Fergus, Visualizing and understanding convolutional networks, in: European Conference on Computer Vision (ECCV), 2014.

[153] J.T. Springenberg, A. Dosovitskiy, T. Brox, M.A. Riedmiller, Striving for simplicity: The all convolutional net, 2014, CoRR, abs/1412.6806.

[154] P. Dabkowski, Y. Gal, Real time image saliency for black box classifiers, in: NIPS, 2017.

[155] R. Fong, A. Vedaldi, Interpretable explanations of black boxes by meaningful perturbation, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 3449–3457.

[156] P.W. Koh, P. Liang, Understanding black-box predictions via influence functions, in: International Conference on Machine Learning, 2017.

[157] M.T. Ribeiro, S. Singh, C. Guestrin, Anchors: High-precision model-agnostic explanations, in: Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[158] J.R. Zilke, E. Loza Mencía, F. Janssen, Deepred – rule extraction from deep neural networks, in: T. Calders, M. Ceci, D. Malerba (Eds.), Discovery Science, Springer International Publishing, Cham, 2016, pp. 457–473.

[159] M. Sato, H. Tsukimoto, Rule extraction from neural networks via decision tree induction, in: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222), Vol. 3, 2001, pp. 1870–1875.

[160] L. Chu, X. Hu, J. Hu, L. Wang, J. Pei, Exact and consistent interpretation for piecewise linear neural networks: A closed form solution, in: ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, ACM, 2018, pp. 1244–1253.

[161] G. Weiss, Y. Goldberg, E. Yahav, Extracting automata from recurrent neural networks using queries and counterexamples, in: International Conference on Machine Learning, 2018.

[162] D. Angluin, Learning regaular sets from queries and counterexamples, Inform. and Comput. 75 (1987) 87–106.

[163] C.W. Omlin, C. Lee Giles, Extraction of rules from discrete-time recurrent neural networks, Neural Netw. 9 (1) (1996) 41–52.

[164] N. Tishby, F.C. Pereira, W. Bialek, The information bottleneck method, 2000, arXiv e-prints, page physics/0004057.

[165] R. Shwartz-Ziv, N. Tishby, Opening the black box of deep neural networks via information, 2017, CoRR, abs/1703.00810.

[166] Z. Goldfeld, E. van den Berg, K. Greenewald, I. Melnyk, N. Nguyen, B. Kingsbury, Y. Polyanskiy, Estimating information flow in neural networks, 2018, arXiv e-prints, page arXiv:1810.05728.

[167] N. Wolchover, New theory cracks open the black box of deep learning, 2017, https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921/.

[168] A.M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B.D. Tracey, D.D. Cox, On the information bottleneck theory of deep learning, in: International Conference on Learning Representations, 2018.

[169] R.A. Amjad, B.C. Geiger, How (not) to train your neural network using the information bottleneck principle, 2018, CoRR, abs/1802.09766.

[170] J. Lu, H. Sibai, E. Fabry, D. Forsyth, NO need to worry about adversarial examples in object detection in autonomous vehicles, in: Conference on Computer Vision and Pattern Recognition, Spotlight Oral Workshop, 2017.

[171] Z.C. Lipton, The mythos of model interpretability, Commun. ACM 61 (2018) 36–43.

[172] V. Bruni, D. Vitulano, An entropy based approach for ssim speed up, Signal Process. 135 (2017) 198–209.

[173] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement for symbolic model checking, J. ACM 50 (5) (2003) 752–794.

[174] R. Könighofer, R. Bloem, Repair with on-the-fly program analysis, in: A. Biere, A. Nahir, T. Vos (Eds.), Hardware and Software: Verification and Testing, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 56–71.

[175] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, D. Hassabis, Human-level control through deep reinforcement learning, Nature 518 (2015) 529, EP –.

[176] T. Schaul, J. Quan, I. Antonoglou, D. Silver, Prioritized experience replay, 2015, CoRR, abs/1511.05952.

[177] Z. Wang, N. de Freitas, M. Lanctot, Dueling network architectures for deep reinforcement learning, in: International Conference on Machine Learning, 2016.

[178] H. van Hasselt, A. Guez, D. Silver, Deep reinforcement learning with double q-learning, in: Association for the Advancement of Artificial Intelligence, 2015.

[179] S.H. Huang, N. Papernot, I.J. Goodfellow, Y. Duan, P. Abbeel, Adversarial attacks on neural network policies, 2017, CoRR, abs/1702.02284.

[180] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, G. Chowdhary, Robust deep reinforcement learning with adversarial attacks, in: International Conference on Autonomous Agents and Multiagent Systems (AAMAS), 2018.

[181] Y. Lin, Z. Hong, Y. Liao, M. Shih, M. Liu, M. Sun, Tactics of adversarial attack on deep reinforcement learning agents, in: International Joint Conference on Artificial Intelligence, 2017.

[182] E. Tretschk, S.J. Oh, M. Fritz, Sequential attacks on agents for long-term adversarial goals, 2018, CoRR, abs/1805.12487.

[183] X. Wei, J. Zhu, H. Su, Sparse adversarial perturbations for videos, 2018, CoRR, abs/1803.02536.

[184] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith, Counterexample-guided abstraction refinement, in: E.A. Emerson, A.P. Sistla (Eds.), Computer Aided Verification, Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, pp. 154–169.

[185] J. Sifakis, Autonomous systems - an architectural characterization, 2018, CoRR, abs/1811.10277.

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

[187] Y. Sun, Y. Zhou, S. Maskell, J. Sharp, X. Huang, Reliability validation of learning enabled vehicle tracking, in: ICRA, 2020.

[188] W. Huang, Y. Zhou, J. Meng, J. Sharp, S. Maskell, X. Huang, Formal Verification of Robustness and Resilience of Learning-Enabled State Estimation System for Robotics, Technical Report, 2020.

[189] X. Zhao, A. Banks, J. Sharp, V. Robu, D. Flynn, M. Fisher, X. Huang, A safety framework for critical systems utilising deep neural networks, 2020, arXiv e-prints, page arXiv:2003.05311.
[190] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016.
[191] D. Amodei, C. Olah, J. Steinhardt, P. Christiano, J. Schulman, D. Mané, Concrete problems in ai safety, 2016, arXiv preprint arXiv:1606.06565.
[192] J.G. Moreno-Torres, T. Raeder, R. Alaiz-Rodríguez, N.V. Chawla, F. Herrera, A unifying view on dataset shift in classification, Pattern Recognit. 45 (1) (2012) 521–530.
[193] M. Abbasi, A. Rajabi, A. Sadat Mozafari, R.B. Bobba, C. Gagne, Controlling over-generalization and its effect on adversarial examples generation and detection, 2018, ArXiv e-prints.
[194] S. Liang, Y. Li, R. Srikant, Enhancing the reliability of out-of-distribution image detection in neural networks, in: International Conference on Learning Representations, 2018.
[195] T. DeVries, G.W. Taylor, Learning confidence for out-of-distribution detection in neural networks, 2018, arXiv e-prints, page arXiv:1802.04865.
[196] C.-H. Cheng, G. Nührenberg, H. Yasuoka, Runtime monitoring neuron activation patterns, 2018, arXiv e-prints, page arXiv:1809.06573.
[197] C. Olah, A. Mordvintsev, L. Schubert, Feature visualization, Distill (2017) https://distill.pub/2017/feature-visualization.
[198] M. Ancona, E. Ceolini, C. Öztireli, M. Gross, Towards better understanding of gradient-based attribution methods for deep neural networks, in: International Conference on Learning Representations, 2018.
[199] P. Tamagnini, J. Krause, A. Dasgupta, E. Bertini, Interpreting black-box classifiers using instance-level visual explanations, in: Proceedings of the 2nd Workshop on Human-in-the-Loop Data Analytics, in: HILDA'17, ACM, New York, NY, USA, 2017, pp. 6:1–6:6.
[200] R. Hardin, Trust and Trustworthiness, Russell Sage Foundation, 2002.
[201] R.C. Mayer, J.H. Davis, F.D. Schoorman, An integrative model of organizational trust, Acad. Manag. Rev. 20 (3) (1995) 709–734.
[202] X. Huang, M. Kwiatkowska, Reasoning about cognitive trust in stochastic multiagent systems, in: AAAI2017, 2017, pp. 3768–3774.