Importance-Driven Deep Learning System Testing

Simos Gerasimou
simos.gerasimou@york.ac.uk
University of York
York, UK

Hasan Ferit Eniser*
hfeniser@mpi-sws.org
MPI-SWS
Kaiserslautern, Germany

Alper Sen
alper.sen@boun.edu.tr
Bogazici University
Istanbul, Turkey

Alper Cakan
alper.cakan@boun.edu.tr
Bogazici University
Istanbul, Turkey

ABSTRACT

Deep Learning (DL) systems are key enablers for engineering intelligent applications due to their ability to solve complex tasks such as image recognition and machine translation. Nevertheless, using DL systems in safety- and security-critical applications requires to provide testing evidence for their dependable operation. Recent research in this direction focuses on adapting testing criteria from traditional software engineering as a means of increasing confidence for their correct behaviour. However, they are inadequate in capturing the intrinsic properties exhibited by these systems. We bridge this gap by introducing DeepImportance, a systematic testing methodology accompanied by an *Importance-Driven* (IDC) test adequacy criterion for DL systems. Applying IDC enables to establish a *layer-wise functional understanding* of the importance of DL system components and use this information to assess the *semantic diversity* of a test set. Our empirical evaluation on several DL systems, across multiple DL datasets and with state-of-the-art adversarial generation techniques demonstrates the usefulness and effectiveness of DeepImportance and its ability to support the engineering of more robust DL systems.

CCS CONCEPTS

• **Deep Neural Networks**; • **Test Adequacy**; • **Safety**; • **Assurance**;

KEYWORDS

Deep Learning Systems, Test Adequacy, Safety-Critical Systems

ACM Reference Format:
Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. 2020. Importance-Driven Deep Learning System Testing. In *ICSE '20: Proceedings of the 42th International Conference on Software Engineering, May 23–29, 2020, Seoul, South Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/1122445.1122456

*Work done while at Bogazici University.

1 INTRODUCTION

Driven by the increasing availability of publicly-accessible data and massive parallel processing power, Deep Learning (DL) systems have achieved unprecedented progress, commensurate with the cognitive abilities of humans [28, 46]. In fact, DL systems can solve challenging real-world tasks such as image classification [22], natural language processing [70] and speech recognition [34]. Consequently, DL systems are becoming key enablers in many applications, including medical diagnostics [47], air traffic control [39], malicious code detection [23] and autonomous vehicles [13].

Despite the manifold potential applications, using DL systems in safety- and security-critical applications requires the provision of assurance evidence for their trustworthy and robust behaviour [15]. Vulnerabilities and defects in these systems, either originating from systematic errors, insufficient generalisation or inadequate training, can endanger human lives, lead to environmental damage or cause significant financial loss [72]. Preliminary reports from safety advisory boards (e.g., the US transportation board [4]) regarding recent unfortunate events involving autonomous vehicles [3, 5] underline not only the challenges associated with using DL systems but also the urgent need for improved assurance evaluation practices.

From a safety assurance perspective, testing has been among the primary instruments for evaluating quality properties of software systems providing a trade off between completeness and efficiency [38]. Domain-specific standards such as ISO26262 [25] and DO-178C [62] prescribe testing principles (e.g., adequacy criteria, testing properties) which should be employed for the verification of applications within the automotive and avionics domains, respectively. Evidence collected as a result of testing is typically used to demonstrate compliance with expected quality assurance levels, thus manifesting the ability of those systems to operate with an acceptable risk of failure within their lifetime.

However, testing DL systems by simply adopting principles recommended by these standards is not straightforward [14, 67]. The lack of a system specification regulating the inference mechanism to be learnt combined with the data-driven programming paradigm makes impossible to explicitly encode the expected DL system behaviour into its control flow structures [64]. The extremely large configuration spaces of modern DL models deteriorates the issue as it is impossible to determine and calibrate the influence of each configurable parameter in completing a task; e.g., LeNet [45] and

VGG-16 [66] have more than 60K and 100M configurable parameters, respectively. Thus, traditional software testing techniques and coverage criteria [7] are inapplicable for DL system verification.

Driven by the need for providing high-quality assurance in DL systems and inspired by traditional software engineering testing paradigms [7], recent research proposes novel testing techniques and coverage criteria [43, 50, 55, 58, 69] (see Section 6 for an overview of related work). The core principle underlying those techniques is that for effective DL system testing, the test set should be characterised by high diversity, thus enabling to exercise different behaviours of the system [50]. For example, DeepXplore [58] estimates the diverse DL system behaviour by calculating *neuron coverage* as the ratio of neurons whose activation values are above a predefined threshold. Similarly, the DeepGauge multi-granularity testing criteria [50] generalise the neuron coverage concept and calculate the ability of the test set to cover (i.e., trigger) major and corner-case neuron regions, given by partitioning the ranges of neuron activation values. Despite their usefulness, these criteria are simply an aggregation of neurons (or neuron regions) whose activation values conform to certain conditions. By focusing only on these constrained neuron properties and ignoring the overall DL system behaviour, the causal relationship between the test set and decision-making is uninformative [43]. Also, the instantiation of recently proposed techniques depends on user-defined conditions (number of regions [50] or upper bounds [43]) which might not represent the actual behaviour of the DL system adequately. Finally, these criteria provide limited information about the testing improvement contributed by individual test inputs as expected for an effective testing adequacy criterion [7, 27].

In this paper, we bridge the gap in existing research by introducing *DeepImportance*, a systematic testing methodology accompanied by an *Importance-Driven* test adequacy criterion for DL systems based on relevance propagation. By analysing the activity of a DL system and its internal neuron behaviours, DeepImportance develops a *layer-wise functional understanding* that signifies the contribution of internal neurons to the output through the layers. This contribution enables to determine the causal relationship between the neurons and the DL system behaviour as more influential neurons have a stronger causal relationship and can *explain* which high-level features influence more the decision-making. DeepImportance establishes this relationship by computing a decomposition of the decision made by the DL system and iteratively redistributing the relevance in a layer-wise manner proportional to how prominent each neuron and its connections are [11]. As we demonstrate in Section 3.1, this importance score is quite different from the neuron activation values used by similar DL testing techniques (e.g., [50, 58]). Using those important neurons, DeepImportance carries out *neuron-wise quantisation* to partition the space of each neuron's activity into an automatically-determined finite set of clusters that captures its behaviour to a sufficient level of granularity. The *Importance-Driven* adequacy criterion instrumented by DeepImportance measures the adequacy of an input set as the ratio of combinations of important neurons clusters covered by the set.

Our empirical evaluation using publicly available datasets (MNIST [45], CIFAR-10 [1], Udacity self-driving challenge [2]) and DL systems whose models size ranges from small-medium (LeNet [45])

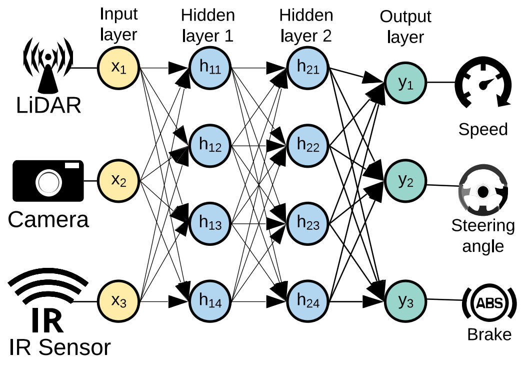

Figure 1: A four layer fully-connected DL system that receives inputs from vehicle sensors (camera, LiDAR, infrared) and outputs a decision for speed, steering angle and brake.

to large (e.g., Dave-2 [13]) demonstrates the ability of DeepImportance to develop a functional understanding of the DL system and evaluate the testing adequacy of a test set. Furthermore, the *Importance-Driven* adequacy criterion is effective in quantifying the ability of a DL system to identify defects as indicated by the coverage difference between the original test set and adversarial examples generated using state-of-the-art adversarial generation techniques [19, 29, 44, 57].

Overall, the main contributions of our paper are:

- The DeepImportance approach for finding important neurons of a DL system that are core contributors in decision-making;
- The *Importance-Driven Coverage* criterion which can establish the adequacy of an input set to trigger different combinations of important neurons' behaviours, thus enabling software engineers to assess the semantic adequacy of a test set;
- Am extensive DeepImportance evaluation on three public datasets (MNIST, CIFAR-10, Udacity) and three DL systems (LeNet, CIFAR-10, Dave-2) showing its feasibility and effectiveness;
- A prototype open-source DeepImportance tool and a repository of case studies, both of which are freely available from our project webpage at https://deepimportance.github.io.

To the best of our knowledge, DeepImportance is the first systematic and automated testing methodology that employs the semantics of neuron influence to the DL system as a means of developing a laywer-wise functional understanding of its internal behaviour and assessing the semantic adequacy of a test set.

The remainder of the paper is structured as follows. Section 2 presents briefly DL systems and coverage criteria in traditional software testing. Section 3 introduces DeepImportance and Section 4 presents its open-source implementation. Section 5 describes the experimental setup and evaluation carried out. Sections 6 and 7 discuss related work and conclude the paper, respectively.

2 BACKGROUND

2.1 Deep Learning Systems

Fig. 1 shows a typical feed-forward DL system consisting of several interconnected neurons arranged into consecutive layers: the input

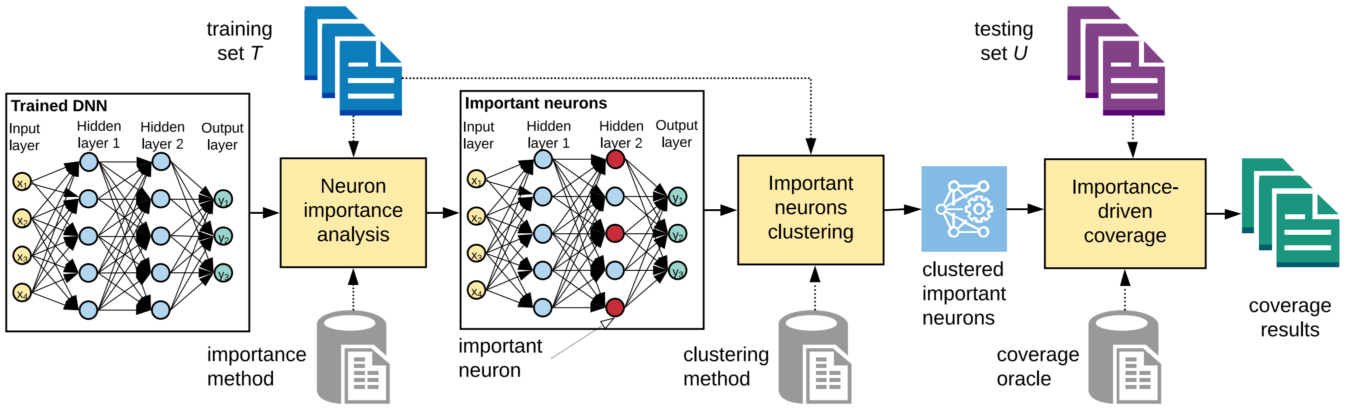

Figure 2: DeepImportance workflow for determining the importance-based testing adequacy of a DL system.

layer, the output layer and at least one hidden layer [28]. Each layer within a DL system comprises a sequence of neurons. A neuron represents a computing unit that applies a nonlinear activation function to its inputs and transmits the result to neurons in the following layer [46]. Commonly used activation functions include sigmoid, hyperbolic tangent and ReLU (Rectified Linear Unit). All neurons, except from those in the input layer, are connected to neurons in the following layer with weights whose values express how strong are the connections among neuron pairs. A DL system's architecture comprises the number of layers, neurons per layer, neuron activation functions and a cost function. Given such an architecture, the DL system carries out an iterative training process through which it consumes labelled input data (e.g, raw image pixels) in its input layer, executes a set of nonlinear transformations in its hidden layers to extract semantic concepts (i.e., features) from the input data, and, finally, generates a decision that matches the effect of these computations in its output layer. The training process aims at finding weight values that minimize the cost function, thus enabling the DL system to achieve high generalisability.

2.2 Coverage Criteria in Software Testing

Since testing a software system exhaustively is, in principle, impossible due to its extremely large number of possible inputs, coverage criteria are typically employed to quantify how well a test suite exercises the system [7, 59]. There are several types of coverage criteria, with the most widely-adopted in industry being: statement coverage, condition coverage, path coverage and branch coverage [38].

Testing techniques and coverage criteria are building blocks of safety standards employed in various safety-critical domains such as automotive and avionics (e.g., ISO26262 [25], DO-178C [62]). Depending on the integrity level associated with a system component, different coverage criteria are mandated. For instance, ISO26262 requires to demonstrate compliance with statement coverage and MC/DC (Modified Condition/Decision coverage) for components whose integrity levels are the lowest (A) and highest (D), respectively. The higher the risk from a component's misbehaviour, the higher its integrity level, and thus, more significant assurance (testing) effort is required to avoid unreasonable residual risk.

3 APPROACH

DeepImportance, whose high-level workflow is shown in Fig. 2, enables the systematic testing and evaluation of DL systems. Using a pre-trained DL system, DeepImportance analyses the training set $\mathcal{T}$ to establish a fundamental understanding of the overall contribution made by internal neurons of the DL system. This enables to identify the most important neurons that are core contributors to the decision-making process (Section 3.1). Then, DeepImportance carries out a quantisation step which produces an *automatically-determined* finite set of clusters of neuron activation values that characterises, to a sufficient level, how the behaviour of the most important neurons changes with respect to inputs from the training set (Section 3.2). Finally, DeepImportance uses the produced clusters of the most important neurons to assess the coverage adequacy of the test set (Section 3.3). Informally, the *Importance-Driven* test adequacy criterion of DeepImportance is satisfied when all combinations of important neurons clusters are exercised.

We use the following notations to present DeepImportance. Let D be a DL system with L layers. Each layer L_i, $1 \leq i \leq L$, comprises $|L_i|$ neurons and the total number of neurons in D is $S = \sum_{i=1}^{L} |L_i|$. Let also $n_{i,j}$ be the j-th neuron in the i-th layer. When the context is clear, we use $n \in D$ to denote any neuron that is a member of D irrespective of its layer. Let X denote the input domain of D and $x \in X$ be a concrete input. Finally, we use the function $\phi(x, n) \in \mathbb{R}$ to signify the output of the activation function of neuron $n \in D$.

3.1 Neuron Importance Analysis

The purpose of *importance analysis* is to identify neurons within a DL system that are key contributors to decision-making. Given an input, information within a DL system is propagated according to the strength of connections (weights) between neurons in successive layers. As such, the activity of some neurons influences more the capabilities of the system to make correct decisions [46].

Although *representation learning* is a key characteristic of DL systems that eliminates the tedious and potentially erroneous process of manual feature extraction, it also means that neurons develop, through backpropagation [63], the ability to learn optimal feature transformations for the given setting on their own [12]. More specifically, raw input data passing through the complex architecture of a

Algorithm 1 Neuron importance analysis

```
 1: function ImportantNeuronsAnalysis($\mathcal{D}$, $\mathcal{X}$, $m$)
 2:     $R \leftarrow \emptyset$                                      ▷ relevant neurons vector
 3:     for all $x \in \mathcal{X}$ do
 4:         $R^x \leftarrow \emptyset$                                ▷ $x \in \mathcal{X}$ importance vector
 5:         $R_L$ = ComputeValue($\mathcal{D}$, $x$)                   ▷ decision value
 6:         for all $i \in \{L_{L-1}, \ldots, L_1\}$ do               ▷ relevance propagation
 7:             $R_L$=Relevance($L_i$, $x$, $R_L$)                     ▷ $L_i$ neurons relevance
 8:             $R^x = R^x \frown R_L$                                ▷ append to vector $R^x$
 9:         $R = R \cup R^x$                                          ▷ collect relevance vectors
10:     $AN$ = Analyse($D$, $R$)                                      ▷ analyse relevance vectors
11:     return Top($AN$, $m$)                                         ▷ select top $m$ neurons
```

modern DL system, with many layers, many neurons per layer and non-linear transformations (e.g., ReLU activation functions [52], max pooling, convolutions), yield abstract and discriminative features that enable the system to make effective decisions in the final layer using a log-linear model (typically softmax) [46]. For instance, neurons within the initial hidden layers learn abstract shapes (e.g., edges, circles) while neurons in deeper layers extract more semantically meaningful features (e.g., faces, objects). Using as an analogy a software system whose architecture adopts conventional software engineering principles, neurons can be considered as functions that execute a distinct functionality. Irrespective of the position of a function into the control flow graph, it receives (transformed) information from functions preceding in the graph and itself applies function-specific transformations before propagating the updated information to subsequent functions in the control flow graph.

We capitalise on this unique characteristic of neurons within a trained DL system to establish the importance of each neuron. To achieve this, we compute a decomposition of the decision $f(x)$ made by the system for input $x \in X$ and use *layer-wise relevance propagation* [11] to traverse the network graph and redistribute the decision value in a layer-wise manner proportional to the contribution made by each neuron within the layer. For a fully-connected layer i, the relevance R_{ij} of the j-th neuron entails redistributing relevance from neurons in layer $i+1$ which is given by [11]:

$$R_{ij} = \sum_k \frac{\phi(x, n_{ij})w_{ijk}}{\sum_i \phi(x, n_{ij})w_{ijk} + \epsilon} R_{i+1,k} \tag{1}$$

where $R_{i+1,k}$ is the relevance score of the k-the neuron in layer $i+1$, w_{ijk} is the weight connecting neuron j to neuron k and ϵ is a small stabilization term (to avoid division by zero).

Intuitively, the relevance attributed to neurons in layer i from neurons in layer $i+1$ is proportional to (i) the neuron activation $\phi(x, n_{ij})$, i.e., neurons with higher activation values receive a larger relevance contribution; and (ii) the strength of the connection w_{ijk}, i.e., more relevance flows through more important connections. While (1) applies to fully-connected layers, we refer interested readers to [54] for definitions of redistribution rules for other layer types including pooling, activation and normalisation layers.

The redistribution process is underpinned by a *relevance conservation* property specifying that at every step of the process (i.e., at every layer L_i) the total amount of relevance (i.e., the prediction) is conserved. No relevance is artificially added or removed.[1]. Therefore, $\sum_j^{|L_1|} R_{1j} = \cdots = \sum_k^{|L_4|} R_{4k} = \sum_l^{|L_5|} R_{5l} = \cdots = f(x)$.

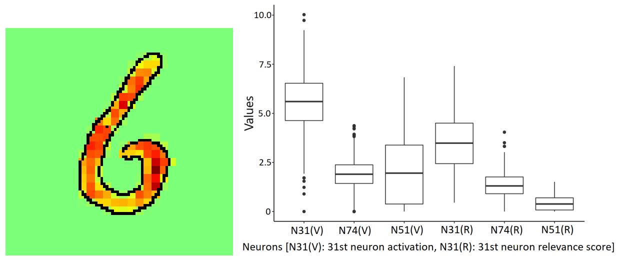

Figure 3: Input from MNIST dataset with the most important pixels contributing to the correct decision highlighted (left), and difference between neuron activation values and relevance scores (right) for the same set of neurons.

Algorithm 1 shows the high-level process for computing the importance scores for neurons of D and selecting the m most important. For any given input $x \in X$, we perform a standard forward pass to compute the decision value, i.e., the magnitude of evidence for a given class before applying softmax (line 5). Next, we perform a backward pass (lines 6–8) considering each layer successively during which the relevance is allocated to neurons of the current layer before being backpropagated from one layer to another until it reaches the input layer. The decomposition is achieved using the layer-specific rules in [11]. The Analyse function (line 10) analyses the relevance scores of all neurons for all inputs and prioritises them based on a priority criterion (e.g., cumulative relevance, normalised relevance). In our evaluation (Section 5), we use cumulative relevance. Finally, the top m neurons are returned (line 11).

The use of relevance for identifying the most important neurons is a key ingredient of our approach. Building on recent research on *explainability* of DL systems, which targets the identification of input parts responsible for a prediction, DeepImportance targets the identification of the most influential neurons; these are *high-risk neurons* that should be tested thoroughly. Albeit being outside the scope of this work, we also highlight that other explainability-driven techniques could be used for the identification of the most important neurons (e.g., DeepLift [65], L2X [20]).

State-of-the-art testing adequacy criteria for DL systems including neuron coverage [58] and k-multisection neuron coverage [50] quantify testing coverage solely based on neuron values, irrespective of the added value of a neuron to the final decision. In other words, a neuron might contribute to increasing the confidence for classes other than the correct one, and this is not distinguished. DeepImportance captures the actual contribution made by each neuron to the decision which in shallow and deeper layers corresponds to raw pixels and concrete features from the input domain, respectively. For instance, Fig. **??** (left) shows the most important pixels and Fig. **??** (right) shows the difference between the activation values and the relevance scores for the same set of most important neurons within the penultimate layer of a LeNet network [45]. DeepImportance exploits this understanding to assess

[1]When neurons with bias contribute to the output, the relevance attribution to the bias is redistributed onto each input of the decomposed layer using the method in [54].

the adequacy of a test set to examine the most critical neurons, i.e., those with the strongest influence on the behaviour of the DL system.

Using relevance is also significantly different compared to sensitivity analysis [54]. While sensitivity analysis cares about *what makes more/less a labelled input (e.g., a dog) to be classified as its target label*, relevance analysis investigates *what actually makes the input to be classified as that label.* The sensitivity scores do not really explain why an input has been predicted in a certain way, but rather to which direction in the input space the output is most sensitive. In contrast, relevance scores indicate which neurons/inputs are pivotal for the classification. Thus, they are a significantly more informative and practicable measure for assessing and explaining the composition about the decision made by the DL systems [11].

3.2 Important Neurons Clustering

Having established the important neurons that are core contributors to the behaviour of the DL system, we are now ready to determine regions within their value domain which are central to the DL system execution. Since each neuron is responsible for perceiving specific features within the input domain [46], we argue that for inputs with similar features the activation values of those important neurons are concentrated into specific regions within their value domain. Informally, those regions form a pattern that captures the activity of the most influential neurons of the DL system.

The purpose of clustering is threefold. First, compared to [50] which partitions the value range of neuron activation values into k buckets of equal width solely based on a randomly selected number of buckets (i.e., k-multisection neuron coverage [50]), the clusters generated by our approach correspond to *semantically different features* of each neuron. Second, since the range of neuron activation values $\phi(x, n)$ could in principle be the entire set of real numbers ($\mathbb{R}_+$ for ReLU activation functions), the cyclomatic complexity for analysing the DL system is very large. Similar to techniques employed in [43, 50], clustering (bucketing in [43, 50]) enables to reduce dimensionality and computational cost, thus making tractable to test the DL system (cf. Section 5). Finally, the identification of clusters for those important neurons could inform the allocation of testing resources to ensure that the regions of those neurons are tested sufficiently, thus increasing our confidence for the robust DL system behaviour.

DeepImportance employs *iterative unsupervised learning* to cluster the vector of activation values from the training set for each important neuron and determine sets of values that can be grouped together. The DeepImportance instantiation we present in this research work (Section 5) employs *k-means* [33], an iterative clustering method that produces k clusters which minimize the within-class sum of squares. To this end, we segment the activation values of each important neuron into groups (clusters) so that activation values within the same group are more similar to other activation values in the same group and dissimilar to those in other groups.

Determining the optimal number of clusters without analysing the data is not a trivial problem [41]. We reinforce cluster extraction with the *Silhouette index* [61], thus supporting the *automatic identification* of a neuron-specific optimal strategy for clustering the activation values of each important neuron in D_m. Silhouette

Algorithm 2 Important Neurons Cluster Extraction

```
1: function ClusterImportantNeurons($\mathcal{D}_m$, $\mathcal{T}$, $C$)
2:     $\Psi \leftarrow \emptyset$                                   ▷ vector for clustered important neurons
3:     for all $n \in \mathcal{D}_m$ do
4:         $\Phi_n = (\phi(t, n))$, $t \in \mathcal{T}$               ▷ $n$-th neuron activation values
5:         $c_n^{\max} = \arg\max_{c \in C}$ Score($\Phi_n$, Labels($\Phi_n$, $c$))
6:         $\Psi_n$ = Cluster($\Phi_n$, $c_n^{\max}$)               ▷ $\Psi_n = \bigcup_{1 \le i \le c_n^{\max}} \Psi_n^i$
7:         $\Psi = \Psi \cup \Psi_n$                                 ▷ collect cluster vectors
8:     return $\Psi$
```

is an internal clustering validation index that computes the goodness of a clustering structure without external information [48]. As such, depending on each neuron's activation values, the optimal number of clusters is determined automatically and can be different between the important neurons. Also, this strategy addresses the weakness of k-means that requires to define the desired number of clusters a priori. More formally, given the n-th important neuron, $n \in D_m$, and the function $C(t)$ indicating for each $t \in T$ the cluster assigned to $\phi(t, n)$ within the n-th neuron's clusters, the Silhouette score for $c \in \mathbb{N}_+$ clusters is defined as follows

$$S_n^c = \frac{1}{|T|} \sum_{t=1}^{|T|} \frac{B(t) - A(t)}{max(B(t), A(t))} \tag{2}$$

where

$$A(t) = \frac{1}{1 - |C(t)|} \sum_{\phi(u,n) \in \{C(t) \setminus \phi(t,n)\}} d(\phi(u, n), \phi(t, n)) \tag{3}$$

is the *intra-cluster cohesion* given by the average L_1 distance of activation value $\phi(t, n)$ to all other values in the same cluster, and

$$B(t) = \min_{C' \neq C(t)} \frac{1}{|C'|} \sum_{\phi(u,n) \in C'} d(\phi(u, n), \phi(t, n)) \tag{4}$$

is the *inter-cluster separation* given by the average L_1 distance between $\phi(t, n)$ and activation values in its nearest neighbour cluster.

Maximising the Silhouette score gives the optimal clustering strategy and correspondingly the optimal number of clusters for the n-th important neuron. Therefore, the higher the score the better the overall quality of the clustering result in terms of cluster cohesion and cluster separation.

Algorithm 2 shows the high-level process underpinning DeepImportance for quantising the vector of neuron activation values and extracting clusters for the most important neurons. Given as inputs the training set $\mathcal{T} \subseteq \mathcal{X}$, the set of possible clusters $C \subset \mathbb{N}_+$ and the set of important neurons $\mathcal{D}_m$ (cf. Section 3.1), DeepImportance produces for each neuron $n \in \mathcal{D}_m$ the vector of activation values Φ_n for all training inputs $t \in \mathcal{T}$ (line 4). Then, through an iterative cluster analysis strategy using the *Silhouette index* [61], we find the optimal clustering strategy for each important neuron's activation values (line 5). Next, we establish the clusters such that $\Psi_n = \bigcup_{1 \le i \le c_n^{\max}} \Psi_n^i$, where Ψ_n^i is the vector containing the activation values for th i-th cluster (line 6). We stop when all important neurons have been analysed.

Our approach is generic and can support different clustering algorithms, including density-based, grid-based and hierarchical clustering [41]. We emphasise, however, the importance of using an iterative strategy that enables to determine the optimum number of clusters. This is an important step that defines the granularity of our importance-driven test adequacy criterion (cf. Section 3.3). Investigating the applicability and effectiveness of other clustering algorithms and clustering validity criteria is left for future work.

3.3 Importance-Driven Coverage

Given an input set Y, we can measure the degree to which it *covers* the clusters of important neurons, termed *Importance-Driven Coverage (IDC)*. Since important neurons are core contributors in decision-making (cf. Section 3.1), it is significant to establish that inputs triggering combinations of activation value clusters of those neurons (cf. Section 3.2) have been covered adequately. Doing this, enables to test the most influential neurons, thus increasing our confidence in the correct operation of the DL system and reducing the risk for wrong decisions. The vector of important neurons cluster combinations (INCC) is given by

$$INCC = \prod_{n \in D_m} \{\textsc{Centroid}(\Psi_n^i) | \forall 1 \leq i \leq |\Psi_n|\} \tag{5}$$

where the function $\textsc{Centroid}(\Psi_n^i)$ measures the "centre of mass" of the i-th cluster for the n-th important neuron.

We define *Importance-Driven Coverage* to be the ratio of INCC covered by all $y \in Y$ over the size of the INCC set. Compared to all other elements in INCC, the j-th INNC element is covered if there exists an input y for which the Euclidean distance between the activation values of all important neurons $n \in D_m$ and the corresponding neuron's clusters centroids in j is minimised. Formally

$$IDC(Y) = \frac{|\{INCC(j) | \exists y \in Y : \forall V_n^i \in INCC(j) \bullet \min d(\phi(y,n), V_n^i\}|}{|INCC|} \tag{6}$$

Following from (6), a test input is always mapped to an element of the semantic feature set given by INCC (5). IDC increases only if the mapped INCC element tests a new semantic feature set not already covered by existing test suite inputs; otherwise, the score remains the same. We provide a proof of IDC soundness on DeepImportance webpage (https://deepimportance.github.io).

Achieving a high IDC score entails a systematically diverse input set that exercises many combinations of important neurons clusters. The covered combinations do not include only those exercised during training, whose activation values have been used for establishing the important neurons, but also new and diverse combinations. These new combinations could represent edge-case behaviours for the DL system. The higher the IDC score, the more INCC combinations have been triggered. Consequently, the more confidence we should have in the DL system's operation.

Another important characteristic of IDC is the *layer-wise* estimation of coverage. By exploiting the combinations of important neurons clusters given by (5), IDC measures how well multiple inputs with semantically different features can trigger those combinations. As such, IDC is significantly different to research which focuses on counting how many neurons have at least once been the most active neurons on a given layer [50, 58].

The granularity with which IDC is specified depends on the number of important neurons m (cf. Algorithm 1). Clearly, setting m to the number of neurons within any layer results in an unmanageable INCC number. For instance, assuming each of the 84 neurons of the penultimate layer of LeNet-5 [45] produces two clusters (cf. Algorithm 6), the number of combinations given by (5) is $INCC = 1.9E{+}25$. Since m is the only IDC hyper-parameter, that affects the combinations of important neurons clusters (5), it enables software engineers to experiment with different testing strategies by specifying how coarse- or fine-grained the analysis should be. In safety-critical systems, for instance, we might opt for a fine-grained IDC coverage, hence a large m, aiming to cover as many combinations as possible. We show in our experimental evaluation that the higher the number of m, the higher the number of combinations and the more testing budget is required to increase the IDC score (cf. Section 5). Investigating training-informed ways for the automatic identification of the number of important neurons is part of our future work.

4 IMPLEMENTATION

To ease the evaluation and adoption of DeepImportance and the Importance-Driven Coverage from Section 3, we have implemented a prototype tool on top of the open-source machine learning framework Keras (v2.2.2) [21] with Tensorflow (v1.10.1) backend [6].

The open-source DeepImportance source code, the full experimental results summarised in the following section, additional information about DeepImportance and the case studies used for its evaluation are available at https://deepimportance.github.io.

5 EVALUATION

5.1 Research Questions

Our experimental evaluation answers the research questions below.

RQ1 (Importance): Can neuron-importance analysis identify the most important neurons? We used this research question to establish if the importance-based algorithm underpinning DeepImportance for the identification of important neurons comfortably outperforms a strategy that selects such neurons randomly.

RQ2 (Diversity): Can DeepImportance inform the selection of a diverse test set? We investigate whether software engineers can employ the *Importance-Driven Coverage* to generate a diverse test set that comprises semantically different test inputs.

RQ3 (Effectiveness): How effective is DeepImportance in identifying misbehaviours in DL systems? With this research question, we examine the effectiveness of DeepImportance to detect adversarial inputs carefully crafted by state-of-the-art adversarial generation techniques [18, 29, 44, 57]. These adversarial inputs should be semantically different than those encountered before, thus increasing the Importance-Driven Coverage metric.

RQ4 (Correlation): How is DeepImportance correlated with existing coverage criteria for DL systems? We analyse the relationship in behaviour between DeepImportance and state-of-the-art coverage criteria for DL systems including neuron coverage [58], k-multisection neuron coverage [50] and surprise adequacy [43].

RQ5 (Layer Sensitivity): How is the behaviour of DeepImportance affected by the selection of specific neuron layers? Given the layer-wise capability of DeepImportance, we investigate whether performing the analysis on shallow or deeper layers has any impact on the Importance-Driven Coverage metric.

5.2 Experimental Setup

Datasets and DL Systems. Table 1 shows the datasets and DL systems used in our experimental evaluation. We evaluate DeepImportance on three popular publicly-available datasets. MNIST [45] is a handwritten digit dataset with 60,000 training inputs and 10,000 testing inputs; each input is a 28x28 pixel image with a class label from 0 to 9. CIFAR-10 [1] is an image dataset with 50,000 training inputs and 10,000 testing inputs; each input is a 32x32 image in ten different classes (e.g., dog, bird, car). The Udacity self-driving car challenge dataset [2] comprises images captured by a camera mounted behind the windshield of a moving car and supported by the steering wheel angle applied by the human driver for each image. Since this is the ground truth, the aim for a DL system is to predict the steering wheel angle; hence, the DL system's accuracy is measured using Mean Squared Error (MSE) between ground truth and predicted steering angles. The Udacity dataset has 101,396 training and 5,614 testing inputs.

To enable a systematic and comprehensive assessment of DeepImportance, we chose DL systems used in related research [43, 50, 58, 74] with different architecture (i.e., number of layers and layer types - fully-connected, convolutional, dropout, max- pooling), complexity (i.e., number of trainable parameters) and performance. For MNIST, we study three DL systems from the Le-Net family [45], i.e., LeNet-1, LeNet-4 and LeNet-5, trained to achieve over 98% accuracy on the provided test set (cf. Table 1). For CIFAR-10, we employ the prototype model in [1] which is a 20 layer convolutional neural network (CNN) trained to achieve 77.68% accuracy. For the Udacity self-driving car challenge, we used the pre-trained Dave-2 [13] self-driving car DL system from Nvidia. Dave-2 comprises nine layers including five convolutional layers and its MSE is 0.096. All experiments were run on an Ubuntu server with 16 GB memory and Intel Xeon E5-2698 2.20GHz.

Table 1: Datasets and DL Systems used in our experiments.

Dataset	DL System	# Params	Performance
MNIST [45]	LeNet-1	7206	98.33%
	LeNet-4	69362	98.59%
	LeNet-5	107786	98.96%
CIFAR-10 [1]	A 20 layer ConvNet with max-pooling and dropout layers.	952234	77.68%
Udacity self-driving car challenge[2]	Dave-2 architecture from Nvidia	2116983	0.096 (*MSE*)

The column 'Params' shows the number of trainable parameters for each DL model. The column 'Performance' shows the accuracy (for for MNIST and CIFAR-10 datasets) and mean squared error (for the Udacity dataset).

Coverage Criteria Configurations. We facilitate a thorough and unbiased evaluation of DeepImportance by comparing it against state-of-the-art coverage criteria for DL systems. To this end, we used DeepXplore's [58] neuron coverage (NC); DeepGauge's [50] k-multisection neuron coverage (KMNC), neuron boundary coverage (NBC), strong neuron activation coverage (SNAC) and top-k neuron coverage (TKNC); and Surprise's Adequacy [43] distance-based (DSC) and likelihood-based surprise coverage (LSC). For each criterion, we use the hyper-parameters recommended in its original research. In particular, we set neuron activation threshold to 0.75 in *NC*, and $k = 3$ and $k = 1000$ in TKNC and KMNC, respectively. For NBC and SNAC we set as lower (upper) bound the minimum (maximum) activation value encountered in the training set. The upper bound for DSC and LSC is fixed to 2 and 2000, respectively, and the number of buckets is set to 1000. Concerning DeepImportance, unless otherwise stated (e.g., RQ5), we always consider the penultimate layer as the subject layer and the number of important neurons $m \in \{6, 8, 10, 12\}$. When running the experiments, we set an upper bound of execution time to three hours. If a criterion exceeds this threshold, we terminate its execution and report that no results have been generated. We facilitate the replication of our findings by making available the implementation of all those metrics on the project webpage.

Synthetic Inputs and Adversarial Examples. We use both synthetic inputs and adversarial examples to evaluate DeepImportance. Synthetic inputs are obtained by applying small perturbations on the original inputs through Gaussian-like injected white noise [9, 10]. Adversarial examples are carefully crafted perturbations to inputs, which albeit being imperceptible to the human, lead a DL system to make an incorrect decision [29]. Adversarial examples are typically used to assess the robustness of DL systems. We employ four widely studied attack strategies to evaluate DeepImportance Fast Gradient Sign Method (FGSM) [29], Basic Iterative Method (BIM) [44], Jacobian-based Saliency Map Attack (JSMA) [57], and Carlini&Wagner (C&W) [19]. Our implementation of these strategies is based on Cleverhans [56], a Python library for benchmarking DL systems against adversarial examples.

5.3 Results and Discussion

RQ1 (Importance). Since identifying the most important neurons within a subject layer is a key principle of DeepImportance, we assess if the neurons identified during neuron-importance analysis (cf. Algorithm 1) have indeed a significant role in decision-making. To answer this research question, we employ DeepImportance to find the $m = 6$ and $m = 8$ most important neurons for the MNIST and Cifar-10, and Udacity datasets, respectively. We select an equivalent number of neurons using a random-selection strategy. Next, we employed the approach from [11], used in the *explainable AI* area to highlight input parts responsible for a decision, and chose inputs (pixels) whose score is above the 90th percentile (i.e., among the top 10%). We then perturbed those pixels, setting their value to zero if their score is above a predefined threshold of 0.5 (i.e., they are relevant) and to one otherwise. We limit the magnitude of perturbation to at most 10% of the total number of pixels, aiming to keep the perturbed input close to the original. Finally, we measured the L2 (Euclidean) distance between the activation values of the

Table 2: Average (std dev.) L2 distance of activation values for neurons selected randomly and using DeepImportance on MNIST (LeNet-1|4|5), Cifar-10 and Udacity (Dave-2).

Strategy	LeNet-1	LeNet-4	LeNet-5	Cifar-10	Dave-2
Random	0.07(±0.05)	1.09(±0.49)	0.93 (±0.51)	47.22(±42.8)	1.16(±0.75)
DeepImportance	0.28(±0.13)	4.79(±1.35)	4.18(±1.61)	112.03(±70.3)	2.83(±1.92)

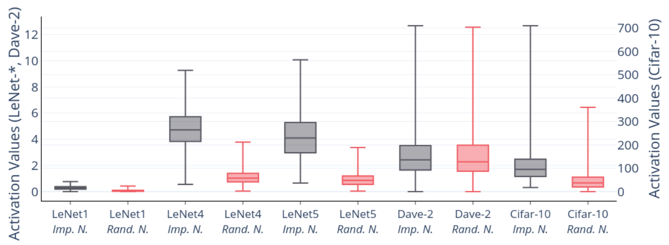

Figure 4: Boxplots comparing activation values distance of important and randomly-selected neurons between original inputs and those with their most relevant pixels perturbed.

original input and the perturbed input both for DeepImportance and random; a higher distance signifies a more significant change.

Figure **??** and Table 2 show boxplots and the average delta (standard deviation in parenthesis) of activation values for the entire testing set (i.e., for all classes) of each dataset, respectively. The reported results are over five independent runs, thus mitigating the risk that they have been obtained by chance. Clearly, the activation values distance for neurons selected by DeepImportance is higher than the equivalent distance for randomly-selected neurons. The difference becomes more evident in LeNet-4 and LeNet-5 that have 120 and 84 neurons in the penultimate layer, respectively, with the distance using DeepImportance exceeding 4.18, whereas the distance using random is between 0.93 and 1.09. Similar observations hold for Cifar-10 (128 feature maps), while the difference is less clear for LeNet-1 (12 feature maps). These observations also provide a useful indication for the number of important neurons m with regards to the total number of neurons in the subject layer.

We conclude that DeepImportance can detect the most important neurons of a DL system and those neurons are more sensitive to changes in relevant pixels of a given input.

RQ2 (Diversity). A useful coverage criterion for DL systems entails the ability to assign higher coverage for test sets that comprise *semantically diverse* test inputs [76]. This is a significant asset for evaluating the ability of a DL system to learn *semantically meaningful features* for the decision-making task rather than *memoising* or learning irrelevant features (i.e., learn to make decisions by exploiting unintended similarity patterns in the test set) [8].

To answer this research question, we measured the IDC metric (6) given the original test set U_O of each dataset and corresponding DL systems (cf. Section 5.2) for multiple values of important neurons $m \in \{6, 8, 10, 12\}$. Then, for each test set we created two 'perturbed' versions. The former is a *semantically diverse* set U_{DI} that consists of inputs whose top 2% pixels (identified similarly as in RQ1), are perturbed by applying small perturbations to the original inputs through Gaussian white noise [9, 10]. The number of perturbed pixels are 15 for MNIST, 20 for Cifar-10 and 200 for Udacity. The other is a *numerically diverse* set U_S that consists of synthetic inputs generated also by injecting Gaussian white noise to an equivalent number of randomly-selected pixels of original inputs. For example, Fig. **??** shows an original input from the Udacity dataset (left), the perturbed input from the U_S set (centre) and the perturbed input within the U_{DI} set (right). The modified pixels in the image on the right are the top 2% pixels that lead the car to steer the wheel to the left (ground truth). We add both of these perturbed test sets to the original set and obtain the test sets U_{O+DI} and U_{O+S} and measured their IDC metric.

Table 3 (top rows with IDC prefix) shows the average IDC value for different DL systems and number of important neurons m. As before, we reduce randomisation bias by reporting results over five independent runs. For all datasets and DL systems, the IDC value for the semantically diverse set (column U_{O+DI}) is always higher than that for the numerically diverse set (U_{O+S} column). In fact, the difference becomes more clear for the more complex DL systems, e.g., LeNet-4 (+1% on average) and Dave-2 (+1.5% on average). This behaviour is also reinforced by a corresponding reduction in accuracy. In particular, in all instances the prediction confidence for the U_{O+S} set is always higher than that of the U_{O+DI} set. These observations signify that IDC is more sensitive to input features that are important to the decision-making task instead of randomly-selected features.

Another interesting observation from Table 3 is that due to the INCC number (5), the IDC value becomes lower as the number of important neurons m increases. Considering LeNet-4, for instance, IDC decreases from 65.8% (64.2%) to 18.1%(16.9%) for $U_{O+DI}(U_{O+S})$ when $m = 6$ and $m = 10$, respectively. For these experiments, the number of clusters of important neurons extracted from Algorithm 2 is between two and four. This is expected since the combinations of important neurons clusters (INCC) increases exponentially as m increases (e.g., $[64, 486]$ for $m = 6$ and $[4096, 69984]$ for $m = 12$). Software engineers can use this information to adjust the available budget and effort needed to test their DL systems.

For completeness, we ran similar experiments using state-of-the-art coverage criteria for DL systems (cf. Section 5.2). Table 3 (bottom) shows their coverage results. Except from LeNet-1, i.e., the DL system with the smallest complexity, the coverage results for all other DL systems are smaller for the *semantically diverse* set U_{DI} compared to the *numerically diverse* set U_S. In contrast to IDC, which is sensitive to perturbations to relevant input features, these criteria are also sensitive to perturbations to random input features.

We conclude that DeepImportance with its IDC coverage criterion can support software engineers to create a diverse test set that comprises semantically different test inputs.

RQ3 (Effectiveness). Building on research in traditional software testing, effective coverage criteria for DL systems should be capable of identifying misbehaviours (i.e., failing test cases) [35]. Coverage criteria satisfying this property have good fault-detection abilities. Thus, they can be used to evaluate the *adequacy* of a test set and provide a quantifiable measurement of confidence in testing [59].

To assess the effectiveness of DeepImportance, we compared the IDC values between an unmodified test set U_O and sets enhanced

Table 3: Average (std dev) coverage results for Importance-Driven Coverage criterion ($m \in \{6,8,10,12\}$) and other coverage criteria for MNIST, Cifar-10 and Udacity datasets; the highest value between U_S and U_{DI} is boldfaced (T/O: timeout, N/A: not applicable).

	LeNet-1 (MNIST)			LeNet-4 (MNIST)			LeNet-5 (MNIST)			Cifar-10			Dave-2 (Udacity)		
	U_O	U_{O+S}	U_{O+DI}	U_O	U_{O+S}	U_{O+DI}	U_O	U_{O+S}	U_{O+DI}	U_O	U_{O+S}	U_{O+DI}	U_O	U_{O+S}	U_{O+DI}
IDC$_6$	34.6%(±2.2)	38.0%(±2.5)	**38.8%(**±2.4)	58.8%(±2.7)	64.2%(±2.7)	**65.8%(**±2.7**)**	47.0%(±3.0)	51.1%(±2.9)	**52.1%(**±2.8**)**	29.4%(±1.3)	37.9%(±1.5)	**39.0%(**±1.4**)**	19.1%(±0.7)	26.0%(±1.1)	**28.5%(**±1.1**)**
IDC$_8$	14.1%(±0.9)	16.5%(±1.1)	**17.5%(**±1.2**)**	26.9%(±1.1)	31.8%(±1.4)	**32.9%(**±1.5**)**	28.0%(±1.2)	33.7%(±1.7)	**34.5%(**±1.7**)**	11.1%(±0.6)	15.5%(±0.8)	**16.6%(**±0.9**)**	7.8%(±0.3)	10.0%(±0.5)	**11.6%(**±0.2**)**
IDC$_{10}$	5.5%(±0.3)	6.6%(±0.4)	**7.0%(**±0.4)	13.5%(±0.8)	16.9%(±1.0)	**18.1%(**±1.0**)**	9.5%(±0.4)	12.2%(±0.5)	**13.1%(**±0.6**)**	5.1%(±0.3)	7.4%(±0.4)	**8.1%(**±0.4**)**	3.3%(±0)	4.2%(±0.1)	**4.9%(**±0.2**)**
IDC$_{12}$	2.1%(±0.1)	2.6%(±0.1)	**2.9%(**±0.2**)**	4.5%(±0.2)	6.3%(±0.3)	**6.9%(**±0.4**)**	4.4%(±0.2)	6.2%(±0.4)	**6.8%(**±0.4**)**	2.0%(±0.2)	3.2%(±0.3)	**3.7%(**±0.3**)**	1.4%(±0)	2.0%(±0)	**2.3%(**±0**)**
NC	17.3%(±0.5)	**20.7%(**±0.3**)**	20.5%(±0.3)	37.9%(±0.7)	**44.1%(**±0.8**)**	43.4%(±0.8)	44.2%(±0.9)	**51.7%(**±0.9**)**	50.6%(±0.9)	20.2%(±0.3)	**35.2%(**±0.2**)**	34.5%(±0.2)	51.6%(±1.8)	**66.7%(**±0.2**)**	64.7%(±0.3)
KMNC	35.1%(±0.3)	**51.3%(**±0.4**)**	48.2%(±0.4)	34.9%(±0.2)	**54.4%(**±0.3**)**	50.8%(±0.3)	32.5%(±0.2)	**52.0%(**±0.3**)**	48.4%(±0.2)	36.8%(±0)	**43.5%(**±0**)**	41.5%(±0)	30.2%(±0)	**50.9%(**±0.1**)**	46.6%(±0.1)
NBC	16.7%(±1.3)	**22.5%(**±1.4**)**	21.5%(±1.3)	9.3%(±0.6)	**12.3%(**±0.6**)**	11.7%(±0.6)	9.3%(±0.5)	**12.1%(**±0.6**)**	11.5%(±0.5)	9.4%(±0)	**9.5%(**±0**)**	**9.5%(**±0**)**	0.8%(±0.1)	**24.7%(**±0.5**)**	21.4%(±0.7)
SNAC	10.9%(±0.6)	**14.6%(**±0.6**)**	13.8%(±0.6)	12.0%(±0.5)	**15.0%(**±0.6**)**	14.3%(±0.6)	14.4%(±0.5)	**18.1%(**±0.6**)**	17.3%(±0.6)	8.8%(±0)	**8.9%(**±0**)**	**8.9%(**±0**)**	1.5%(±0.2)	**46.9%(**±0.9**)**	41.0%(±1.4)
TKNC	100.0%(±0.0)	**100.0%(**±0.0**)**	**100.0%(**±0.0**)**	91.3%(±0.0)	**91.7%(**±0.2**)**	91.6%(±0.2)	88.8%(±0.0)	**89.2%(**±0.0**)**	89.1%(±0.1)	15.2%(±0.0)	**17.0%(**±0.1**)**	16.6%(±0.1)	40.8%(±0.1)	**52.1%(**±0.2**)**	50.0%(±0.2)
DSC	86.3%(±0.0)	91.7%(±0.3)	**92.3%(±0.5)**	60.2%(±0.3)	**66.8%(±0.1)**	66.7%(±0.2)	54.9%(±0.0)	60.9%(±0.2)	**61.4%(±0.2)**	TO	TO	TO	N/A	N/A	N/A
LSC	2.8%(±0.1)	**3.3%(**±0.1**)**	3.2% (±0.1)	14.6%(±0.1)	**16.7%(±0.1)**	16.5%(±0.2)	% 13.8(±0.0)	16.5%(±0.1)	**16.8%(±0.2)**	TO	TO	TO	4.2%(±0.1)	4.6%(±0.1)	**4.7%(±0.1)**

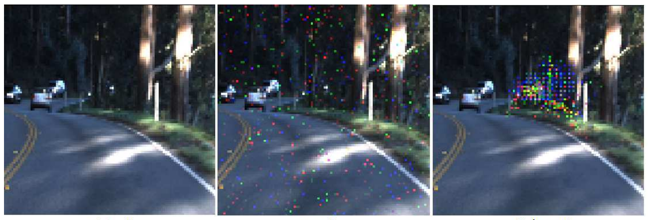

Figure 5: Example image from the Udacity dataset showing the original input (left), an input from the U_S set with Gaussian noise in random pixels (centre), and an input from the U_{DI} set with Gaussian noise to relevant pixels (right).

with perturbed inputs using white noise and adversarial inputs carefully crafted using state-of-the-art adversarial generation techniques. More specifically, for each dataset and each DL system, we generated four adversarial test sets using FGSM [29], BIM [44], JSMA [57] and CW [18] and another *numerically diverse* test U_S via Gaussian white noise with standard deviation=0.3 (as in RQ1). Unlike adversarial inputs, the set U_S is correctly classified with accuracy 97.4% on average. We extended the original set with each of these synthesised test sets and measured their IDC values for the corresponding DL systems.

Table 4 (columns IDC_6 and IDC_8) shows the average IDC coverage results for $m \in \{6, 8\}$ across the six enhanced test sets of each DL system. Compared to the original test set U_O, there is a considerable increase in the IDC result for the enhanced test sets for all DL systems. As expected, the IDC result for $m = 6$ (IDC_6) is higher than that for $m = 8$ (IDC_8) as the number of combinations INCC (5) grows exponentially with the number of important neurons. The increase is more significant in test sets involving adversarial inputs than that with Gaussian-like noisy inputs (U_{O+S}) . Consequently, adversarial inputs lead to higher coverage for our IDC criterion, thus signifying the *sensitivity* to adversarial inputs and its fault detection abilities (conforming to testing criteria in traditional software testing).

We conclude that IDC is sensitive to adversarial inputs and is effective in detecting misbehaviours in test sets with inputs semantically different than those encountered before.

Table 4: Effectiveness of coverage metrics. (‘+Y’ means adding Y-based adversarial inputs to the original test set U_O)

		IDC$_6$	***IDC$_8$***	NC	KMNC	NBC	SNAC	TKNC	LSA	DSA
LeNet-1	U_O.	34.6%	14.1%	23.8%	62.7%	15.1%	18.6%	100%	2.6%	86.2%
	U_{O+S}	36.3%	16.1%	23.8%	70.8%	25.0%	18.6%	100%	4.0%	87.6%
	+FGSM	42.3%	20.9%	23.8%	81.1%	46.5%	55.8%	100%	13.7%	85.3%
	+BIM	43.2%	20.8%	23.8%	71.6%	45.3%	53.4%	100%	9.6%	86.8%
	+JSMA	41.0%	19.0%	23.8%	80.5%	31.3%	37.2%	100%	13.9%	86.9%
	+CW	37.9%	17.0%	23.8%	64.9%	16.6%	19.0%	100%	5.2%	86.2%
LeNet-4	U_O.	58.8%	27.0%	63.7%	69.2%	7.9%	12.3%	91.3%	14.4%	61.5%
	U_{O+S}	62.0%	29.1%	64.4%	72.6%	10.8%	12.3%	91.3%	10.9%	67.0%
	+FGSM	65.6%	33.4%	64.4%	79.4%	38.8 %	65.4%	93.4%	39.3%	83.7%
	+BIM	66.5%	33.4%	79.3%	74.2%	41.0%	69.7%	92.7%	45.1%	78.8%
	+JSMA	64.7%	32.2%	63.7%	76.8%	14.3%	20.8%	91.3%	64.4%	88.8%
	+CW	62.8%	31.0%	63.7%	70.5%	7.9%	12.3%	91.3%	14.4%	60.1
LeNet-5	U_O.	47.0%	28.0%	75.3%	69.2%	7.6%	13.8%	88.8%	13.8%	54.9%
	U_{O+S}	48.1%	29.1%	75.3%	71.5%	10.0%	13.8%	88.8%	10.8%	57.3%
	+FGSM	51.6%	32.3%	75.3%	79.6%	40.7%	71.3%	89.1%	40.4%	83.5%
	+BIM	51.6%	32.3%	84.7%	76.1%	46.2%	83.2%	89.1%	42.4%	74.3%
	+JSMA	49.8%	32.3%	75.3%	74.3%	12.8%	21.1%	89.1%	61.0%	85.7%
	+CW	49.8%	31.8%	75.3%	70.6%	7.6%	13.8%	88.8%	15.2%	61.2%

RQ4 (Correlation). We report results on how state-of-the-art coverage criteria behave across the six tests sets for MNIST in Table 4. Similarly to IDC, most of the criteria, i.e., KMNC, NBC, SNAC, LSA, DSA, experience a similar increase to their coverage results when evaluated using test sets augmented with adversarial inputs (e.g., FGSM, BIM, JSMA, CW). As such, IDC is consistent with criteria

Table 5: IDC coverage results for different layers with the best coverage between the U_O and U_{O+DI} sets typeset in bold.

	LeNet1 (IDC_4)		LeNet4 (IDC_4)		LeNet5 (IDC_6)	
	U_O	U_{O+DI}	U_O	U_{O+DI}	U_O	U_{O+DI}
Conv1	35.3%	**38.3%**	33.9%	**35.9%**	12.5%	**15.6%**
Conv2	76.2%	**80.8%**	81.6%	**84.0%**	31.0%	**36.6%**
FC1	-	-	86.0%	**90.0%**	37.0%	**44.2%**
FC2	-	-	-	-	35.8%	**43.4%**

Conv*: Convolutional layer; FC*: Fully-connected layer
LeNet-4 has only one FC layer; LeNet-1 has none.

based on input surprise (e.g., LSA, DSA) and aggregation of neuron property values (e.g., KMNC, NC). However, while the IDC result for the test set U_{O+S} is always lower than that with adversarial inputs (with the exception of BIM for IDC_8 on LeNet-4), there are several instances in which U_{O+S} produces higher results than the adversarial-augmented sets (e.g., KMNC, NBC DSA for LeNet-1). This is an interesting finding that requires further investigation.

Another interesting observation is that NC and TKNC are insensitive to either Gaussian-like noisy inputs or adversarial inputs, irrespective of the employed adversarial technique. The results for NC are not surprising and conform to results reported in existing research [43, 50]. Nevertheless, the plateau shown in TKNC is particularly important since it is a layer-wise coverage criterion, like IDC. In contrast to IDC, TKNC measures how many neurons have at least once been the most active k neurons on a target (or all) layer. Considering these results, IDC is more informative than TKNC.

In general, we conclude that IDC shows a similar behaviour to state-of-the-art coverage criteria for DL systems; hence, there is a positive correlation between them.

RQ5 (Layer Sensitivity). Since DeepImportance operates layer-wise, we investigated how IDC varies for different layers of a DL system. Table 5 shows the coverage results for $m \in \{4, 6\}$ across layers, ordered by their depth for the three DL systems, using the original test set U_O and that augmented with semantically diverse inputs U_{O+DI}. First, we observe that IDC value increases when the analysis is performed on deeper instead of shallow layers. For instance, in LeNet-4 and the U_O test set, IDC increases from 33.9% in Conv1 to 81.6% in Conv2 until it reaches 86.0% in FC1. We consider this observation as a confirmation of the ability of DL systems to extract more meaningful features in deeper layers.

Furthermore, IDC is more sensitive to the test set with semantically diverse inputs (U_{O+DI}). In fact, we can observe a steady increase in the delta in IDC values between U_{O+ID} and U_O for more deeper layers. For Lenet-5, for instance, the IDC delta grows from 3.1% in Conv1 to 5.6% and 7.2% in Conv2 and FC1, respectively, until it reaches 7.6% for FC2. This behaviour persists despite the slight decrease in IDC value between FC1 and FC2 for both U_O and U_{O+DI}. This observation reasserts our findings in RQ2 (cf. Table 3).

Overall, the chosen target layer affects the result of IDC. Since the penultimate layer is responsible to understand semantically-important high-level features, we argue it is a suitable choice to assess the adequacy of a test set using IDC.

5.4 Threats to Validity

We mitigate **construct validity** threats that could occur due to simplifications in the adopted experimental methodology using widely-studied datasets, i.e., MNIST [45], Cifar-10 [1] and Udacity self-driving car challenge [2]. Also, we employed publicly-available DL systems including LeNet [45] and Dave-2 [13] that have different architectures and achieve competitive performance results [28]. Also, we mitigate threats related to the identification of important neurons (Algorithm 1) by adapting techniques from the *explainable AI* area for identifying input parts responsible for a decision [54].

We limit **internal validity** threats that could introduce bias when establishing the causality between our findings and the experimental study by designing independent research questions to evaluate DeepImportance. Hence, we illustrate the performance of DeepImportance in RQ1 and RQ2 for different values of important neurons $m \in \{6, 8, 10, 12\}$ and by augmenting the original test sets with both *numerically diverse* and *semantically diverse* perturbed inputs. The granularity of IDC increases exponentially with higher m values, thus requiring a substantially larger number of inputs to be satisfied. We also assessed the effectiveness of IDC to detect adversarial examples and confirmed its positive correlation with state-of-the-art coverage criteria for DL systems in RQ3 and RQ4, respectively. Furthermore, we investigate the effect of layer selection on IDC result in a structured manner in RQ5. Finally, when randomness can play a factor (e.g., in RQ1 and RQ2), we reduce threats that the observations might have been obtained accidentally by reporting results over five independent runs per experiment.

We mitigate **external validity** threats that could affect the generalisation of IDC by developing DeepImportance on top of the open-source frameworks Keras and Tensorflow which enable white-box DNN analysis. We further reduce the risk that DeepImportance might be difficult to use in practice by validating it against several DL systems trained on three popular datasets (MNIST [45], Cifar-10 [1], Udacity [2]). However, more experiments are needed to assess the performance of DeepImportance using other techniques to identify the important neurons (e.g., DeepLift [65]), to extract clusters within important neurons (e.g., hierarchical clustering) and to validate the cohesion and separation of those clusters. These experiments are part of our future work.

6 RELATED WORK

Trustworthiness issues in DL systems urged researchers to develop techniques that enable their effective and systematic testing [76]. Existing research in the area adapts testing techniques and criteria from traditional software engineering (e.g., [24, 49, 51, 69]) while other proposes novel test adequacy criteria [35]. For instance, DeepXplore [58] introduces neuron coverage for measuring the ratio of neurons whose activation values are above a predefined threshold. Similarly, DeepGauge [50] introduces a family of adequacy criteria based on a more detailed analysis of neuron activation values. DeepCT [49] proposes a combinatorial testing approach, while DeepCover [68] adapts MC/DC from traditional software testing and defines adequacy criteria that investigate the changes of successive pairs of layers. Recent research also proposes testing criteria and techniques driven by symbolic execution [31], coverage guided fuzzing [55, 75] and metamorphic transformations [71], while other research explores test prioritization [16] and fault localisation [24].

Although the objective of existing research is to guide testing of DL systems, eventually improving their accuracy and robustness, the majority concerns testing adequacy based on neuron-level properties. In contrast, DeepImportance, driven by the fact that the behaviour of a DL system is determined layer-wise [28], proposes a layer-wise and importance-based test adequacy criterion. In our experimental study (cf. Section 5), we compare the performance of IDC against other layer-wise criteria (e.g., TKNC) and show that IDC is more informative. The recent research on using surprise adequacy to guide testing [43] is complementary to DeepImportance.

A closely-related research branch is the provision of guarantees for the trustworthinesss of DL systems via formal verification [36]. AI^2 [26] uses abstract interpretation to verify safety properties, while [60] employs abstraction refinement. Other research uses SMT solvers to identify safe regions in the input space and thus establish the robustness of DL systems [30, 40]. Instead of SMT solvers, ReluVal [73] finds bounds for security properties using interval arithmetic. Finally, DLV [74] verifies local robustness based on user-defined manipulations. DeepImportance identifies important neurons using techniques from the *explainable AI* area (e.g., [11]); thus, it is orthogonal to existing research on DL system verification.

Test adequacy is a widely-studied topic within traditional software engineering [59]. Interested readers can find comprehensive reviews of relevant research in this area in related surveys and books [7, 32, 37, 53].

7 CONCLUSION

Ensuring the trustworthiness of DL systems requires their thorough and systematic testing. DeepImportance is a systematic testing methodology reinforced by an *Importance-Driven* (IDC) test adequacy criterion for DL systems. DeepImportance analyses the internal neuron behaviour to create a *layer-wise functional understanding* and automatically establish a finite set of clusters that represent the behaviour of the most important neurons to an adequate level of granularity. The Importance-Driven adequacy criterion measures the adequacy of a test set as the ratio of combinations of important neurons clusters covered by the set. Our experimental evaluation shows that IDC achieves higher results for test sets with *semantically-diverse* inputs. IDC is also sensitive to adversarial inputs and, thus, effective in detecting misbehaviour in test sets.

Our future work involves (1) investigating methods to automatically determine the number of important neurons; (2) improving the robustness of IDC; (3) evaluating DeepImportance on other DL systems and datasets; and (4) examining how DeepImportance results can be incorporated into safety cases [17, 42].

ACKNOWLEDGEMENTS

This research was supported in part by Semiconductor Research Corporation under task 2020-AH-2970.

REFERENCES

[1] [n. d.]. CIFAR10 Model in Keras. https://keras.io/examples/cifar10_cnn/. Accessed: 13-05-2019.

[2] [n. d.]. The udacity open source self-driving car project. https://github.com/udacity/self-driving-car

[3] 2016 (accessed April 30, 2019). A Google self-driving car caused a crash for the first time. https://www.wired.com/2016/02/googles-self-driving-car-may-caused-first-crash

[4] 2018 (accessed April 30, 2019. National Transportation Safety Board.Preliminary report: Highway HWY18MH010. https://www.ntsb.gov/investigations/AccidentReports/Reports/HWY18MH010-prelim.pdf

[5] 2019 (accessed April 30, 2019. Death of Elaine Herzberg. https://en.wikipedia.org/wiki/Death_of_Elaine_Herzberg

[6] Martin Abadi, Paul Barham, Jianmin Chen, et al. 2016. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*. 265–283.

[7] Paul Ammann and Jeff Offutt. 2016. *Introduction to software testing*. Cambridge University Press.

[8] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. 2016. Concrete problems in AI safety. *arXiv preprint arXiv:1606.06565* (2016).

[9] Guozhong An. 1996. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation* 8, 3 (1996), 643–674.

[10] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 233–242.

[11] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. 2015. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one* 10, 7 (2015), e0130140.

[12] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[13] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, et al. 2016. End to End Learning for Self-Driving Cars.

[14] Markus Borg, Cristofer Englund, Krzysztof Wnuk, Boris Duran, Christoffer Levandowski, Shenjian Gao, Yanwen Tan, Henrik Kaijser, Henrik Lönn, and Jonas Törnqvist. 2018. Safely entering the deep: A review of verification and validation for machine learning and a challenge elicitation in the automotive industry. *arXiv preprint arXiv:1812.05389* (2018).

[15] Simon Burton, Lydia Gauerhof, and Christian Heinzemann. 2017. Making the Case for Safety of Machine Learning in Highly Automated Driving. In *Computer Safety, Reliability, and Security*. 5–16.

[16] Taejoon Byun, Vaibhav Sharma, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. 2019. Input Prioritization for Testing Neural Networks. *arXiv preprint arXiv:1901.03768* (2019).

[17] Radu Calinescu, Danny Weyns, Simos Gerasimou, Muhammad Usman Iftikhar, Ibrahim Habli, and Tim Kelly. 2018. Engineering trustworthy self-adaptive software with dynamic assurance cases. *IEEE Transactions on Software Engineering* 44, 11 (2018), 1039–1069.

[18] Nicholas Carlini and David Wagner. 2017. Adversarial examples are not easily detected: Bypassing ten detection methods. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*. ACM, 3–14.

[19] Nicholas Carlini and David Wagner. 2017. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (S&P)*. 39–57.

[20] Jianbo Chen, Le Song, Martin J Wainwright, and Michael I Jordan. 2018. Learning to explain: An information-theoretic perspective on model interpretation. *arXiv preprint arXiv:1802.07814* (2018).

[21] François Chollet et al. 2015. Keras. https://keras.io.

[22] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. 2012. Multi-column Deep Neural Networks for Image Classification. In *Conference on Computer Vision and Pattern Recognition (CVPR)*. 3642–3649.

[23] Z. Cui, F. Xue, X. Cai, Y. Cao, et al. 2018. Detection of Malicious Code Variants Based on Deep Learning. *IEEE Transactions on Industrial Informatics* 14, 7 (2018), 3187–3196.

[24] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. 2019. DeepFault: Fault Localization for Deep Neural Networks. In *International Conference on Fundamental Approaches to Software Engineering*. Springer, 171–191.

[25] International Organization for Standardization. 2011. ISO 26262: Road Vehicles - Functional Safety.

[26] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, et al. 2018. AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation. In *IEEE Symposium on Security and Privacy (S&P)*. 1–16.

[27] John B Goodenough and Susan L Gerhart. 1975. Toward a theory of test data selection. *IEEE Transactions on software Engineering* 2 (1975), 156–173.

[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[29] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. 2015. Explaining and Harnessing Adversarial Examples. In *International Conference on Learning Representations (ICLR)*.

[30] Divya Gopinath, Guy Katz, Corina S Pasareanu, and Clark Barrett. 2017. DeepSafe: A data-driven approach for checking adversarial robustness in neural networks. *arXiv preprint arXiv:1710.00486* (2017).

[31] D. Gopinath, K. Wang, M. Zhang, C. S. Pasareanu, and S. Khurshid. 2018. Symbolic Execution for Deep Neural Networks. In *arXiv preprint arXiv:1807.10439*.

[32] Mats Grindal, Jeff Offutt, and Sten F Andler. 2005. Combination testing strategies: a survey. *Software Testing, Verification and Reliability* 15, 3 (2005), 167–199.

[33] John A Hartigan and Manchek A Wong. 1979. Algorithm AS 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28, 1 (1979), 100–108.

[34] G. Hinton, L. Deng, D. Yu, G. E. Dahl, et al. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine* 29, 6 (2012), 82–97.

[35] Xiaowei Huang, Daniel Kroening, Marta Kwiatkowska, Wenjie Ruan, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2018. Safety and Trustworthiness of Deep Neural Networks: A Survey. *arXiv preprint arXiv:1812.08342* (2018).

[36] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu. 2017. Safety Verification of Deep Neural Networks. In *International Conference on Computer Aided Verification (CAV)*. 3–29.

[37] Yue Jia and Mark Harman. 2011. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering* 37, 5 (2011), 649–678.

[38] Paul C Jorgensen. 2013. *Software testing: a craftsman's approach*. Auerbach Publications.

[39] Kyle D Julian, Jessica Lopez, Jeffrey S Brush, Michael P Owen, and Mykel J Kochenderfer. 2016. Policy compression for aircraft collision avoidance systems. In *IEEE Digital Avionics Systems Conference (DASC)*. 1–10.

[40] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. 2017. Reluplex: An efficient SMT solver for verifying deep neural networks. In *International Conference on Computer Aided Verification (CAV)*. 97–117.

[41] Leonard Kaufman and Peter J Rousseeuw. 2009. *Finding groups in data: an introduction to cluster analysis*. Vol. 344. John Wiley & Sons.

[42] Timothy Patrick Kelly. 1999. *Arguing safety: a systematic approach to managing safety cases*. Ph.D. Dissertation. University of York York.

[43] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing using Surprise Adequacy. In *Proceedings of the 41th International Conference on Software Engineering (ICSE 2019)*.

[44] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. 2016. Adversarial Examples in the Physical World. *arXiv preprint arXiv:1607.02533* (2016).

[45] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist* (1998).

[46] Yann Lecun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[47] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, et al. 2017. A survey on deep learning in medical image analysis. *Medical Image Analysis* 42 (2017), 60–88.

[48] Yanchi Liu, Zhongmou Li, Hui Xiong, Xuedong Gao, and Junjie Wu. 2010. Understanding of internal clustering validation measures. In *2010 IEEE International Conference on Data Mining*. IEEE, 911–916.

[49] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic Combinatorial Testing for Deep Learning Systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.

[50] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, et al. 2018. DeepGauge: Multi-Granularity Testing Criteria for Deep Learning Systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*.

[51] L. Ma, F. Zhang, J. Sun, M. Xue, et al. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. In *IEEE International Symposium on Software Reliability Engineering (ISSRE)*.

[52] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. 2013. Rectifier Nonlinearities Improve Neural Network Acoustic Models. In *International Conference on Machine Learning (ICML)*, Vol. 30. 3.

[53] Aditya P Mathur. 2013. *Foundations of software testing, 2/e*. Pearson Education India.

[54] Grégoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Müller. 2017. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition* 65 (2017), 211–222.

[55] A. Odena and I. Goodfellow. 2018. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *arXiv preprint arXiv:1807.10875*.

[56] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, and Patrick McDaniel. 2016. cleverhans v1.0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768* (2016).

[57] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, et al. 2016. The Limitations of Deep Learning in Adversarial Settings. In *International Symposium on Security and Privacy (S&P)*. 372–387.

[58] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *Symposium on Operating Systems Principles (SOSP)*. 1–18.

[59] Mauro Pezze and Michal Young. 2008. *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons.

[60] Luca Pulina and Armando Tacchella. 2010. An Abstraction-refinement Approach to Verification of Artificial Neural Networks. In *International Conference on Computer Aided Verification (CAV)*. 243–257.

[61] Peter J Rousseeuw. 1987. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics* 20 (1987), 53–65.

[62] RTCA/EUROCAE. 2011. DO-178C: Software Considerations in Airborne Systems and Equipment Certification.

[63] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 1986. Learning Representations by Back-propagating Errors. *Nature* 323, 6088 (1986), 533.

[64] Rick Salay, Rodrigo Queiroz, and Krzysztof Czarnecki. 2017. An Analysis of ISO26262: Using Machine Learning Safely in Automotive Software. *arXiv preprint arXiv:1709.02435* (2017).

[65] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. 2017. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 3145–3153.

[66] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[67] Y. Sun, X. Huang, and D. Kroening. 2018. Testing Deep Neural Networks. In *arXiv preprint arXiv:1803.04792*.

[68] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2018. Testing Deep Neural Networks. arXiv:cs.LG/1803.04792

[69] Y. Sun, M. Wu, W. Ruan, X. Huang, et al. 2018. Concolic Testing for Deep Neural Networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. 109–119.

[70] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to Sequence Learning with Neural Networks. In *International Conference on Neural Information Processing Systems*. 3104–3112.

[71] Y. Tian, K. Pei, S. Jana, and B. Ray. 2018. DeepTest: Automated Testing of Deep-Neural-Network-Driven Autonomous Cars. In *International Conference on Software Engineering (ICSE)*. 303–314.

[72] Kush R Varshney. 2016. Engineering safety in machine learning. In *2016 Information Theory and Applications Workshop (ITA)*. IEEE, 1–5.

[73] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. 2018. Formal security analysis of neural networks using symbolic intervals. In *27th USENIX Security Symposium 18)*. 1599–1614.

[74] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 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 (TACAS)*. 408–426.

[75] X. Xie, L. Ma, F. Juefei-Xu, H. Chen, et al. 2018. DeepHunter: Hunting Deep Neural Network Defects via Coverage-Guided Fuzzing. In *arXiv preprint arXiv:1809.01266*.

[76] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. 2019. Machine Learning Testing: Survey, Landscapes and Horizons. *arXiv preprint arXiv:1906.10742* (2019).