arXiv:1812.08342v1 [cs.LG] 18 Dec 2018

Safety and Trustworthiness of Deep Neural Networks: A Survey[*]

Xiaowei Huang[1], Daniel Kroening[2], Marta Kwiatkowska[2],
Wenjie Ruan[2], Youcheng Sun[2], Emese Thamo[1], Min Wu[2], and
Xinping Yi[1]

[1]University of Liverpool, UK,
{xiaowei.huang, emese.thamo, xinping.yi}@liverpool.ac.uk
[2]University of Oxford, UK,
{daniel.kroening, marta.kwiatkowska, wenjie.ruan,
youcheng.sun, min.wu}@cs.ox.ac.uk

Contents

---

[*]Authors are listed in alphabetical order

Symbols and Acronyms

0.1   List of Symbols

$\mathcal{N}$	a neural network
f	function represented by a neural network
W	weight
b	bias
$n_{k,l}$	l-th neuron on the k-th layer
$v_{k,l}$	activation value of the l-th neuron on the k-th 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

0.2   List of Acronyms

DNN	Deep Neural Network
ML	Machine Learning
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

1 Introduction

In the past few years, significant progress has been made on deep neural networks (DNNs) in achieving human-level intelligence on several long-standing tasks such as image classification [Russakovsky et al., 2015], natural language processing [Collobert et al., 2011], the ancient game of Go [Silver et al., 2017], etc. With broader deployment of DNNs on various applications, the concerns on its safety and trustworthiness have been raised, particularly after the fatal incidents of self-driving cars [Tes, 2018, Ube, 2018]. Research to address these concerns is very active, with many papers released in the past few years. It is therefore infeasible, if not impossible, to cover all the research activities. This survey paper is to conduct a review[1] of the current research efforts on making DNNs safe and trustworthy, by focusing on those works that are aligned with our humble visions about the safety and trustworthiness of DNNs. Figure 1 gives a yearly change on the number of papers surveyed (started from 2008). In total, we surveyed 178 papers, most of which were published in the most recent two years, i.e., 2017 and 2018.

Trust, or trustworthiness, is a general term and its definition varies in different contexts. We base our definition on a practise that has been widely adopted in established industries, e.g., automotive and avionics. In these established industries, trustworthiness is addressed mainly with two processes: a certification process and an explanation process. A **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. An **explanation** process is held whenever needed in the lifetime of the product. The user manual explains a set of expected behaviour 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 similar practise should be carried out when working with data-driven deep learning systems. That is, in this survey, we address trustworthiness based on the following vision:

$$\text{Trustworthiness} = \text{Certification} + \text{Explanation}$$

In other word, 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 interactions to the trust.

The survey will be concerned with the advance of enabling techniques for 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] The readers are welcomed to point to us any sources that are relevant to this survey but were not picked up, and/or provide any comments.
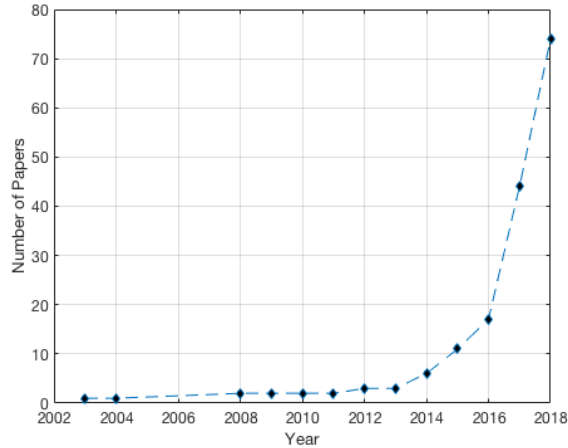
Figure 1: Number of publications surveyed with respect to the year

1.1 Certification

For certification, an important low-level requirement for DNNs is the robustness to input perturbations. DNNs have been shown suffering from poor robustness because of their susceptibility to *adversarial examples* [Szegedy et al., 2014]. These are small modifications to an input, sometimes imperceptible to humans, that make the network unstable. Significant efforts have been developed in the area of machine learning on the attack and defence techniques. Attack techniques aim to find adversarial examples which the DNN is unknown about by e.g., classifying them with high probability to wrong classes, while defence techniques aim to enhance the DNN so that it can identify or eliminate adversarial examples. These techniques cannot be directly applied to certify a DNN, for their inability to provide assurance to their results. Nevertheless, we review some prominent methods since they provide useful insights to certification techniques.

The certification techniques we cover in this survey include verification and testing, both of which have been proved useful for checking the dependability of software and hardware systems. However, traditional techniques developed in these two areas, see e.g., [Ammann and Offutt, 2008, Clarke Jr et al., 2018], cannot be be directly applied to deep learning systems, which exhibit complex internal behaviours that are not commonly seen for traditional verification and testing.

DNN verification techniques are to determine whether a property, e.g., the local robustness for a given input x, holds on a DNN. If holds, they are able to supplement the answer with a mathematical proof. Otherwise, they 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. While verification techniques are promising, they suffer from the scalability problem,

due to the high computational complexity of the verification problems and the large size of the DNN. Up to now, DNN verification techniques are either working with small scale DNNs or working with approximate methods with convergence guarantees on the bounds.

DNN testing techniques arise as a complement to the verification techniques. Instead of providing mathematical proofs to the satisfiability of a property on the system, testing techniques aim to either find bugs (i.e., counterexamples to the property) or provide assurance cases [Rushby, 2015], by exercising the system with a large set of test cases. They are computationally less expensive and therefore are able to work with state-of-the-art systems. In particular, coverage-guided testing generates test cases according to a set of pre-specified coverage metrics. Intuitively, a high coverage suggests that most of the DNN's behaviours have been tested and therefore the DNN has a lower chance of containing undetected bugs. Research is needed to clarify how such coverage-guided testing can contribute to assurance cases.

1.2   Explanation

Explainable AI [exp, 2018], or interpretability problem of AI [Voosen, 2017], is to explain why the AI arrived at a specific decision, say to give or not give loan to a person. EU's General Data Protection Regulation (GDPR) [GDR, 2016] mandates a "right to explanation" from machine learning models, meaning that an explanation of how the model reached its decision can be asked for. While this "explainability" request is definitely beneficial to the end consumer, it can be hard for the system developers who design systems with ML components.

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, and interpretability. This is followed by Section 3, which discusses safety problems and safety properties. In Section 4 and Section 5, we review DNN verification and DNN testing techniques, respectively. The attack and defence techniques are reviewed in Section 6. This is followed by Section 7, which reviews a set of interpretability techniques for DNNs. Finally, we discuss future challenges in Section 8 and conclude in Section 9.

Figure 2 gives a causality relation between sections. We use dashed arrows from attack and defence techniques (Section 6) to a few other sections for its technical supports to certification and explanation techniques.

9
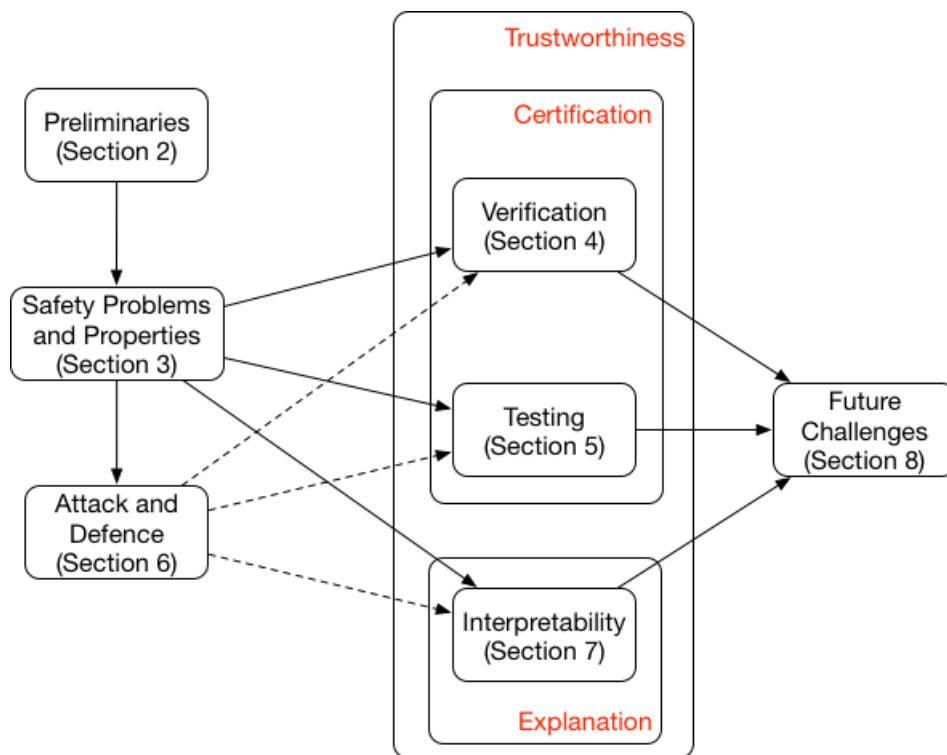
Figure 2: Relationship between sections

2 Preliminaries

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

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 l-th 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) [Nair and Hinton, 2010] 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 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 h-th node of layer $k-1$) and $n_{k,l}$ (i.e., the l-th node of layer k), and $b_{k,l}$ the the so-called *bias* for node $n_{k,l}$. We note that this definition can express both fully-connected functions and convolutional functions[2]. The function ϕ_k is the combination of Equation (1) and (2). Owing to the use of the ReLU as in (1), the behavior 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 $f : D_1 \rightarrow D_K$ such that

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

---

[2]Many of the surveyed techniques can work with other types of functional layers. Here for simplicity, we omit their expressions.

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:

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

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

Example 1 *Figure 3 is a simple DNN with four layers. Its input space is $D_1 = \mathbb{R}^2$.*

Input layer   Hidden layer   Hidden layer   Output layer
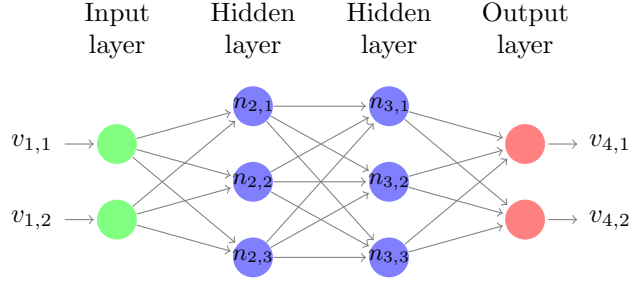
Figure 3: A simple neural network

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 denoted as $u_{k,l}[x]$ and $v_{k,l}[x]$, respectively. Thereby, the activation or deactivation of each ReLU operation in the network is also 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 Figure 3. Assume that the weights for the first three layers are as follows:*

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

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 a set of techniques to check whether the property C holds on $\mathcal{N}$. A verification technique needs to provide provable guarantees to its results. A provable guarantee can be in the form of either a Boolean guarantee or a statistical guarantee. A Boolean guarantee means that the verification technique is able to provide a mathematical proof, when the property holds, or a counterexample, otherwise. When a mathematical proof is hard to achieve, a statistical guarantee provides a quantitative error tolerance bound on the resulting claim. It might also be the case that this error bound can be continuously improved until converged.

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

2.3 Testing

Verification problems usually have high computational complexity, such as NP-hard when the properties are simple input-output constraints [Katz et al., 2017, Ruan et al., 2018a]. This, compounded with the high-dimensionality and the high non-linearity of DNNs, makes the exisitng verification techniques hard to work with industrial scale DNNs. This computational intensity can be partially alleviated by considering testing techniques, at the price of the provable guarantees. Instead, assurance cases are pursued, as is done for existing safety critical systems [Rushby, 2015].

The goal of testing DNNs is to generate a set of test cases, so that the developer can be more confident on the performance of a developed DNN when it passes the test cases. Usually, the generation of test cases is guided by coverage metrics. Let N be a set of DNNs, R a set of test condition 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 [Zhu et al., 1997] for software testing.

Definition 1 *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 conditions. Usually, the greater the number $M(\mathcal{N}, \mathcal{R}, \mathcal{T})$, the more adequate the testing. We may use criterion and metric interchangeably.

We let F be a set of covering methods, and $\mathcal{R} = \mathsf{O}(\mathcal{N})$ be the set of test conditions to be covered. In [Sun et al., 2018b] $\mathsf{O}(\mathcal{N})$ is instantiated as the set of causal relationships between feature pairs while in [Pei et al., 2017a] $\mathsf{O}(\mathcal{N})$ is instantiated as the set of statuses of hidden neurons.

Definition 2 (Test Suite) *Given a DNN $\mathcal{N}$, a test suite $\mathcal{T}$ is a finite set of input vectors, i.e., $\mathcal{T} \subseteq D_1 \times D_1 \times ... \times D_1$. Each vector is called a test case.*

Usually, a test case is a single input, e.g., in [Pei et al., 2017a], or a pair of inputs, e.g., in [Sun et al., 2018b]. Ideally, given the set of test conditions $\mathcal{R}$ according to some covering method cov, we run a test case generation algorithm to find a test suite $\mathcal{T}$ such that

$$\forall \alpha \in \mathcal{R} \exists (x_1, x_2, ..., x_k) \in \mathcal{T} : cov(\alpha, (x_1, x_2, ..., x_k)) \qquad (6)$$

In practice, we might want to compute the degree to which the test conditions are satisfied by a generated test suite $\mathcal{T}$.

Definition 3 (Test Criterion) *Given a DNN $\mathcal{N}$ with its associated function f, a covering method cov, test conditions in $\mathcal{R}$, and a test suite $\mathcal{T}$, the test criterion $M_{cov}(\mathcal{R}, \mathcal{T})$ is as follows:*

$$M_{cov}(\mathcal{R}, \mathcal{T}) = \frac{|\{\alpha \in \mathcal{R} | \exists (x_1, x_2, ..., x_k) \in \mathcal{T} : cov(\alpha, (x_1, x_2, ..., x_k))\}|}{|\mathcal{R}|} \qquad (7)$$

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.

We will elaborate various covering methods and test objectives in Section 5. Moreover, a testing oracle is a mechanism that determines whether the DNN behaves correctly for a test case. It depends on the properties to be tested, and therefore will be discussed in Section 3.

2.4 Interpretability

Interpretability is an issue aroused due to the black-box nature of the DNNs. Intuitively, it is to provide a human-understandable explanation to the behaviour of a DNN. An explanation procedure can be separated into two steps: an extraction step and an exhibition step. The *extraction step* is to obtain an intermediate representation, and the *exhibition step* is to present the obtained intermediate presentation in a way easy for human users to understand. Given the fact that DNNs are usually high-dimensional and that simpler information can be easier to be understood by human users, the intermediate representation needs to be lower dimensional. Since the exhibition step is closely related to the intermediate representation and are usually conducted by e.g., visualising the representation, we will focus on the extraction step.

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

Definition 4 *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, it 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 Section 7.1, Section 7.2, and Section 7.3 fit with this definition.

Definition 5 *Given a function $f : \mathbb{R}^{s_1} \to \mathbb{R}^{s_K}$, which represents a DNN $\mathcal{N}$, a model explanation $\mathtt{expl}(f)$ 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.*

Intuitively, for model explanation, it 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 fits with this definition. There are other model explanations such as the influence function based approach reviewed in Section 7.4, which explains 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 method for the explanation of information flow in the DNN training process.

Definition 6 *Given a function family $\mathcal{F}$, which represents a stochastic DNN, an information-flow explanation $\mathtt{expl}(\mathcal{F})$ 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.*

Intuitively, for information-flow explanation, it 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 such that $||x||_0 = |\{x_i \mid x_i \neq 0, i = 1..n\}|$, i.e., the number of non-zero elements. Note that, L_0 norm does not satisfy the triangle inequality.

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

Definition 7 *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 wrt L_p*

$$\eta(x, L_p, d) = \{\hat{x} \mid ||\hat{x} - x||_p \leq d\} \tag{8}$$

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

3   Safety Problems and Safety Properties
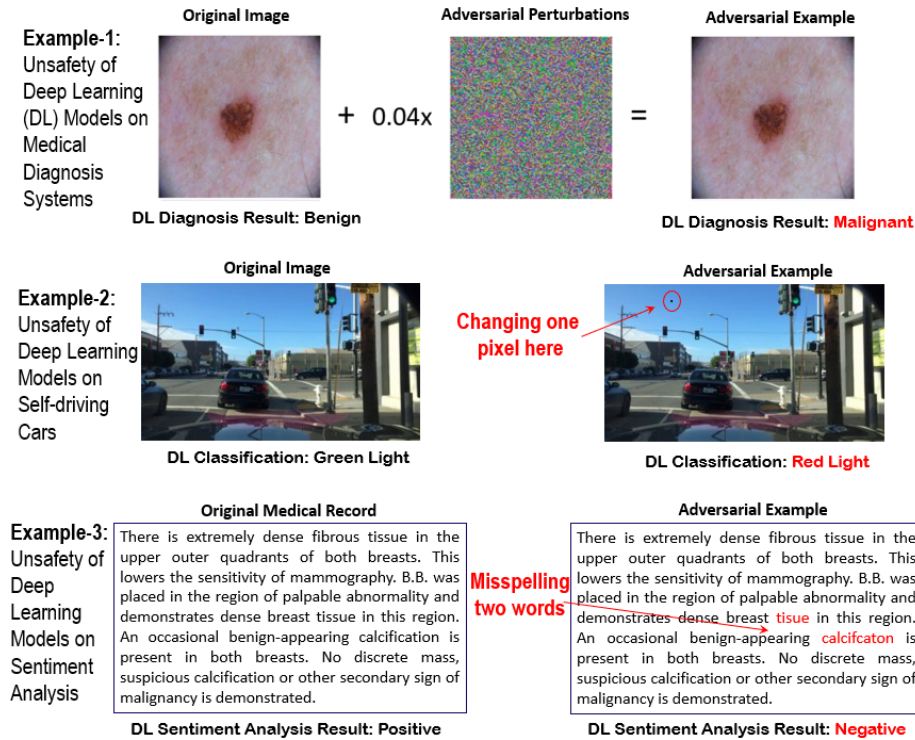
Figure 4: Examples of Erroneous Behaviour on Deep Learning Models. *Example-1 [Finlayson et al., 2018]: In a medical diagnosis system, a "Benign" tumour is misclassified as "Malignant" after adding a small amount of human-imperceptible perturbations; Example-2 [Wu et al., 2018]: By just changing one pixel in a "Green-Light" image, a state-of-the-art DNN misclassifies it as "Red-Light"; Example-3 [Ebrahimi et al., 2018]: In a sentiment analysis task for medical record, with two misspelling words, a well-trained deep learning model classifies a "Positive" medical record as "Negative".*

Despite the success of deep learning (DL) in many areas, serious concerns have been raised on 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 the analysis techniques are working with.

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

Definition 8 (Erroneous Behavior 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 an*

legitimate input $x \in \mathbb{R}^{s_1}$, an erroneous behavior of DNNs is such that

$$\arg\max_j f_j(x) \neq \arg\max_j \mathcal{H}_j(x) \tag{9}$$

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

Adversarial examples [Szegedy et al., 2014] represent a class of erroneous behaviour.

Definition 9 (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 of DNNs is defined as:*

$$
\begin{aligned}
\exists \hat{x} : \quad & \arg\max_j \mathcal{H}_j(\hat{x}) = \arg\max_j \mathcal{H}_j(x) \\
& \wedge \; ||x - \hat{x}||_p \leq d \\
& \wedge \; \arg\max_j f_j(\hat{x}) \neq \arg\max_j f_j(x)
\end{aligned} \tag{10}
$$

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 $\hat{x}$, which is classified differently with x by the network f (i.e., $\arg\max_j f_j(\hat{x}) \neq \arg\max_j f_j(x)$) even when human believes that they should be the same (i.e., $\arg\max_j \mathcal{H}_j(\hat{x}) = \arg\max_j \mathcal{H}_j(x)$) and they are semantically similar (i.e., $||x - \hat{x}||_p \leq d$).

Figure 4 shows three concrete examples of such safety concerns brought by DNNs on safety-critical application scenarios including medical diagnosis systems, self-driving cars and automated sentimental analysis on medical records.

Example 3 *In classification tasks, by adding a small amount of adversarial perturbations (w.r.t. L_p-norm distance), the DNNs will misclassify an image of traffic sign "red light" into "green light" [Wicker et al., 2018, Wu et al., 2018]. 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 4 *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" [Zhang et al., 2018]. However, it is clear that, from human driver's point of view, adding "rain" should not change the driving decision of a car.*

Example 5 *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 [Finlayson et al., 2018].*

As we can see, those unsafe, or erroneous, phenomenon acting on deep neural networks are essentially caused by the inconsistency of the decision boundaries from DL models (that are learned from various training datasets) and human oracles. They inevitably raise significant concerns on when deep learning models can be 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 adapts from that of [Huang et al., 2017b].

Definition 10 (Verification of Local Robustness) *Given a DNN $\mathcal{N}$ with its associated function f, and an input region $\eta \subseteq [0,1]^{s_1}$, the (un-targeted) local robustness of f on η is defined as*

$$Robust(f, \eta) \triangleq \exists l \in [1..s_K] \forall x \in \eta \forall j \in [1..s_K] : f_l(x) \geq f_j(x) \qquad (11)$$

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

$$Robust_j(f, \eta) \triangleq \forall x \in \eta \exists l \in [1..s_K] : f_l(x) > f_j(x) \qquad (12)$$

Intuitively, local robustness says that all inputs in the region η have the same class label. Usually, the region η is defined with respect to an input x and a norm L_p, as in Definition 7. If so, it means that all inputs in η have the same class with the 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 test oracle for local robustness property. Note that, all existing testing approaches are on local robustness, and therefore we only provide the test oracle for local robustness.

Definition 11 (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 $(x_1, ..., x_k) \in \mathcal{T}$ passes the oracle if*

$$\forall 1 \leq i \leq k \exists x_0 \in D : x_i \in \eta(x_0, L_p, d) \qquad (13)$$

Intuitively, a test case $(x_1, ..., x_k)$ passes 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 7.

3.3   Output Reachability Property

Output reachability is to compute the set of outputs with respect to a given set of inputs. We follow the name from [Xiang et al., 2018, Ruan et al., 2018a]. Formally, we have the following definition.

19

Definition 12 (Output Reachability) *Given a DNN $\mathcal{N}$ with its associated function f, an input region $\eta \subseteq [0,1]^{s_1}$, the output reachable set of f and η is to compute a set $Reach(f, \eta)$ such that*

$$Reach(f, \eta) \triangleq \{f(x) \mid x \in \eta\} \tag{14}$$

The continuity of the region η suggests the existence of infinite number of inputs in η. The output reachability problem is *highly non-trivial* for the facts that η is a continuous region and f is highly non-linear (or black-box). Based on this, we can define the following verification problem.

Definition 13 (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*

$$Reach(f, \eta) = \mathcal{Y}. \tag{15}$$

Intuitively, the verification of reachability is to check whether all inputs in η are mapped onto $Reach(f, \eta)$, and at the meantime, whether all outputs in $Reach(f, \eta)$ has a corresponding x in η.

3.4   Interval Property

Interval property is to compute a convex over-approximation of the output reachable set. We follow the name from interval-based approaches which are a typical class of methods to compute this property. Formally, we have the following definition.

Definition 14 (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 to compute a convex set $Interval(f, \eta)$ such that*

$$Interval(f, \eta) \supseteq \{f(x) \mid x \in \eta\} \tag{16}$$

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 15 (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 interval property on f, η, and $\mathcal{Y}$ is to determine if*

$$\mathcal{Y} \supseteq \{f(x) \mid x \in \eta\} \tag{17}$$

In other word, it is to determine whether the given $\mathcal{Y}$ is an interval satisfying Expression (16).

Intuitively, the verification of interval property is to check whether all inputs in η are mapped onto $\mathcal{Y}$.

3.5 Lipschitzian Property

Lipschitzian property, inspired by the Lipschitz continuity (see textbooks such as [OSearcoid, 2006]), is to monitor the changes of the output with respect to small changes of the inputs.

Definition 16 (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,*

$$Lips(f, \eta, L_p) \equiv \sup_{x_1, x_2 \in \eta} \frac{|f(x_1) - f(x_2)|}{||x_1 - x_2||_p} \qquad (18)$$

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 17 (Verification of Lipschitzian Property) *Given a Lipschitzian metric $Lips(f, \eta, L_p)$ and a real value $d \in \mathbb{R}$, it is to determine whether*

$$Lips(f, \eta, L_p) \geq d. \qquad (19)$$

3.6 Relationship between Properties

Figure 5 gives the relationship between the four properties we 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

$$Interval(f, \eta) = [f(x) - Lips(f, \eta, L_p) \cdot d, f(x) + Lips(f, \eta, L_p) \cdot d] \qquad (20)$$

It can be checked 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:

$$Interval(f, \eta) \subseteq \mathcal{Y}_l = \{y \in \mathbb{R}^{s_K} \mid \forall j \neq l : y_l \geq y_j\}, \text{ for some } l \qquad (21)$$

$$Reach(f, \eta) \subseteq \mathcal{Y}_l = \{y \in \mathbb{R}^{s_K} \mid \forall j \neq l : y_l \geq y_j\}, \text{ for some } l \qquad (22)$$

where y_l is the l-entry of the output vector y. The relation between $Reach(f, \eta)$ and $Interval(f, \eta)$ is actually an implication relation such that every $Reach(f, \eta)$ is also an interval $Interval(f, \eta)$.

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 [Ruan et al., 2018a, Wicker et al., 2018, Weng et al., 2018].
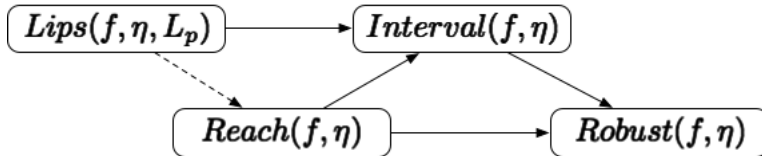
Figure 5: Relationship between properties

3.7 Instancewise Interpretability

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

Definition 18 (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 $eval_{\mathcal{H}} : \mathbb{R}^{s_1} \times \mathcal{E} \to [0,1]$, 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 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 as the distance metric based on human perception, it is hard to approximate. Based on this, we have the following definition.

Definition 19 (Validity of Explanation) *Let f be the associated function of a DNN $\mathcal{N}$, x an input, and $\epsilon > 0$ a real number, $\mathtt{expl}(f, x) \in \mathcal{E} \subseteq \mathbb{R}^t$ is a valid instance-wise explanation if*

$$eval_{\mathcal{H}}(x, \mathtt{expl}(f, x)) > 1 - \epsilon. \tag{23}$$

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 on DNNs. According to the underlying techniques, existing works on the verification of DNNs largely fall into the following categories: constraints solving, search based approach, global optimisation, and over-approximation. Figure 6 classifies works surveyed in this paper into different categories.
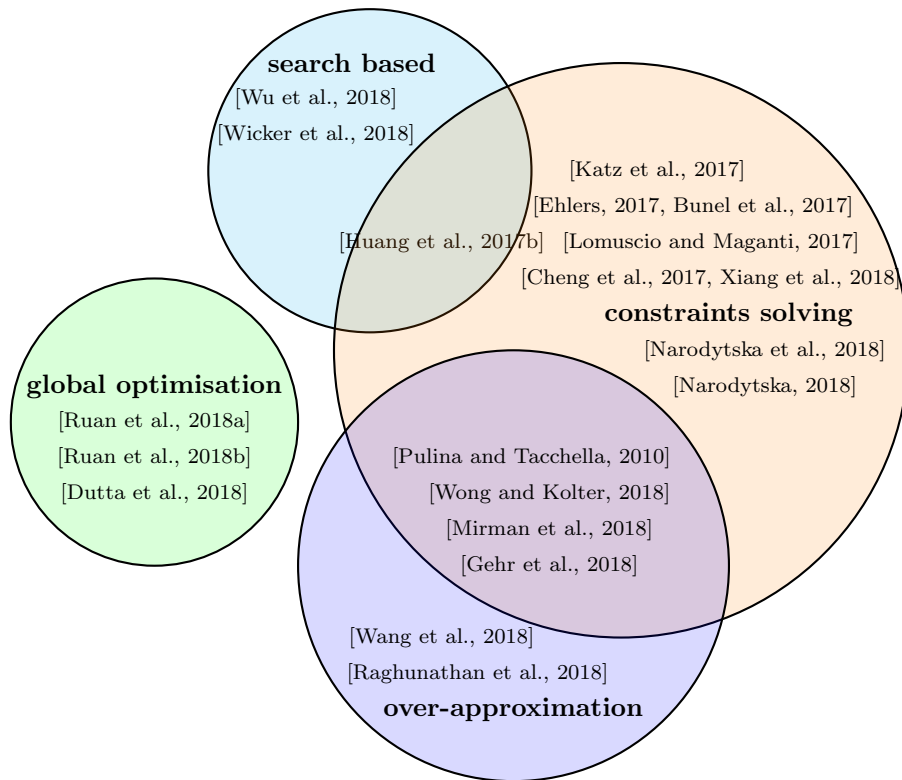
Figure 6: A taxonomy of verification works for DNNs.

In this survey, we classify these techniques with respect to the type of guarantees they can achieve. Basically, the guarantees can be

- an *exact deterministic* guarantee, which states exactly whether a property holds; We will omit the word *exact* and call it deterministic guarantee below.
- an *one-sided* guarantee by providing either a lower bound or an upper bound to a variable, as a sufficient condition for a property to hold;
- a guarantee with *converging* lower and upper bounds to a variable; or
- a *statistical* guarantee quantifying the probability that a property holds.

Note that, algorithms with the one-sided guarantee and the bound-converging guarantee are to compute the real values for e.g., output reachability property (Definition 12), interval property (Definition 14), or Lispchitzian property (Definition 16). Their respective verification problems are based on these values, see Definition 13, Definition 15 and Definition 17.

Remark. All existing verification works focus on *feedforward* neural networks.

4.1 Approaches with Deterministic Guarantees

The *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 usually return a deterministic answer to an 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, etc.

4.1.1 SMT/SAT

An abstraction-refinement approach based on SMT solving. A solution to the verification of interval property (can be easily extended to work with reachability property for ReLU activation functions) is proposed in [Pulina and Tacchella, 2010] by abstracting a DNN into a set of Boolean combinations of linear arithmetic constraints. It is shown that whenever the abstracted model is declared to be safe, the same holds for the concrete one. Spurious counterexamples, on the other hand, trigger refinements and can be leveraged to automate the correction of misbehaviour. The approach is validated on DNNs with fewer than 10 neurons.

SMT solvers for DNNs. Two SMT solvers Reluplex [Katz et al., 2017] and Planet [Ehlers, 2017] were put forward to verify DNNs on properties expressible with SMT constraints. SMT solvers often have good performance on problems which 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 DNNs 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 rule out conflict clauses, while they are slightly different in dealing with the intersection. For Reluplex, it inherits rules in the algorithm of Simplex and add 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. Differently, Planet uses linear approximation

to over-approximate the neural network, and manage the condition of ReLU and max pooling node with a logic formula.

SAT approach. [Narodytska et al., 2018, Narodytska, 2018] propose to verify properties of a class of neural networks, i.e., binarised neural networks on 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 deep neural networks used in image classification tasks.

4.1.2 Mixed Integer Linear Programming (MILP)

MILP formulation for DNNs. [Lomuscio and Maganti, 2017] encodes the behaviours of the fully connected neural networks with MILP. For instance, a hidden layer $z_{i+1} = \mathrm{ReLU}(W_i z_i + b_i)$ 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 the bounds, and therefore this approach can work with both reachability property and interval property.

However, it is not efficient to simply use MILP to verify DNNs or to compute the output range. In [Cheng et al., 2017], a number of MILP encoding heuristics are developed to speed up solving process, and moreover, parallelisation of MILP-solvers are used to result in an almost linear speed-up in the number (up to a certain limit) of computing cores in experiments. In [Dutta et al., 2018], Sherlock alternately conducts local and global search to efficiently calculate the output range. In a local search phase, it 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.

Moreover, [Bunel et al., 2017] presents a branch and bound (B&B) algorithm and claims that both SAT/SMT-based approaches and MILP-based approaches can be regarded as its special cases.

4.2 Approaches to Compute a Lower Bound

The approaches to be 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. While these approaches can only have a bounded estimation to the value of some variable, it is able to work with larger models, for e.g., up to 10,000 hidden neurons. The other advantage is its 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 [Neumaier and Shcherbina, 2004]. It may happen that a solver wrongly claims the satisfiability or un-satisfiability of a property. For example, [Dutta et al., 2018] reports several false positive results in Reluplex, and mentions that this may come from unsound floating point implementation.

4.2.1 Abstract Interpretation

Abstract interpretation is a theory of sound approximation of the semantics of computer programs [Cousot and Cousot, 1977]. It has been used in static analysis to verify properties of a program without actually running it. The basic idea of abstract interpretation is to use abstract domains (represented as e.g., boxes, zonotopes, polyhedra) to over-approximate the computation of a set of inputs. It has been explored in a few papers, including [Gehr et al., 2018, Mirman et al., 2018, Li et al., 2018].

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 20 *A pair of functions $\alpha : \mathcal{C} \rightarrow \mathcal{A}$ and $\gamma : \mathcal{A} \rightarrow \mathcal{C}$ 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.

- **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 6 *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

A method is proposed in [Wong and Kolter, 2018] to learn deep ReLU-based classifiers that are provably robust against norm-bounded adversarial perturbations on the training data. For previously unseen examples, the approach is guaranteed to detect all adversarial examples, though it may flag some non-adversarial examples as well. Therefore, the approach works with interval property, but not reachability property. The basic idea is to consider a convex outer over-approximation of the set of activations reachable through a norm-bounded perturbation, and the authors develop a robust optimisation procedure that minimises the worst case loss over this outer region (via a linear program). Crucially, it is shown that the dual problem to this linear program 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 end result is that by executing a few more forward and backward passes through a slightly modified version of the original network (though possibly with much larger batch sizes), a classifier can be learned that is provably robust to any norm-bounded adversarial attack. They illustrate the approach on a number of tasks to train classifiers with robust adversarial guarantees (e.g. 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., 2018] 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 [Wang et al., 2018], 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 [Peck et al., 2017], 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 [Xiang et al., 2018], 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 12. 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 Equation (14). 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 computed via solving convex optimisation problems. 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

[Weng et al., 2018] 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.

28

$$N, \eta_0 \models x \longleftarrow N, \eta_1 \models x \longleftarrow N, \eta_2 \models x \longleftarrow \quad \dots \quad \longleftarrow N, \eta_k \models x$$

Δ_k is minimal

$$N, \eta_1, \Delta_1 \models x \longleftarrow N, \eta_2, \Delta_2 \models x \longleftarrow \quad \dots \quad \longleftarrow N, \eta_k, \Delta_k \models x$$
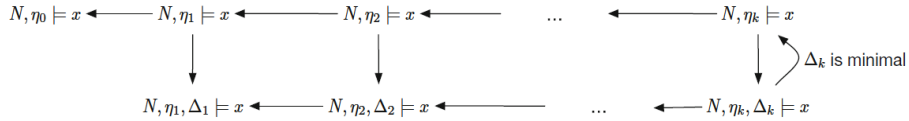
Figure 7: The layer-by-layer refinement framework of [Huang et al., 2017b].

4.3 Approaches with Converging Upper and Lower 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 Section 4.3 and Section 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., 2017b] develops an automated verification framework for feedfor-
ward 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 a region surrounding the
input, Δ a set of manipulations, and subscript k means at layer k. Later, in
[Wicker et al., 2018, Wu et al., 2018], 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, as shown in Figure 7. 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 [Wicker et al., 2018, Wu et al., 2018], the search problem is alleviated by
Monte-Carlo tree search.
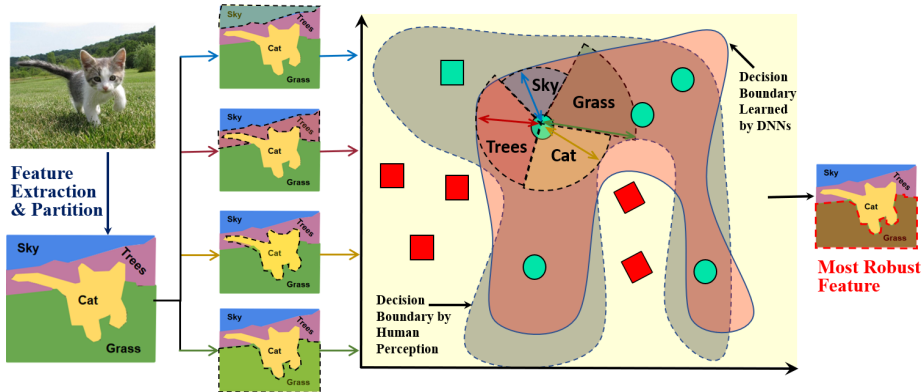
Figure 8: The *feature robustness* (FR) problem [Wu et al., 2018], which aims to find, on an original image x, a feature, or a subset of features, that is the most robust against adversarial perturbations. Given a benign image, first apply feature extraction or semantic partitioning methods to produce a set of disjoint features ('Sky', 'Trees', 'Cat', etc.), then find a set of robust features that is most resilient to adversarial perturbations ('Grass' in the figure), which quantifies the most robust direction in a safe norm ball.

4.3.2 Reduction to A Two-Player Turn-based Game

In [Wu et al., 2018], 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, as shown in Figure 8.

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.

30

4.3.3 Global Optimisation Based Approaches

DeepGO [Ruan et al., 2018a] 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 [Ruan et al., 2018b], the authors focus on the L_0 norm, 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., 2018] 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, 2018], 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., 2016] 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 [Ruan et al., 2018a].

4.5 Computational Complexity of Verification

There are two ways to measure the complexity of conducting formal verification. The first, appeared in [Katz et al., 2017], 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 [Ruan et al., 2018a], 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 the 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.

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

	Guarantees	Algorithm		Property			
				Robustness	Reachability	Interval	Lipschitzian
[Pulina and Tacchella, 2010]	Deterministic Guarantees	Constraints Solving	SMT	✓	✓	✓	
[Katz et al., 2017]				✓	✓	✓	
[Ehlers, 2017]				✓	✓	✓	
[Narodytska et al., 2018] [Narodytska, 2018]			SAT	✓	✓	✓	
[Lomuscio and Maganti, 2017]		MILP		✓	✓	✓	
[Cheng et al., 2017]				✓	✓	✓	
[Dutta et al., 2018]				✓	✓	✓	
[Bunel et al., 2017]				✓	✓	✓	
[Gehr et al., 2018]	Lower/Upper Bound	Abstract Interpretation				✓	
[Mirman et al., 2018]						✓	
[Li et al., 2018]						✓	
[Wong and Kolter, 2018]		Convex Optimisation				✓	
[Wang et al., 2018]		Interval Analysis				✓	
[Peck et al., 2017]						✓	
[Xiang et al., 2018]		Set Estimation				✓	
[Weng et al., 2018]		Linear Approximation				✓	✓
[Huang et al., 2017b]	Converging Bounds	Search Based	Layer-by-Layer Refinement	✓	✓	✓	
[Wicker et al., 2018]			Two-Player Turn-based Game	✓	✓	✓	✓
[Wu et al., 2018]				✓	✓	✓	✓
[Ruan et al., 2018a]		Global Optisimation		✓	✓	✓	✓
[Ruan et al., 2018b]				✓	✓	✓	
[Weng et al., 2018]	Statistical Guarantees	Extreme Value Theory					✓
[Bastani et al., 2016]		Robustness Estimation		✓			

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 has 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 test software. Please refer to [Zhu et al., 1997, Jia and Harman, 2011, Su et al., 2017] for comprehensive reviews. In white-box testing, the structure of a program 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 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 [Zhu et al., 1997]. 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[Hayhurst et al., 2001] and has been widely adopted. It is used in avionics software development guidance to ensure adequate testing of applications with the highest criticality [RTCA, 2011].

5.1.1 Neuron Coverage

Neuron coverage [Pei et al., 2017a] is the first coverage criterion designed for DNNs. It can be seen as the statement coverage variant for DNN testing.

Definition 21 *[Pei et al., 2017a] 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 $O(\mathcal{N}) = \{\exists x : N(n_{k,i}, x) \mid 2 \leq k \leq K-1, 1 \leq i \leq s_k\}$. 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 [Pei et al., 2017a] 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 $f_k(x)[c]$ be the class probability that f_k predicts x to be c. The optimisation objective is as follows.

$$obj(x) = (\sum_{k \neq j} f_k(x)[c] - \lambda_1 f_j(x)[c]) + \lambda_2 v_{j,i}(x) \qquad (24)$$

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

Moreover, in [Tian et al., 2018] greedy search combining image transformations is used to increase neuron coverage, and is applied to DNNs for autonomous driving. In [Sun et al., 2018c], the conclic testing algorithm can work with a set of coverage metrics including neuron coverage and some other coverage metrics to be surveyed below.

5.1.2 Safety Coverage

In [Wicker et al., 2018], 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 $x \in rec$.*

Let $Rec(\mathcal{N})$ be the set of hyper-rectangles. The set of objectives to be covered is $\mathsf{O}(\mathcal{N}) = \{\exists x : S(rec, x) | rec \in Rec(\mathcal{N})\}$. 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 [Ashmore and Hill, 2018], 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 [Wicker et al., 2018] 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 [Ma et al., 2018a], several coverage criteria are proposed by following similar rationale as neuron coverage to focus on individual neurons' activation values.

Definition 23 *[Ma et al., 2018a] A node $n_{k,i}$ is neuron boundary covered by a test case x, denoted as $NB(n_{k,i}, x)$, if $v_{k,i}[x] > v_{k,i}^u$.*

Let $rank(n_{k,i}, x)$ be the rank of $v_{k,i}[x]$ among those values of the nodes at the same layer, i.e., $\{v_{k,j}[x] \mid 1 \leq j \leq s_k\}$.

Definition 24 *[Ma et al., 2018a] For $1 \leq m \leq s_k$, a node $n_{k,i}$ is top-m neuron covered by x, denoted as $TN^m(n_{k,i}, x)$, if $rank(n_{k,i}, x) \leq m$.*

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

Definition 25 *[Ma et al., 2018a] Given $m \geq 1$, 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 $\forall 1 \leq j \leq m \exists x \in \mathcal{T} : v_{k,i}[x] \in I_{k,i}^j$, 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 [Ma et al., 2018a] for generating test cases for the criteria proposed; instead, they apply adversarial attack methods (e.g., [Goodfellow et al., 2014b]) to generate an extra set of new inputs that is shown to increase the coverage. Following [Ma et al., 2018a], an exploratory study on combinatorial testing is conducted in [Ma et al., 2018c] 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) [Hayhurst et al., 2001] 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 7 *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 six test cases provide 100% MC/DC coverage:

*1. $(a > 3)$=**false**, $(b = 0)$=**true**, $(c \neq 4)$=**false***

*2. $(a > 3)$=**true**, $(b = 0)$=**false**, $(c \neq 4)$=**true***

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

*4. $(a > 3)$=**false**, $(b = 0)$=**true**, $(c \neq 4)$=**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 [Sun et al., 2018a], which is further refined in [Sun et al., 2018b]. Different from these criteria in [Pei et al., 2017a, Ma et al., 2018a] that only consider individual neurons' activations, the criteria in [Sun et al., 2018a, Sun et al., 2018b] 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 set of subsets of nodes at layer k. Without loss of generality, each element of Ψ_k, i.e., a subset of nodes in the k-th 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 $nsc(\psi_{k,l}, x_1, x_2) \neq \neg sc(\psi_{k,l}, x_1, x_2)$. 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 $g : \Psi_k \times D_{L_1} \times D_{L_1} \to$ {true, 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 8 *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 $\frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} > d \lor \frac{u_{k,j}[x_1]}{u_{k,j}[x_2]} < 1/d$ (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 9 *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 [Wang et al., 2003]. 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*

- $g(\psi_{k,l}, x_1, x_2)$=**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 [Sun et al., 2018a, Sun et al., 2018b], 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* $nsc(L_k, x_1, x_2)$;

- $sc(\psi_{k+1,j}, x_1, x_2)$.

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* $nsc(L_k, x_1, x_2)$;

- $vc(g_2, \psi_{k+1,j}, x_1, x_2)$ *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., 2018a] develops an algorithm based on liner programming (LP). This is complemented with an adaptive gradient descent (GD) search algorithm in [Sun et al., 2018b] and a concolic testing algorithm in [Sun et al., 2018c].

5.1.5 Quantitative Projection Coverage

In [Cheng et al., 2018b], 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 proposed 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 [Cheng et al., 2018b] are based on self-driving scenes, [Cheng et al., 2018c] 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 [Cheng et al., 2018a].

5.1.6 Surprise Coverage

[Kim et al., 2018b] 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 [Sun et al., 2018a] between inputs. Given a training set $\mathbf{T} \subseteq D_{L_1}$, a layer k, and a new input x, one of the measurements is to compute the following value

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

where $v_k(x)$ is the vector of activation values for neurons in layer k when the input is x. Moreover, $\mathcal{K}$ is a Gaussian kernel function and H is a bandwidth matrix, used in Kernel Density Estimation [Wand and Jones, 1994]. Based on this, the coverage is defined, similar as the m-multisection [Ma et al., 2018a], 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., 2018b] utilises a few existing algorithms for adversarial attack, including FGSM [Goodfellow et al., 2014b],

Basic iterative method [Kurakin et al., 2016], JSMA [Papernot et al., 2016c], CW attack [Carlini and Wagner, 2017a].

5.1.7 Comparison between Existing Coverage Criteria

Figure 9 gives a diagrammatic illustration of the relationship between most of the 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 [Salay and Czarnecki, 2018], 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).
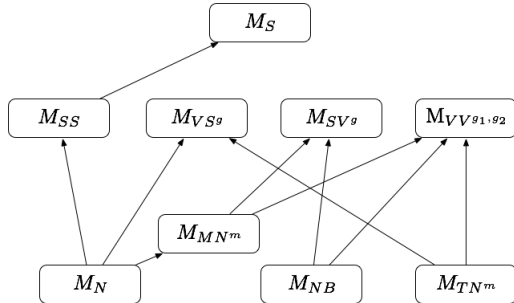
Figure 9: Relationship between test criteria

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.

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., 2018] 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 [Udeshi et al., 2018] 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 program, 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 program structure, the fuzzer can be white/grey/block-box. There are recent works that adopt fuzz testing to deep neural networks.

TensorFuzz [Odena and Goodfellow, 2018] 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 quantized versions, and surfacing undesirable behaviour in DNNs. Similar to TensorFuzz, DeepHunter [Xie et al., 2018] is another coverage-guided grey-box DNN fuzzer, which utilises these extensions of neuron coverage from [Ma et al., 2018a]. Moreover, DLFuzz [Guo et al., 2018] 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 program to determine what inputs cause each part of a program to execute. It assumes symbolic values for inputs rather than obtaining actual inputs as normal execution of the program would, and thus arrives at expressions in terms of those symbols for expressions and variables in the program, 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. This idea still holds for deep neural networks. In DeepConcolic [Sun et al., 2018c, Sun et al., 2018d], 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 [Gopinath et al., 2018] is to translate a DNN into an imperative program, thereby enabling program 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 [Agarwal et al., 2018], 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. Deep-Road [Zhang et al., 2018] 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 [Pei et al., 2017a] and [Guo et al., 2018] 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., 2018d], 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 [Ma et al., 2018d] 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., 2018] 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., 2018b] considers data mutations, program 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 program mutations, a few operations are considered, including adding or deleting a layer and remvoing activation function. Model-level mutations include changing the weights, shuffling the weights between neurons in neighboring layers, etc. Moreover, [Cheng et al., 2018d] simulates program 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 [Goodfellow et al., 2014b] and JSMA [Papernot et al., 2016c]) 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 [Pei et al., 2017a, Guo et al., 2018] are in principle based on differential analysis of multiple DNNs, thus more than one DNN inputs are expected.

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.

Table 2: Comparison between different DNN testing methods

	Test generation	Coverage criteria	DNN inputs	Distance metric
[Pei et al., 2017a]	dual-objective search	neuron coverage	multiple	L_1
[Tian et al., 2018]	greedy search	neuron coverage	single	Jaccard distance
[Wicker et al., 2018]	MCTS	safety coverage	single	n/a
[Ma et al., 2018a]	adversarial attack	neuron coverage extensions	single	n/a
[Sun et al., 2018a, Sun et al., 2018b]	LP, adaptive GA search	MC/DC	single	L_∞
[Cheng et al., 2018b]	0-1 ILP	quantitative projection coverage	single	n/a
[Kim et al., 2018b]	adversarial attacks	surprise coverage	single	n/a
[Sun et al., 2018c]	concolic testing	MC/DC, neuron coverage and its extensions	single	L_0, L_∞
[Odena and Goodfellow, 2018]	fuzzing	fast approximate nearest neighbour	single	n/a
[Xie et al., 2018]	fuzzing	neuron coverage extensions	single	n/a
[Guo et al., 2018]	fuzzing	neuron coverage	multiple	n/a
[Gopinath et al., 2018, Agarwal et al., 2018]	symbolic execution	n/a	single	n/a
[Zhang et al., 2018]	GAN	n/a	single	n/a
[Ma et al., 2018d]	GAN	n/a	single	n/a

6 Attack and Defence

Attack techniques are to provide evidence (i.e., adversarial example) to the lack of robustness of a DNN without having a provable guarantee. Defence techniques are dual to the attacking techniques, by either improving the robustness of the DNN to reduce the adversarial examples or differentiating the adversarial examples from the correct inputs. From Section 6.1 to Section 6.3, we review the attack techniques from different aspects. These techniques are compared in Section 6.4 with a few other techniques from verification. Then, in Section 6.5 and Section 6.6 we review defence techniques and certified defence techniques, respectively.

6.1 Adversarial Attacks

Given an input, an adversarial attack (or attacker) is 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 roughly classified into two groups based on the choice of misclassification.

- For targeted perturbation, the attacker is able to control the resulting misclassification label;

- For 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: an attacker needs to access the parameters and the internal structure of the trained DNN, and may also need to access the training dataset;

- Black-box perturbation: an attacker can only query the trained DNNs 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 classified as L_0, L_1, L_2 or L_∞-attack. Please noted that all perturbations can be measured with the norms, but an attack technique can produce adversarial examples which are better measured with a particular norm.

Most previous attacks focus on adversarial examples for computer vision tasks. Multiple techniques to create such adversarial examples have been developed recently. Broadly, such attacks can be categorised, in terms of technical point of view, as either using costs gradients such as works in [Goodfellow et al., 2014b, Moosavi-Dezfooli et al., 2017, Biggio et al., 2013], or the forward gradient of the neural network such as work in [Papernot et al., 2016c], or perturbing

along most promising direction or directly solving an optimisation problem (possibly using gradient ascent/descent) to find a perturbation such as works in [Moosavi-Dezfooli et al., 2016, Carlini and Wagner, 2017c].

In addition, adversarial examples have been shown to be transferable between different network architectures, or DNNs trained on disjoint subsets of data [Szegedy et al., 2014, Papernot et al., 2016c]. Adversarial examples have also been shown to be transferable to real world scenarios [Kurakin et al., 2016], specifically, adversarial images remain misclassified even after being printed out and recaptured with a cell phone camera. In the following, we will review a few notable works with greater detail.

6.1.1 Limited-memory BFGS Algorithm (L-BFGS)

[Szegedy et al., 2014] noticed the existence of adversarial examples, and described them as 'blind spots' in DNNs. Adversarial examples are misclassified by the DNNs and appear in the neighbourhood of correctly-classified examples. It is claimed that, since the adversarial examples have low probability of occurrence, they cannot be found efficiently by sampling around correctly-classified inputs. However, the adversarial examples can be found via an optimisation scheme. Formally, assume a classifier $f : \mathbb{R}^{s_1} \to \{1 \dots s_K\}$, mapping inputs to one of s_K class labels, a given input $x \in \mathbb{R}^{s_1}$, and the target label $t \in \{1 \dots s_K\}$ such that $t \neq \arg\max_l f_l(x)$, the goal is to find additive adversarial perturbation $r \in \mathbb{R}^{s_1}$ with the following optimization expression:

- Minimize $||r||_2$ subject to:

    1. $\arg\max_l f_l(x + r) = t$
    2. $x + r \in \mathbb{R}^{s_1}$

Since exact computation is hard, an approxmate algorithm based on limited-memory Broyden–Fletcher–Goldfarb–Shanno algorithm (L-BFGS) is used. Furthermore, [Szegedy et al., 2014] notices that adversarial examples are abundant and, with the above framework, one could generate an unlimited number of adversarial examples. They also note that these adversarial examples generalise across both model and training sets - an adversarial example generated for one DNN classifier will likely be an adversarial example for another classifier with different architectures or training dataset.

6.1.2 Fast Gradient Sign Method (FGSM)

Fast Gradient Sign Method [Goodfellow et al., 2014b] 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 argue that this provides a linear explanation to the existence of adversarial examples. They highlight that since the precision of an individual input feature is typically limited, e.g., digital images often use only 8 bits per pixel and therefore are precise up to $1/255$, it is

therefore unreasonable for a classifier to respond differently to two inputs if they only differ on each feature by an amount that is less than the level of precision. However, consider the dot product between a weight vector w and an adversarial example $x' = x + r$:

$$w^T x' = w^T x + w^T r$$

and let $r = \epsilon \, \text{sign}(w)$, the activation growth can be maximized. If w has n dimensions with elements having average magnitude m, the activation growth is $\epsilon m n$, i.e. increases linearly with respect to the dimensionality of the problem, whereas $||\eta||_\infty$ remains less than ϵ. Thus for high-dimensional problems, FGSM can make many small changes to the input to produce a large difference in model output. Based on this linear explanation, [Goodfellow et al., 2014b] suggests a fast linear algorithm to generate adversarial examples. Denoting θ as the model parameters, x an input to the model, y the label associated with x, and $J(\theta, x, y)$ the cost function used to train the model, then an adversarial perturbation r can be generated by

$$r = \epsilon \, \text{sign}\left(\nabla_x J(\theta, x, y)\right) \qquad (27)$$

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 into a targeted and iterative version [Kurakin et al., 2016].

6.1.3 Jacobian Saliency Map based Attack (JSMA)

Papernot et al [Papernot et al., 2016c] present an algorithm based on the *forward derivative* of a DNN, defined as the Jacobian matrix of the output probability distribution (over the set $\mathcal{L}$ of labels) with respect to the input dimensions, which is used to highlight those features that the DNN's prediction is most sensitive to and are therefore most likely to cause miss-classification when perturbed. For a given target class $c \in \mathcal{L}$ and a given input $x \in [0, 1]^{s_1}$, each dimension of the input is assigned a salient value based on the forward derivative. The salient value captures, for each input dimension, the sensitivity of the output probability assigned to a class c. For the adversarial perturbation, the input dimension with largest salient value is perturbed by a *maximum distortion parameter* $\tau > 0$. If this perturbation results in a miss-classification, then the algorithm terminates. Otherwise, the forward derivative is computed again over the distorted input and the algorithm proceeds. The algorithm may also terminate when a maximum distance threshold $d > 0$ is reached. This algorithm does not require the computation of the derivative of the perturbation measured with L_p-norm, and can be used to generate adversarial perturbations that are minimised under L_0 norm. This method is generally slower than FGSM and aims to find an adversarial image that has a lower L_0-norm distance to the legitimate image.

6.1.4 DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks

[Moosavi-Dezfooli et al., 2016] proposes an iterative algorithm to generate untargeted adversarial examples that are minimized under the L_p-norm, where $p \in [1, \infty)$. Firstly, they consider generating adversarial examples for simple case of affine binary classifiers: $g(x) = \text{sign}(w^T \cdot x + b)$. In this case, the optimal adversarial example for a given input image x_0 can be computed analytically as the orthogonal projection of x_0 onto the hyperplane $\mathcal{G} = \{x | w^T \cdot x + b = 0\}$. This is generalised to the multi-class case:

$$g(x) = \underset{i \in \{1...k\}}{\text{argmax}}(W^T x + b)$$

where $W \in \mathbb{R}^{m \times k}$ and $b \in \mathbb{R}^k$. Now in order to find the optimal adversarial example, the inupt x_0 is projected onto the nearest face of the hyper-polyhedron P, defined as follows:

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

where $k_0 = g(x_0)$. In other words, P is the set of all inputs that are classified with the same label as x_0. In order to generalise to non-affine multi-class classifiers, e.g. DNNs, the optimal adversarial example is found iteratively, where at each step the adversarial example is updated by linearly approximating the classifier and then affine-projected as described above. Although this is a greedy heuristic algorithm and as such is not guaranteed to find the optimal adversarial examples, the perturbations are observed to be small and believed to be good approximation of the minimal.

6.1.5 Carlini & Wagner Attack

C&W Attack [Carlini and Wagner, 2017c] 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. Specifically, it defines an optimisation problem over loss function

$$\ell(v) = ||v||_p + c \cdot f(x + v), \tag{28}$$

where f is a function that is negative when the DNN $\mathcal{N}$ miss-classifies $x+v$, under the constraint that $x+v$ is a valid input. The optimisation problem is solved using the gradient-descent method described in Adam [Kingma and Ba, 2014]. This approach is applied for three distance metrics - L_2, L_0 and L_∞ - the algorithm is adjusted slightly in each case. In particular for the L_0 case, an iterative algorithm identifies a subset of features having low impact on classification and are therefore not considered candidates for perturbation, this subset grows with each iteration, until its complement set is sufficiently small, giving a minimal feature subset salient to classification. At each iteration the feature i selected for

Figure 10: Rotation-Translation: Original (L) 'automobile', adversarial (R) 'dog' from [Engstrom et al., 2017]. *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 [He et al., 2016] classifies as 'dog'.*

exclusion is the one that minimises $\nabla f(x+v)_i \cdot v_i$. 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 $[0, 1]$). This approach is similar in intuition to [Papernot et al., 2016c] in that a subset of salient features is identified based on the first-order derivatives, and [Carlini and Wagner, 2017c] is shown to be more effective than [Papernot et al., 2016c]. C&W Attack is able to find an adversarial example that has a significant smaller image distance, especially based 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., 2017] argue that many existing adversarial attacking techniques produce adversarial examples that are particularly contrived and as such highly unlikely to occur 'naturally'. It shows that DNNs are vulnerable to examples that might occur in less-contrived settings, for example rotating and/or translating input images the target classifier's performance significantly degrades. Technically, [Engstrom et al., 2017] aims to find the optimal angle of rotation and magnitude of translation of a given image that causes misclassification within an acceptable range: $\pm 30° \times \pm 3$ pixels. A few methods are proposed, including (i) a first-order iterative method using the gradient of the DNN's loss function, (ii) a grid search by descretising the parameters space and exhaustively testing all possibilities, and (iii) a worst-of-k method by randomly sampling k possible parameter values and choosing the value that cause the DNN to perform the worst.
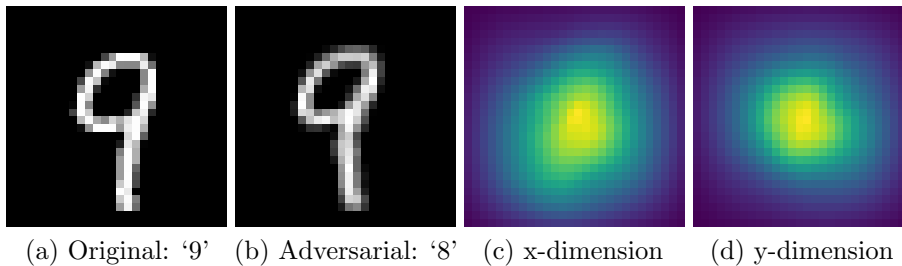
(a) Original: '9'   (b) Adversarial: '8'   (c) x-dimension   (d) y-dimension

Figure 11: Applying spatial transformation to MNIST image of a '9'. *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 [Papernot et al., 2018] 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.*

6.2.2 Spatially Transformed Adversarial Examples

[Xiao et al., 2018] propose to generate realistic adversarial examples by changing the location of pixels through spatial transformations rather than directly altering pixel values. These spatial transformations are defined by a displacement field (or 'flow field'), which defines for each pixel location a displacement of the pixel value to a new location. Using a bi-linear interpolation technique the resulting adversarial output is differentiable with respect to the flow field - this facilitates an optimisation-problem approach to generating the adversarial flow field. Technically, [Xiao et al., 2018] introduce a distance measure $L_{flow}(\cdot)$ (rather than the usual L_p distance) to capture the local geometric distortion. The flow field is generated in a similar manner as [Carlini and Wagner, 2017c] via an optimisation problem whereby the loss function is defined to balance between the adversarial loss function and the L_{flow} loss. Through human perceptual study, [Xiao et al., 2018] show that spatially transformed adversarial images are more difficult for humans to distinguish from original images, comparing to the adversarial images generated in e.g., [Goodfellow et al., 2014b] and [Carlini and Wagner, 2017c].

6.2.3 Towards Practical Verification of Machine Learning: The Case of Computer Vision Systems (VeriVis)

Pei et al [Pei et al., 2017b] propose a generic framework (VERIVIS) for evaluating the robustness of DNNs with a set of twelve 'real-world' transformations: Average smoothing, Median smoothing, Erosion, Dilation, Contrast, Brightness, Occlusion, Rotation, Shear, Scale, Translation and Reflection. Each transformation is defined by a critical parameter having a *polynomial-sized* domain. These

are applied exhaustively to a set of inputs which are used to assess the robustness of a given DNN. VERIVIS is used to test a number of state-of-the-art classifiers, and their results show that all classifiers exhibit a significant number of safety violations. It is also argued that, VERIVIS is capable of generating significantly more violations than gradient-based adversarial techniques.

6.3   Input-Agnostic Adversarial Attacks

A key characteristic of the above approaches is that an adversarial perturbation is generated with respect to a specific input, and therefore cannot be applied to the other input. It is not surprising that an *input-agnostic* perturbation would be a far more powerful tool.

6.3.1   Universal Adversarial Perturbations

[Moosavi-Dezfooli et al., 2017] define *universal* adversarial perturbations (UAP) as those that can mis-classify *any* example input with high probability. Technically, the algorithm iterates over a subset $\mathbf{T}$ of inputs sampled from the input distribution $\mathcal{X}$. At each iteration it updates the perturbation v as follows: for each $x_i \in \mathbf{T}$, the algorithm updates v by finding the minimal Δv_i (with respect to L_2-norm) such that $x_i + v + \Delta v_i$ is miss-classified by the DNN $\mathcal{N}$. Once computed, $v + \Delta v_i$ is projected onto the L_p-ball of radius d to ensure that the perturbation is sufficiently small, i.e.,

$$v \leftarrow \arg\min_{v'}\{||v' - (v + \Delta v_i)||_2 \mid ||v'||_p \leq d\}. \tag{29}$$

The algorithm continues until the empirical error of the sample set is sufficiently large, i.e., no less than $1 - \delta$ for a pre-specified threshold δ. The optimisation problem of finding the minimal Δv_i is solved using the DeepFool algorithm [Moosavi-Dezfooli et al., 2016].

6.3.2   Generative Adversarial Perturbations

[Hayes and Danezis, 2018] 'universalize' the approach in [Carlini and Wagner, 2017c] by training a universal adversarial network (UAN), and generates input-agnostic, rather than input-specific, perturbations. Given a maximum perturbation distance d and an L_p norm, a UAN $\mathcal{U}_\theta$ samples a random input vector z from a normal distribution and outputs a raw perturbation v, which is then scaled by $w \in [0, \frac{d}{||v||_p}]$ to have $v' = w \cdot v$. Then, the new input $x + v'$ needs to be checked with DNN $\mathcal{N}$ to see if it is an adverarial example. The parameters θ is learned using the gradient descent algorithm on a loss function similar to the one used in [Carlini and Wagner, 2017c].

[Poursaeed et al., 2018] adopt a similar technique to [Hayes and Danezis, 2018] for generating universal adversarial perturbations. A random noise image is fed into a UAN, the raw output is scaled to satisfy an L_p constraint, then added to input data, clipped, and then fed into a trained classifier. Their approach differs to [Hayes and Danezis, 2018] in two aspects. Firstly, they test

two UAN architectures U-Net [Ronneberger et al., 2015] and ResNet Generator [Johnson et al., 2016], and find that the ResNet Generator outperforms in most cases. Secondly, they develop a method for training the UAN using multiple target classifiers so that the generated UAP is explicitly trained to be able to fool multiple classifiers. This is achieved by a 'multi-fool' loss function $l_{multi-fool}$:

$$l_{multi-fool}(\lambda) = \lambda_1 \cdot l_{fool_1} + \cdots + \lambda_m \cdot l_{fool_m} \qquad (30)$$

where m is the number of target classifiers, and l_{fool_i} is the loss pertaining to target classifier i. Moreover, λ_i is a weight parameter for setting target classifier priority, for example one may want to set higher weights for those classifiers that are harder to fool.

6.4 Summary of Adversarial Attack Techniques

Table 3: Comparison between different adversarial attacks

	Distance Metric	Targeted/ Untargeted	Accessed Information	Dataset Tested	Core Method
L-BFGS [Szegedy et al., 2014]	L_2	Untarget	Model Parameters	MNIST	L-BFGS
FGSM [Goodfellow et al., 2014b]	L_∞	Untarget	Model Parameters	MNIST, CI-FAR10	Fast linear algorithm
DeepFool [Moosavi-Dezfooli et al., 2016]	$L_p, p \in [1, \infty)$	Both	Model Parameters	MNIST, CI-FAR10	Iterative linearization
C & W [Carlini and Wagner, 2017c]	L_0, L_2, L_∞	Both	Logits	MNIST, CI-FAR10	Adam Optimizer
JSMA [Papernot et al., 2016c]	L_0	Both	Model Parameters	MNIST, CI-FAR10	Jacobian Saliency
DeepGame [Wu et al., 2018]	L_0, L_1, L_2, L_∞	Untarget	Logits	MNIST, CI-FAR10	Game-based approach
L0-TRE [Ruan et al., 2018b]	L_0	Untarget	Logits	MNIST, CI-FAR10, ImageNet	Tensor-based grid search
DLV [Huang et al., 2017b]	L_1, L_2	Untarget	Model Parameters	MNIST, CI-FAR10, GT-SRB	Layer-by-layer search
SafeCV [Wicker et al., 2018]	L_0	Both	Logits	MNIST, CI-FAR10	Stochastic search
[Engstrom et al., 2017]	N/A (Natural Transformations)	Both	Logits	MNIST, CI-FAR10, ImageNet	Rotating and/or translating input images
[Xiao et al., 2018]	$L_{flow}(\cdot)$ (Measuring geometric distortion)	Both	Logits	MNIST, CI-FAR10, ImageNet	Minimising adversarial and L_{flow} loss
VeriVis [Pei et al., 2017b]	N/A (Natural Transformations)	Both	Logits	MNIST, CI-FAR10, ImageNet	A set of 12 'real-world' transformations
UAP [Moosavi-Dezfooli et al., 2017]	L_2 (Universal perturbation)	Both	Logits	MNIST, CI-FAR10, ImageNet	Generalizing DeepFool into universal adversarial attacks
UAN [Hayes and Danezis, 2018]	L_p (Universal perturbation)	Both	Logits	MNIST, CI-FAR10, ImageNet	Generalizing C&W into universal adversarial attacks
[Poursaeed et al., 2018]	L_p (Universal perturbation)	Both	Logits	MNIST, CI-FAR10, ImageNet	Training a generative network

This section summarises the main similarities and differences of the well-established 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. The details are in Table 3.

We roughly classify the adversarial attacks into four categories according to their technical differences. One line of attacking techniques is featured by a fact that adversarial perturbations are designed based on a specific input image, including L-BFGS [Szegedy et al., 2014], FGSM [Goodfellow et al., 2014b], DeepFool [Moosavi-Dezfooli et al., 2016], JSMA [Papernot et al., 2016c] and C&W [Carlini and Wagner, 2017c], etc. Most of those attacking methods are well-established and widely-recognised, which are also the very first works that discover the vulnerability of DNNs to adversarial perturbations. Another line of works are generalised the point-wise attacks into input-agnostic attacks (or called Universal Adversarial Perturbations). The representative research works includes UAP [Moosavi-Dezfooli et al., 2017], UAN [Hayes and Danezis, 2018, Poursaeed et al., 2018], etc. The basic ideas of those works either generalise the techniques in point-wise attacks into universal ones, or training generative neural networks to directly produce such universal perturbations. The third line of attacks however focus on generating more "natural" or "human-visually similar" adversarial perturbations. The most notable works are including [Engstrom et al., 2017], [Xiao et al., 2018] and VeriVis [Pei et al., 2017b]. The final technical line of adversarial attacks are stemmed from various search-based safety verification works for DNNs, including DeepGame [Wu et al., 2018], L0-TRE [Ruan et al., 2018b], DLV [Huang et al., 2017b] and SafeCV [Wicker et al., 2018], etc. The distinction of those works to the other attacks lies on that the adversarial images are served as counter-examples to verify the unsafety of DNNs otherwise the tested DNNs are safe (w.r.t. a certain safety requirement) with provable guarantees.

6.5 Defence

In response to the susceptibility of neural networks to adversarial examples [Szegedy et al., 2014, Biggio et al., 2013], there have been significant interests recently in developing defence techniques, which are to either identify or reduce adversarial examples so that the decision of the DNN can be more robust. The development of attack and defence techniques has been seen as an "arm-race". For example, most defences against adversarial examples in the white-box setting, including [Papernot et al., 2016a, Metzen et al., 2017, Hendrycks and Gimpel, 2016, Meng and Chen, 2017a], have been demonstrated to be vulnerable to e.g., iterative optimisation-based attacks [Carlini and Wagner, 2017b, Carlini and Wagner, 2017a].

6.5.1 Adversarial Training

Originally proposed by [Goodfellow et al., 2014b], adversarial training solves a min-max game through a conceptually simple process: train on adversarial examples until the model learns to classify them correctly. Given a training data $\mathbf{T}$ and a loss function $\ell(\cdot)$, standard training chooses network weights θ as

$$\theta^* = \arg\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\in\mathbf{T}} \ell(x;y;f_\theta). \tag{31}$$

where f_θ is a DNN parameterized by θ. The adversarial training approach of [Madry et al., 2017] is, for a given ϵ-ball (represented as a d-Neighbourhood), to solve

$$\theta^* = \arg\min_{\theta} \mathop{\mathbb{E}}_{(x,y)\in\mathbf{T}} \left[\max_{\delta\in[-\epsilon,\epsilon]^{s_1}} \ell(x+\delta;y;f_\theta) \right]. \tag{32}$$

Intuitively, it is to assume that all neighbours within the ϵ-ball should have the same class label and should be considered during the training. To approximately solve this formulation, the authors solve the inner maximization problem by generating adversarial examples using projected gradient descent.

Aiming at defending iterative attacks, cascade adversarial machine learning [Na et al., 2018] is proposed to generate adversarial examples at each mini-batch. Technically, it trains a first model, generates adversarial examples (with iterative attack methods) on that model, adds these to the training set, then trains a second model on the augmented dataset, and so on. Moreover, [Tramèr et al., 2018] introduces ensemble adversarial training, which augments training data with perturbations transferred from other models.

6.5.2 Defensive Distillation

Distillation [Hinton et al., 2015] a training procedure which trains a DNN using knowledge transferred from a different DNN. Based on this idea, [Papernot et al., 2016b] 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.

6.5.3 Dimensionality Reduction

[Xu et al., 2018] explores two feature squeezing methods to map original inputs into: reducing the color bit depth of each pixel and spatial smoothing, and shows that by comparing a DNN model's prediction on the original input with that on squeezed input, adversarial examples can be detected.

[Bhagoji et al., 2017] reduces the high dimensional inputs (e.g., 784 for MNIST) to a much smaller k-dimensional input (e.g., 20) and trains a DNN on the smaller inputs.

6.5.4 Input Transformations

A popular defence approach is to do input transformations before feeding an input into the DNN. [Meng and Chen, 2017b] observes 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 classfied correctly.

[Song et al., 2018] argues that adversarial examples mainly lie in the low-probability region of the data distribution, and neural density models are good at detecting imperceptible image perturbations. In their tool PixelDefend, a PixelCNN generative model [van den Oord et al., 2016] is applied to project a potential adversarial example onto the data manifold before feeding it into a classifier. Technically, a greedy decoding procedure is developed to approximate finding the highest probability example within an ϵ-ball of the input image. Along this line, instead of using a PixelCNN, [Samangouei et al., 2018] uses a Generative Adversarial Network [Goodfellow et al., 2014a].

[Guo et al., 2017] exercises 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., 2017] propose to defend against adversarial examples by adding a randomisation layer before the input to the classifier. For a classifier that takes a 299×299 input, the defence first randomly rescales the image to a $r \times r$ image, with $r \in [299, 331)$, and then randomly zero-pads the image so that the result is 331×331. The output is then fed to the classifier.

6.5.5 Combining Input Discretisation with Adversarial Training

Input discretization 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 discretization based approaches apply a non-differentiable and non-linear transformation (discretization) to the input, before passing it into the model. [Buckman et al., 2018] suggests the approach of combining thermometer encoding with adversarial training. Given an image x, for each pixel colour $x_{i,j,c}$, the l-level *thermometer encoding* $\tau(x_{i,j,c})$ is a l-dimensional vector where $\tau(x_{i,j,c})_k = 1$ if $x_{i,j,c} > k/l$, and 0 otherwise (e.g., for a 10-level thermometer encoding, $\tau(0.66) = 1111110000$). Due to the discrete nature of thermometer encoded values, it is not possible to directly perform gradient descent on a thermometer encoded neural network. The authors therefore construct Logit-Space Projected Gradient Ascent (LS-PGA) as

an attack over the discrete thermometer encoded inputs. Using this attack, the authors perform the adversarial training of [Madry et al., 2017] on thermometer encoded networks, and show that the vulnerability of neural network models to adversarial attacks is reduced.

6.5.6 Activation Transformations

Stochastic activation pruning (SAP) [Dhillon et al., 2018] introduces randomness into the evaluation of a DNN, 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 increasing robustness at the price of slightly decreaseing clean classification accuracy.

6.5.7 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 [Ma et al., 2018e], 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., 2017] uses Kernel Density (KD) estimation as a measure to identify adversarial subspaces. Given an input x and its label t, it is to compute

$$KDE(x) = \frac{1}{|\mathbf{T}_t|} \sum_{x \in \mathbf{T}_t} \exp(\frac{|f^{K-1}(x) - f^{K-1}(x)|^2}{\sigma^2}) \tag{33}$$

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 bandwidth. With a threshold τ, it reports x as adversarial if $KDE(x) < \tau$, otherwise reporting x as natural.

[Ma et al., 2018e] uses Local Intrinsic Dimensionality (LID) [Houle, 2017] to measure the adversarial region by considering the local distance distribution from a reference point to its neighbours

6.5.8 Defence against Data Poisoning Attack

Instead of defencing against adversarial attacks, [Jacob Steinhardt, 2017] considers a data poisoning attack in which the attacker is to manipulate a percentage ϵ of the training dataset $\mathbf{T}$ to have a new dataset $\mathbf{T}_p$ such that $|\mathbf{T}_p| = \epsilon|\mathbf{T}|$. The purpose of the attacker is to mislead the defender who trains the model over the set $\mathbf{T} \cup \mathbf{T}_p$. The success of defence or attack is defined with respect to a loss function ℓ.

6.6 Certified Defence

The approaches in Section 6.5 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.

6.6.1 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, 2017] 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

$$\mathcal{G}(f) = \frac{1}{nK^2} \sum_{i=1}^{n} \sum_{l,k=1}^{s_K} ||\nabla f_c(x_i) - \nabla f_k(x_i)||_2^2 \tag{34}$$

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 [Raghunathan et al., 2018], 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.

6.6.2 Robustness through Training Objective

[Sinha et al., 2018] considering the problem

$$\min_{\theta} \sup_{\mathcal{X} \in \mathcal{P}} \mathbb{E}_{\mathcal{X}}[\ell(\theta; x, y)] \tag{35}$$

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.

6.7 Summary of Defence Techniques

In summary, defence techniques in are either to identify adversarial examples 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

The 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 on their predictions. However, to gain the trust from human users, it is essential to enable human users to understand the decisions a DNN has made. Moreover, it is also possible that the results in interpretability can enhance the other techniques such as verification, testing, and adversarial attacks.

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

7.1 Instancewise 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 4, it is to let $t = s_1$ and $\mathtt{expl}(f, x) \in \mathbb{R}^{s_1}$.

7.1.1 Optimising over Hidden Neuron

Recall that $u_{k,l}(x)$ is the activation of the l-th neuron on the k-th layer for the input x. [Erhan et al., 2009] synthesises images that cause high activations for particular neurons by treating the synthesis problem as an optimisation problem as follows.

$$x^* = \arg\max_x u_{k,l}(x) \tag{36}$$

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 synthesize inputs that cause higher and higher activations for the neuron $n_{k,l}$, and eventually terminates at some x^* which is deemed to be a preferred input stimulus for the neuron in question.

7.1.2 Inverting Representation

[Mahendran and Vedaldi, 2015] computes an approximate inverse of an image representation. Let $rep : \mathbb{R}^{s_1} \to \mathbb{R}^{a_1}$ be a representation function, mapping an

input to a representation of a_1 dimensions. Now given a representation T_0 to be inverted, it is to find an input x^* such that

$$x^* = \arg \min_{x \in \mathbb{R}^{s_1}} \ell(rep(x), T_0) + \lambda \mathcal{G}(x) \tag{37}$$

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

$$\mathcal{G}_\alpha(x) = ||x||_\alpha^\alpha \tag{38}$$

where $\alpha = 6$ is taken in their experiments, or a total variation, which encourages images to consist of piece-wise constant patches. For the algorithm to solve the above optimsation expression (37), the gradient descent procedure is extended to use momentum.

[Dosovitskiy and Brox, 2015] proposes to analyse which information is preserved by a feature representation and which information is discarded. Given a feature vector, it trains 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 $\{x_i, \phi_i\}$, it is to learn the weight w of a network $f(\phi, w)$ to minimise a Monte-Carlo estimate of the loss as follows:

$$\hat{w} = \arg \min_w \sum_i ||x_i - f(\phi_i, w)||_2^2 \tag{39}$$

Moreover, it has been argued in [Yosinski et al., 2015] 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, it is to find a mapping $r_x : \Psi \to \mathbb{R}$ such that each feature $\psi \in \Psi$ has a rank $r_x(\psi)$. Intuitively, the ranking r_x provides a total order between features on their contributions to a decision making of $\mathcal{N}$ on the input x.

7.2.1 Local Interpretable Model-agnostic Explanations (LIME)

Local Interpretable Model-agnostic Explanations (LIME) [Ribeiro et al., 2016] 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, and to compute the rank, it minimises over the following objective function:

$$\texttt{expl}(f, x) = \arg \min_{e \in \mathcal{E}} \ell(f, e, \pi_x) + \mathcal{G}(e) \tag{40}$$

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 reguralisation 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., 2017] suggests that a good explanation should satisfy the following two axiomatic attributes:

- sensitivity, which requires that, for every input and baseline that differ in one feature but have different predictions, the differing feature should be given a non-zero attribution.

- implementation invariance, which requires that the attributions are always identical for two functionally equivalent networks.

Based on these axiomatic attributes, they propose the integrated gradient and show that it is the only method to satisfy certain desirable axioms including sensitivity and implementation invariance. Technically, the integrated gradient along the i-th dimension for an input x and a baseline x' is as follows.

$$\text{IntGrad}_i(x) = (x_i - x_i') \times \int_0^1 \frac{\partial f(x' + \alpha \times (x - x'))}{\partial x_i} d\alpha \qquad (41)$$

where $\frac{\partial f(x)}{\partial x_i}$ is the gradient of $f(x)$ along the i-th dimension. 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

$$\sum_{i=1}^{n} \text{IntGrad}_i(x) = f(x) - f(x'). \qquad (42)$$

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 [Bach et al., 2015] is a concept of pixel-wise decomposition

$$f(x) \approx \sum_{d=1}^{s_1} R_d \qquad (43)$$

to understand how much a single pixel d 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. It is suggested in [Shrikumar et al., 2017] that LRP is equivalent to the Gradient $\odot$ 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., 2017] is a recursive prediction explanation method. It attributes to each input x_i a value $C_{\Delta x_i \Delta y}$ that represents the effect of that input being set to a reference value as opposed 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:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta y} = \Delta o, \tag{44}$$

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) [Selvaraju et al., 2016] computes a feature-importance map (i.e., a coarse localization) 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.

$$\alpha_k^c = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k} \tag{45}$$

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. 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) [Lundberg and Lee, 2017] suggests take 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.

$$\texttt{expl}_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} [f_x(z') - f_x(z' \setminus i)] \tag{46}$$

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'.

7.2.7 Prediction Difference Analysis

[Zintgraf et al., 2017] 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

$$p(c|x_{-i}) = \sum_{x_i} p(x_i|x_{-i})p(c|x) \tag{47}$$

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., 2018a] 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

$$S_{C,k,l}(x) = \nabla h_{l,k}(f_l(x)) \cdot v_C^l \tag{48}$$

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

$$\text{TCAV}_{Q_{C,k,l}} = \frac{|\{x \in X_k : S_{C,k,l}(x) > 0\}|}{|X_k|} \tag{49}$$

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) [Chen et al., 2018] 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. 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, the problem is to find a solution for the following optimisation problem:

$$\max_e I(X_S; Y) \text{ subject to } S \sim e(X) \tag{50}$$

where $I(X_S; Y)$ represents the mutual information between features X_S and the output Y. Intuitively, the explainer is to find a set of features to maximise the mutual information between selected features and the output.

Because the problem is computationally hard to solve, a variational approximation is developed to compute it efficiently, and a subset sampling trick is applied to relax the categorical distribution of discrete feature subsets to continuous differentiable approximation. The final objective is to optimise the following approximation

$$\max_{\theta, \alpha} E_{X,Y,\xi}[\log g_\alpha(V(\theta, \xi) \odot X, Y)] \tag{51}$$

where ξ is a set of auxiliary random variables sampled independently from the Gumbel distribution, $V(\theta, \xi)$ is a multi-dimensional random vector, g_α is a neural network for parameterising the family of conditional probabilities invoked by mutual information quantities, the element-wise product $V(\theta, \xi) \odot X$ is used to approximate a feature subset sampling X_S, and θ is associated with a weight vector $w_\theta(X)$ that ranks the importance of the corresponding features in X.

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.

7.3.1 Gradient-based Methods

Gradient [Simonyan et al., 2013]. 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

$$\max_x S_c(x) - \lambda ||x||_2^2 \tag{52}$$

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 a tiny change of each pixel of the input x affects the classification score c.

To sharpen the sensitive maps, **SmoothGrad** [Smilkov et al., 2017] 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, **Grad $\odot$ Input** [Shrikumar et al., 2017] method yields another explanation $\hat{w}_c(x) = x \odot w_c(x)$ to reduce visual diffusion. **DeConvNet** [Zeiler and Fergus, 2014] highlights the portions of a particular image that are responsible for the firing of each neural unit. **Guided Backpropagation** [Springenberg et al., 2014] builds upon DeConvNet and sets negative gradients to zero during backpropagation.

7.3.2 Perturbation-based Methods

[Dabkowski and Gal, 2017] proposed 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, 2017] proposed 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

$$
\Phi(u) = \begin{cases}
m(u)x(u) + (1 - m(u))\mu, & \text{Constant mapping} \\
m(u)x(u) + (1 - m(u))\eta(u), & \text{Noising} \\
\int g(v - u)x(v)\mathrm{d}v, & \text{Blurring}
\end{cases}
\tag{53}
$$

where the constant μ is the average color, $\eta(u)$ 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, 2017] 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,

$$
\hat{\theta}_{\epsilon,z} = \arg\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} \ell(z_i, \theta) + \epsilon\ell(z, \theta)
\tag{54}
$$

and evaluate the influence by

$$
\mathcal{I}(z) = \frac{d\hat{\theta}_{\epsilon,z}}{d\epsilon}\bigg|_{\epsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta\ell(z, \hat{\theta})
\tag{55}
$$

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

$$\hat{\theta}_{-z} - \hat{\theta} \approx -\frac{1}{n}\mathcal{I}(z) \tag{56}$$

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.

7.5.1 Rule Extraction

[Ribeiro et al., 2018] 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

$$\mathop{\mathbb{E}}_{\mathcal{D}(z|A)} \left[\mathbb{1}_{f(x)=f(z)} \right] > 1 - \epsilon \text{ and } A(x) = 1 \tag{57}$$

where $\mathcal{D}(\cdot|A)$ is the conditional distribution when the rule A applies. 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 [Zilke et al., 2016], 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) [Sato and Tsukimoto, 2001] and C4.5 decision tree learning algorithm.

7.5.3 Linear Classifiers to Approximate Piece-wise Linear Neural Networks

[Chu et al., 2018] 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 the decision features of the polytope boundaries.

7.5.4 Automata Extraction from Recurrent Neural Networks

[Weiss et al., 2018] extends the L* automata learning algorithm [Angluin, 1987] 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 Giles, 1996] 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., 2000].

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 [Tishby et al., 2000] is to figure out the stochastic encoder $p(t|x)$, i.e.,

$$\text{IB:} \min_{p(t|x)} \quad \mathcal{L}_{\text{IB}} = I(X;T) - \beta I(T;Y) \tag{58a}$$

$$\text{subject to} \quad T \leftrightarrow X \leftrightarrow Y \tag{58b}$$

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

$$p(t|x) = \frac{p(t)}{Z(x, \beta)} \exp[-\beta D_{KL}[p(y|x) \mid p(y|t)]] \tag{59}$$

$$p(y|t) = \frac{1}{p(t)} \sum_x p(t|x)p(x, y) \tag{60}$$

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

$$Z(x, \beta) = \exp[\frac{\lambda(x)}{p(x)} - \beta \sum_y p(y|x) \log \frac{p(y|x)}{p(y)}] \tag{61}$$

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.

7.6.2 Information Plane

Given a K-layer deterministic DNN with T_k being a multivariate random variables representing the output of k-th hidden layer, [Shwartz-Ziv and Tishby, 2017] introduced the concept of information plane with coordinates $(I(X; T_k), I(T_k, Y))$.[3]

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

$$\text{IB:} \quad \min_{p(t_k|x)} \quad \mathcal{L}_{\text{IB}} = I(X; T_k) - \beta I(T_k; Y) \tag{62a}$$

$$\text{subject to} \quad Y \leftrightarrow X \leftrightarrow T_1 \leftrightarrow \cdots \leftrightarrow T_K \leftrightarrow \hat{Y} \tag{62b}$$

where the coordinates of the information planes follow

$$I(X; Y) \geq I(T_1; Y) \geq \cdots \geq I(T_K; Y) \geq I(\hat{Y}; Y) \tag{63}$$

$$H(X) \geq I(X; T_1) \geq \cdots \geq I(X; T_K) \geq I(X; \hat{Y}). \tag{64}$$
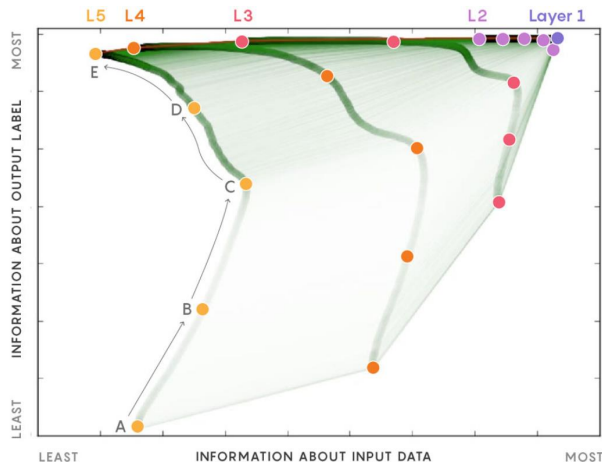
Figure 12: The evolution of Information Plane $(I(X; T_k), I(X_k; Y))$ during the training procedure of a 5-layer DNN [qua, 2017], where A-B-C indicates the fitting phase, and C-D-E is the compression phase.

---

[3]According to [Goldfeld et al., 2018], the quantity computed and plotted in [Shwartz-Ziv and Tishby, 2017] 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."

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, 2017] that the training procedure consists of two phases:

- 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 $I(X_k; Y)$ 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 of the output.

It was also claimed in [Shwartz-Ziv and Tishby, 2017] 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 triggered 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., [Saxe et al., 2018, Amjad and Geiger, 2018, Goldfeld et al., 2018]) 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 constaint value (when X is discrete) or infinite (when X is continous), given a deterministic DNN.

To resolve this issue, several statistical 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., 2018] suggested injecting an independent Gaussian noise with user-defined variance to T_k followed by quantisation. [Amjad and Geiger, 2018] 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., 2018] 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.

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., 2017] 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., 2015] 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, 2018] 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 [Wang et al., 2003], 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, 2017] become available.

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 [Clarke et al., 2003], etc) or to provide useful information to the system designers for them to improve the system. Similar for the software testing, where bugs are utilised to repair implementation errors (e.g., automatic repair techniques [Könighofer and Bloem, 2013], 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 6.5 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., [Sun et al., 2018b, Sun et al., 2018c], 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 [Mnih et al., 2015, Schaul et al., 2015, Wang et al., 2016, van Hasselt et al., 2015] 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 observations instead of a single observation. A few attack techniques, such as [Huang et al., 2017a, Pattanaik et al., 2018, Lin et al., 2017, Tretschk et al., 2018], have been proposed, with the key techniques based on generalising the idea of FGSM [Goodfellow et al., 2014b]. For recurrent neural networks, there are a few attack techniques such as [Wei et al., 2018]. Less works have 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 technqiues. Similar with 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 [Huang et al., 2017b, Katz et al., 2017, Lomuscio and Maganti, 2017, Ehlers, 2017], recent efforts have been on approximate techniques to alleviate the computational problem while still achieve tangible bounds, e.g., [Wicker et al., 2018, Ruan et al., 2018a, Gehr et al., 2018]. 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, like [Clarke et al., 2000] did for concurrent systems, and it will be interesting to see how it is related to the layer-by-layer refinement [Huang et al., 2017b].

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 with 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 [Sun et al., 2018a]. Such semantic difference suggests 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 [Sun et al., 2018b] 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

Real-world systems contain both logic components, to handle e.g., high-level 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 such as 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.

8.8 Distributional Shift 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., [Goodfellow et al., 2016] for a comprehensive overview), as suggested in e.g., [Amodei et al., 2016, Ashmore and Hill, 2018, Moreno-Torres et al., 2012], 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., 2018] suggests to have an extra class "dustbin" for such inputs.

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.

8.9 Formulation of Interpretability

Interpretability has been widely discussed without a rigorous definition, although there are comprehensive surveys such as [Lipton, 2018] aiming to study this concept from many different angles. Existing research is able to provide various partial information about the DNNs, and these works can hardly be compared

with each other, due to the lack of a systematic definition. Recently, a few works have been done towards accommodating several similar approaches with a single, unified approach, to enable a comparison. For example, [Olah et al., 2017] suggest that many visualization methods are based on optimisation with different regularisation terms, [Lundberg and Lee, 2017] use Shapley value to explain a few attribute-based approaches with an additive model, and [Ancona et al., 2018] use a modified gradient function to accommodate a few gradient-based approaches. While it may be infeasible to have a single definition for interpretability, it is necessary to formally define it from a few aspects.

8.10 Application of Interpretability to other Tasks

Except for the size of DNNs, the key difficulty of verifying, testing, or attacking DNNs are on the fact that DNNs are black-box. It is therefore reasonable to expect that, the information provided by interpretability techniques 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 application of interpretability in attack techniques have been seen in e.g., [Papernot et al., 2016c] and [Ruan et al., 2018b], 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 increases 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 can human users help on improving the trustworthiness of the DNNs, is less explored, with only a few works such as [Tamagnini et al., 2017], 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 [Hardin, 2002]. A classical definition from organisation theory [Mayer et al., 1995] 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 [Huang and Kwiatkowska, 2017] with a comprehensive reasoning framework.

9   Conclusions

In this survey, we review 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 several existing areas which do not have much overlaps before, including formal verification, software testing, machine learning, and logic reasoning.

References

[GDR, 2016] (2016). General data protection regulation. `http://data.europa.eu/eli/reg/2016/679/oj`.

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

[exp, 2018] (2018). Explainable artificial intelligence. `https://www.darpa.mil/program/explainable-artificial-intelligence`.

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

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

[Abbasi et al., 2018] Abbasi, M., Rajabi, A., Sadat Mozafari, A., Bobba, R. B., and Gagne, C. (2018). Controlling Over-generalization and its Effect on Adversarial Examples Generation and Detection. *ArXiv e-prints*.

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

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

[Ammann and Offutt, 2008] Ammann, P. and Offutt, J. (2008). *Introduction to Software Testing*. Cambridge University Press.

[Amodei et al., 2016] Amodei, D., Olah, C., Steinhardt, J., Christiano, P., Schulman, J., and Mané, D. (2016). Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*.

[Ancona et al., 2018] Ancona, M., Ceolini, E., Öztireli, C., and Gross, M. (2018). Towards better understanding of gradient-based attribution methods for deep neural networks. In *International Conference on Learning Representations*.

[Angluin, 1987] Angluin, D. (1987). Learning regaular sets from queries and counterexamples. *Information and Computation*, 75:87–106.

[Ashmore and Hill, 2018] Ashmore, R. and Hill, M. (2018). Boxing clever: Practical techniques for gaining insights into training data and monitoring distribution shift. In *First International Workshop on Artificial Intelligence Safety Engineering*.

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

[Bastani et al., 2016] Bastani, O., Ioannou, Y., Lampropoulos, L., Vytiniotis, D., Nori, A. V., and Criminisi, A. (2016). Measuring neural net robustness with constraints. In *NIPS*.

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

[Biggio et al., 2013] Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., and Roli, F. (2013). Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer.

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

[Buckman et al., 2018] Buckman, J., Roy, A., Raffel, C., and Goodfellow, I. (2018). Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations*.

[Bunel et al., 2017] Bunel, R., Turkaslan, I., Torr, P. H., Kohli, P., and Kumar, M. P. (2017). Piecewise linear neural network verification: A comparative study. *arXiv preprint arXiv:1711.00455*.

[Carlini and Wagner, 2017a] Carlini, N. and Wagner, D. (2017a). Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 3–14. ACM.

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

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

[Chen et al., 2018] Chen, J., Song, L., Wainwright, M. J., and Jordan, M. I. (2018). Learning to explain: An information-theoretic perspective on model interpretation. In *International Conference on Machine Learning*.

[Cheng et al., 2018a] Cheng, C.-H., Huang, C.-H., and Nührenberg, G. (2018a). nn-dependability-kit: Engineering neural networks for safety-critical systems. *arXiv preprint arXiv:1811.06746*.

[Cheng et al., 2018b] Cheng, C.-H., Huang, C.-H., and Yasuoka, H. (2018b). Quantitative projection coverage for testing ML-enabled autonomous systems. In *International Symposium on Automated Technology for Verification and Analysis*. Springer.

[Cheng et al., 2018c] Cheng, C.-H., Nührenberg, G., Huang, C.-H., and Yasuoka, H. (2018c). Towards dependability metrics for neural networks. In *Proceedings of the 16th ACM-IEEE International Conference on Formal Methods and Models for System Design*.

[Cheng et al., 2017] Cheng, C.-H., Nührenberg, G., and Ruess, H. (2017). Maximum resilience of artificial neural networks. In D'Souza, D. and Narayan Kumar, K., editors, *Automated Technology for Verification and Analysis*, pages 251–268. Springer.

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

[Chu et al., 2018] Chu, L., Hu, X., Hu, J., Wang, L., and Pei, J. (2018). Exact and consistent interpretation for piecewise linear neural networks: A closed form solution. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1244–1253. ACM.

[Clarke et al., 2000] Clarke, E., Grumberg, O., Jha, S., Lu, Y., and Veith, H. (2000). Counterexample-guided abstraction refinement. In Emerson, E. A. and Sistla, A. P., editors, *Computer Aided Verification*, pages 154–169, Berlin, Heidelberg. Springer Berlin Heidelberg.

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

[Clarke Jr et al., 2018] Clarke Jr, E. M., Grumberg, O., Kroening, D., Peled, D., and Veith, H. (2018). *Model checking*. The MIT Press.

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

[Cousot and Cousot, 1977] Cousot, P. and Cousot, R. (1977). 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)*, pages 238–252.

[Dabkowski and Gal, 2017] Dabkowski, P. and Gal, Y. (2017). Real time image saliency for black box classifiers. In *NIPS*.

[Dhillon et al., 2018] Dhillon, G. S., Azizzadenesheli, K., Lipton, Z. C., Bernstein, J., Kossaifi, J., Khanna, A., and Anandkumar, A. (2018). Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations*.

[Dosovitskiy and Brox, 2015] Dosovitskiy, A. and Brox, T. (2015). Inverting convolutional networks with convolutional networks. *CoRR*, abs/1506.02753.

[Dutta et al., 2018] Dutta, S., Jha, S., Sanakaranarayanan, S., and Tiwari, A. (2018). Output range analysis for deep neural networks. *arXiv preprint arXiv:1709.09130*.

[Dvijotham et al., 2018] Dvijotham, K., Stanforth, R., Gowal, S., Mann, T. A., and Kohli, P. (2018). A dual approach to scalable verification of deep networks. *CoRR*, abs/1803.06567.

[Ebrahimi et al., 2018] Ebrahimi, J., Rao, A., Lowd, D., and Dou, D. (2018). 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)*, volume 2, pages 31–36.

[Ehlers, 2017] Ehlers, R. (2017). Formal verification of piece-wise linear feedforward neural networks. In D'Souza, D. and Narayan Kumar, K., editors, *Automated Technology for Verification and Analysis*, pages 269–286, Cham. Springer International Publishing.

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

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

[Feinman et al., 2017] Feinman, R., Curtin, R. R., Shintre, S., and Gardner, A. B. (2017). Detecting Adversarial Samples from Artifacts. *arXiv e-prints*, page arXiv:1703.00410.

[Finlayson et al., 2018] Finlayson, S. G., Kohane, I. S., and Beam, A. L. (2018). Adversarial attacks against medical deep learning systems. *arXiv preprint arXiv:1804.05296*.

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

[Gehr et al., 2018] Gehr, T., Mirman, M., Drachsler-Cohen, D., Tsankov, P., Chaudhuri, S., and Vechev, M. (2018). Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (S&P2018)*, pages 948–963.

[Goldfeld et al., 2018] Goldfeld, Z., van den Berg, E., Greenewald, K., Melnyk, I., Nguyen, N., Kingsbury, B., and Polyanskiy, Y. (2018). Estimating Information Flow in Neural Networks. *arXiv e-prints*, page arXiv:1810.05728.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[Goodfellow et al., 2014a] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Goodfellow, 2018] Goodfellow, I. J. (2018). Gradient masking causes CLEVER to overestimate adversarial perturbation size. *CoRR*, abs/1804.07870.

[Goodfellow et al., 2014b] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572.

[Gopinath et al., 2018] Gopinath, D., Wang, K., Zhang, M., Pasareanu, C. S., and Khurshid, S. (2018). Symbolic execution for deep neural networks. *arXiv preprint arXiv:1807.10439*.

[Guo et al., 2017] Guo, C., Rana, M., Cisse, M., and van der Maaten, L. (2017). Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*.

[Guo et al., 2018] Guo, J., Jiang, Y., Zhao, Y., Chen, Q., and Sun, J. (2018). DLFuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 12nd Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2018*.

[Hardin, 2002] Hardin, R. (2002). *Trust and trustworthiness*. Russell Sage Foundation.

[Hayes and Danezis, 2018] Hayes, J. and Danezis, G. (2018). Learning universal adversarial perturbations with generative models. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE.

[Hayhurst et al., 2001] Hayhurst, K., Veerhusen, D., Chilenski, J., and Rierson, L. (2001). A practical tutorial on modified condition/decision coverage. Technical report, NASA.

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

[Hein and Andriushchenko, 2017] Hein, M. and Andriushchenko, M. (2017). Formal guarantees on the robustness of a classifier against adversarial manipulation. In *NIPS*.

[Hendrycks and Gimpel, 2016] Hendrycks, D. and Gimpel, K. (2016). Early methods for detecting adversarial images. *arXiv preprint arXiv:1608.00530*.

[Hinton et al., 2015] Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the Knowledge in a Neural Network. *arXiv e-prints*, page arXiv:1503.02531.

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

[Huang et al., 2017a] Huang, S. H., Papernot, N., Goodfellow, I. J., Duan, Y., and Abbeel, P. (2017a). Adversarial attacks on neural network policies. *CoRR*, abs/1702.02284.

[Huang and Kwiatkowska, 2017] Huang, X. and Kwiatkowska, M. (2017). Reasoning about cognitive trust in stochastic multiagent systems. In *AAAI2017*, pages 3768–3774.

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

[Jacob Steinhardt, 2017] Jacob Steinhardt, Pang Wei Koh, P. L. (2017). Certified defenses for data poisoning attacks. In *Advances in Neural Information Processing Systems 30*.

[Jia and Harman, 2011] Jia, Y. and Harman, M. (2011). An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678.

[Johnson et al., 2016] Johnson, J., Alahi, A., and Fei-Fei, L. (2016). Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer.

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

[Kim et al., 2018a] Kim, B., Wattenberg, M., Gilmer, J., Cai, C., Wexler, J., Viegas, F., and sayres, R. (2018a). Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (TCAV). In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2668–2677, Stockholmsmässan, Stockholm Sweden. PMLR.

[Kim et al., 2018b] Kim, J., Feldt, R., and Yoo, S. (2018b). Guiding deep learning system testing using surprise adequacy. *arXiv preprint arXiv:1808.08444*.

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

[Koh and Liang, 2017] Koh, P. W. and Liang, P. (2017). Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning*.

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

[Kurakin et al., 2016] Kurakin, A., Goodfellow, I. J., and Bengio, S. (2016). Adversarial examples in the physical world. *CoRR*, abs/1607.02533.

[Li et al., 2018] Li, J., Liu, J., Yang, P., Chen, L., and Huang, X. (2018). Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. *Submitted*.

[Lin et al., 2017] Lin, Y., Hong, Z., Liao, Y., Shih, M., Liu, M., and Sun, M. (2017). Tactics of adversarial attack on deep reinforcement learning agents. In *International Joint Conference on Artificial Intelligence*.

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

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

[Lu et al., 2017] Lu, J., Sibai, H., Fabry, E., and Forsyth, D. (2017). NO Need to Worry about Adversarial Examples in Object Detection in Autonomous Vehicles. In *Conference on Computer Vision and Pattern Recognition, Spotlight Oral Workshop*.

[Lundberg and Lee, 2017] Lundberg, S. and Lee, S. (2017). A unified approach to interpreting model predictions. *NIPS*.

[Ma et al., 2018a] Ma, L., Juefei-Xu, F., Sun, J., Chen, C., Su, T., Zhang, F., Xue, M., Li, B., Li, L., Liu, Y., Zhao, J., and Wang, Y. (2018a). 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*.

[Ma et al., 2018b] Ma, L., Zhang, F., Sun, J., Xue, M., Li, B., Juefei-Xu, F., Xie, C., Li, L., Liu, Y., Zhao, J., et al. (2018b). DeepMutation: Mutation testing of deep learning systems. In *Software Reliability Engineering, IEEE 29th International Symposium on*.

[Ma et al., 2018c] Ma, L., Zhang, F., Xue, M., Li, B., Liu, Y., Zhao, J., and Wang, Y. (2018c). Combinatorial testing for deep learning systems. *arXiv preprint arXiv:1806.07723*.

[Ma et al., 2018d] Ma, S., Liu, Y., Zhang, X., Lee, W.-C., and Grama, A. (2018d). 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.

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

[Madry et al., 2017] Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*.

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

[Mayer et al., 1995] Mayer, R. C., Davis, J. H., and Schoorman, F. D. (1995). An integrative model of organizational trust. *Academy of management review*, 20(3):709–734.

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

[Meng and Chen, 2017b] Meng, D. and Chen, H. (2017b). Magnet: a two-pronged defense against adversarial examples. In *ACM Conference on Computer and Communications Security*.

[Metzen et al., 2017] Metzen, J. H., Genewein, T., Fischer, V., and Bischoff, B. (2017). On detecting adversarial perturbations. *arXiv preprint arXiv:1702.04267*.

[Mirman et al., 2018] Mirman, M., Gehr, T., and Vechev, M. (2018). Differentiable abstract interpretation for provably robust neural networks. In *International Conference on Machine Learning*, pages 3575–3583.

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

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

[Moosavi-Dezfooli et al., 2016] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582.

[Moreno-Torres et al., 2012] Moreno-Torres, J. G., Raeder, T., Alaiz-Rodríguez, R., Chawla, N. V., and Herrera, F. (2012). A unifying view on dataset shift in classification. *Pattern Recogn.*, 45(1):521–530.

[Na et al., 2018] Na, T., Ko, J. H., and Mukhopadhyay, S. (2018). Cascade adversarial machine learning regularized with a unified embedding. In *International Conference on Learning Representations (ICLR)*.

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

[Narodytska, 2018] Narodytska, N. (2018). Formal analysis of deep binarized neural networks. In *IJCAI*, pages 5692–5696.

[Narodytska et al., 2018] Narodytska, N., Kasiviswanathan, S. P., Ryzhyk, L., Sagiv, M., and Walsh, T. (2018). Verifying properties of binarized deep neural networks. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

[Neumaier and Shcherbina, 2004] Neumaier, A. and Shcherbina, O. (2004). Safe bounds in linear and mixed-integer linear programming. *Math. Program.*, 99(2):283–296.

[Odena and Goodfellow, 2018] Odena, A. and Goodfellow, I. (2018). TensorFuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875*.

[Olah et al., 2017] Olah, C., Mordvintsev, A., and Schubert, L. (2017). Feature visualization. *Distill*. https://distill.pub/2017/feature-visualization.

[Omlin and Giles, 1996] Omlin, C. W. and Giles, C. L. (1996). Extraction of rules from discrete-time recurrent neural networks. *Neural Networks*, 9(1):41–52.

[OSearcoid, 2006] OSearcoid, M. (2006). *Metric Spaces*. Springer Science & Business Media.

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

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

[Papernot et al., 2016b] Papernot, N., McDaniel, P., Wu, X., Jha, S., and Swami, A. (2016b). Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597.

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

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

[Peck et al., 2017] Peck, J., Roels, J., Goossens, B., and Saeys, Y. (2017). Lower bounds on the robustness to adversarial perturbations. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 804–813. Curran Associates, Inc.

[Pei et al., 2017a] Pei, K., Cao, Y., Yang, J., and Jana, S. (2017a). DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18. ACM.

[Pei et al., 2017b] Pei, K., Cao, Y., Yang, J., and Jana, S. (2017b). Towards practical verification of machine learning: The case of computer vision systems. *arXiv preprint arXiv:1712.01785.*

[Poursaeed et al., 2018] Poursaeed, O., Katsman, I., Gao, B., and Belongie, S. J. (2018). Generative adversarial perturbations. In *Conference on Computer Vision and Pattern Recognition.*

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

[Raghunathan et al., 2018] Raghunathan, A., Steinhardt, J., and Liang, P. (2018). Certified defenses against adversarial examples. *arXiv preprint arXiv:1801.09344*.

[Ribeiro et al., 2016] Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). "why should I trust you?": Explaining the predictions of any classifier. In *HLT-NAACL Demos*.

[Ribeiro et al., 2018] Ribeiro, M. T., Singh, S., and Guestrin, C. (2018). Anchors: High-precision model-agnostic explanations. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*.

[Ronneberger et al., 2015] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.

[RTCA, 2011] RTCA (2011). Do-178c, software considerations in airborne systems and equipment certification.

[Ruan et al., 2018a] Ruan, W., Huang, X., and Kwiatkowska, M. (2018a). Reachability analysis of deep neural networks with provable guarantees. In *IJCAI*, pages 2651–2659.

[Ruan et al., 2018b] Ruan, W., Wu, M., Sun, Y., Huang, X., Kroening, D., and Kwiatkowska, M. (2018b). Global robustness evaluation of deep neural networks with provable guarantees for L0 norm. *arXiv preprint arXiv:1804.05805*.

[Rushby, 2015] Rushby, J. (2015). The interpretation and evaluation of assurance cases. Technical report, SRI International.

[Russakovsky et al., 2015] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252.

[Salay and Czarnecki, 2018] Salay, R. and Czarnecki, K. (2018). Using machine learning safely in automotive software: An assessment and adaption of software process requirements in iso 26262. *arXiv preprint arXiv:1808.01614*.

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

[Sato and Tsukimoto, 2001] Sato, M. and Tsukimoto, H. (2001). Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222)*, volume 3, pages 1870–1875 vol.3.

[Saxe et al., 2018] Saxe, A. M., Bansal, Y., Dapello, J., Advani, M., Kolchinsky, A., Tracey, B. D., and Cox, D. D. (2018). On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*.

[Schaul et al., 2015] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *CoRR*, abs/1511.05952.

[Selvaraju et al., 2016] Selvaraju, R. R., Das, A., Vedantam, R., Cogswell, M., Parikh, D., and Batra, D. (2016). 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*.

[Shen et al., 2018] Shen, W., Wan, J., and Chen, Z. (2018). MuNN: Mutation analysis of neural networks. In *International Conference on Software Quality, Reliability and Security Companion, QRS-C*. IEEE.

[Shrikumar et al., 2017] Shrikumar, A., Greenside, P., and Kundaje, A. (2017). Learning important features through propagating activation differences. In *Proceedings of Machine Learning Research 70:3145-3153*.

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

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

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

[Sinha et al., 2018] Sinha, A., Namkoong, H., and Duchi, J. (2018). Certifiable distributional robustness with principled adversarial training. In *International Conference on Learning Representations*.

[Smilkov et al., 2017] Smilkov, D., Thorat, N., Kim, B., Viégas, F. B., and Wattenberg, M. (2017). Smoothgrad: removing noise by adding noise. *CoRR*, abs/1706.03825.

[Song et al., 2018] Song, Y., Kim, T., Nowozin, S., Ermon, S., and Kushman, N. (2018). Pixeldefend: Leveraging generative models to understand and defend against adversarial examples. In *International Conference on Learning Representations*.

[Springenberg et al., 2014] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. A. (2014). Striving for simplicity: The all convolutional net. *CoRR*, abs/1412.6806.

[Su et al., 2017] Su, T., Wu, K., Miao, W., Pu, G., He, J., Chen, Y., and Su, Z. (2017). A survey on data-flow testing. *ACM Computing Surveys*, 50(1):5:1–5:35.

[Sun et al., 2018a] Sun, Y., Huang, X., and Kroening, D. (2018a). Testing deep neural networks. *arXiv preprint arXiv:1803.04792*.

[Sun et al., 2018b] Sun, Y., Huang, X., Kroening, D., Shap, J., Hill, M., and Ashmore, R. (2018b). Structural test coverage criteria for deep neural networks.

[Sun et al., 2018c] Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., and Kroening, D. (2018c). Concolic testing for deep neural networks. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on.*

[Sun et al., 2018d] Sun, Y., Wu, M., Ruan, W., Huang, X., Kwiatkowska, M., and Kroening, D. (2018d). Deepconcolic: Testing and debugging deep neural networks. In *41st ACM/IEEE International Conference on Software Engineering (ICSE2019).*

[Sundararajan et al., 2017] Sundararajan, M., Taly, A., and Yan, Q. (2017). Axiomatic attribution for deep networks. In *International Conference on Machine Learning.*

[Szegedy et al., 2014] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2014). Intriguing properties of neural networks. In *In ICLR*. Citeseer.

[Tamagnini et al., 2017] Tamagnini, P., Krause, J., Dasgupta, A., and Bertini, E. (2017). Interpreting black-box classifiers using instance-level visual explanations. In *Proceedings of the 2Nd Workshop on Human-In-the-Loop Data Analytics*, HILDA'17, pages 6:1–6:6, New York, NY, USA. ACM.

[Tian et al., 2018] Tian, Y., Pei, K., Jana, S., and Ray, B. (2018). DeepTest: Automated testing of deep-neural-network-driven autonomous cars. *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 303–314.

[Tishby et al., 2000] Tishby, N., Pereira, F. C., and Bialek, W. (2000). The information bottleneck method. *arXiv e-prints*, page physics/0004057.

[Tramèr et al., 2018] Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., and McDaniel, P. (2018). Ensemble Adversarial Training: Attacks and Defenses. In *International Conference on Learning Representations.*

[Tretschk et al., 2018] Tretschk, E., Oh, S. J., and Fritz, M. (2018). Sequential attacks on agents for long-term adversarial goals. *CoRR*, abs/1805.12487.

[Udeshi et al., 2018] Udeshi, S., Arora, P., and Chattopadhyay, S. (2018). Automated directed fairness testing. In *Automated Software Engineering (ASE), 33rd IEEE/ACM International Conference on.*

[van den Oord et al., 2016] van den Oord, A., Kalchbrenner, N., Vinyals, O., Espeholt, L., Graves, A., and Kavukcuoglu, K. (2016). Conditional image generation with pixelcnn decoders. In *NIPS*.

[van Hasselt et al., 2015] van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double q-learning. In *Association for the Advancement of Artificial Intelligence*.

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

[Wand and Jones, 1994] Wand, M. P. and Jones, M. C. (1994). *Kernel smoothing*. Chapman and Hall/CRC.

[Wang et al., 2018] Wang, S., Pei, K., Whitehouse, J., Yang, J., and Jana, S. (2018). Formal security analysis of neural networks using symbolic intervals. In *USENIX Security Symposium*. USENIX Association.

[Wang et al., 2016] Wang, Z., de Freitas, N., and Lanctot, M. (2016). Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*.

[Wang et al., 2003] Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multi-scale structural similarity for image quality assessment. In *Signals, Systems and Computers, Conference Record of the Thirty-Seventh Asilomar Conference on*.

[Wei et al., 2018] Wei, X., Zhu, J., and Su, H. (2018). Sparse adversarial perturbations for videos. *CoRR*, abs/1803.02536.

[Weiss et al., 2018] Weiss, G., Goldberg, Y., and Yahav, E. (2018). Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning*.

[Weng et al., 2018] Weng, T.-W., Zhang, H., Chen, H., Song, Z., Hsieh, C.-J., Boning, D., Dhillon, I. S., and Daniel, L. (2018). Towards fast computation of certified robustness for relu networks. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, pages 5273–5282.

[Weng et al., 2018] Weng, T.-W., Zhang, H., Chen, P.-Y., Yi, J., Su, D., Gao, Y., Hsieh, C.-J., and Daniel, L. (2018). Evaluating the Robustness of Neural Networks: An Extreme Value Theory Approach. In *International Conference on Learning Representations (ICLR)*.

[Wicker et al., 2018] Wicker, M., Huang, X., and Kwiatkowska, M. (2018). Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 408–426. Springer.

[Wong and Kolter, 2018] Wong, E. and Kolter, Z. (2018). Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5283–5292.

[Wu et al., 2018] Wu, M., Wicker, M., Ruan, W., Huang, X., and Kwiatkowska, M. (2018). A game-based approximate verification of deep neural networks with provable guarantees. *arXiv preprint arXiv:1807.03571*.

[Xiang et al., 2018] Xiang, W., Tran, H.-D., and Johnson, T. T. (2018). Output reachable set estimation and verification for multi-layer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29:5777–5783.

[Xiao et al., 2018] Xiao, C., Zhu, J.-Y., Li, B., He, W., Liu, M., and Song, D. (2018). Spatially transformed adversarial examples. *arXiv preprint arXiv:1801.02612*.

[Xie et al., 2017] Xie, C., Wang, J., Zhang, Z., Ren, Z., and Yuille, A. (2017). Mitigating adversarial effects through randomization. In *International Conference on Learning Representations (ICLR)*.

[Xie et al., 2018] Xie, X., Ma, L., Juefei-Xu, F., Chen, H., Xue, M., Li, B., Liu, Y., Zhao, J., Yin, J., and See, S. (2018). Coverage-guided fuzzing for deep neural networks. *arXiv preprint arXiv:1809.01266*.

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

[Yosinski et al., 2015] Yosinski, J., Clune, J., Nguyen, A. M., Fuchs, T. J., and Lipson, H. (2015). Understanding neural networks through deep visualization. In *International Conference on Machine Learning (ICML) Deep Learning Workshop*.

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

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

[Zhu et al., 1997] Zhu, H., Hall, P. A., and May, J. H. (1997). Software unit test coverage and adequacy. *ACM Computing Surveys*, 29(4):366–427.

[Zilke et al., 2016] Zilke, J. R., Loza Mencía, E., and Janssen, F. (2016). Deepred – rule extraction from deep neural networks. In Calders, T., Ceci, M., and Malerba, D., editors, *Discovery Science*, pages 457–473, Cham. Springer International Publishing.

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