Project Number 101017258

D6.1 Assurance of Data-Driven and Learning Components of EDDIs

Version 2.0
12 April 2023
Final

Public Distribution

University Of York

Project Partners: Aero41, ATB, AVL, Bonn-Rhein-Sieg University, Cyprus Civil Defence, Domaine Kox, FORTH, Fraunhofer IESE, KIOS, KUKA Assembly & Test, Locomotec, Luxsense, The Open Group, Technology Transfer Systems, University of Hull, University of Luxembourg, University of York

Project Partner Contact Information

Aero41 Frédéric Hemmeler Chemin de Mornex 3 1003 Lausanne Switzerland E-mail: frederic.hemmeler@aero41.ch	**ATB** Sebastian Scholze Wiener Strasse 1 28359 Bremen Germany E-mail: scholze@atb-bremen.de
AVL Martin Weinzerl Hans-List-Platz 1 8020 Graz Austria E-mail: martin.weinzerl@avl.com	**Bonn-Rhein-Sieg University** Nico Hochgeschwender Grantham-Allee 20 53757 Sankt Augustin Germany E-mail: nico.hochgeschwender@h-brs.de
Cyprus Civil Defence Eftychia Stokkou Cyprus Ministry of Interior 1453 Lefkosia Cyprus E-mail: estokkou@cd.moi.gov.cy	**Domaine Kox** Corinne Kox 6 Rue des Prés 5561 Remich Luxembourg E-mail: corinne@domainekox.lu
FORTH Sotiris Ioannidis N Plastira Str 100 70013 Heraklion Greece E-mail: sotiris@ics.forth.gr	**Fraunhofer IESE** Daniel Schneider Fraunhofer-Platz 1 67663 Kaiserslautern Germany E-mail: daniel.schneider@iese.fraunhofer.de
KIOS Panayiotis Kolios 1 Panepistimiou Avenue 2109 Aglatzia, Nicosia Cyprus E-mail: kolios.panayiotis@ucy.ac.cy	**KUKA Assembly & Test** Michael Laackmann Uhthoffstrasse 1 28757 Bremen Germany E-mail: michael.laackmann@kuka.com
Locomotec Sebastian Blumenthal Bergiusstrasse 15 86199 Augsburg Germany E-mail: blumenthal@locomotec.com	**Luxsense** Gilles Rock 85-87 Parc d'Activités 8303 Luxembourg Luxembourg E-mail: gilles.rock@luxsense.lu
The Open Group Scott Hansen Rond Point Schuman 6, 5th Floor 1040 Brussels Belgium E-mail: s.hansen@opengroup.org	**Technology Transfer Systems** Paolo Pedrazzoli Via Francesco d'Ovidio, 3 20131 Milano Italy E-mail: pedrazzoli@ttsnetwork.com
University of Hull Yiannis Papadopoulos Cottingham Road Hull HU6 7TQ United Kingdom E-mail: y.i.papadopoulos@hull.ac.uk	**University of Luxembourg** Miguel Olivares Mendez 2 Avenue de l'Universite 4365 Esch-sur-Alzette Luxembourg E-mail: miguel.olivaresmendez@uni.lu
University of York Simos Gerasimou & Nicholas Matragkas Deramore Lane York YO10 5GH United Kingdom E-mail: simos.gerasimou@york.ac.uk nicholas.matragkas@york.ac.uk	

Document Control

Version	Status	Date
0.1	Document outline	15 May 2022
0.2	First draft	08 June 2022
0.3	First full draft	17 June 2022
0.4	Further editing draft	22 June 2022
1.0	Version 1	30 June 2022
1.1	Revision after review	20 January 2023
1.2	Final revision after review	22 February 2023
2.0	Version 2	31 March 2023

Table of Contents

List of Figures

List of Acronyms

AI	Artificial Intelligence
ML	Machine Learning
DL	Deep Learning
DNN	Deep Neural Networks
CNN	Convolutional Neural Networks
DEEPKNOWLEDGE	systematic testing methodology for DNNs
TrKnw	Transfer Knowledge neurons
OOD	Out-Of-Domain dataset
ID	In-Domain dataset
HD	Hellinger Distance
TCC	TrKnw neurons cluster combinations

EXECUTIVE SUMMARY

This document details the first version of techniques for assuring data-driven and learning components developed in Task 6.1 for the SESAME project.

Task 6.1 Machine Learning Testing Task: Assurance of Data-Driven and Learning Components of EDDIs.

This document provides information on the challenges we tackle and the gap in the literature we bridge, the benefits of the tool to the EDDIs and the SESAME project, the way we infer and analyze Machine Learning (ML) system properties for the development of quantitative objectives for ML testing, the set of test obligations and, finally, the performed empirical study to test the validity of the proposed approach. Results and discussion of the study are also reported in this document. The list of requirements emerging from SESAME partners that are addressed by our tool-supported technique are also detailed in this document.

The tool-supported technique we develop in this context is a white-box systematic testing technique for ML-based software, and in particular, Deep Learning (DL) models. We focus on DL since this technology is increasingly used in robotic systems to support classification and detection tasks. Furthermore, most of the SESAME use case partners employ DL to support similar activities for their robotic systems. The tool-supported technique aims to automatically uncover different erroneous behaviours of DL-based systems. The approach has a good fault-revealing ability, which enables to support the assurance of data-driven and learning components employed by SESAME partners for different prediction tasks.

We introduce DEEPKNOWLEDGE, a novel test adequacy criterion for testing DL-based systems. The key insight of DEEPKNOWLEDGE is on analysing the generalisation behaviour of *Deep Neural Networks* (DNN) models under domain shift. We measure the generalisation ability of a DNN model as the statistical distance between knowledge abstracted from In-Domain input sets (i.e., what was learnt during training) and Out-Of-Domain (i.e., potentially unexpected data not seen during training). This helps identifying a set of neurons that are key contributors to decision-making and crucial to knowledge generalisation within a DNN. We call this neurons, *transfer knowledge neurons* and use them to inform our test adequacy criterion as a means of assessing the quality and diversity of the test set and the generalisation capabilities of a DNN. A good test set should exercise extensively those key critical computational units (neurons) within a DNN. This approach is extensively evaluated through a set of empirical studies using different popular publicly-available datasets, e.g., MNIST, Cifar and SVHN, and prevalent DNN models such as the LeNet family [38]. We perform this study to evaluate the effectiveness of DEEPKNOWLEDGE and its correlation with the state-of-art approaches.

On the implementation front, we pay special attention to developing our tool in a simple multi-paradigm technology, i.e., using a set of Python scripts. This enables quick adaptation and code maintenance by our partners. The developed prototype tool is also compatible with most systems and architectures used by our industrial partners.

STRUCTURE OF THE DOCUMENT. In this document:

- We provide a general overview of the software we developed and its connection with the SESAME use cases in Section 1;
- We give an overview of the used technology, including an overview of existing approaches and their relation to testing in conventional software systems in Section 2
- We present the architecture and methodology of DEEPKNOWLEDGE in Section 3;
- We detail the research questions we address in our experimental study and the experimental setup in Section 4.
- We present and discuss the obtained results in Section 5;
- We present how we satisfy the project requirements in Section 7
- We conclude the report in Section 8.

1 INTRODUCTION

1.1 OVERVIEW

This document summarises the activities concerning the testing of data-driven and learning components aspect of the project, in particular, our focus on Machine Learning (ML) models and algorithms used in any of the MRS components. The work has focused on the development of a practical and useful approach, supported by a prototype tool for testing data-driven and learning components deployed as part of the SESAME project.

This deliverable reports the work carried out within Task 6.1, focusing on the description and integration of the Deep Neural Networks (DNNs) testing approach developed in Work Package 6. The key idea underpinning the approach, called DEEPKNOWLEDGE, is on identifying neurons within a DNN that are capable of abstracting knowledge and assessing how well these neurons have been exercised by the given test test.

In the following sections, we first describe the motivation that guides the development of the proposed approach alongside the assumptions and hypotheses described in more detail later on in this document.

1.2 TESTING OF LEARNING COMPONENTS IN SESAME: GOALS, MOTIVATION AND CHALLENGES

Machine Learning (ML) models and in particular *deep neural networks* (DNNs) have become prevalent in robotic applications providing a diverse range of functionalities, including, but not limited to, classification and detection, that complement traditional data-driven components. Notwithstanding their wide adoption, the DNN deployed in our industrial use cases, are susceptible to errors and bias in the data [70]. These DNN components are at the center of MRS operations as they provide critical functionality. For example, the LOCOMOTEC use case deploys DNNs in the *Person Detection module* as a computer vision tool to ensure the safety of human beings during the disinfection task. Similarly, the DKOX use case employs DNNs for the classification of vine leaves into healthy leaves and those that have a particular disease. Consequently, and despite the wide range of data-driven and learning components that can be deployed in the operating environment of the MRS, the main focus of SESAME Task 6.1 is on the assurance of the data-driven components that employ ML.

Given their black-box nature, the software engineering techniques, which are supposed to guarantee functionality, safety as well as fairness of traditional data-driven components, are not applicable in DNNs context.

As described in the SESAME deliverable D1.1 (pp92-94), in Task 6.1 we devise techniques for testing ML tools and more specifically, developing systematic software testing techniques for DNN-based software. On the other hand, data-driven components that are not ML (as described in WP2) can be tested during simulation using techniques developed such as those developed in deliverable D6.2 (e.g., simulation-based testing). More details about our motivations, the SESAME partners' needs, and the faced challenges will be provided in the following.

Machine Learning (ML) is a subset of Artificial Intelligence (AI) leveraging algorithms that help discover patterns and insights from data so that computers can independently perform tasks without being explicitly programmed to do so. Recently, ML models have become the primary solution to support data-driven decision-making procedures in the industry. Using ML can speed up any development process while improving success rates. ML models also offer great tools for predictive analytics, but resistance to them is growing. Indeed, researchers, as well as civil liberties advocates, express ethical concerns about ML transparency, explainability and overall mistrust issues [10], exposing its harms and highlighting the urgent need for techniques that provide evidence for its safe and reliable use.

In fact, ML, and DNNs in particular (cf. Figure 1), are currently being used in a wide spectrum of safety- and security-critical applications ranging from drug discovery and the analysis of digital pathology data in clinical trials to flight control systems [32] and autonomous cars [9]. This unprecedented expansion witnessed in the adoption of DNNs is partly due to the extra computational power of modern hardware components, substantial algorithmic improvements in the engineering of DNNs and extensive amounts of available data [39, 82].

According to [11], in Industry 4.0 ML-enabled systems are more versatile and are capable of working in changing environments and adapting to them. For instance, vision systems and robotics are combined with ML algorithms to improve processes and increase productivity. This allows the automation of tasks that traditional robots with fixed algorithms and well-defined processes could not carry out before. Despite these noteworthy advances, the main challenge is to maintain the reliability-explainability trade-off without significant losses in predictive performance. In safety- and security-critical applications such as pesticide spraying in viticulture or security inspection of critical infrastructures, more challenges arise related to the correctness, robustness, privacy, efficiency and fairness of the deployed ML models. For instance, the autonomous vehicle driving industry has seen several safety incidents that led to high uncertainty and mistrust in ML technology and AI in general. For example, Tesla's fatal Autopilot crash [4], Uber car's fatal incident [12], and Google's self-driving car crash [1] are all safety incidents caused by failures of the DNNs-based autonomous system. These reliability, safety and security issues in DNN models have led to an urgent need for new approaches to testing the safety, correctness, reliability and robustness of DNN-based systems.

Besides the possible risks of inconsistency in DNN models' performance, adversarial attacks represent an additional threat to their robustness. More specifically, adversarial attacks attempt to mislead DNN models with deceptive data. The deceptive data can take many formats (i.e., images or videos) and are inputs specifically designed to look close to genuine data but which cause misclassification issues to DNN models (i.e., forcing these models to make incorrect predictions).

According to the U.S. National Security Commission on Artificial Intelligence's report [63], adversarial examples are maliciously designed data deployed to cause a malfunction of DNN models and 'turn AI capabilities against us'. More specifically, adversarial attacks have shown harmful effects on self-driving cars and medical analysis systems, e.g., by classifying a benign mole as malignant [18]. Through imperceptible changes to images, adversarial techniques were highly effective in deceiving computer vision systems into the wrong classification, which can endanger human lives and cause significant financial losses. Overall, the increasing interest in the migration of DNN in industrial applications and safety- and security-critical domains requires providing testing evidence for their robustness and dependable operation. DNN Testing has imposed itself as a technique that can automatically provide assurance evidence for the robustness of DNNs, demonstrating their ability to cope with erroneous inputs.

DNN testing refers to any activity designed to reveal bugs in these models [81]. This emerging field remains an area with unsolved questions at several levels. In the following section, we discuss in more detail these levels along with the challenges DNN testing faces in safety- and security-critical applications, challenges related to algorithmic and computational capabilities, industrial applications and domain-specific challenges in relation to the SESAME context and use cases.

1.2.1 Algorithmic and Computational Challenges

Unlike traditional software and rule-based ML models, where software developers and ML engineers understand precisely the system requirements and can define them within the system's logic, DNNs learn their rules automatically from data (cf. Figure 2). Historical data is used to train models and extract patterns for predicting future events. This makes the DNNs' behaviour uncontrollable and unexplainable, giving rise to the so called 'black box' issue [27]. In particular, the black box metaphor is shorthand for data-driven models that use complex and not interpretable mathematical and statistical operations to infer predictions. Unavoidably, this creates a huge challenge for DNN testing, making the traditional software testing techniques inapplicable.

A related challenge involves test oracles which rely on a component to test (e.g., code where the bug may exist) and assume that the software's output can be verified against the expected values designed by the developer [82]. This is not applicable in the case of DNNs as they are data-driven models for which it is difficult

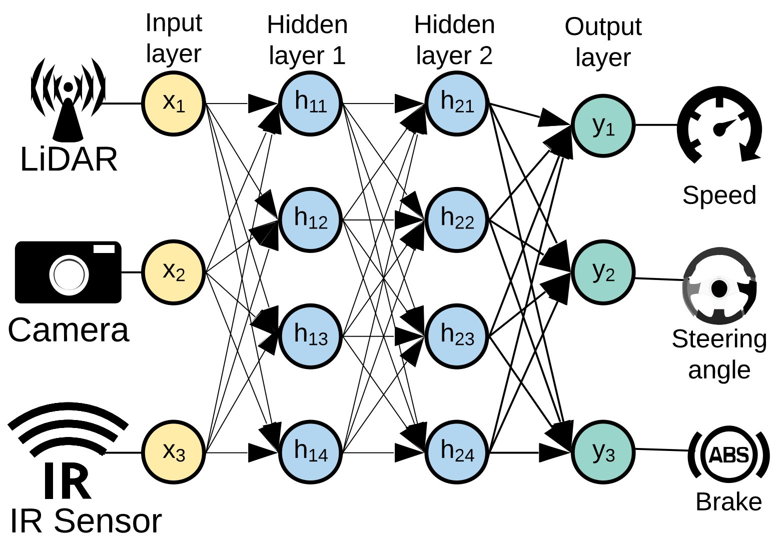


Figure 1: A four layer fully connected DL system that receives inputs from vehicle sensors (camera, LiDAR, infrared) and outputs a decision for speed, steering angle and brake [19]

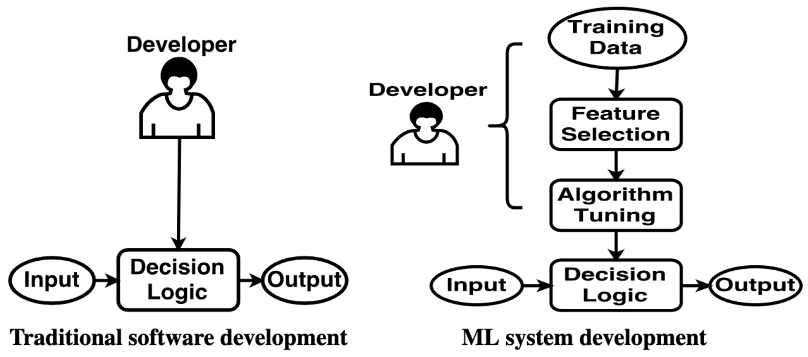


Figure 2: Traditional software Vs ML system development processes [57].

to trace the logic. This challenge prevents the judgement of whether a bug exists and creates the so-called 'oracle problem' [82]. On the other hand, traditional testing techniques are highly reliant on labelled test sets. Deriving these labelled test sets is predominantly a manual and engineer-driven activity, which despite its usefulness, is tedious and time-consuming. Unavoidably, this challenge results in reduced number of labelled datasets. The limited amount of manually labelled data makes these techniques insufficient to expose the DNN system's erroneous behaviour for rare inputs.

Recently, a new stream of coverage criteria has been proposed to solve data limitations and avoid unreasonable residual risk of black box DNN systems [82]. Notwithstanding their usefulness, the majority of the existing approaches focuses on a limited set of criteria that observe how well the testing dataset is able to satisfy certain proprieties, i.e., the activation level of DNN neurons. Accordingly, the resulting findings show that test inputs often fail to uncover different erroneous behaviours of a DNN system, due to both:

- data availability issues, i.e., collecting real-world data is challenging and, more often than not, cost-prohibitive;
- data quality, i.e., limited availability of high-quality data, e.g., 3D object/image, etc [40].

In Section 2, we further present related work and discuss the challenges and limitations of traditional DNN testing methods.

1.2.2 Industrial Challenges

Deploying AI-enabled systems (and DNNs, in particular) in production is difficult, due to the ML lifecycle complexity. The ML lifecycle consists of many complex components such as data acquisition, data wrangling, model training, and tuning, then, model deployment, and monitoring. Furthermore, the lifecycle includes steps to assure the quality and reliability of the prediction. This requires collaboration across teams from Data Engineering, Data Science to ML Engineering [20]. Thus, organisations adopting ML in their industrial activities face both new opportunities and unique challenges [33]. Key challenges are related to:

1. operational rigour to keep all ML lifecycle's components synchronised and working in tandem;
2. stringent infrastructure for testing and experimentation that enables systematic and effective safety assurance for ML models, alongside the continuous improvement of its lifecycle.

In relation to (2), one of the key challenges in ML testing is the lack of skilled resources well equipped to test ML models. Unlike other ML lifecycle components, for which organisations can deploy off-the-shelf tools and libraries (that require minimum knowledge to be used), ML testing suffers from the lack of scalable frameworks and tools. This important challenge contributes to the need for devising ML testing frameworks and/or libraries that are easily deployed and maintained and can easily scale. Accordingly, the work undertaken in Task 6.1 and described in this deliverable aims at devising such a technique. Towards addressing this challenge of the lack of scalable tools, the SESAME ML testing component is a testing technique, supported by a coverage criterion, that is developed to quantify the semantic adequacy of a test set with respect to the training set used to train a DNN system.

1.2.3 Domain-Specific Challenges

The most common ML challenge that businesses face is the availability of data. Thus, most organisations deploy popular publicly-available datasets and prevalent DNN models, including some of our use case partners in SESAME, including:

- **Locomotec's KELO ARODIS** uses pre-trained available convolutional neural networks to detect persons using its Person Detection sub-module. This DNN model is based on the publicly available COCO dataset [65].
- **LuxSense (LXSNS)** deploys their DNN model to monitor the environment and detect blocking objects, such as birds, humans, cars, vegetation [65].
- **KIOS** and **CYPCD** deploy real-world datasets, e.g., a vehicle tracking dataset for their Task Manager module. Thus, their deployed DNN model aims to detect several types of objects with sufficient reliability [65].

The diversity of tasks and industrial activities run by our partners poses further challenges for the DNN testing component in SESAME. Publicly available datasets mostly suffer from underrepresented and distributionally skewed data, which can lead to misleading conclusions about the efficiency and performance of the DNN model. Accordingly, quantifying the semantic adequacy of a test set has to rely on rigorous approaches that determine the DNN model's ability to generalise, i.e., to adapt properly to new, previously unseen data, rather than how well the test set activates parts of the DNN's model beyond a certain threshold [81]. The developed SESAME ML testing approach (described later in Section 3) relies on a learning theory[76] that suggests that the DNN model's generalisation capability should not be related to the "complexity" of the hypothesis space, i.e., the dataset, but on specific DNN's parameters and computational units. This means that identifying these computational units helps assessing the generalisation capabilities of a DNN's model and consequently determining its ability to respond appropriately when unexpected inputs, corner cases or erroneous behaviours appear in the operational setting.

1.3 Benefits to the Project and Desired Industrial Improvements

The proposed DNN testing approach is designed to address the shortcomings described above. We provide a self-contained approach that allows our partners to test data-driven and learning components of EDDIs. The developed tool-supported technique allows us to systematically test real-world DNN-based components. It provides the capabilities for offline testing along with DNN improvement based on a novel test adequacy criterion developed by our team. In response to the abovementioned industrial challenges, our tool-supported technique is developed as a set of Python scripts that can be easily integrated into any system with no required hands-on experience in DNN development or testing. Concerning the domain-specific challenges, our tool satisfied our partners' requirements as described in Section 7.

Overall, our DEEPKNOWLEDGE approach improves the confidence in the DNN model functioning and reduces confusion and uncertainty about its behaviour to unique inputs and corner cases. These capabilities of DEEPKNOWLEDGE allow our industrial partners to gain a better understanding of their system's results. Furthermore, the DNN testing component benefits the research community and DNN testing developers by providing a tool-supported approach supporting quantitative testing of DNNs.

2 Related Work

This section presents an overview of the state of the art in DNNs, software testing and DNN testing. The main ideas, approaches, and research in this field related to our work are also presented.

2.1 Foundations of Neural Networks and Key Notions of Deep Learning

Deep Learning (DL) is a sub-field of ML that encompasses a wide range of algorithms called *Artificial Neural Networks*. Due to recent advances in computational power and available datasets, multi-layer learning models termed *Deep Neural Networks* (DNNs) have taken a dominant position in research, innovation and application of ML techniques.

Conventionally, all software systems that are empowered by at least one DNN component are referred to as DL-based systems. These systems have benefited from the great achievements DL have been making in multiple fields, including achieving human-level performance in many challenging tasks such as machine translation [68], image classification [28] and sentiment analysis [23], or even surpassing it (e.g., in game playing [66, 75]).

Unlike traditional ML algorithms, DNNs do not require an intensive feature engineering process [40]. Inspired by neurobiology, DNNs employ statistical computations and graph technologies simultaneously in order to build up a multi-layer architecture that allows the network to automatically extract features from raw data without manual intervention or support from domain experts. A DNN uses mathematically complex computations to automatically extract high level patterns from within the inputs to produce its prediction effectively [22].

DNNs can be classified into three main categories: multi-layer perceptrons (MLP) [37], convolutional neural networks (CNN) [25], and recurrent neural networks (RNN) [51]. As shown in Figure 1, independently of its type, a DNN is composed of multiple interconnected neurons organised in hidden, input and output layers. Neurons are considered computing units that combine multiple inputs and produce a single output by applying non linear transformations in the form of activation functions such as sigmoid, hyperbolic tangent and rectified linear unit (ReLU). Despite their similarities, the DNN's architecture has different characteristics depending on its type (MLP, CNN, RNN). CNNs are feed-forward networks that have convolutional layers and use filters and pooling, which makes them suitable for spatial data such as images and computer vision tasks. In contrast, the architecture of RNNs, such as gated recurrent units (GRUs) and long short-term memory (LSTM), enables them to feed the results back into the network, making them more suitable for temporal and sequential data such as text. Irrespective of their architecture type, the engineering of DNNs follows two main phases, i.e., *training* (or learning), where the DNNs abstract knowledge (i.e., high level features) from training data, and *inference* (or prediction), where the network applies the extracted knowledge to new and most likely unforeseen inputs.

Advances in DNN architectures, ranging from GANs [22] to transformers [74], have made DNNs the 'gold standard' in the ML research community and industrial practice. To this end, DNNs revolutionise industrial markets and contribute significantly to efficiency improvements in various tasks. According to McKinsey[1], DNNs have the potential to create between £1-2 trillion annually in manufacturing. However, the adaptation of DNNs requires well-established assurance techniques that:

- mitigate business and operational risks of deploying the DNNs technology;
- ensure and improve safety, efficiency and correctness of DNN models.

[1]https://www.mckinsey.com/featured-insights/artificial-intelligence/notes-from-the-ai-frontier-applications-and-value-of-deep-learning

2.1.1 Generalization

An open problem in the field of Deep Learning is the provision of guarantees for the correctness of DNNs in predicting the values of future inputs, given the high risk of possible over-fitting to training data. As a result many research efforts focus on studying and developing methods to enhance their generalisation, e.g., by calibrating the regularisation function during training.

Generalisation refers to the DNN's ability to extract knowledge gained during training and apply it to a different but related problem, i.e., data [72]. Some learning theories suggest that the model's generalisation ability is closely related to the 'complexity' of the hypothesis/data space [53]. More recent studies have empirically demonstrated that the generalisation ability of a neural network is related to the spectrum of the Hessian [76]. Besides these theories, our focus in this task is on analysing the generalisation ability of DNNs under domain shift at a neuron level. Our aim is to increase the confidence in the robustness of DNN inference through the evaluation of its generalisation ability following recent advances [48, 16]. In particular, our focus is on identifying the computational units of a DNN that are responsible for the generalisation and use the conditions affecting this ability as a by-product.

2.1.2 ZeroShot Learning

ZeroShot learning is a promising ML technique that leverages supervised learning with no additional training data. As a methodology, ZeroShot learning allows a DNN model to perform predictive functions on classes that have not been seen during training, i.e., learn how to classify images without any explicit labeling [43]. ZeroShot is another methodology that has been developed to decrease the model's reliance on labelled data. Many approaches have been developed ranging from traditional embeddings [43] and generative-based techniques [35] to more relatively new methods such as CLIP [59].

To achieve its goal, ZeroShot-based methods associate non-observed classes with auxiliary information that allows encoding observable distinguishing properties of the input. In fact, in the ZeroShot setting, the model will be used to predict a number of unseen classes using only abstracted knowledge about a small set of classes and external knowledge about class relations.

Despite the advancements made, these ZeroShot methodologies suffer from important limitations such as the issue of bias and domain shift, which urge for further advances in this field. In our DEEPKNOWLEDGE approach, ZeroShot is used to simulate the real-world settings where the DNN model can face new and previously unseen inputs. We analyse what knowledge subset a DNN model is applying to a new domain without fine-tuning. To do this, we deployed a trained DNN model for prediction in a ZeroShot setting.

2.1.3 Activation Maximization

As 'interpretability' matters in order to build trust in the technologies underpinning DNNs, explaining the reasoning behind a neural network prediction has become a main research topic within the DNN community. In response to this, many techniques have been proposed including *sensitivity-based analysis* [2] that analyses the model's prediction gradient and *layer-wise relevance propagation* [52] which is designed to interpreting the feed-forward graph structure of the deep neural network.

Alongside the above-mentioned methods, one of the foundational techniques proposed for explaining the DNN's prediction is activation maximization. To this end, activation maximization is an optimisation technique that allows finding representations for features that neurons/filters in neural networks have learned [16].

In practice, activation maximization is a technique that generates an input image that maximizes the filter output activations. In particular, using gradient ascent [6], the activation maximization method seeks to maximize the predicted output by iteratively making incremental changes to the input image. For instance, consider the visualisation of a CNN model. Recall that a CNN model is mainly deployed for tasks related to computer

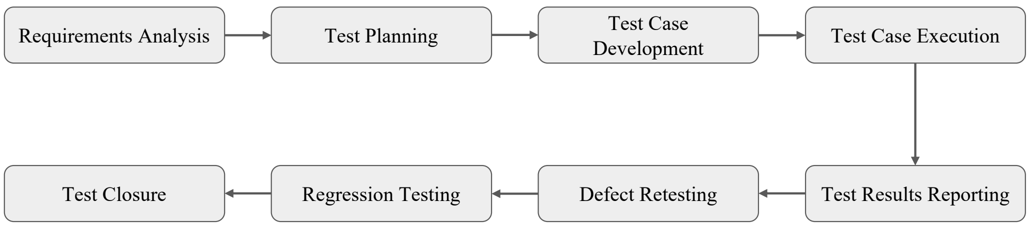


Figure 3: Software testing lifecycle [54].

vision, e.g., image recognition and classification. A CNN encompasses convolutional layers, where each layer has several learned patterns matching filters that maximise their output when a similar template pattern is found in the input image [84, 21]. To visualise and analyse what a filter is learning/abstracting from an input image, activation maximization can be applied to update the input and highlight specific patterns within it.

Activation maximisation is one of the pillars of DEEPKNOWLEDGE, and it is used to understand what sort of input patterns activate a particular neuron in the DNN model and how that changes under domain shift.

2.2 SOFTWARE TESTING

The growing complexity of software and its extensive adoption in many security- and safety-critical systems has created new assurance challenges for establishing the trustworthiness of developed software systems [3].

Software testing is a well-defined methodology, encompassing validation and verification steps, to establish whether the system under test meets a set of specified requirements [58]. More informally, it is an in-depth investigation that involves examining the behaviour of a system to find bugs, errors or missing requirements in the developed software. Software engineers can use insights (e.g., execution traces) from identified bugs or errors to improve the target system. The overall effectiveness and cost of software testing depends on the type and number of test cases executed on the software [58].

According to [30], the major classes of testing are black-box testing, white-box testing and grey-box testing. While white-box testing methods test the internal structure of the software system under test in addition to its functionality, black-box testing focuses only on the functionality without investigating its implementation-level details. Grey-box testing sits between black- and white-box testing and uses limited information from the internal system's execution.

Another dimension for classifying software testing techniques is based on the part of the system tested. Typical examples include functional testing, non-functional testing, automation testing, agile testing, and their sub-types. For example, unit testing [62] is a type of software testing that aims to tackle the smallest testable parts of an application, called units, and assess their correctness. Many test automation tools have been proposed such as NUnit [29], Xunit [50], JUnit [47] for the test execution at a unit level, i.e., at method, function, procedure, or object level.

Another important factor in software testing is the test oracle [5], which helps determining whether the software executed correctly for a test case. Conceptually, this mechanism encompasses an oracle procedure that compares the oracle information with the actual (and expected) system output.

Besides their types, all software testing techniques share the same steps shown in Figure 3. Among these steps, *Test Planning* is the key phase where the testing strategy is defined, and high-level objectives of the test activities are planned following a quality assurance perspective. Planning for traditional software testing is doable given the clarity of the systems and the knowledge of the developer about all the system units. Yet, following a purely traditional software testing methodology to test DNNs is not applicable due to the following reasons:

- Black-box issue: DNNs are data-driven and non-interpretable predictive models for which their decision-making process is unexplainable.

- Volume-related issue: DNN models are typically large structures whose parameters (number of layers and neurons per layer) can reach billion in size.
- Data bias issue: During training, the weights and biases are iteratively tuned to minimize the error, or loss, of the DNN. This process is based on the dataset with the aim to adapt the model decision to the data labelled. As one can notice, any bias in the data would be then automatically mirrored into the DNN decision-making process leading to discriminatory outputs. In fact, data bias is a type of error in which certain labels/classes in the dataset are more heavily weighted than others. Thus, testing DNN would require first and foremost to ensure the quality of the training data set [13].

These issues drive the demand for DNN-specific assurance techniques. Testing techniques became vital building blocks for creating predictive systems that perform efficiently and effectively. Therefore, a substantial research effort has been made towards DNN testing in order to ensure their safety, correctness and robustness. The following section will give an overview of these efforts alongside their shortcomings.

2.3 Recent Advances in DNN Testing

The development of safety-critical DNN systems like autonomous cars urge for testing both functional and non-functional DNN properties, including correctness, robustness and security of DNNs against adversarial attacks [24, 15]. As described above, DNNs are different in many aspects from conventional software systems, therefore adapting traditional software testing techniques to DNNs is not doable for many reasons, including the difficulty of tracing the DNN-based system's logic. While traditional software behaviour results from the code/functionalities that are fixed by the requirement and encoded in software, the DNN model's behaviour is highly dependable on the training data and learnt parameters (weights and biases).

Another consideration is that testing a DNN-based system is more involved and time-intensive than testing traditional software systems. In fact, testing DNN requires validating the training data, parameters, and the code altogether.

According to [81], DNN testing refers to all techniques designed to reveal a DNN bug (Figure 4). A DNN bug can manifest in the data (e.g., in form of bias, mislabeled training data), in the model, or in the framework. Thus, a DNN testing activity can cover any of these elements through test input generation, test oracle identification, test adequacy evaluation, and bug triage.

Test adequacy is a technique that has been widely adapted to DNN testing. More specifically, test adequacy criteria provide a quantitative measurement on the degree of testing of a target software [81], i.e., the percentage of code/model parts that have been exercised by the test cases. The test adequacy criterion is a predicate that can be 'satisfied' or not by a (program, test suite) pair [58]. Eventually, it helps software testers to select properties of a program to focus on during test.

In this section, we provide a brief review of testing approaches for DL-based systems. More specifically, we review testing approaches based on the aggregation of neuron property values as test adequacy criteria, and their ability to detect erroneous behaviours in DNNs.

Although the current state-of-the-art in DL testing is still at its early stage, some notable achievements have been made including white-box techniques [57]. For instance, DeepXplore presents a neuron coverage (NC) measure to identify the parts of DNN logic exercised by a test set. DeepXplore pioneered research by proposing coverage criteria for DNN. Neuron coverage has become a reference in DNN testing because of the significant advances over manual ad-hoc testing of DNN and its effectiveness in detecting the diversity of test inputs. Simply put, NC measures the ratio of neurons whose activation values are above a predefined threshold. Furthermore, the proposed technique offers the possibility of generating inputs that could augment the training set and improve the model's accuracy. The authors demonstrated that a DeepXplore-generated test set covers 34.4% more neurons than the same size of randomly picked test inputs.

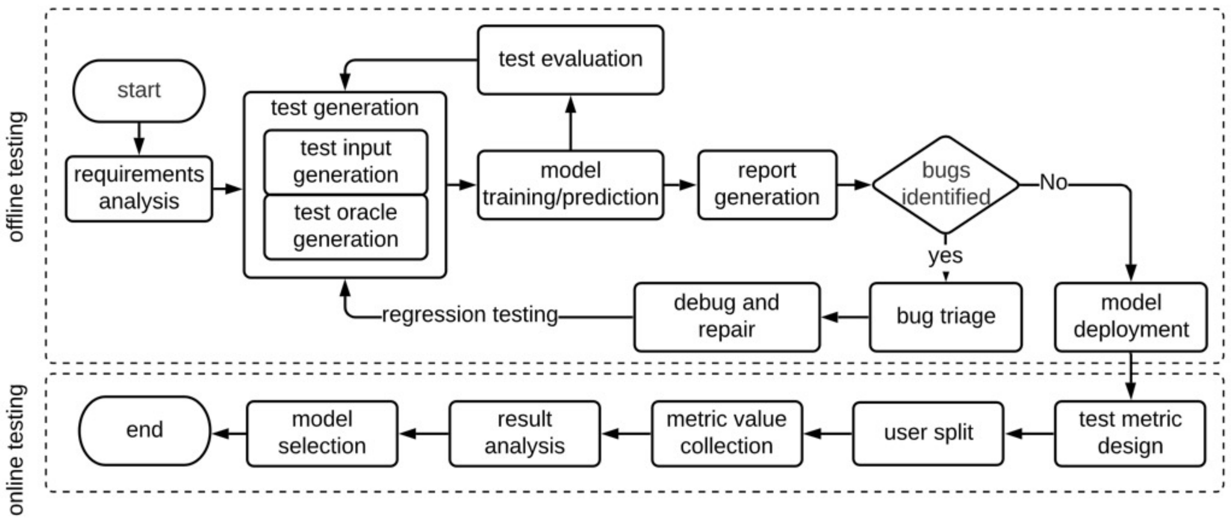


Figure 4: ML testing workflow (adapted from [81])

Subsequently, more fine-grained approaches were introduced, including DeepGauge [45], DeepGini [17], DeepTest [71] and DeepImportance [19]. DeepGauge [45] has introduced multi-granularity testing criteria based on a more detailed analysis of neuron activation values. It assumes that the output of a neuron is located in a defined interval with low and high bounds defined based on the training process, i.e., major and corner-case neuron regions. The defined interval needs to be divided into k equal sections, and the coverage of the test set is estimated based on its ability to cover these sections.

DeepImportance [19] is based on a similar assumption where a set of important neurons is identified then their vector of activation values is used to produce k clusters. DeepImportance measures the diversity of the input set through the degree to which it covers the clusters of important neurons. In particular, given a test set Y, DeepImportance calculates a coverage score IDC that measures the systematic diversity of Y. Higher IDC means more combinations of important neuron clusters have been exercised by the test set.

DeepTest [71] is another systematic testing tool for automatically detecting erroneous behaviours of DL-based self-driving cars. The approach treats erroneous corner-case behaviours in DNN models as analogous to logic bugs in traditional software systems. The approach generates realistic synthetic images by applying image transformations on seed images, e.g., rotation, blurring, changing brightness, changing contrast, translation, scaling, fog effect, horizontal shearing, and rain effect. Furthermore, DeepTest leverages neuron coverage as a mechanism for partitioning the input space, and assumes that an image transformation should increase neuron coverage, through which the erroneous behaviour can be detected. Overall, DeepTest succeeded using the Udacity challenge dataset to detect over 1000 erroneous behaviours in the employed CNN model.

Apart from neuron coverage property, a line of research work has focused on adapting concepts from traditional software testing. DeepCover [67] proposed an adequacy criterion that investigates the changes of successive pairs of layers adapting the Modified Condition/Decision Coverage (MC/DC) code coverage criterion from traditional software testing. DeepCT [44] proposes a combinatorial testing approach, that simply detects surprise via increase in coverage. In the same line of thinking, the surprise adequacy criterion for deep learning (SADL) [36] enables quantifying the amount of surprise the input presents to DNNs with respect to the training data.

Overall, test adequacy criteria is a growing field in DNN testing and verification. For a more in-depth review, we refer interested readers to [83] which provides a comprehensive survey on DNN testing approaches. As a final observation, the above-mentioned approaches could to a certain extent support the detection of erroneous behaviour in DNN models. Existing DNN testing approaches are limited to neurons coverage criteria which is not sufficient to explain the software internal states causing erroneous behaviour. Most of the existing approaches focus only on constrained neuron properties and ignore the overall DNN model behaviour. Thus, the relationship between the test set and the system decision-making process is not well investigated [36].

3 DeepKnowledge

3.1 Intentions and Motivation

Most existing test adequacy criteria for DNN testing focus on the ratio of neurons, i.e., computational units, that have been activated across the entire DNN model. Although this could reflect, to a certain extent, the diversity of inputs provided during testing, these criteria are limited and not sufficient to explain the DNN internal states causing erroneous behaviours. By focusing only on constrained neuron properties and ignoring the overall DNN model behaviour, the relationship between the test set and the final prediction of the system is uninformative [36] and the quantitative testing results are less objective.

In particular, DNNs typically encompass connections and/or neurons that are 'unnecessary' or 'redundant', and have limited or no direct effect on the model's final decision-making process. Accordingly, an input should be considered diverse if it covers neurons that have a high impact on the model's accuracy in real-world scenarios. These neurons are core contributors in the DNN decision-making [19]. We argue that test adequacy criteria should account for the DNN knowledge generalisation capabilities, i.e., identify and reason about neurons that are key components of the DNN functionality and responsible for knowledge transfer to unforeseen input examples.

3.1.1 Knowledge Generalization

Our approach is inspired by recent research in knowledge generalisation [53]. In particular, we leverage the concept of out-of-distribution generalisation which is based on the idea that "*Generalization is a crucial component of learning a language ... so learners must be able to generalize to sentences that they have never encountered before*". Our assumption is based on the fact that these 'learners', i.e., neurons, that are able to generalise knowledge abstracted during self-supervised training, and apply it to a new domain without re-training, e.g., in a zeroshot setting, are the core computational units within a DNN. Their ability to generalise knowledge to new domains indicates that they are core units that highly contribute to the DNN final prediction and accuracy during the deployment of the model. The next section details our definition of knowledge generalisation-driven test adequacy criterion for DNN systems and its use for the systematic testing approach we propose.

3.1.2 Addressing the Reality Gap

Another important limitation of applying DNNs in industrial production that we aim to tackle is the 'reality gap'. Despite its widespread use, the definition of the reality gap concept differs from one domain to another. Concerning DNN models and their application in computer vision and robotics, the reality gap is a natural result to the challenge of obtaining sufficient training data of high enough quality.

Researchers and ML practitioners mitigated this challenge by using simulation to augment the training/testing datasets [8]. This practice has become more popular in recent years, but the gap between simulation and reality is still a major issue. Transferring the knowledge learned by a DNN model in a simulated environment to the real world is a real challenge that leads to erroneous behaviour when the model faces corner case inputs in the real world.

In our approach, we propose to mitigate this issue by exploiting a ZeroShot dataset during the knowledge Transfer Analysis phase. The rationale behind its use is that ZeroShot learning [77] enables us to analyse what part of the knowledge gained during training the DNN can employ in a new domain without re-training or fine-tuning. To this end, this part enables analysing the knowledge generalisation capabilities of the model in a new domain. We simulate the real world through ZeroShot samples from datasets and classes which were not observed during the training phase. The ZeroShot setting exposes the DNN model to unique samples similar to a real world setting where the model can encounter unique and corner cases.

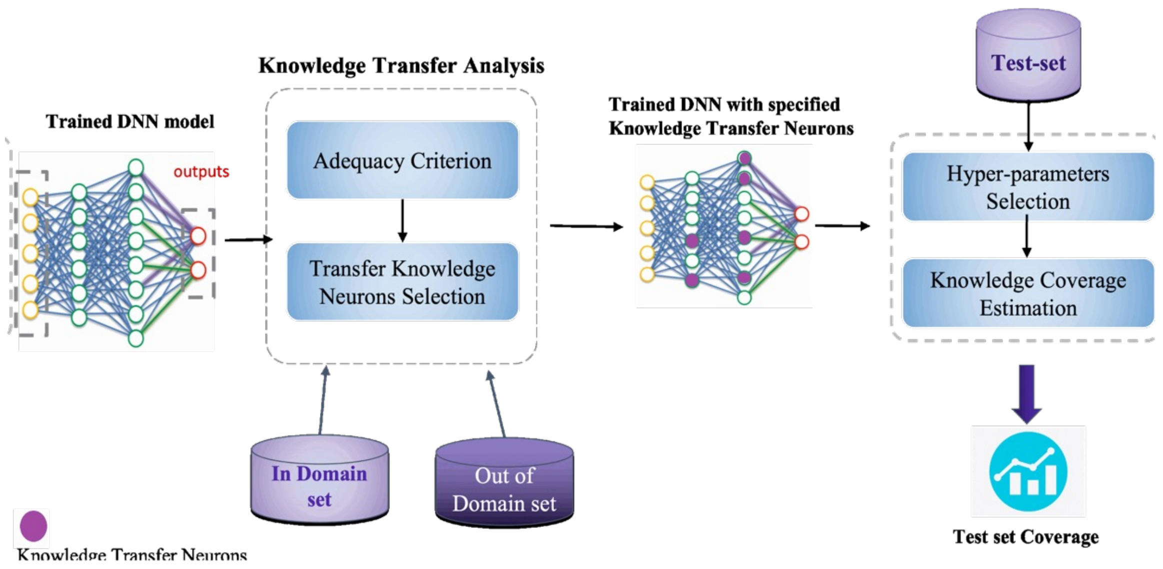

Knowledge Transfer Analysis
Trained DNN model
outputs
Adequacy Criterion
Transfer Knowledge Neurons Selection
Trained DNN with specified Knowledge Transfer Neurons
Test-set
Hyper-parameters Selection
Knowledge Coverage Estimation
In Domain set
Out of Domain set
Knowledge Transfer Neurons
Test set Coverage

Figure 5: High-level worklflow of the DEEPKNOWLEDGE approach.

At neuron level, the ZeroShot setting allows quantifying the neurons' knowledge distributions - so called preference distributions later on - over input features. Neurons with certain statistical characteristics are considered core components of DNN system execution in real-world scenarios, and should be used as a test adequacy criterion.

3.2 The Proposed Approach

DEEPKNOWLEDGE is a systematic testing approach for DNN systems. As shown in Figure 5, using a pre-trained model and two sets of in-domain (ID) and out-of-domain (OOD) data (also referred to as out-of-distribution data [79]), DEEPKNOWLEDGE analyses how the model builds and transfers knowledge abstractions under domain shift. Through this analysis, we establish a fundamental understanding of knowledge generalisation at the level of individual neurons, and quantify the individual contribution of each neuron to this process. Through additional filtering steps, DEEPKNOWLEDGE identifies a set of neurons, termed 'transfer knowledge neurons', that are considered core computational units of the DNN.

The next step in our approach is to apply an established fine-grained method for determining activation value clusters that reflect the changes in neurons' behaviour with respect to new inputs. The defined clusters of the transfer knowledge neurons are then used to assess the coverage adequacy of the test set, similarly to the method defined in DeepImportance [19].

3.2.1 Knowledge Transfer Analysis

Identifying the ***Transfer Knowledge*** (TrKnw) neurons across the DNN trainable layers is a key principle of DEEPKNOWLEDGE. This analysis aims to identify neurons that are able to generalise knowledge abstracted during training and apply it to a new domain without re-training/fine-tuning. These are core contributors to the DNN's behaviour, and, consequently, affect the accuracy of the prediction.

Knowledge Transfer Quantification The identification of *TrKnw neurons*, i.e., neurons capable of transferring knowledge, is loosely inspired by recent research on quantifying knowledge change in DNN under

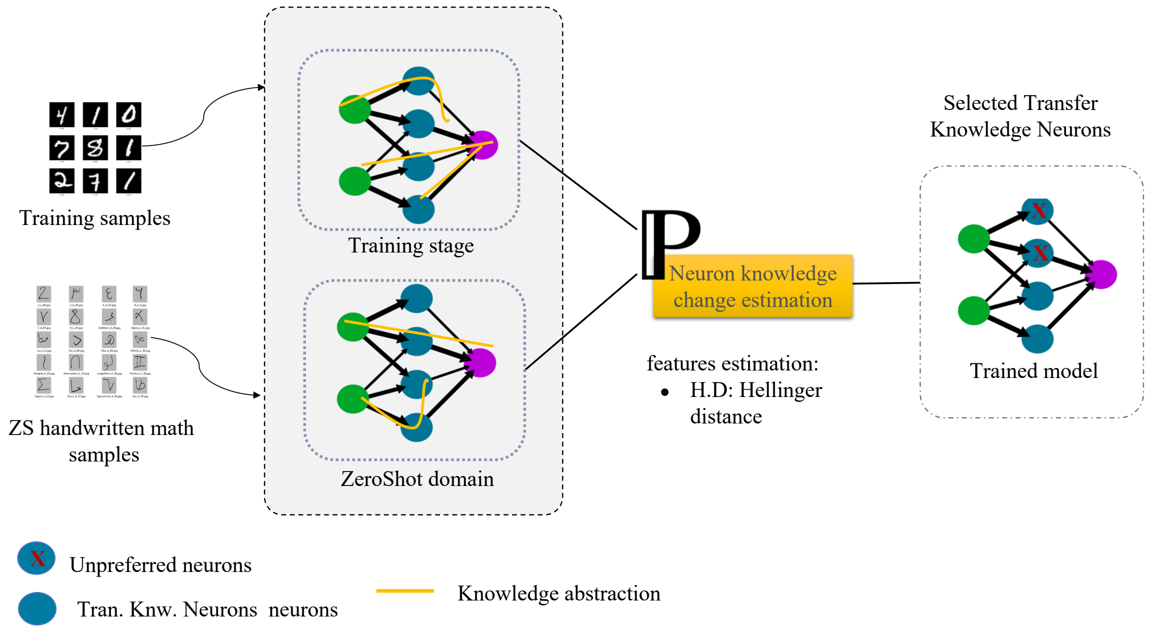

Training samples
ZS handwritten math samples
Training stage
ZeroShot domain
Neuron knowledge change estimation
features estimation:
• H.D: Hellinger distance
Selected Transfer Knowledge Neurons
Trained model
Unpreferred neurons
Tran. Knw. Neurons neurons
Knowledge abstraction

Figure 6: Process for executing knowledge transfer analysis

domain shift at the level of individual neurons [60]. This approach is based on the activation maximization explainability method [16], which allows identifying input instances that produce the highest activation for a neuron, which are called 'preferred' inputs.

As shown in Figure 6, for this analysis we employ two different datasets, i.e., training samples that correspond to *In-Domain data*, and ZeroShot samples that correspond to *Out-Of-Domain data*. These datasets are respectively used to quantify feature preference distributions (yellow lines) over input features (e.g., image pixels for CNN models) for both the training and ZeroShot stages. This is the knowledge quantification step that allows identifying how the DNN model computational units, i.e., neurons, abstract/learn knowledge from the input feature space.

In particular, we employ the *Activation maximization* method (cf. Section 2) to derive a representation for features that neurons/filters in the neural network have learned during the training and ZeroShot stages. Then, we turn these representations into probability distributions to easily measure the change/shift in the abstracted knowledge by employing a statistical metric on the defined space of probability distributions at an individual level, i.e., per neuron. Neurons able to achieve a certain threshold (that is empirically defined) of the statistical measure are selected as transfer knowledge neurons; the remaining neurons are defined as unpreferred neurons.

Now, let's formally explain this process. Given a trained DNN model D with $|L|$ trainable layers, each layer $L_i, 1 \leq i \leq L$ has $|L_i|$ neurons and the total number of neurons in D is $S = \sum_{i=1}^{L} |L_i|$. Let also n_{ij} 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, representing either the in-domain (ID) or out-of-domain (OOD) dataset, and $x_k \in X$ be the k-th concrete input from X (also termed input feature). Finally, we use the function $\phi(x_k, n) \in \mathbb{R}$ to signify the output of the activation function of neuron $n \in D$.

Given the input set X, $A_{x_k} = D(x_k) = \left(\phi\left(x_k, n_{11}\right), \ldots, \phi\left(x_k, n_{L_L|L_L|}\right)\right)$ enables calculating the activation trace of an input $x_k \in X$. Given this information, we signify with $\overline{n^{x_k}} = argmax((D(x_k)))$ the ('preferred') maximally activated neuron and with $\overline{\alpha^{x_k}} = max\left(D(x_k)\right)$ the corresponding maximum activation value.

Applying these steps, we can construct the matrix $M = [(x_k, \overline{n^{x_k}}, \overline{\alpha^{x_k}})]_{\forall x_k \in X}$. We denote the r-th entry of M using $m_r = \left(x_{k,r}, \overline{n^{x_{k,r}}}, \overline{\alpha^{x_{k,r}}}\right)$.

Adopting the definition of *preferred input distribution per neuron* [60], we calculate the probability distribution by aggregating the maximum activation values $\overline{\alpha^{x_k}}$ for each neuron n_{ij} over its maximally activated input

Algorithm 1 Knowledge Transfer Quantification

```
 1: function KnowledgeTransferQuantification($D$, $X$)
 2:     $M \leftarrow \emptyset$
 3:     for $x_k \in X$ do                                        ▷ $X$ = $ID$ or $OOD$
 4:         $A_{x_k} = D(x_k)$                                      ▷ activation trace
 5:         $\overline{n^{x_k}} = argmax(A_{x_k})$
 6:         $\overline{\alpha^{x_k}} = max(A_{x_k})$
 7:         $v \leftarrow \left(x^k, \overline{n^{x_k}}, \overline{\alpha^{x_k}}\right)$            ▷ single input's activation row vector
 8:         $M = M \cup v$                                          ▷ neuron *preferred* inputs matrix
 9:     end for
10:     $P \leftarrow \emptyset$
11:     for  $n \in D$ do                          ▷ using $n$ instead of $n_{ij}$ for simplification
12:         $A(n) \leftarrow \emptyset$
13:         for $x_k \in X$ do
14:             $M_{x_k} = $ ExtractFromM$(x_k, n)$
15:             $\mu_{x_k} = \frac{1}{|M_{x_k}|} \sum_{(x,n,\alpha) \in M_{x_k}} \alpha$
16:             $A(n) = A(n) \cup (x_k, \mu_{x_k})$
17:         end for
18:         $P_n = \{(x_k, p_k)\}_{x_k \in X}$                          ▷ using (1) and (2)
19:         $P = P \cup P_n$
20:     end for
21:     return $P$
22: end function
```

features x_k. Accordingly, $A(n_{ij}) = \{(x_k, \mu_{x_k}) \mid \forall x_k \in X : M_{x_k} = \bigcup_{1 \le r \le |M|}(x_k, n_{ij}, \overline{\alpha^{x_k,r}}) \bullet \mu_{x_k} = \frac{1}{|M_{x_k}|} \sum_{(x,n,\alpha) \in M_{x_k}} \alpha\}$ gives the discrete input activation distribution per neuron where μ_{x_k} is the mean maximum activation of input x_k.

Using $A(n_{ij})$, we can then calculate each neuron's n_{ij} normalised probability distribution per x_k as:

$$P_{n_{ij}} = \{(x_k, p_k)\}_{x_k \in X} \tag{1}$$

with p_k being the activation probability of an input feature x_k calculated as follows:

$$p_k = \frac{\mu_{x_k}}{s_{\overline{\mu}}} \tag{2}$$

where $s_{\overline{\mu}} = \sum_{(x_l, \mu_{x_l}) \in A(n_{ij})} \mu_{x_l}$ is the sum of its feature activation means.

Finally, P describes the per-neuron activation distributions defined as $P = P_{n_{ij}}, \cdots, P_{n_{L_L|L_L|}}$ over the whole input set.

Algorithm 1 shows the high-level process DeepKnowledge employs for quantifying knowledge change in model D. Given X as the input set, each neuron n is expressed as a distribution over preferred inputs. A matrix M of neurons input max activations is produced, that we aggregate later to express each neuron as a probability distribution over maximally activated inputs as defined in equations (1) and (2).

3.2.2 Transfer Knowledge Neurons Selection

To identify the transfer knowledge (*TrKnw*) neurons we quantify the shift/change in the abstracted knowledge for each neuron n_{ij} from the ID dataset to the OOD dataset. We perform this activity using the symmetric divergence measure *Hellinger Distance* (HD) between the discrete activation distribution probabilities over both datasets $p = P^{ID}_{n_{ij}}$ and $q = P^{OOD}_{n_{ij}}$

$$HD(p,q) = \frac{1}{\sqrt{2}}\sqrt{\sum_{j=1}^{X}(\sqrt{p_j} - \sqrt{q_j})^2} \tag{3}$$

HD is used to measure the per-neuron knowledge change caused by different model training stages (i.e., pre-training using ID and ZeroShot learning using OOD datasets). We also quantify for each neuron the total number of unique preferred inputs $l = |X|$ is the number of distinct x_k occurring in X. l enables us to quantify the knowledge diversity for each neuron, with n being selected if l is greater than a given threshold (typically 0). This condition enables to eliminate neurons whose Hellinger distance is low but do not have many preferred inputs.

The use of Hellinger distance allows us to analyse knowledge transfer as a shift between the ID and OOD distributions. Given this measure of distribution shift, we define two types of neurons based on the knowledge shifted per neuron:

- *Preferred neurons* which have a *low* Hellinger distance entailing that the neuron is able to apply its knowledge abstracted during pre-training stage to the new domain (OOD data) without being fine-tuned to do so
- *'Gained'* neurons which have a *high* Hellinger distance between $p = P^{ID}_{n}$ and $q = P^{OOD}_{n}$. This tell us that the neuron has abstracted new knowledge from the new domain inputs without being fine-tuned.

An empty distribution over one of the domains, i.e., $P^{ID}_{n} = 0$ or $P^{OOD}_{n} = 0$, results in an invalid HD value.

Algorithm 2 shows the high-level process to express a neuron as a probability distribution over maximally activated input P_n. Quantifying differences between discrete input preference probability distributions P^{ID}_{n} and P^{OOD}_{n} of neuron n is achieved using the Hellinger distance divergence HD.

We begin by performing the Knowledge Transfer Quantification step using Algorithm 1 (lines 3 and 4). This quantification enables expressing each neuron as a probability distribution over maximally activated features. Given this information for the ID and OOD datasets, we perform the Hellinger distance calculation according to equation (3) and if a given threshold (experimentally identified) is exceeded the neuron is added to the list of candidate neurons capable of transfer knowledge. In the final step, we sort these candidate neurons based on their distance and return the top z (given as a parameter to the algorithm).

Although we use HD, our approach is generic and can support different divergence metrics, including Wasserstein distance [73], Kolmogorov–Smirnov [7] and Jensen–Shannon divergence [49]. We emphasise the importance of using symmetric measures of distance as they have the advantage of providing a unique interpretation of the knowledge shift, i.e., $distance(P^{ID}_{n}||P^{OOD}_{n}) = distance(P^{OOD}_{n}||P^{ID}_{n})$, while also requiring less computation time than asymmetric divergence measures such as Kullback-Leiber [31].

3.2.3 Knowledge Coverage Estimation

To increase our confidence for the robust DNN model D behaviour, we assess how well a test set exercises the set of Transfer Knowledge neurons identified previously. The main premise we adopt here is that inputs that are semantically similar, i.e., having similar features, generate activation values with a similar statistical distribution at the neuron-level [19, 46]. Essentially, these combinations are triggered by the training input set on the most influential neurons of the DNN system, i.e., TrKnw neurons (cf. Section 3.2.1). Accordingly, each combination of cluster reflects different knowledge abstracted during the training phase, and we can assess the knowledge diversity of the test set by assessing if it adequately covers these combinations.

Thus, we adopt the combinatorial neuron coverage method employed in [19] to first iteratively cluster the vector of activation values of each TrKnw neuron from the training set. Then, given a set of inputs, we measure

Algorithm 2 Transfer Knowledge Neurons Selection

```
 1: function TrKnwNeuronsSelection($D$, $ID$, $OOD$, $HD^{thr}$, $z$)
 2:     $TK \leftarrow \emptyset$
 3:     $P^{ID}$ = KnowledgeTransferQuantification($D$, $ID$)
 4:     $P^{OOD}$ = KnowledgeTransferQuantification($D$, $OOD$)
 5:     for $n \in D$ do
 6:         $HD_n$ = CalculateHD($P_n^{ID}, P_n^{OOD}$)                ▷ cf equation 3
 7:         if $HD_n < HD^{thr}$ then
 8:             $TK = TK \cup \{n\}$
 9:         end if
10:     end for
11:     $TK^z$ = TOP($TK, z$)
12:     return $TK^z$
13: end function
```

the number of combinations this set is able to trigger over the combinations of activation value clusters of TrKnw neurons. Given that the cluster combinations of activation values generated correspond to semantically different features of the input set, this combinatorial analysis could be used to assess the knowledge diversity of the test set.

Formally, we first determine regions within the TrKnw value domain that are central to the DNN system execution, i.e., clusters of activation values using the k-means clustering algorithm [41]. Clustering enables the reduction of the dimensionality of neuron activation values. Subsequently, the Silhouette index [61] is used to reinforce the clustering output.

Silhouette is an internal clustering validation index that computes the goodness of a clustering structure without external information, so it helps to determine the optimal number of clusters.

The Silhouette score for $c \in \mathbb{N}^+$ clusters is defined as follows:

$$S_c^n = \frac{1}{|X|} \sum_{x=1}^{X} \frac{B(x) - A(x)}{max\left(B(x), A(x)\right)} \tag{4}$$

where $A(x)$ is the intracluster cohesion given by the average distance of activation value $\phi(x, n)$ to all other values in the same cluster, and $B(x)$ is the inter-cluster separation calculated as the distance between $\phi(x, n)$ and activation values in its nearest neighbour cluster.

Next, we measure the degree to which a test set X covers the clusters of TrKnw neurons, by evaluating the combinations of activation value clusters triggered by inputs $x \in X$. Thus, having the set of clusters extracted and optimized, we identify the vector of TrKnw neurons cluster combinations (TCC) as follows:

$$TCC = \prod_{n \,\in\, TK^z} \{Centroid\left(\psi_i^n \middle| \, \forall 1 \leqslant i \leqslant |\psi_n|\right)\} \tag{5}$$

where the $Centroid(\psi_i^n)$ measures the centre of mass of the i-th cluster for the n-th TrKnw neuron. For more details on how the clusters are extracted and ψi^n is determined please refer to the *DeepImportance* paper [19].

Finally, given the input set X, the DeepKnowledge coverage is computed as the ratio of TCC covered by all $x \in X$ over the size of the TCC set as follows:

$$Score_{Knw} = \frac{\left|\left\{TCC(i) | \exists x \in X : \forall V_n^j \in TCC(i) \bullet min\ d\left(\phi(x,n), V_n^j\right)\right\}\right|}{|TCC|} \tag{6}$$

$TCC(i)$ is covered if there is an input x for which the Euclidean distance $d\left(\phi\left(x,n\right),V_n^j\right)$ between the activation values of all TrKnw neurons $n \in TK^z$ and the i-th neuron's clusters centroids is minimised.

Overall, the calculated coverage score $Score_{Knw}$ quantifies if the combinations of activation value clusters of TrKnw neurons have been covered adequately by the test set. A higher $Score_{Knw}$ entails a knowledge-diverse input set that exercises different combinations of Trknw neurons clusters. Thus, it helps assessing how knowledge diverse the test set inputs are.

4 Experimental Study

In this section, we describe the experimental study designed to showcase DeepKnowledge's usefulness for analysing and quantifying knowledge transfer in DNN, and its effectiveness in using this analysis as a test adequacy criterion. We experimentally validate DeepKnowledge using the following research questions.

4.1 Research Questions

1. **RQ1 – Knowledge Generalisation: Is DeepKnowledge capable of capturing the most influential neurons of a DNN system?** This research question is used to evaluate the selected transfer knowledge neurons' ability to generalise the abstracted knowledge to a new domain. Consequently, we aim to verify whether the selected neurons are core contributors to the decision-making process of DNN under domain shift, and that DeepKnowledge can outperform a strategy that selects such neurons randomly.

2. **RQ2 – Hyper-parameters Sensitivity: Does the selection of hyper-parameters used in DeepKnowledge have any impact on how accurately DeepKnowledge reflects the behaviour of the DNN systems?** As shown in Figure 5, the computation of DeepKnowledge coverage relies on a filtering step that uses a set of hyper-parameters including: (i) the percentage of knowledge transfer neurons used denoted by $Top - N$; (ii) the feature length threshold, i.e., the number of input features that maximally activated the Trknw neurons; and (iii) the Hellinger distance range.

3. **RQ3 – Effectiveness: Can knowledge transfer neurons be effectively used as an adequacy criterion?** Based on state-of-the-art studies [83], effective coverage criteria for DL systems should be capable of identifying misbehaviours (i.e., failing test cases). Hence, this question aims to study how effective is DeepKnowledge in identifying misbehaviours in DNN systems. To this end, we evaluate whether it is possible to detect adversarial examples based on DeepKnowledge's coverage values. We expect adversarial examples to cause different behaviours in DNN systems.

4. **RQ4 – Correlation: How is DeepKnowledge coverage correlated to existing coverage criteria for DL systems?** The DeepKnowledge coverage aims mainly to detect input diversity and its ability to cover the main DNN features. To evaluate this, we need to evaluate the consistency of DeepKnowledge with existing coverage criteria against different input sets. To this end, we evaluate the correlation between DeepKnowledge and state of the art approaches, including DeepImportance (IDC) [19], Neuron Coverage (NC) [57], k-Multisection Neuron Coverage (KMNC) [45], Neuron Boundary Coverage (NBC) [45], and Strong Neuron Activation Coverage (SNAC) [45].

4.2 Implementation

All experiments in our study were conducted on a high-performance computer running a cluster GPU with NVIDIA 510.39. We implement the proposed DeepKnowledge approach and the knowledge transfer neuron analysis method based on the state-of-the-art framework, open-source machine learning framework Keras (v2.2.2) [26] with Tensorflow (v2.6) backend. To allow for reproducibility, our full implementation and evaluation subjects are available on Github[2]. In the following section, we report the full experimental results.

[2]https://github.com/sesame-project/MLTesting

Table 1: Datasets and DNN models used in our experiments

Dataset	Description	DNN model	#Layers	Out-Of-Domain Data set
SVHN	Street View Hous Numbers	LeNet 5	7	EMNIST: hand written digits and letters
MNIST	Digits 0∼9	LeNet 1	5	Fashion MNIST: Zalando's article
CIFAR-10	Colored Images with 10 classes	ResNet 18	18	CIFAR-100 images with 100 classes

4.3 Experimental Setup

We extensively evaluate DeepKnowledge on four different DNN-based systems using: (i) the original test sets, and (ii) adversarial examples generated by four attack strategies. This section describes the studied DL systems and the input generation methods.

4.3.1 Datasets and DNN-based Systems

The proposed method for extracting knowledge transfer neurons is based on analysing the DNN knowledge generalisation process under domain shift. The method, as explained in Section3, estimates the change in knowledge abstraction from pre-training to ZeroShot stages [55] using, respectively, in-domain-data ID and out-of-domain OOD data for each DNN.

For convenience, we did not re-train the used DNN model to calculate the 'preferred input distribution' and estimate the knowledge abstraction per neuron. Instead, we quantify the abstracted knowledge during the validation phase over a relatively smaller dataset compared to the training set. This step helps reduce the computational cost of DeepKnowledge and allows its application easily in low-resource scenarios. As listed in Table 1, this methodology leads to the use of six widely adopted datasets.

Overall we extensively evaluate DeepKnowledge on: MNIST (28x28 grayscale images of handwritten digits), SVHN (32x32 coloured images of real-world Street View House Numbers), and CIFAR-10 (32x32 coloured images classified in 10 classes) as in-domain data, and three respective out-of-domain datasets: Fashion MNIST (a dataset of Zalando's article in format of 28x28 grayscale images), EMNIST (Extension of MNIST that contains both handwritten digits and letters in the format of 28x28 pixel images), CIFAR-100 (dataset of 32x32 colour images with 100 different classes). These datasets have been selected as OOD based on their labelled classes that have to be unseen in the corresponding in-domain datasets.

Then, for performing a comprehensive assessment of the DeepKnowledge approach, we adopted DNN-based systems used in related research [83]. The majority of employed DNN models are convolutional DNNs, and these models have different architectures and are trained with different datasets, which make them suitable for the extensive analysis on the effectiveness of TrKnw neurons as an adequacy criterion. Overall, we study three DNNs from the Le-Net family, i.e., LeNet-1, LeNet-4 and LeNet-5 [38], and the ResNet18 model [69] trained on the CIFAR-10 dataset[3].

4.3.2 Adversarial Inputs

We adopt four different techniques to evaluate DeepKnowledge through adversarial examples, i.e., Fast Gradient Sign Method (FGSM) [42], Basic Iterative Method (BIM) [24], Momentum Iterative Method (MIM) [15], and Projected Gradient Descent (PGD) [14]. Adversarial inputs are crafted carefully using the Cleverhans

[3]https://www.cs.toronto.edu/ kriz/cifar.html

Python library [56] to generate perturbed examples based on given original and benign inputs. For each of the original test sets (SVHN, MNIST and CIFAR-10), we generate an adversarial set with similar size.

4.3.3 DeepKnowledge Configurations

Naturally DL models are extremely sensitive to hyper-parameters. Thus, extensive hyper-parameters analysis was carried out for selecting *TrKnw neurons* which can provide us with more confidence in deploying these TrKnw neurons as coverage criteria.

More specifically, the granularity with which DeepKnowledge is specified depends on HD values and the number of selected $Trknw$ neurons $n \in \{5, 6\}$. Concerning the layers, we always consider the model's trainable layers as subject layers from which we extract the transfer knowledge (TrKnw) neurons.

When running the experiments to compare DeepKnowledge against state-of-the-art coverage criteria for DNN systems, we used for each approach the hyper-parameters recommended in its original research paper. For instance, for NC [57], we set the neuron activation threshold to 0.75. For TKNC approach the hyperparameter k was set as $k = 3$ and for KMNC approach $k = 1000$. For NBC and SNAC, we set as the lower bound the minimum activation value encountered in the training set, and the maximum value for the upper bound. More details on these parameters will be covered in the following sections. Also to facilitate the replication of our findings we make available the implementation of all those metrics on the project's Github repository.

5 Results and Discussion

RQ1: Knowledge Generalisation

We employ this question to assess if the neurons identified during Transfer Knowledge analysis have a significant role in decision-making, and thus whether they could be used as a test adequacy criterion.

Since identifying the knowledge transfer neurons within the DNN model is a key principle of DeepKnowledge, we assess if the neurons identified during neuron-knowledge transfer analysis (cf. Algorithm 2) have indeed a significant role in decision-making.

To answer this research question, we employ the DeepKnowledge-based model retraining strategy. This strategy aims to augment the training set of a DNN using a set of selected inputs generated by the DeepKnowledge approach. Then, we deploy these inputs to retrain the DNN and evaluate if they help improving the DNN's accuracy.

As explained in Algorithm 3, we select Transfer Knowledge neurons for the SVHN, MNIST and CIFAR-10 datasets, respectively. As shown in line (6), we select an equivalent number of neurons using a random-selection strategy. We deploy these TrKnw neurons (with $LowHD$, then $HighHD$) for each dataset to generate respectively knw_{Low} and knw_{High} sets. knw_{Low} and knw_{High} are each a sample set of size 6K, 3K and 5K, respectively for MNIST, SVHN and CIFAR-10. The next step is to augment the original training data of each DNN model (i.e., LeNet1, LeNet5, ResNet18) with the DeepKnowledgebased inputs, i.e., $knwset_{Low}$ and $knwset_{High}$ for neurons with $LowHD$ and $HighHD$.

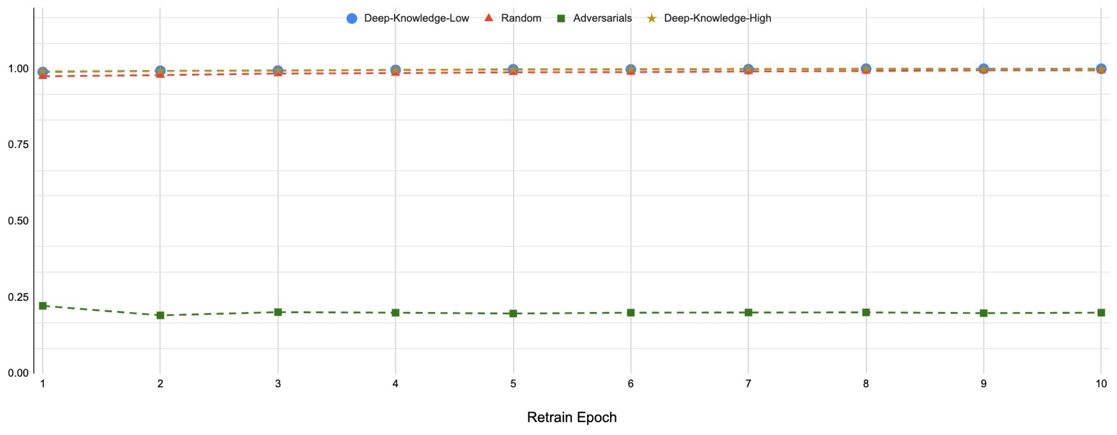
Similarly, we generate an equivalent number of inputs *Non-Knw Set* using the randomly selected neurons. We make sure that the random set inputs are not included in the knw_{Low} or knw_{High} sets. A fourth set is generated from adversarial inputs. Finally, we retrain the DNN using the four generated sets separately for 10 epochs. We measure the resulting accuracy, with the assumption that both sets knw_{Low} and knw_{High} are able to improve the DNN's accuracy.

Algorithm 3 DeepKnowledge based retraining strategy

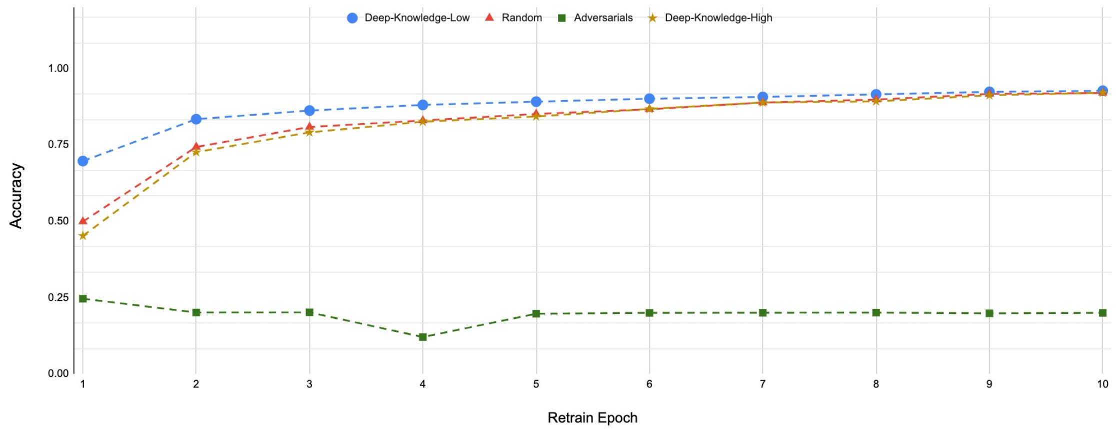
```
1: Input:$D$, $X$, $\mathcal{W}$, $\mathcal{T}$ /*$X$: test set*/
2: Generate $\kappa/\ \kappa \subset X$
3: /* $\kappa$: inputs that maximally activate the transfer knowledge neurons $\mathcal{W}$ */
4: Generate $\gamma/\ \kappa \cap \gamma = \emptyset$
5: Generate Adversarial inputs $\varrho$
6: Check $|\gamma|$ == $|\kappa|$ $==$ $|\varrho|$
7: repeat
8:     Augment Training set $\mathcal{T}^{'} = \mathcal{T} + \kappa$
9:     Retrain $D$ with $\mathcal{T}^{'}$
10:    $D \leftarrow D^{'}$
11:    Measure Accuracy
12:    $\mathcal{A} \leftarrow Accuracy(D^{'})$
13: until $\gamma$, $\kappa$ and $\varrho$ are processed
14: Return $\mathcal{A}$
```

Our experimental results comparing the three training set augmentation strategies, i.e., DeepKnowledge using low HD values ('DeepKnowledge-Low'), and high HD values ('DeepKnowledge-High'), random selection ('Random'), and adversarial inputs ('Adversarial') are shown in Table 2 and Figure 7.

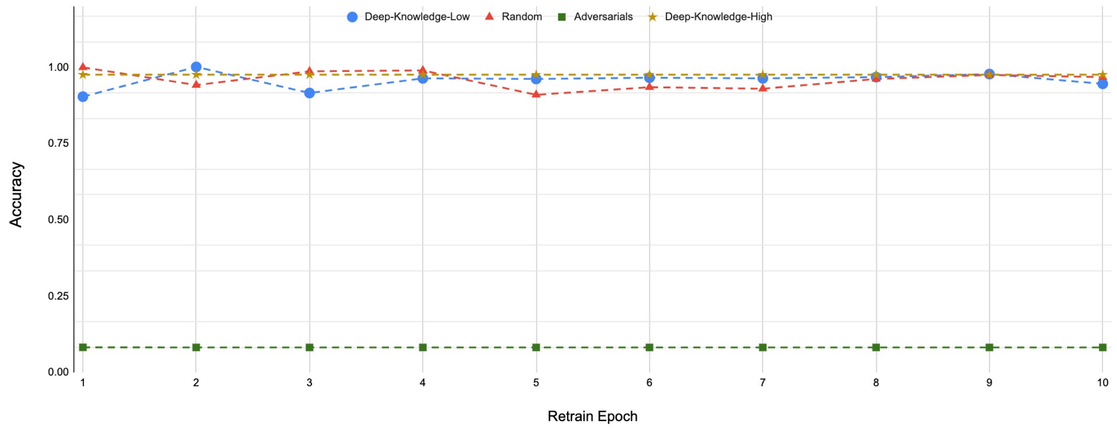
Figure 7 reports the detailed training accuracies of the different augmentation strategies throughout 10 epochs. Clearly, the DeepKnowledge-Low strategy helped achieving a better accuracy overall for the LeNet5 model. For the LeNet1 and ResNet18 models, the results of DeepKnowledge-Low, DeepKnowledge-High and Random are very close.



(a) LeNet1



(b) LeNet5



(c) ResNet18

Figure 7: Improvement in accuracy of LeNet1, LeNet5 and ResNet18 when the training set is augmented with the same number of inputs generated by random selection, adversarial testing, and DEEPKNOWLEDGE (with Low and High HD).

Table 2: DEEPKNOWLEDGE-based retraining dataset effect on model accuracy
**values between brackets shows improvement compared to the initial accuracy values

DNN Model	Dataset	Initial Accuracy	Knw_set 'Low'	#Knw_set 'High'	Random set	Adversarials
LeNet1	MNIST	97,9%	100% (+2.1)	99.77% (+1.8)	97.06% (-0.3)	19.84 %(-79.06)
LeNet5	SVHN	91.09%	92.69% (+1.6)	92.05% (+0.15)	91.97% (+0.88)	19.99 % (-71.1)
ResNet 18	CIFAR-10	93%	100% (+7)	99.80% (+6.2)	99.74% (+6.74)	20.08 % (-72.92)

The accuracy of each of the models after retraining with a separate test set shows that DEEPKNOWLEDGE-based training set augmentation achieved on average 1.5% more accuracy improvement over random/adversarial augmentation. Although this improvement is not very significant, it shows that both knw_{Low} and knw_{High} helped to fix the erroneous behaviours and therefore improve the accuracy. Notice that the sizes of these data sets are limited, e.g., only 3K instances have been generated as knw_{High} for the CIFAR-10 dataset. Since these inputs were selected based on the DEEPKNOWLEDGE approach, we have insights that TrKnw neurons are sensitive to relevant input features. We conclude that DEEPKNOWLEDGE is able to detect neurons that are core contributors to the decision-making process. This information can be used to adjust the DNN training phase by augmenting the training dataset.

RQ2: HYPER-PARAMETER SENSITIVITY

Since DEEPKNOWLEDGE relies on measuring the knowledge change under domain shift, we investigated how DEEPKNOWLEDGE varies for different divergence values, i.e., Hellinger Distance HD range and different number of selected transfer knowledge neurons (TrKnw). To evaluate the sensitivity of our approach to these hyper-parameters, we executed a set of experiments, where the granularity with which the DEEPKNOWLEDGE's coverage is specified depends on the percentage of TrKnw neurons and the feature length of each of them. Experiments are run on different publicly-available datasets and prevalent DNN models as described in Table 1.

To help us understand the effect of HD on DEEPKNOWLEDGE values, we measure our coverage with different sets of neurons where each set belongs to a different HD values range, i.e., High HD values and Low HD values. Table 3 shows the coverage results for a number of selected TrKnw neurons $n \in \{5, 6\}$ across all trainable layers for the three DNN systems. To analyse the effectiveness of DEEPKNOWLEDGE with different hyperparameters, the TrKnw neurons are being ordered by their HD values (i.e., High HD and Low HD). Then, we have selected n neurons with the lowest HD values (preferred neurons) (i.e., on average $HD \in [0, 0.5]$) to be considered as neurons with *Low HD*. We also select the top n neurons with the highest HD values (gained neurons) (i.e., on average $HD \in [0.8, 0.9]$) and label them as neurons with *High HD*. Note that these ranges differ from one dataset to another.

First, as shown in Table 3, we observe that DEEPKNOWLEDGE values decrease when the analysis is performed on a bigger number of neurons. For instance, in ResNet18 and the CIFAR-10 test set, the DEEPKNOWLEDGE coverage value decreases from 14.06% to 10.16% for $n = 5$ and $n = 6$, respectively. We observe the same behaviour with LeNet1 and the MNIST test set, as well as with LeNet5 and the SVHN test set. Although the growth in n size goes by 1 at a time, the shift on the DEEPKNOWLEDGE coverage value is still notable.

This can be explained by the fact that the combinations of TrKnw neurons clusters increases exponentially as n increases (e.g., for CIFAR-10 we have $[64]$ for $n = 5$ and $[288]$ for $n = 6$). This can also explain the Knw value for the MNIST test set, where the coverage behaviour is not consistent with the rest of the datasets for $HighHD$. In fact, the Knw score goes from 12.50%, 17.19% to 13.67% for $n = 4, 5, 6$, respectively.

Table 3: Average (std dev) coverage results for DeepKnowledge with different hyper-parameters configurations.
* Low HD (Preferred neurons) and High HD (gained neurons)

	SVHN+LeNet5			MNIST + LeNet1			CIFAR-10 + ResNet 18		
# neurons	4	5	6	4	5	6	4	5	6
Low HD*	100%	26.56%	13.67%	21.88%	14.06%	4.69%	18.75%	14.06%	10.16%
High HD*	62.50%	37.50%	18.75%	12.50%	17.19%	13.67%	6.25%	3.65%	1.22%

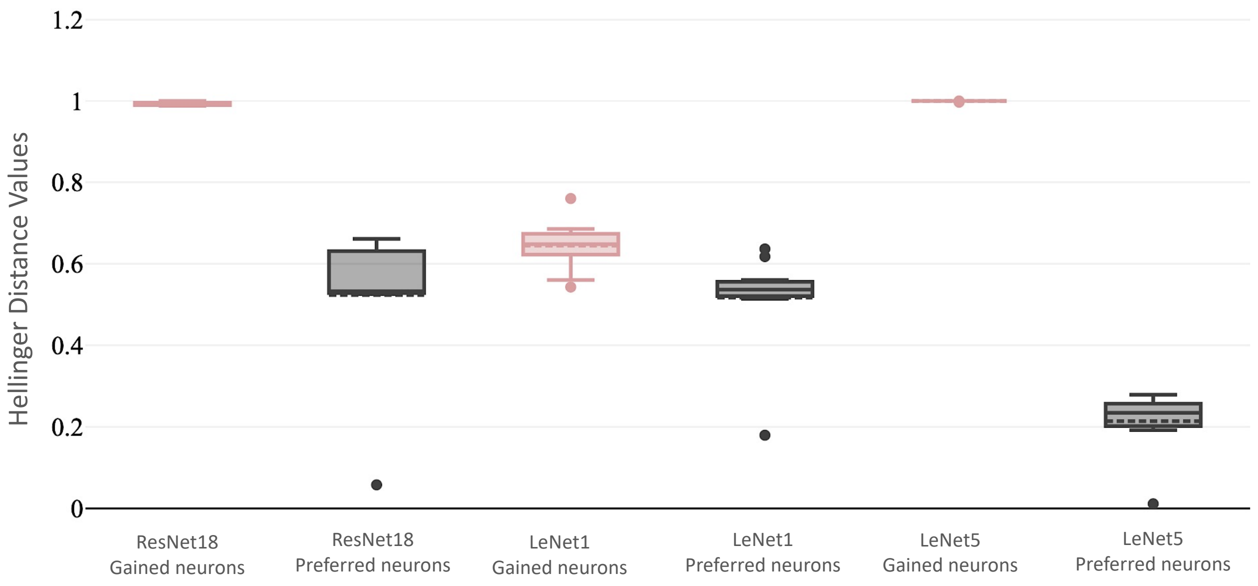


Figure 8: Comparing the average Hellinger distance HD values between preferred (Low HD) and gained (High HD) neurons for different datasets.

Indeed, if we take a closer look at the combinations of TrKnw neurons clusters we can see that they go from $[8, 64]$ for $n = 4$, $[22, 128]$ for $n = 5$ to $[35, 256]$ for $n = 6$. As we can observe, each of the added neurons has doubled the combinations of clusters. However, the number of covered combinations is still logarithmic which explains the shift in DeepKnowledge's coverage for the MNIST test set.

Similarly to the number of neurons n, we observe a big difference in DeepKnowledge coverage values when varying the HD, i.e., select neurons with $LowHD$, then the same size of neurons with $HighHD$. For instance, for ResNet18 and the CIFAR-10 test set, we notice that the Knw score goes from 100% to 62.5% with $n = 4$ for $LowHD$ to $HighHD$, respectively. To investigate further the reasons behind this big shift we looked closer to the HD values of each of the n selected TrKnw neurons as shown in Figure 8 and we plot the average HD for each dataset. We can notice that the average HD between the lowest values and highest values in each dataset is high. For instance, for the CIFAR-10 dataset, the average (std) HD equals 0.52 for $n = 10$ neurons with the lowest HD, and $std = 0.99$ for $n = 10$ neurons with the highest HD.

One important observation is that DeepKnowledge coverage values for the MNIST dataset are on average similar for both $LowHD$ (preferred) neurons and $HighHD$ (gained) neurons, which are opposite to the values for the SVHN and CIFAR-10 datasets. If we take a closer look to the HD values for both preferred and gained neurons sets for MNIST (cf. Figure 8), we can notice that medians for preferred (gray boxplot) and gained (red boxplot) are very close, i.e., 0.53 and 0.64, respectively. First, this means both sets of neurons are very similar in terms of HD values and as test adequacy criteria their proprieties are the same. As a consequence, DEEPKNOWLEDGE values are very close for both neurons sets. This shows that DEEPKNOWLEDGE values are highly correlated with HD values.

Based on these results, we answer RQ2 that DEEPKNOWLEDGE is sensitive to the selection of HD values and the number of selected neurons n it is computed from. In the following experiments, we investigate further the effect of HD on the effectiveness of DEEPKNOWLEDGE's testing criterion.

RQ3: Effectiveness of Knowledge Transfer Neurons as adequacy criteria

The effectiveness of TrKnw as an adequacy criterion is tested through the test of its sensitivity to adversarial inputs and how effective it can be in detecting misbehaviours in test sets with inputs semantically different than the original test inputs.

By implementing state-of-the-art adversarial approaches [15, 24] multiple sets of adversarial inputs are being crafted. Given that the generated adversarial inputs are semantically different, we expect an increase in the computed DEEPKNOWLEDGE coverage similarly to state-of-the-art testing criteria [57, 19, 71].

Our results reported in Table 5 (rows Knw 5 and Knw 6) show the average Knw coverage results for $n \in \{5, 6\}$ across the three augmented test sets MNIST, SVHN and CIFAR-10 with four different adversarial attacks (FGSM) [42], (BIM) [24], (MIM) [15], and (PGD) [14]. Overall, the results shows the consistency between the DEEPKNOWLEDGE's behaviour with the state-of-the-art coverage criteria based on neuron property in response to perturbed inputs using adversarial attacks. For instance, the DEEPKNOWLEDGE values have a notable increase in values with LeNet1 and MNIST once adversarial examples are introduced to the test set. For example, DEEPKNOWLEDGE coverage with original inputs is 13.67 and reaches up to 85.93 when FGSM examples are introduced. Considering CIFAR-10 and ResNet 18, the DEEPKNOWLEDGE values still increase when adding adversarial examples but with less margins, e.g., with $n = 6$, DEEPKNOWLEDGE (denoted Knw in the Table) equals 1.22 (S_0), 1.4 (FGSM), 1.56 (MIM), 1.4 (BIM). It is important to remind the reader that for each attack, we augmented the data using a set of adversarial examples with similar size to the original input set S_0.

DEEPKNOWLEDGE is also consistent with its own behaviour and decreases with the increase of n even when adversarial examples are introduced. An important observation is that DEEPKNOWLEDGE with preferred neurons, i.e., $lowHD$, is less sensitive to adversarial attacks. For instance, for the MNIST dataset with $n = 5$ the DEEPKNOWLEDGE value is equal to 14.06 for S_0 and 28.12 for all adversarial examples FGSM, BIM, MIM, and PGD. Similarly, for CIFAR-10 and SVHN the coverage values did not show a big shift in behaviour.

We notice that DEEPKNOWLEDGE saturates quickly when deploying preferred neurons with $lowHD$ value as a test adequacy criterion. We investigated this behaviour further by looking at HD values for preferred neurons in the ZeroShot domain (i.e., OOD data) and adversarial testing. Table 4 shows the median of HD calculated between OOD data and adversarial data following equation (3), i.e., HD= $HD(P_n^{ADV}, P_n^{OOD})$, with ADV denoting data generated using one of the adversarial techniques (FGSM, BIM, MIM, PGD).

For all three DNN models, the median of HD values between ZeroShot and adversarial datasets is minimal. For instance, for LeNet1, the median of HD between FGSM data and ZeroShot equals 1.80, while for ZeroShot and MIM data it is equal to 2.34. As previously discussed, the preferred neurons with $LowHD$ distance are able to transfer the learnt knowledge to a new domain without huge changes in their activation distributions. Having the same behaviour within the adversarial testing, we can conclude that preferred neurons are less sensitive to adversarial attacks. Thus, preferred neurons are less adequate to be used as a test adequacy criterion.

Table 4: Median values of HD for '*preferred*' neurons between ZerShot and Adversarial Testing

	LeNet1 + MNIST		LeNet5 + SVHN		ResNet18 + CIFAR-10	
	preferred neurons	gained neurons	preferred neurons	gained neurons	preferred neurons	gained neurons
Zeroshot/FGSM	0.2017	0.535	0.208	0.966	0.252	0.995
Zeroshot/BIM	0.191	0.684	0.180	0.882	0.202	0.992
Zeroshot/MIM	0.204	0.6447	0.234	0.768	0.211	0.989
Zeroshot/PGD	0.221	0.6225	0.199	0.815	0.179	0.988

RQ4: Correlation

In these experiments, we assess how much DEEPKNOWLEDGE is correlated to existing coverage criteria for DNN systems. We evaluate this correlation in response to semantically diverse input sets. We control the input diversity by cumulatively adding inputs from the test dataset (i.e., 25%, 50%, 75% and 100%). Hence, we report results on how state-of-the-art coverage criteria behave across the three tests sets SVHN, MNIST and CIFAR-10 in Table 6 versus DEEPKNOWLEDGE coverage for the different test sets sizes.

We have noticed that DEEPKNOWLEDGE with $HighHD$ values, i.e., average $HD \in [0.8, 0.9]$, is consistent with the state-of-the-art coverage criteria based on neuron activation/property values such as NC [57], KMNC [45] and NBC [45]. Overall the DEEPKNOWLEDGE's values increase with the increase of test set size, although for some datasets this shift in value is very minimal. For instance, with $n = 6$ and S_0 is MNIST, the DEEPKNOWLEDGE moves from 12.89 to 13.28 when cumulatively adding 25% of the test set and reaches its maximum 13.67 for the MNIST test set as soon as we reach 75% of its size, i.e., $|S_0| = 10K$.

Another interesting observation is that DEEPKNOWLEDGE values with TrKnw adequacy criterion selected based on $lowHD$ has a consistent behaviour where it reaches its maximum with only 25% of S_0. This can be noticed with three datasets and could be mitigated with using a relatively bigger size of $n = 6$ (i.e., number of TrKnw neurons). For instance, DEEPKNOWLEDGE increases with ± 0.7, i.e., 9.38 to 10.16 when cumulatively adding 25% of the CIFAR-10 test set inputs when $n = 6$. A similar behaviour is observed with the SVNH dataset. We notice an almost similar behaviour with TKNC and NC coverage as well.

By taking a closer look at our results, we noticed that the number of inputs that covers unique combinations of clusters of TrKnw neurons is very limited. Most of the test set inputs are covering the same combinations which leads to a low DEEPKNOWLEDGE coverage value that stagnates with only 25% of S_0 in the case of MNIST. This behaviour confirms that the selected TrKnw neurons, based on low Hellinger distance and a $lowHD$ between *In-domain* and *Out-of-Domain* datasets, signify a very fine-grained behaviour for the DNN and may not be useful a test adequacy criterion.

This adequacy criterion analyses the effect of each test set input and assesses it based on its uniqueness and ability to activate the specific combinations of TrKnw neurons. In other words, the TrKnw test adequacy criterion assesses an input based on the unique knowledge (i.e., features) that can be abstracted from it by the transfer knowledge neurons. Thus, we can conclude that DEEPKNOWLEDGE's coverage is able to test the diversity of input, in particular, the diversity of knowledge abstracted from the test set.

At a higher level, DEEPKNOWLEDGE coverage shows similar behaviour to state-of-the-art coverage criteria for DNN systems. Thus, we can conclude that there is a positive correlation between them.

Table 5: Effectiveness of DEEPKNOWLEDGE coverage (denoted Knw) versus other metrics. ('+X' means adding x-based adversarial inputs to the original test set S_0)

	ResNet18					LeNet1				
	S_0	FGSM	MIM	BIM	PGD	S_0	FGSM	MIM	BIM	PGD
Knw (n=5)	2.43	3.12	3.12	4.68	4.687	17.19	89.06	85.93	85.93	84.37
Knw (n=6)	1.22	1.4	1.56	1.4	1.41	13.67	85.93	83.59	84.37	81.25
NC	74.28	77.85	75.71	77.85	79.28	20.05	23.8	23.8	23.8	23.8
KMNC	62.7	81.1	71.6	80.5	64.9	69.2	71.5	79.6	76.1	74.3
NBC	15.1	46.5	45.3	31.3	15.1	22.85	40.7	46.2	21.5	42.4
SNAC	13.8	71.3	83.2	18.92	21.1	18.6	53.4	37.2	23.0	19.0
TKNC	74.741	74.741	92.14	92.14	92.14	88.8	91.8	91.8	91.8	91.8

5.1 THREATS TO VALIDITY

Construct Validity. The primary threat to internal validity of our methodology is the correctness of the studied DNN models, as well as the used datasets. We mitigate these construct validity threats using widely-studied datasets, i.e., SVHN [64] MNIST [4], CIFAR-10 and CIFAR-100 [5], and Fashion MNIST [78]. Also, we employed publicly-available architectures and pre-trained models that achieved competitive performance results including LeNet1 and LeNet5 [80], and ResNet18 [69]. Further, we mitigate threats to the process of identifying core units responsible for the DNN prediction, i.e., TrKnw neurons (cf. Algorithm 2) by adapting the state-of-the-art approach on analysing knowledge generalisation under domain shift [34].

Internal Validity. The primary internal threats that could introduce bias to DEEPKNOWLEDGE is the computation of the coverage value. To mitigate this threat, we design a study based on specific set of research questions to evaluate DEEPKNOWLEDGE. We first, illustrate the effectiveness of the DEEPKNOWLEDGE selection methodology of TrKnw neurons (core contributors to DNN decision) then its effectiveness as a test adequacy criterion in **RQ1** and **RQ4**, respectively. For **RQ4**, we illustrate DEEPKNOWLEDGE's effectiveness by augmenting the original test sets with numerically diverse inputs. **RQ3** was also designed to illustrate the effectiveness of DEEPKNOWLEDGE to detect adversarial examples and confirmed its positive correlation with state-of-the-art coverage criteria for $HighHD$ as a hyperparameter. The performance of our approach is also demonstrated through **RQ2** for different values of TrKnw neurons $n \in \{4, 5, 6\}$ and Hellinger Distance

[4]http://yann.lecun.com/exdb/mnist/
[5]https://www.cs.toronto.edu/ kriz/cifar.html

Table 6: DeepKnowledge coverage results (denoted Knw) for different size of the test set (in % of the original test set) versus other metrics, with the notable shift in Knw in bold.

	MNIST (LeNet1)				SVHN (LeNet5)				CIFAR-10 (ResNet18)			
	25% S_0	50% S_0	75% S_0	S_0	25% S_0	50% S_0	75% S_0	S_0	25% S_0	50% S_0	75% S_0	S_0
Knw (n=5, Low HD)	14.06	14.06	14.06	14.06	26.56	26.56	26.56	26.56	14.06	14.06	14.06	14.06
Knw (n=5, High HD)	17.19	17.19	17.19	17.19	31.25	31.25	37.50	37.50	2.08	**2.08**	2.43	2.43
Knw (n=6, Low HD)	4.69	4.69	4.69	4.69	13.28	13.28	**13.67**	13.67	9.38	9.38	9.38	**10.16**
Knw (n=6, High HD)	**12.89**	**13.28**	**13.67**	**13.67**	**15.62**	**15.62**	**18.75**	**18.75**	**0.69**	**0.69**	**0.81**	**1.22**
IDC (n=4)	12.50	12.50	12.50	12.50	12.50	12.50	18.75	18.75	20.75	20.75	23.50	23.50
NC	23.80	23.80	23.80	23.80	67.85	70.71	71.42	71.78	69.28	71.78	72.85	74.28
KMNC	49.50	56.19	59.99	62.71	30.39	37.18	40.99	43.60	31.10	38.02	41.97	44.73
NBC	3.57	9.52	11.90	15.47	2.32	5.35	7.85	12.5	1.78	4.10	5.89	9.10
SNAC	7.14	11.90	14.28	19.04	4.28	10	14.28	21.42	3.21	7.5	10.71	16.78
TKNC	100	100	100	100	84.64	86.42	87.5	87.85	86.42	88.21	88.92	89.64

$HD \in \{LowHD, HighHD\}$. Thus, we illustrate the granularity of DEEPKNOWLEDGE coverage, which increases exponentially with higher n values.

External Validity. Threats to external validity have been mostly mitigated by developing DEEPKNOWLEDGE on top of the open-source frameworks Keras and Tensorflow. Also, we used several trained DNN models and publicly-available datasets (SVHN, MNIST, Cifar-10 and Fashion MNIST). As a part of our future work, more experiments are planned to validate the performance of DEEPKNOWLEDGE for TrKnw neurons extraction using *Supervised transfer learning* rather than *ZeroShot transfer learning*. This will enable the assessment of the ability of neurons to generalise knowledge under domain shift.

6 Integration in the EDDIs

This section describes how DEEPKNOWLEDGE can be integrated into an EDDI, also explaining how the technique will allow for more efficient and effective operation while minimizing the overall risks of deploying DNN components and assuring their safety.

As shown in Figure 9, DEEPKNOWLEDGE is a design time activity, focusing on testing the robustness and safety of DNNs at design time. To help address the challenge of DNN robustness and correctness, the DEEPKNOWLEDGE is developed to support MRS applications, e.g., in computer vision operations such as person detection, by simulating the runtime configurations. Given that DNNs are generally tested at design time using testing datasets (in-distribution data – ID) and with configurations that cannot be predicted or analysed at run time, their evaluation can be faulty. In fact, this evaluation may drift if data encountered at runtime does not match what was used at design time. This can result in runtime performance that is significantly less than intended, leading to possible safety implications. DEEPKNOWLEDGEaddresses this anomaly using out-of-domain generalisation principles to measure the accuracy of DNNs in simulated settings with out-of-distribution datasets (OOD).

The DEEPKNOWLEDGE is a part of a task manager that generates test adequacy estimation reflecting on the fault-revealing ability of the testset. We focus on instance-wise errors, which are single inputs that result in a DNN model's erroneous outputs, that are found to be related to many real-world errors without malicious attackers. As detailed in Figure 9, DEEPKNOWLEDGE is designed to determine the test adequacy of the DNN component using training/testing datasets during design time (steps 1, 2, and 3).

By doing so, DEEPKNOWLEDGE can estimate the confidence in the predicted accuracy of DNN results for each outcome at runtime (step 5 in Figure), alongside the SafeML estimation. This information can then be used to make runtime decisions about safety, e.g., to generate warnings for the operator (step 5).

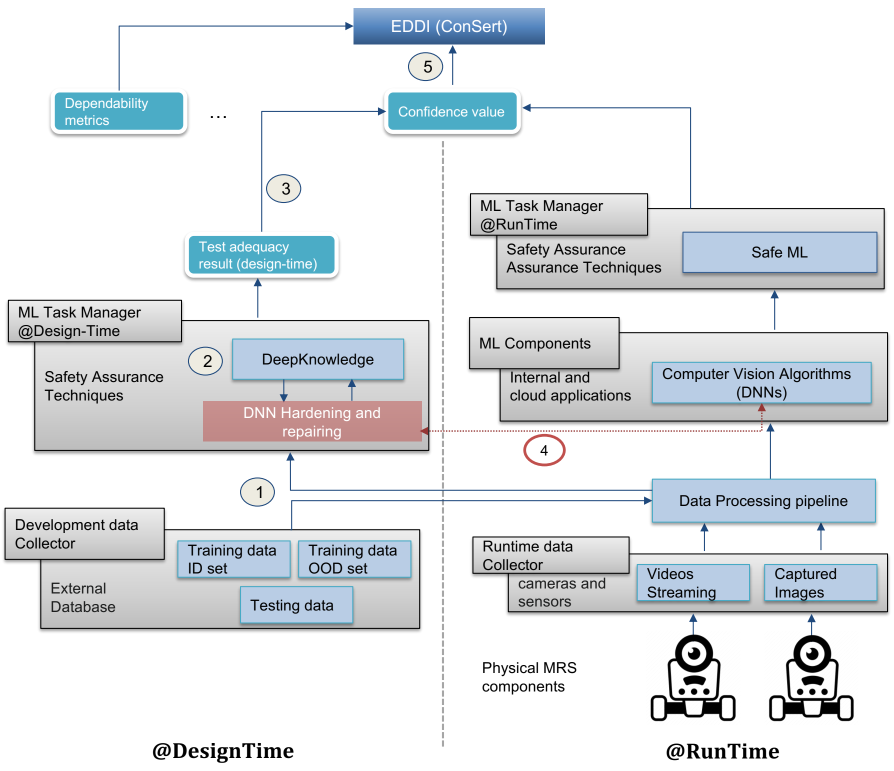


Figure 9: The process of integrating DEEPKNOWLEDGE quality assurance results into the EDDIs (steps and process on red are still under development).

Figure 10 illustrates a concrete example of the usability of DEEPKNOWLEDGEassurance results in collaboration with other WPs to assure the safety of fungicide spraying tasks. In DKOX's viticulture use case, a robotic team will spray chemical compounds in vineyard fields. This task needs to be guaranteed that no drone will get in contact with surfaces outside the area of interest. In this scenario, object detection algorithms will be deployed in a way that a certain level of safety is guaranteed. The safety of trajectory for fungicide spraying is assured through a collaboration between WP2, WP3, and WP6. When obstacles are detected within the flight path (using ML components for object detection), the WP2 updates the trajectory and provides robots with an optimal object bypass. This prediction is evaluated and its correctness us assured using DEEPKNOWLEDGE testing results.

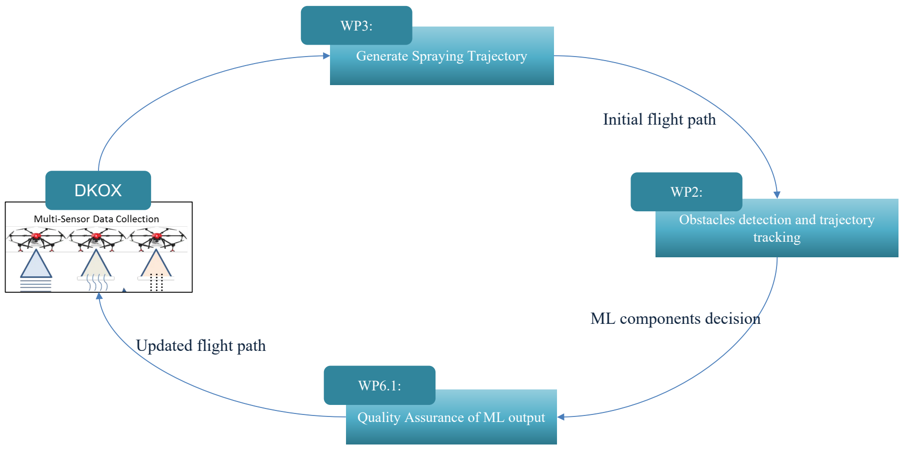

WP3:
Generate Spraying Trajectory
Initial flight path
WP2:
Obstacles detection and trajectory tracking
ML components decision
WP6.1:
Quality Assurance of ML output
Updated flight path
DKOX
Multi-Sensor Data Collection

Figure 10: Illustration example of collaboration between SESAME WPs for safety assuring of MRS trajectory planning and tracking for fungicide spraying in DKOX's viticulture use case.

7 Satisfaction of SESAME Requirements

Table 7: Satisfaction of SESAME Requirements from deliverable D.1.1: Assurance of Machine Learning

ID	Requirement	Priority	Status
U94	SESAME provides a fail-safe monitoring of program changes with fall-back option	SHOULD	In progress
U95	SESAME provides a tool that monitors if a person detection system based on a Convolutional Neural Network works in similar conditions to the tests cases	SHALL	Partial: The tool enables monitoring of computer vision models for image recognition and detection. The next step is to test it on some SESAME use cases with person detection algorithms
U96	The assurance of Machine Learning monitor is executable on a Jetson Xavier platform.	SHOULD	Full: The tool is developed in Python which helps developers to run its scripts easily on the platform (cf. deliverable D6.3 for more information).
U97	SESAME provides facilities to improve assurance of existing machine learning models (e.g., object detection and tracking, hotspot detection, etc.)	SHALL	Full: The tool provides facilities to improve assurance of deployed DNN models, through a novel test adequacy criterion and a data augmentation technique.
U98	SESAME supports analysis of adversarial machine learning of cyber-physical vulnerabilities (e.g. security)	SHOULD	Full: The tool has been tested against adversarial attacks and proven to be sensitive to adversarial inputs and is effective in detecting misbehaviours in test sets.

8 Conclusion

This document details the developed assurance technique for data-driven and learning components developed in Task 6.1. To this end, the report gives a comprehensive presentation of the developed DeepKnowledge approach, which is a systematic testing methodology for DNNs that can quantitatively measure the knowledge diversity of a test set.

DeepKnowledge is built on the principle of *knowledge generalisation* (cf. Section 2.1.1) in DNNs for analyzing how neurons transfer knowledge under domain shift. We identify a finite set of neurons responsible of knowledge generalisation and use this set to instrument a test adequacy criterion. Our experimental evaluation demonstrates the effectiveness of our approach in establishing the knowledge diversity of a test set and guiding the improvement of a DNN's accuracy. In particular, DeepKnowledge can be used to guide the selection of inputs and support effective retraining of DNN systems. DeepKnowledge is also sensitive to adversarial inputs and, thus, effective in detecting misbehaviours in test sets.

The proposed approach will be applied to the SESAME uses cases. The used methodology for DNN testing and DeepKnowledge-based augmentation technique empirically validated in this deliverable will be repurposed for Locomotec's case study first with a few adaptations to support object detection. Then DeepKnowledge will be applied to all other suitable SESAME uses cases, once their real-world data is available.

References

[1] https://www.bbc.co.uk/news/technology-35692845.

[2] Markus J Ankenbrand, Liliia Shainberg, Michael Hock, David Lohr, and Laura M Schreiber. Sensitivity analysis for interpretation of machine learning based segmentation models in cardiac mri. *BMC Medical Imaging*, 21(1):1–8, 2021.

[3] Nahid Anwar and Susmita Kar. Review paper on various software testing techniques & strategies. *Global Journal of Computer Science and Technology*, 2019.

[4] Victoria A Banks, Katherine L Plant, and Neville A Stanton. Driver error or designer error: Using the perceptual cycle model to explore the circumstances surrounding the fatal tesla crash on 7th may 2016. *Safety science*, 108:278–285, 2018.

[5] Earl T Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. The oracle problem in software testing: A survey. *IEEE transactions on software engineering*, 41(5):507–525, 2014.

[6] Syed Muzamil Basha, Dharmendra Singh Rajput, and Vishnu Vandhan. Impact of gradient ascent and boosting algorithm in classification. *International Journal of Intelligent Engineering and Systems (IJIES)*, 11(1):41–49, 2018.

[7] Vance W Berger and YanYan Zhou. Kolmogorov–smirnov test: Overview. *Wiley statsref: Statistics reference online*, 2014.

[8] Alex Bewley, Jessica Rigley, Yuxuan Liu, Jeffrey Hawke, Richard Shen, Vinh-Dieu Lam, and Alex Kendall. Learning to drive from simulation without real world labels. In *2019 International conference on robotics and automation (ICRA)*, pages 4818–4824. IEEE, 2019.

[9] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, et al. End to end learning for self-driving cars, 2016.

[10] Jason Borenstein and Ayanna Howard. Emerging challenges in AI and the need for AI ethics education. *AI and Ethics*, 1(1):61–65, 2021.

[11] Inés Sittón Candanedo, Elena Hernández Nieves, Sara Rodríguez González, M Martín, and Alfonso González Briones. Machine learning predictive model for industry 4.0. In *International Conference on Knowledge Management in Organizations*, pages 501–510. Springer, 2018.

[12] Rory CELLAN-JONES. Uber's self-driving operator charged over fatal crash, 2020.

[13] Sam Corbett-Davies and Sharad Goel. The measure and mismeasure of fairness: A critical review of fair machine learning. *arXiv preprint arXiv:1808.00023*, 2018.

[14] Yingpeng Deng and Lina J Karam. Universal adversarial attack via enhanced projected gradient descent. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 1241–1245. IEEE, 2020.

[15] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[16] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[17] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. Deepgini: prioritizing massive tests to enhance the robustness of deep neural networks. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 177–188, 2020.

[18] Samuel G Finlayson, John D Bowers, Joichi Ito, Jonathan L Zittrain, Andrew L Beam, and Isaac S Kohane. Adversarial attacks on medical machine learning. *Science*, 363(6433):1287–1289, 2019.

[19] Simos Gerasimou, Hasan Ferit Eniser, Alper Sen, and Alper Cakan. Importance-driven deep learning system testing. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pages 702–713. IEEE, 2020.

[20] Ali Ghoroghi, Yacine Rezgui, Ioan Petri, and Thomas Beach. Advances in application of machine learning to life cycle assessment: a literature review. *The International Journal of Life Cycle Assessment*, pages 1–24, 2022.

[21] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

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

[23] Ian Goodfellow and Nicolas Papernot. The challenge of verification and testing of machine learning, 2017.

[24] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. 2015.

[25] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, et al. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[26] Antonio Gulli and Sujit Pal. *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[27] Guy S Handelman, Hong Kuan Kok, Ronil V Chandra, Amir H Razavi, Shiwei Huang, Mark Brooks, Michael J Lee, and Hamed Asadi. Peering into the black box of artificial intelligence: evaluation metrics of machine learning methods. *American Journal of Roentgenology*, 212(1):38–43, 2019.

[28] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[29] Andy Hunt and Dave Thomas. *Pragmatic unit testing in c# with nunit*. The Pragmatic Programmers, 2004.

[30] Irena Jovanović. Software testing methods and techniques. *The IPSI BgD Transactions on Internet Research*, 30, 2006.

[31] James M Joyce. Kullback-leibler divergence. In *International encyclopedia of statistical science*, pages 720–722. Springer, 2011.

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

[33] Ziqiu Kang, Cagatay Catal, and Bedir Tekinerdogan. Machine learning applications in production lines: A systematic literature review. *Computers & Industrial Engineering*, 149:106773, 2020.

[34] Kenji Kawaguchi, Leslie Pack Kaelbling, and Yoshua Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[35] Varun Khare, Divyat Mahajan, Homanga Bharadhwaj, Vinay Kumar Verma, and Piyush Rai. A generative framework for zero shot learning with adversarial domain adaptation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 3101–3110, 2020.

[36] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 1039–1049. IEEE, 2019.

[37] Rudolf Kruse, Sanaz Mostaghim, Christian Borgelt, Christian Braune, and Matthias Steinbrecher. Multi-layer perceptrons. In *Computational Intelligence*, pages 53–124. Springer, 2022.

[38] Yann LeCun. The mnist database of handwritten digits. *http://yann. lecun. com/exdb/mnist/*, 1998.

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

[40] Ian Lenz, Honglak Lee, and Ashutosh Saxena. Deep learning for detecting robotic grasps. *The International Journal of Robotics Research*, 34(4-5):705–724, 2015.

[41] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[42] Yujie Liu, Shuai Mao, Xiang Mei, Tao Yang, and Xuran Zhao. Sensitivity of adversarial perturbation in fast gradient sign method. In *2019 IEEE symposium series on computational intelligence (SSCI)*, pages 433–436. IEEE, 2019.

[43] Yang Long, Li Liu, Ling Shao, Fumin Shen, Guiguang Ding, and Jungong Han. From zero-shot learning to conventional supervised classification: Unseen visual data synthesis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1627–1636, 2017.

[44] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. Deepct: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 614–618. IEEE, 2019.

[45] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 120–131, 2018.

[46] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2018.

[47] Vincent Massol and Ted Husted. *JUnit in action*. Manning, 2004.

[48] R. Thomas McCoy, Junghyun Min, and Tal Linzen. BERTs of a feather do not generalize together: Large variability in generalization across models with similar test set performance. pages 217–227, November 2020.

[49] ML Menéndez, JA Pardo, L Pardo, and MC Pardo. The jensen-shannon divergence. *Journal of the Franklin Institute*, 334(2):307–318, 1997.

[50] Gerard Meszaros. *xUnit test patterns: Refactoring test code*. Pearson Education, 2007.

[51] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[52] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. Layer-wise relevance propagation: an overview. *Explainable AI: interpreting, explaining and visualizing deep learning*, pages 193–209, 2019.

[53] Behnam Neyshabur, Srinadh Bhojanapalli, David McAllester, and Nati Srebro. Exploring generalization in deep learning. *Advances in neural information processing systems*, 30, 2017.

[54] Giuseppe De Nicola, Pasquale di Tommaso, Esposito Rosaria, Flammini Francesco, Marmo Pietro, and Orazzo Antonio. A grey-box approach to the functional testing of complex automatic train protection systems. In *European Dependable Computing Conference*, pages 305–317. Springer, 2005.

[55] Arghya Pal and Vineeth N Balasubramanian. Zero-shot task transfer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2189–2198, 2019.

[56] Nicolas Papernot, Ian Goodfellow, Ryan Sheatsley, Reuben Feinman, Patrick McDaniel, et al. cleverhans v2. 0.0: an adversarial machine learning library. *arXiv preprint arXiv:1610.00768*, 10, 2016.

[57] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *proceedings of the 26th Symposium on Operating Systems Principles*, pages 1–18, 2017.

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

[59] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[60] Nils Rethmeier, Vageesh Kumar Saxena, and Isabelle Augenstein. Tx-ray: Quantifying and explaining model-knowledge transfer in (un-) supervised nlp. In *Conference on Uncertainty in Artificial Intelligence*, pages 440–449. PMLR, 2020.

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

[62] Per Runeson. A survey of unit testing practices. *IEEE software*, 23(4):22–29, 2006.

[63] Eric Schmidt, Bob Work, Safra Catz, Steve Chien, Chris Darby, Kenneth Ford, Jose-Marie Griffiths, Eric Horvitz, Andrew Jassy, William Mark, et al. National security commission on artificial intelligence (ai). Technical report, National Security Commission on Artificial Intellegence, 2021.

[64] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st international conference on pattern recognition (ICPR2012)*, pages 3288–3291. IEEE, 2012.

[65] SESAME Project Partners. SESAME: Deliverable D1.2: Evaluation Plan. Technical report, University of York, Sep 2021.

[66] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[67] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Testing deep neural networks. *arXiv preprint arXiv:1803.04792*, 2018.

[68] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *International Conference on Neural Information Processing Systems*, pages 3104–3112, 2014.

[69] Sasha Targ, Diogo Almeida, and Kevin Lyman. Resnet in resnet: Generalizing residual architectures. *arXiv preprint arXiv:1603.08029*, 2016.

[70] Yuchi Tian. Repairing confusion and bias errors for dnn-based image classifiers. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1699–1700, 2020.

[71] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th international conference on software engineering*, pages 303–314, 2018.

[72] Siddhaling Urolagin, Prema KV, and NV Reddy. Generalization capability of artificial neural network incorporated with pruning method. In *International Conference on Advanced Computing, Networking and Security*, pages 171–178. Springer, 2011.

[73] SS Vallender. Calculation of the wasserstein distance between probability distributions on the line. *Theory of Probability & Its Applications*, 18(4):784–786, 1974.

[74] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[75] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[76] Huan Wang, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. Identifying generalization properties in neural networks. *arXiv preprint arXiv:1809.07402*, 2018.

[77] Yongqin Xian, Christoph H Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning—a comprehensive evaluation of the good, the bad and the ugly. *IEEE transactions on pattern analysis and machine intelligence*, 41(9):2251–2265, 2018.

[78] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[79] Jingkang Yang, Kaiyang Zhou, Yixuan Li, and Ziwei Liu. Generalized out-of-distribution detection: A survey. *arXiv preprint arXiv:2110.11334*, 2021.

[80] Chuan-Wei Zhang, Meng-Yue Yang, Hong-Jun Zeng, and Jian-Ping Wen. Pedestrian detection based on improved lenet-5 convolutional neural network. *Journal of Algorithms & Computational Technology*, 13:1748302619873601, 2019.

[81] Jie M Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 2020.

[82] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering*, 48(1):1–36, 2022.

[83] Jin Zhang and Jingyue Li. Testing and verification of neural-network-based safety-critical control software: A systematic literature review. *Information and Software Technology*, 123:106296, 2020.

[84] Zhiming Zhou, Han Cai, Shu Rong, Yuxuan Song, Kan Ren, Weinan Zhang, Yong Yu, and Jun Wang. Activation maximization generative adversarial nets. *arXiv preprint arXiv:1703.02000*, 2017.