Enhancing Deep Neural Network Reliability: Test Adequacy, Selection, and Generation Strategies for Fault Detection:

by

Zohreh Aghababaeyan

Thesis submitted to the Faculty of Engineering
In partial fulfillment of the requirements
For the Doctorate of Philosophy degree in
Computer Science

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Deep Neural Networks (DNNs) have achieved remarkable success in fields such as image recognition, medical diagnostics, and autonomous systems. However, like traditional software, DNNs are susceptible to failures, requiring rigorous testing to ensure their reliability. Although their potential to bring change is substantial, their weaknesses can lead to severe outcomes in practical applications, such as accidents in self-driving cars or incorrect treatment in AI health system recommendations. These shortcomings underscore the pressing necessity for thorough testing of DNNs to guarantee dependability and safety in essential uses.

However, testing DNNs comes with unique challenges. One significant challenge is the high cost of data labeling, as manually labeling large datasets for fault detection is both expensive and resource-intensive. Another challenge lies in the complexity of test case selection, where identifying the most fault-revealing inputs requires advanced methods to avoid inefficiencies. Additionally, model comparison poses its own difficulties, as comparing AI models for updates, compression, or selection demands specialized techniques to uncover their behavioral differences.

This thesis addresses these challenges through practical and innovative solutions:

- **Fault Estimation:** A novel clustering-based approach that groups similar mispredictions, enhancing the precision of fault detection and improving model reliability assessment.
- **Test Adequacy Metrics:** A thorough evaluation of existing metrics and the introduction of a diversity-based adequacy metric that excels in detecting critical faults, particularly in safety-critical systems.
- **DeepGD Framework:** A black-box test selection framework that reduces labeling costs by identifying diverse and fault-revealing inputs, significantly improving testing efficiency.
- **DiffGAN for Model Comparison:** A GAN-based method to generate test inputs that expose behavioral differences between models, ensuring reliable updates and robust model selection, especially in domains like autonomous driving and healthcare.

By tackling the real-world challenges of DNN testing, this work provides scalable, practical solutions that enhance safety, reliability, and robustness in high-risk applications. It directly addresses the pressing need for rigorous testing in DNNs, where failures can have life-altering consequences. These contributions mark a significant step forward in making DNN models safer and more dependable.

Acknowledgements

This research was supported by a grant from General Motors, as well as the Canada Research Chair and Discovery Grant programs of the Natural Sciences and Engineering Research Council of Canada (NSERC). Lionel Briand's contribution was partially funded by the Science Foundation Ireland grant 13/RC/2094-2. We are grateful to Amirhossein Zolfagharian for his valuable assistance in evaluating the validity of the generated inputs in our experiments. Additionally, we would like to thank Kim *et al.* [89] for their help and support in replicating the surprise adequacy coverage results.

Dedication

To my mother and father, the CEO of my success story since day one—your prayers have guided me from kindergarten scribbles to the final draft of this PhD thesis. Mom, it's finally done, and I have no doubt your unwavering support made it all possible.

To my husband, my technical support hotline, my best friend, and my unwavering mental health advocate. It was truly perfect to have my best friend, best colleague, and best husband all wrapped into one. Thank you for tolerating me from morning until night, listening to endless discussions about technical news, ideas, and frustrations. Your infinite patience, boundless encouragement, and belief in me kept me grounded. I couldn't have done this without you.

To my postdoc, mentor, friend, and colleague—you've been a one-person support system, seamlessly offering both sound advice and laughter. Reliable doesn't even begin to describe you. Thank you, Manel, for being there through it all.

To my academic supervisor Lionel, who has been like a strict yet kind boss—always challenging us with demanding tasks and pushing our boundaries. Your guidance has been invaluable, and I am truly grateful for your unwavering support.

To my industry supervisor, Ramesh who always encourages me to think about new ideas and problem definitions while reminding me to take a breather. Your guidance has meant the world to me.

To my sister, who always called with her "urgent" problems, skillfully distracting me from my own and keeping me focused on what she assured me were the "real priorities." Thank you for being my unintentional stress relief—I'm so happy you were here and having you around me made this journey so much better!

To my younger brother, who somehow always managed to capture our parents' attention at the most critical moments when I needed them the most. Thank you for your incredibly active presence and for always living your life so independently and organized, without ever relying on me. It's been a unique kind of support that I deeply appreciate!

To my amazing friends, for the inspiring collaborations, thought-provoking discussions, and—let's be honest—those procrastination sessions that magically turned productive. A special thanks to Ehsan, whose fascinating tangents contributed to one technical part of my thesis. Your unique mix of insight and humor made every discussion both productive and entertaining.

And finally, to everyone who endured the chaos of my thesis journey—this work is dedicated to all of you. It's as much yours as mine (but I'm keeping the degree).

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context and Motivation

Deep Neural Networks (DNNs) are widely used in various fields, including image processing [7], medical diagnostics [103], speech recognition [117], and autonomous driving [68]. Despite their widespread success, DNNs may exhibit unpredictable behaviors or produce incorrect outputs, particularly in safety-critical domains like autonomous driving and healthcare, where errors can have severe consequences. Unlike traditional software, where testing relies on explicit code structures, DNNs operate based on complex, learned patterns, thus making their internal decision processes harder to interpret and test. As a result, detecting faults in DNNs becomes even more challenging, given their vast input space and the complexity of their decision-making tasks.

Traditional software testing techniques, which often rely on code coverage metrics [72], fall short of capturing the intricate, data-driven nature of DNNs. These metrics are not directly applicable to DNNs because their behavior is driven by data rather than predefined logic. This limitation necessitates new testing methodologies that extend beyond code coverage-based techniques to ensure DNN reliability and safety in real-world environments. Moreover, access to labeled data for DNN testing is often limited or expensive, particularly in real-world operational contexts where massive amounts of data may be unlabeled [33]. This lack of labeled data becomes especially problematic when comparing models for tasks like model selection. Without enough inputs that can expose differences between versions of models designed for the same task, it becomes difficult to make an informed selection, increasing the risk of failure in real-world deployments.

Key areas that need to be addressed include how to assess fault detection, test selection, and test generation for model selection. These areas are crucial for advancing the state of DNN testing and ensuring that models can be safely deployed in real-world settings. To enhance the reliability of DNNs, we focus on the following problems:

1.1.1 Adequacy Metrics and Fault Detection

Traditional software testing methods often use metrics like code coverage to assess whether all parts of the software have been sufficiently tested [72]. In the context of DNNs, neuron coverage has been proposed as a similar metric to guide test case selection [89, 104, 125, 141, 144]. However, empirical studies suggest that neuron coverage and other white-box adequacy metrics do not reliably correlate with the presence of faults in DNNs [34, 100]. For instance, achieving high neuron coverage does not guarantee that a DNN has been adequately tested for its ability to handle real-world inputs, and such metrics may overlook critical failure modes. However, the implicit decision logic in DNNs makes it difficult to clearly define what constitutes a fault, as failures often result from complex interactions between learned features and inputs.

Thesis Objective 1: The goal is to develop new adequacy metrics that correlate more strongly with fault detection and can be used to guide DNN testing processes, particularly in identifying real-world faults. This includes improving our ability to define and detect faults within DNNs, and ensuring that test cases selected for evaluation are more representative of potential faults in practice.

1.1.2 Test Selection and Labeling

DNN testing is particularly challenging due to the vast input space and the necessity of labeling large datasets. For many real-world applications, such as autonomous driving or medical image analysis, ground truths (i.e., correct labels for inputs) are either unavailable or extremely expensive to obtain. Since we cannot rely on models to generate labels for test inputs, due to the lack of guarantees that these labels will be accurate, the only reliable solution is manual labeling, which is both time-consuming and costly. Existing test selection approaches often rely on neuron coverage or other white-box techniques that assume access to the DNN's internal structure, which may not be feasible for third-party or proprietary models. Moreover, DNN testing is often constrained by limited budgets, especially when testing on real hardware or with costly simulators in the loop.

Thesis Objective 2: The objective is to develop an efficient black-box test selection strategy that maximizes fault-revealing power while minimizing labeling costs, enabling practical DNN testing even when access to the model's internal details is unavailable.

1.1.3 Differential Testing

When comparing multiple DNN models, for example, to support model selection or compression, using the original test dataset alone is often insufficient to expose subtle behavioral differences between models [143, 154, 162]. For instance, two models with similar accuracy on standard test sets may exhibit significantly different behaviors in critical, real-world scenarios, which especially matters in safety-critical applications. These discrepancies are particularly crucial in ensemble learning systems or when compressed models must retain

performance while improving efficiency. Differential testing addresses this issue by generating test inputs specifically designed to highlight these behavioral differences [154, 162].

Thesis Objective 3: The goal is to develop a differential testing method capable of generating images revealing subtle yet important differences in model behavior, thus enabling better-informed decisions regarding model selection, optimization, and deployment in real-world environments.

1.1.4 Research Contributions

This thesis provides several contributions that address the challenges outlined above, providing novel and validated solutions to improve DNN fault detection, test selection, and differential testing:

- **Improved Adequacy Metrics and Fault Detection**: We propose a novel fault estimation method that clusters similar mispredictions, improving fault detection by identifying common root causes of failures. Additionally, we introduce a new diversity-based adequacy metric and conduct a thorough evaluation of existing metrics to determine their fault-revealing capabilities. This evaluation will help identify which metrics are most effective for guiding DNN testing.
- **Efficient Test Selection with DeepGD**: We present DeepGD, a black-box, multi-objective test selection framework that prioritizes test cases based on their diversity and uncertainty. This framework reduces the need for extensive labeling efforts while maximizing the likelihood of detecting distinct faults, making it a more practical solution for DNN testing in real-world applications. This approach is validated against multiple DNN models and datasets, demonstrating significant improvements in fault detection compared to existing methods and compared with all existing baselines.
- **Differential Testing with DiffGAN**: We introduce DiffGAN, a differential testing framework based on Generative Adversarial Networks (GANs), which generate realistic and diverse synthetic data. By guiding the GAN generation process through diversity and maximizing models' differences, DiffGAN generates test cases that expose subtle behavioral discrepancies between models. These test cases help reveal differences in models' behaviors, enabling better model selection and optimization, particularly in scenarios such as model compression or ensemble systems.

1.1.5 Dissemination

The findings from this research have been published in leading journals and presented at leading conferences. Our evaluation of adequacy metrics and fault estimation methods was published in IEEE Transactions on Software Engineering (TSE). The DeepGD test selection framework was published in ACM Transactions on Software Engineering and Methodology (TOSEM). These works were presented at ICSE 2024 and FSE 2024 in their journal-first track. The DiffGAN method for differential testing has been submitted to

IEEE Transactions on Software Engineering (TSE) and is currently under review. All tools, datasets, and code developed during this research are publicly available to the research community for further validation and use.

1.2 Thesis Structure

The rest of this thesis is structured as follows:

- chapter 2 addresses Thesis Objective 1 by providing:
  - Background and motivation for testing DNNs and identifying limitations of current test adequacy criteria,
  - Proposal and introduction of diversity metrics as an alternative,
  - A thorough empirical evaluation of proposed techniques against existing criteria,
    - Proposal and validation of new fault detection techniques,
  - Demonstration of the effectiveness of diversity metrics in fault detection,
  - Discussion of practical implications, recommendations, and threats to validity,
  - Review of related work and comparison with proposed techniques,
  - Conclusion and suggestions for future research directions.
- chapter 3 starts addressing Thesis Objective 2 by providing:
  - Background and motivation for testing DNNs, identifying the challenges in selecting high fault-revealing test inputs from large unlabeled datasets,
  - Proposed black-box multi-objective test selection approach using diversity and uncertainty,
  - Discussion of the practical implications, and addresses potential threats to the validity of the study,
  - Review of the related work and comparison with our proposed technique,
  - Conclusion of our work and suggestions for future work.
- chapter 4 addresses Thesis Objective 3 by providing:
  - Background and motivation for differential testing of DNNs, identifying challenges,
  - Proposal of GAN-based image generation techniques for differential testing, addressing the lack of diversity and novelty in generated test cases,
  - Empirical evaluation of DiffGAN's effectiveness in generating diverse test inputs,
  - Discussion of practical implications, computational efficiency, and potential threats to validity,
  - Conclusion and suggestions for future improvements and broader application of DiffGAN.
- chapter 5 provides:

  - A discussion of the broader practical implications of the proposed techniques across different domains, including applicability to text data and large language models (LLMs),
  - Consideration of key challenges such as computational cost, scalability, and the complexity of testing in safety-critical domains,
  - Reflection on the limitations of the current work and suggestions for extending these methods to new applications.

- chapter 6 provides:
  - A summary of the key contributions of the thesis,
  - An analysis of the impact of the proposed techniques on DNN reliability,
  - Directions for future research, and final remarks on the significance of the work.

Chapter 2

Exploring the relation between test adequacy metrics and faults in DNNs

This chapter addresses TO_1, i.e., exploring the correlation between test adequacy metrics and faults in DNNs. The content of this chapter has been published in *2023 IEEE Transactions on Software Engineering (TSE)* [2] under the title of *Black-Box Testing of Deep Neural Networks Through Test Case Diversity*.

2.1 Overview

As discussed in chapter 1, traditional software testing methods use code coverage criteria to guide test case selection, and generation and assess the adequacy of a test suite. For DNNs, test adequacy metrics like neuron coverage have been introduced. In traditional software, testers rely on coverage metrics based on two assumptions: (1) test inputs that only cover the same section of the source code are considered homogeneous, meaning either all of these inputs will trigger a failure, or none of them will, and (2) the inputs used in testing should be diverse to ensure high coverage [100]. However, these assumptions break down in DNN testing because (1) unlike code coverage, neuron coverage does not fully exercise the implicit logic embedded in DNNs; (2) the homogeneity assumption is broken with adversarial inputs, and (3) increasing the diversity of inputs does not necessarily increase DNN coverage [100]. Further, most coverage studies rely on adversarial inputs to validate their proposed criteria [65, 89, 104, 125, 141]. However, these inputs are mostly unrealistic and used to study the robustness of the DNN model instead of its accuracy. While state-of-the-art coverage criteria have been largely validated with artificial inputs generated based on adversarial methods, their claimed sensitivity to adversarial inputs does not necessarily mean that they relate to the fault detection capability of natural test input sets. This is confirmed by various studies [34, 100] that have failed to find a significant correlation between coverage and the number of misclassified inputs in a natural test input set, despite a positive correlation in the presence of adversarial test inputs. Consequently, coverage criteria may be ineffective in guiding DNN testing to increase the fault-detection

capability of natural test input sets. Further, another study [49] found that retraining DNN models with new input sets that improve coverage does not increase the robustness of the model to adversarial attacks.

Furthermore, coverage criteria require full access to the internals of the DNN state or training data, both of which are often not available to testers, especially when the DNN model is proprietary and provided by a third party. Thus, in our project, we focus on black-box input diversity metrics to provide guidance on how to assess test suites or select test cases for DNNs. We target diversity because it has been successfully used in testing software systems [19,56,74]. Intuitively, relying on diverse test inputs should increase the exploration of the fault space and thus increase the fault detection capability of a given test input set. Further, we target black-box metrics that do not require executing test inputs on the DNN under test since this is a strong practical impediment in many application contexts, such as when dealing with large models and large databases of unlabeled inputs. We also target black-box metrics that are model-independent and do not rely on the outputs of DNNs under test because they cannot be trusted when the models are not accurate [101]. Based on these requirements, we propose and investigate black-box diversity metrics for DNNs that rely on inputs' features, investigate their relationships with coverage metrics and analyze their association with fault detection. In other words, this research topic focuses on the fundamental assumptions related to the relationship between testing criteria (i.e. coverage and diversity metrics) and faults in DNNs. However, in this first research topic, we do not investigate how these testing criteria might be used for specific testing scenarios such as the selection, minimization or generation of test sets. Nonetheless, investigating the relationship between DNN faults and testing criteria is an essential step for selecting proper criteria, independent of any specific purpose.

In traditional software systems, some of the inputs causing failures are usually very close to each other [35, 36]. Similarly, it has been observed that many mispredicted inputs in DNNs fail due to the same causes [55]. Counting such inputs to assess the fault detection capability of a test suite is therefore misleading. However, the notion of fault, though rather straightforward in regular software, is elusive in DNNs. For this reason, we rely on a clustering-based fault estimation approach to group similar mispredicted inputs based on their features and misprediction behaviour [55]. We assume that each cluster corresponds to a fault because similar mispredicted inputs belonging to the same cluster are assumed to be mispredicted for similar reasons. To assess test suites for DNNs, we use and adapt three diversity metrics. As we evaluate datasets composed of images, commonly used as inputs in many DNNs (e.g. the perception layer of cyber-physical systems), we rely on a feature extraction model to extract features from images that will be used to compute the diversity of test input sets. We evaluate the selected metrics in terms of their capability to measure actual diversity based on extracted features. We then analyze their associations with fault detection in DNNs using four widely used datasets and five different DNN models. We further study state-of-the-art white-box coverage metrics and their associations with diversity and fault detection.

Based on our experiments, we show that diversity metrics, and geometric diversity (GD) [94] in particular, though black-box and without the use of any DNN internal information, far outperform existing coverage criteria in terms of fault-revealing capability and

computational time. We also show that state-of-the-art coverage metrics are not correlated to faults or diversity in natural test input sets.

Overall, the main contributions of this work are as follows:

- We propose and study the use of black-box diversity metrics to guide the testing of DNN models. We show that geometric diversity is the best option to guide the testing of DNN models because it is positively correlated to faults in subsets.
- We introduce and validate a clustering-based approach to estimate faults in DNNs as test input sets typically contain many similar mispredicted inputs caused by the same problems in the DNN model. We explain why this is a requirement to evaluate any test set evaluation criterion.
- We study state-of-the-art coverage criteria and show that there is no correlation between coverage and faults in DNN models. Further, coverage is not correlated with diversity in input sets. Our results question the reliability of coverage, as it is currently defined, to guide DNN testing if the objective is to detect as many faults as possible.

Chapter Structure. The remainder of the chapter is structured as follows. Section 2.2 presents our approach and describes the selected diversity metrics. Section 2.3 presents our empirical evaluation and results. Section 2.4 discusses the implications of our results and our recommendations for guiding the testing of DNN models. Section 2.5 describes the threats to the validity of our study. Sections 2.6 and 2.7 contrast our work with related work and conclude the work, respectively.

2.2 Approach

A central problem in software testing, especially when test oracles (verdicts) are not automated, is the selection of a small set of test cases that sufficiently exercise a software system. Intuitively, testers should select a set of diverse test cases because selecting similar test cases does not bring extra benefits to fault detection. In this chapter, we study diversity metrics with the ultimate aim of using them to guide DNN testing, relying on the best diversity metric in both the capacity to uncover erroneous behavior and computational complexity. We target black-box diversity metrics that are model-independent because we cannot rely on DNNs output when the models are not accurate [100]. We also target black-box metrics that do not require executing the model with all inputs because it would impede their application when working with large models and datasets. Therefore, we use and adapt three diversity metrics that have been applied widely in other contexts and are based on inputs. We rely on a feature extraction model to extract features from images that we use to compute diversity. In section 2.3, we will first evaluate the selected metrics in terms of their capability to measure the actual diversity of a test input set. Then we

will study their relationships with state-of-the-art white-box coverage metrics and analyze their associations with fault detection in DNNs.

In this section, we describe the feature extraction method and the diversity metrics used, followed by a detailed evaluation process in the next section.

2.2.1 Feature Extraction For Diversity Calculation

For diversity to account for the content of images, we need to extract features from each input image in the test input set. Consequently, we rely on VGG-16 [137], which is one of the most-used and accurate state-of-the-art feature extraction models [86, 115]. It is a pre-trained convolutional neural network model and consists of 16 weight layers, including 13 convolutional layers with a filter size of 3×3, and three fully connected layers. The model is trained on ImageNet [1], which is a dataset of over 14 million labeled images belonging to 22,000 categories.

We use VGG-16 to extract the features of images. A feature is an activation value on the layer after the last convolutional layer of the VGG-16 model. A set of features can characterize semantic elements such as shapes and colors. We extract the features in the test input set S and build the related feature matrix Vs where (1) each row of the matrix corresponds to the feature vector of an input in the test set, and (2) each column corresponds to a feature.

After generating the feature matrix, we normalize it by applying *Min-Max normalization* per feature, which is one of the most common and simple ways to normalize data. For each feature in Vs, the maximum and minimum values of that feature are transformed to one and zero, respectively, and every other value is transformed to a real value between zero and one. The *Min-Max normalization* is defined as follows. For every feature in the feature matrix Vs where $j \in \{1, 2, \ldots, m\}$ and m is the number of features, the normalized feature Vs'_j is calculated as follows:

$$Vs'_j = \frac{Vs_j(i) - min(Vs_j)}{max(Vs_j) - min(Vs_j)} \tag{2.1}$$

We normalize the feature matrix (1) to make the computation of the selected diversity metrics more scalable, and (2) to eliminate the dominance effect of features with large value ranges.

2.2.2 Diversity metrics

In this section, we describe the selected diversity metrics: Geometric Diversity [67, 94], Normalized Compression Distance [40, 56], and Standard Deviation.

[1] https://image-net.org/index.php

We chose these metrics based on the following criteria. First, we targeted diversity metrics that measure diversity within a subset. We did not consider metrics that measure diversity in relation to another subset (e.g. Kullback-Leibler [76], Jensen-Shannon divergence [32]). Second, we selected diversity metrics that can be applied to our datasets, specifically targeting metrics that can be applied to images. Third, we selected diversity metrics that do not depend on the DNN model under test and do not require the execution of this model with all inputs. Finally, we targeted diversity metrics that are widely used in a variety of other application contexts. For instance, the geometric diversity metric has been used in a variety of machine learning applications such as the selection of training sets with the Determinantal Point Process method [94], data summarization [102] and data clustering [85, 155]. Furthermore, the standard deviation metric is considered to be a common diversity metric that has been successfully applied in different contexts to measure text and image similarity [116]. The Normalized Compression Distance metric has been employed in many application domains such as image processing [40], security [22] and clustering [38, 40]. This metric supports any type of input and has been used recently to guide the selection of diverse input tests for regular software systems [56].

In this section, we will describe each of these metrics and discuss their strengths and limitations.

2.2.2.1 Geometric Diversity

The geometric diversity metric measures the diversity of the selected inputs [94]. As mentioned previously, this metric is widely used to select diverse input sets with the Determinantal Point Process (DPP) method [67, 94]. DPP is applied to guide the selection of diverse subsets from a fixed ground set [52] and has been used in a variety of machine learning applications for images [94], videos [66], documents [102], recommendation systems [169] and sensor placement [92]. The key characteristic of DPP is that including one item makes including other similar items less likely (i.e. a DPP assigns a greater probability to subsets of items that are diverse). Thus, a DPP value of a subset indicates its diversity, where the higher this value, the more diverse the subset. The key component in DPP is geometric diversity which measures the diversity of an input set in terms of the (hyper)volume spanned by the input feature vectors (feature matrix).

2.2.2.1.1 Definition

The geometric diversity *G(.)* is defined as follows. Given a dataset X, a number of inputs n, a number of features m, and feature vectors $V \in R^{n*m}$, the geometric diversity of a subset $S \subseteq X$ is defined as:

$$G(S) = det(Vs * Vs^T) \tag{2.2}$$

which corresponds to the squared volume of the parallelepiped spanned by the rows of Vs, since they correspond to vectors in the feature space. The larger the volume, the

more diverse S is in the feature space, as illustrated in Figure 2.1. It is expected that very different (similar) images result in very different (similar) feature vectors.

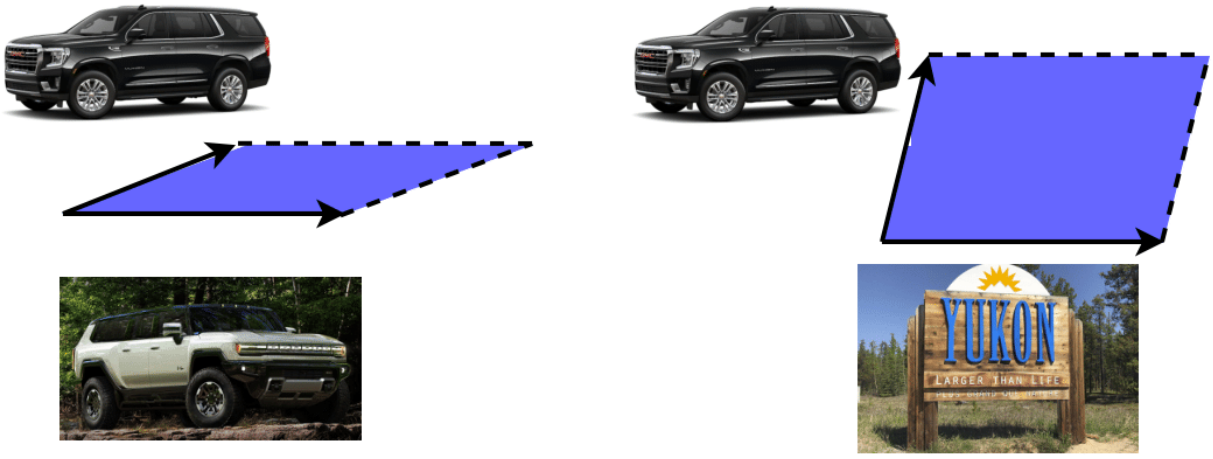


Figure 2.1: Illustration of the geometric diversity metric

2.2.2.1.2 Calculation

Geometric diversity takes as input the feature matrix of the test input set, as generated using the feature extraction model. Because geometric diversity relies on the calculation of the determinant of a matrix, we need to handle several challenges related to such processing.

Determinant Overflow. The determinant is likely to run into overflow when we work with large feature matrices. The main cause of the problem is that the determinant value is too large to be represented by a real number. To overcome this problem, we follow the recommendations of Celis *et al.* [31] and use the logarithm of this value rather than the determinant itself. We also overcome the determinant overflow problem by using the normalized feature matrix, *Vs'* thus making the geometric diversity computation more scalable.

Mathematical Limitations. If a matrix contains at least two linearly dependent vectors, its determinant will be equal to zero. Consequently, we cannot calculate the geometric diversity score of an input set that contains duplicate inputs. The feature extraction model predicts features for each test input. If the feature values are the same for two test inputs, we have duplicate inputs. This means the two test images are redundant in terms of this feature extraction model. We therefore have to delete redundant inputs before calculating the diversity score. This kind of pre-processing is acceptable in our context because (1) duplicate inputs that do not add any value to our testing model, and (2) we aim to test the DNN model with a diverse input set to detect faults.

Further, the maximum subset size for which we calculate GD must be less than the number of the features in *Vs*. This is also due to the mathematical limitations of the determinant and the rank of matrices.

Proof: In linear algebra, the rank of a matrix A of size $n * m$ refers to the number of linearly independent rows or columns in the matrix. Consequently, $Rank(A_{n*m}) <= min(n, m)$, where n is the number of lines in the matrix A, and m is the number of columns. Consider a square matrix B of size $n * n$. By definition, If $Rank(B) < n$ then $Det(B) = 0$. Let us assume that $B = A * A^T$. By definition $Rank(B) = Rank(A * A^T) = Rank(A)$. If $n > m$ then $Rank(B) \leq m < n$. As a result $Det(B) = Det(A * A^T) = 0$.

To mitigate this mathematical limitation, we can select one of the internal layers of the feature extraction model where the number of linearly independent features is equal to or greater than the size of the subset. We propose to use the deepest hidden layer, which provides enough features because, as noted by Bengio *et al.* [16,89], deeper layers represent higher-level features of the input. Specifically, we can select hidden layers that are possible candidates for feature extraction because their number of linearly independent features is greater than or equal to the size of the subset. From these candidates, we then select the deepest hidden layer because it is likely to contain the most semantically significant and helpful features for characterizing an input.

2.2.2.2 Normalized Compression Distance

The Normalized Compression Distance (NCD) is a similarity metric based on the Kolmogorov complexity [91] and information distance [17] where we measure the information required to transform one object into another to assess the similarity between these objects. Because of the complexity in calculating the Kolmogorov complexity, we approximate it by using real-world compressors [40, 98]. This leads to the normalized compression distance [38], which has been extended by Cohen *et al.* [40] to support the calculation of multisets' similarity.

2.2.2.2.1 Definition

The NCD metric for a multiset S is calculated via an intermediate measure NCD_1 [40, 56, 57]:

$$NCD_1(S) = \frac{C(S) - min_{s \in S}\{C(s)\}}{max_{s \in S}\{C(S \backslash \{s\})\}} \tag{2.3}$$

$$NCD(S) = max\Big\{NCD_1(S), max_{Y \subset S}\{NCD(Y)\}\Big\} \tag{2.4}$$

where $C(S)$ denotes the length of S after compression [40, 56].This metric is interpreted by Cohen *et al.* [40] as follows. For example, if a multiset S of strings (inputs) of about 1,000,000 bits each have pairwise information distances of 1,000 bits between each pair of inputs, then those strings can be considered relatively similar. If, on the other hand, a multiset S contains strings of about 1,200 bits each, and each pair of strings in S has a pairwise information distance of 1,000 bits, then we can conclude that the inputs in S are quite diverse [40]. NCD supports any type of input (e.g text, images, execution traces) and has many applications, such as in pattern recognition [42,90], clustering [38,40], security [22] and measuring the diversity of test sets [56, 75].

2.2.2.2.2 Calculation

We have re-implemented the NCD metric for multisets based on the original paper [56]. NCD takes the normalized feature matrix of an input set and measures its diversity score. It takes values in the range $[0, 1]$. The more diverse the input set, the larger the NCD score. However, one limitation of this metric is its high computational cost, such that its application on large input sets becomes prohibitive [40, 56]. NCD is highly sensitive to the used compression tool [56]. Different compression tools determine various performance aspects of NCD, such as computation time, used memory and compression distance. Following the recommendations of existing papers on NCD [22, 40, 56], we tried different compression tools like *Lzm*, *Bzip2* and *Zlip*. We tested their efficiency in computational cost and correctness in generating diversity scores. We evaluated the correctness of the diversity scores by controlling the actual diversity of input sets in terms of features and compared the corresponding NCD scores. We compared the NCD score of input sets with similar images to other sets with different images. The NCD score was expected to increase when the input set was more diverse in terms of features. The best results were obtained with *Bzip2*, which we used in our experiments.

2.2.2.3 Standard Deviation

Standard deviation (STD) is a statistical measure of how far from the mean a group of data points is, determined by calculating the square root of the variance.

2.2.2.3.1 Definition

STD is a straightforward measure of the diversity of a test input set based on the statistical variation of the inputs' features. We define the STD metric as the norm of the standard deviation of each feature in the test input set. Formally, we define the STD of an input set S of size n as follows:

$$STD(S) = \left\| \left(\sqrt{\sum_{i=1}^{n} \frac{Vs_{i,j} - \mu_j}{n}}, 1 \leq j \leq m \right) \right\| \tag{2.5}$$

where Vs is the feature matrix of the input set S, m is the number of features, and μ_j is the mean value of feature j in Vs.

2.2.2.3.2 Calculation

To calculate STD for an input set S, we first extract the feature matrix for S and normalize it. Then we calculate the norm of the standard deviations of each feature in the matrix to measure the diversity of the input set. The higher the STD, the more diverse the input set. One of the limitations of the standard deviation is its dependence on the mean, which

introduces unwanted bias in some cases. To explain this, we use two same-size subsets, A and B, where (1) in subset A we have two sets of similar inputs, and these two sets are far from each other in the features space, and (2) in subset B all inputs are different from one another. The variance of the inputs in subset A with respect to the mean could be larger than the one in subset B. In such a case, $STD(A)$ would be larger than $STD(B)$ though subset B is more diverse than A, as the latter only contains two truly distinct groups of inputs.

2.3 Empirical Evaluation

This section describes the empirical evaluation of our approach, including research questions, datasets, DNN models, experiments, and results.

Dataset	Description	DNN Model	Accuracy
MNIST	Handwritten digit images composed of 60,000 images for training and 10,000 images for testing.	LeNet-5	87.85%
		LeNet-1	84.5%
Fashion-MNIST	Grayscale images in 10 different classes of clothes composed of 60,000 images for training and 10,000 images for testing.	LeNet-4	88%
Cifar-10	Object recognition dataset in ten different classes composed of 50,000 images for training and 10,000 images for testing.	A 12-layer ConvNet with max-pooling and dropout layers.	82.93%
		ResNet20	86%
SVHN	A real-world image dataset for recognizing house numbers obtained from Google Street View images. It is composed of 73,257 training images and 26,032 testing images.	LeNet-5	88%

Table 2.1: Datasets and models used for evaluation

2.3.1 Research Questions

Our empirical evaluation is designed to answer the following research questions.

- **RQ1. To what extent do the selected diversity metrics measure actual diversity in input sets?** We want to assess, in a controlled way, the reliability of the selected diversity metrics for measuring the actual diversity of an input set in terms of the features the images contain. Only the metrics that reliably reflect changes in image diversity will be retained for the next research questions.

- **RQ2. How does diversity relate to fault detection?** Similar to other studies in different contexts [28, 56, 97], we aim to investigate the correlation between diversity and faults to assess whether diverse input sets lead to higher fault coverage. We do not investigate in this research question the correlation between diversity and the number of mispredicted inputs, as this is misleading. Many mispredictions result from the same problems in the DNN model and are therefore redundant. This is similar to failures in regular software. In classification problems, for example, guiding the selection of test inputs to maximize misprediction rates (the number of mispredicted inputs / total number of inputs) could thus be misleading. However, the notion of fault in DNN models is not as straightforward as it is in regular software, where we can identify statements responsible for failures. Therefore, to investigate this research question, we need to first define a mechanism to compare how effective test sets are in detecting faults in DNNs, so that we can then investigate the relationship between diversity and faults.

- **RQ3. How does coverage relate to fault detection?** Similar to diversity, we aim to assess the association between state-of-the-art coverage metrics and faults. This enables us to compare black-box diversity and white-box coverage in selecting test sets with high fault-revealing power. Note that recent studies questioned the use of coverage metrics to assess DNN test inputs [71, 100]. Most state-of-the-art coverage metrics strongly rely on artificial inputs generated based on adversarial methods [65, 89, 104, 125, 141]. However, their positive correlation with the presence of adversarial inputs does not necessarily mean that they are efficient enough to reveal the fault detection capability of natural test input sets. Several studies [34,100] failed to find a strong correlation between coverage and misprediction rates when using only natural input sets. Furthermore, coverage metrics showed poor performance in guiding the retraining of DNN models to improve the robustness of the model to adversarial attacks [49,159]. Therefore, there is still no consensus on which coverage metrics are suitable for different DNN testing-related tasks such as test selection, minimization, and generation.

- **RQ4. How do diversity and coverage perform in terms of computation time?** We aim to compare the computation times of selected diversity and coverage metrics. Most importantly, we aim to study how these computation times scale as the sizes of the test sets increase. Excessive computation times may limit applicability, though what is acceptable depends on the context.

- **RQ5. How does diversity relate to coverage?** Though diversity is black-box and therefore has inherent practical advantages, it is interesting to study the correlation between diversity and coverage to determine if they essentially capture

the same thing. Though this question can be answered indirectly by some of the previous questions (correlations are transitive), such correlation analysis can provide additional insights to explain and support previous results.

2.3.2 Subject Datasets and DL Models

Table 2.1 shows the characteristics of the datasets and models in our experiments. We used four common image recognition datasets, Cifar-10 [8], MNIST [47], Fashion-MNIST [47] and SVHN [118]. We use these datasets with five state-of-the-art DNN models: 12 layers Convolutional Neural Network (12-layer ConvNet), LeNet-1, LeNet-4, LeNet-5 and ResNet20.

Cifar-10 contains 50,000 images for training and 10,000 for testing. These images belong to 10 different classes (e.g. cats, dogs, trucks).

We also used MNIST, which contains 70,000 images (60,000 for training and 10,000 for testing). Each of these images represents a handwritten digit and belongs to one of the 10 classes. We included Fashion-MNIST, which contains grayscale images in 10 different classes of clothes. It is composed of 60,000 images for training and 10,000 images for testing. Finally, we use SVHN, a real-world image dataset for recognizing house numbers. It contains 99,289 images where 73,257 are for training and 26,032 are for testing.

For Cifar-10, we used a 12-layer ConvNet and ResNet20 that we trained for 50 and 100 epochs, respectively. For MNIST, we used the LeNet-1 and LeNet-5 models that we trained for 50 epochs. We trained the LeNet-4 model with Fashion-MNIST for 20 epochs. Finally, for SVHN, we used the LeNet-5 model, which we trained for 100 epochs. The different combinations of models and datasets, along with the models' accuracy, are detailed in Table 2.1.

We selected these datasets and models because they are widely used in the literature [65, 89, 104, 125]. Further, all the inputs in the selected datasets are correctly labelled. These datasets and models are considered good baselines to observe key trends, as they offer a wide range of diverse inputs (in classes and domain concepts) and different models (in terms of internal architecture).

2.3.3 Evaluation and Results

Before addressing our research questions, one essential issue was how to count faults in DNNs. A misprediction implies the existence of a fault in the DNN. However, identifying faults is not as straightforward as in regular software, where faulty statements that cause failures can be identified. Nevertheless, estimating fault detection effectively is essential to compare coverage and diversity metrics. Simply comparing misprediction rates is misleading as many test inputs are typically mispredicted for the same reasons [55]. Typically, with regular software, a tester does not select input tests to maximize the failure rate (equivalent to the misprediction rate in our context) but rather wants to maximize the

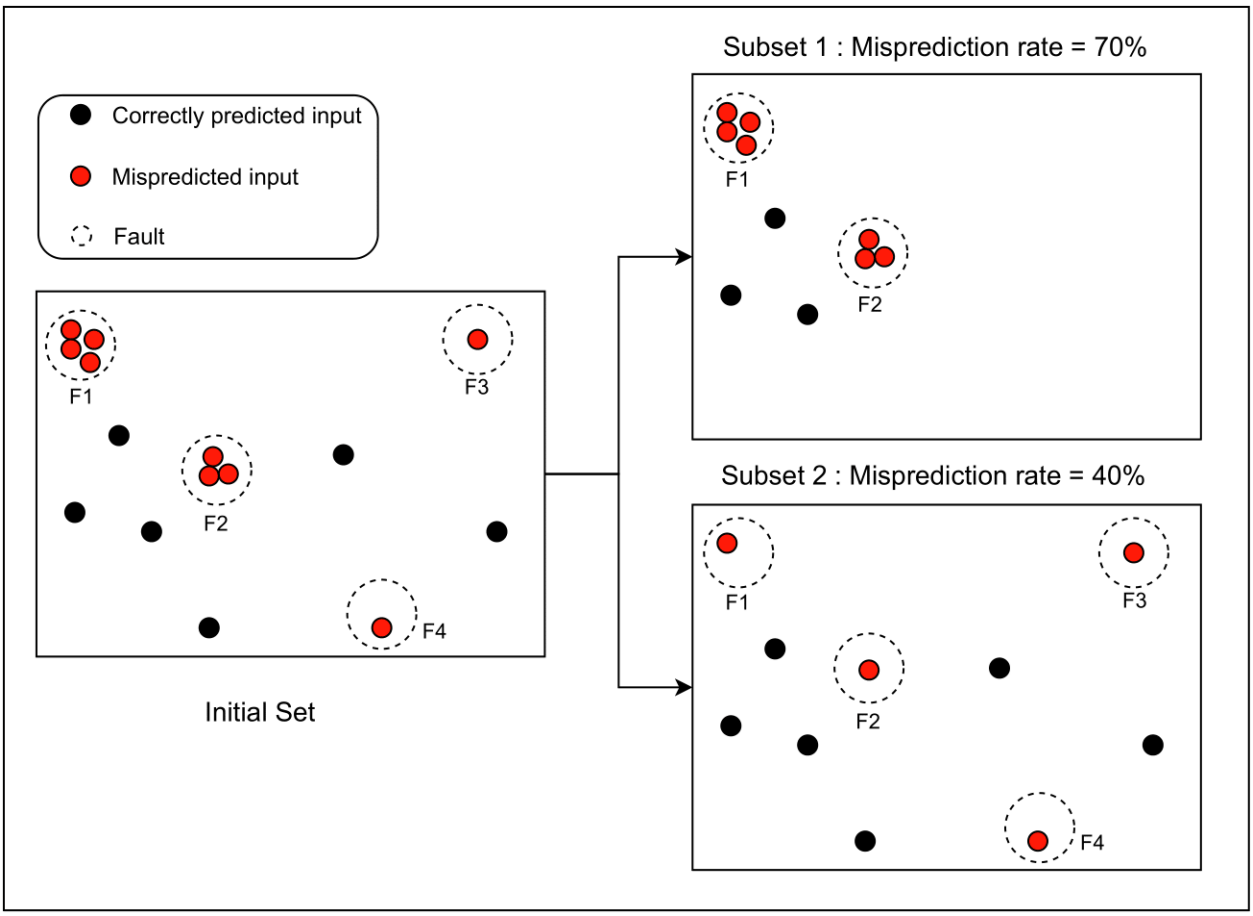


Figure 2.2: Relying on misprediction rates is misleading

number of distinct detected faults. This should be not different with DNNs, where we want to detect the distinct causes of mispredictions.

We illustrate this issue in Figure 2.2 where we represent an example of a test input set in a two-dimensional feature space. Black dots refer to the inputs correctly predicted by the DNN under test, and red dots represent the mispredicted ones. We select two subsets from the initial set and measure their corresponding misprediction rates. As shown in Figure 2.2, subset 1 is less diverse than subset 2 but has a higher misprediction rate. However, some of the mispredicted inputs are very similar and somewhat redundant.

As a result, it can be argued that subset 2 is more diverse than subset 1 and is more informative for testing the model because its mispredicted inputs potentially reveal more faults in the DNN model. In preliminary experiments, we evaluated the computation of misprediction rates in test input sets and studied their correlation with diversity and coverage and found no statistically significant correlation for both diversity and coverage metrics. We suspected that accounting for numerous redundant test inputs affected our correlation analysis. In practice, selecting or generating test inputs that trigger failures (i.e mispredictions) is far more useful when these failures are diverse [170]. A test set that repeatedly exposes the same problem in the DNN model is a waste of computational

resources, especially when we have a limited testing budget and a high labeling cost for testing data [170]. This is why, similar to other studies comparing the effectiveness of test strategies with regular software, we want here to address the notion of faults detected in DNNs and study their association with diversity and coverage.

2.3.3.1 Estimating Faults in DNNs

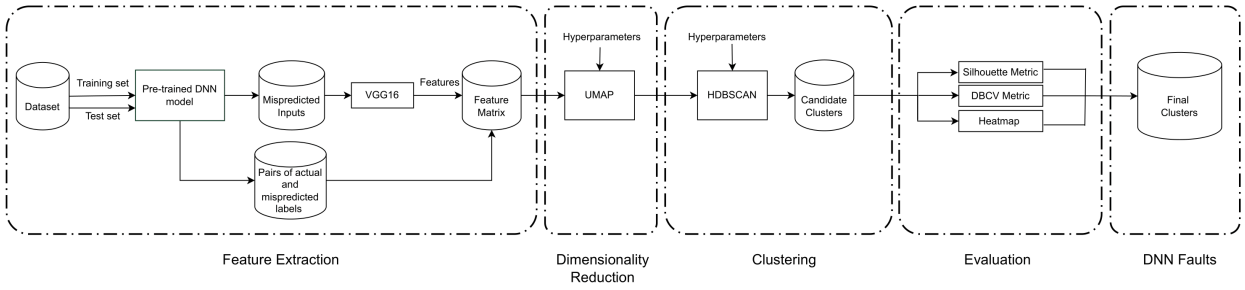


Figure 2.3: Estimating faults in DNNs

Following a similar approach to the work of Fahmy *et al.* [55] and Attaoui *et al.* [12], we rely on a clustering approach to group similar mispredicted inputs presenting a common set of characteristics that are plausible causes for mispredictions. We approximate the number of detected faults in a DNN through such clustering. Although many mispredicted test inputs are redundant and result from the same causes, we assume that test inputs belonging to different clusters are mispredicted due to distinct problems [55] in the DNN model. This is an approximation but a practical and plausible way to estimate and compare the number of detected faults across coverage and diversity strategies. Although faults can only be addressed by retraining in DNNs, as opposed to debugging, clusters nevertheless capture common causes for mispredictions and are thus comparable to faults in regular software. Figure 2.3 depicts how faults are counted in DNNs and we describe below each step in detail.

2.3.3.1.1 Feature Extraction

We start by training our model using the training dataset. We then run our pre-trained model on the test and training datasets to identify all mispredicted inputs. We not only use mispredicted inputs from the test set but also mispredicted inputs from the training dataset to extract the best clusters and estimate detected faults as accurately as possible. We rely on VGG16 to extract the mispredicted inputs' features and build the corresponding feature matrix as described in section 2.2.1. We add two extra features to the matrix from the DNN model to capture actual and mispredicted classes (labels) related to each misclassified input. This adds information to the feature matrix about the misprediction behaviour of the model under test for each mispredicted input, which we believe builds better clusters to reflect common misprediction causes.

2.3.3.1.2 Dimensionality Reduction

By definition, the number of input features for a dataset corresponds to its dimensionality. Low density in high-dimensional spaces makes it difficult, in general, for typical clustering algorithms to find a continuous boundary that separates the different clusters [84]. Therefore, employing dimensionality reduction techniques can help clustering algorithms make the inputs and their related clusters more distinguishable. Because we are working with high-dimensional inputs (512 features from the VGG model and two features from the DNN model), we rely on the Uniform Manifold Approximation and Projection (UMAP) [109] dimensionality reduction technique. We selected UMAP because several studies [48, 79] have shown its effectiveness as a pre-processing step to boost the performance of clustering algorithms when compared to other state-of-the-art dimensionality reduction techniques, such as PCA [83] and t-SNE [145]. PCA is a linear dimensionality reduction technique that performs poorly on features with nonlinear relationships. To work with high-dimensionality data to obtain low-dimensionality and nonlinear manifolds, some nonlinear dimensionality reduction algorithms, such as UMAP and t-SNE, should be used [79]. However, t-SNE is more computationally expensive than UMAP and PCA. It is used in practice for data visualization and data reduction to two or three dimensions. Furthermore, it involves hyperparameters that are not always easy to tune in order to get the best results. Therefore, we relied on UMAP for dimensionality reduction as an effective pre-processing step to boost the performance of density-based clustering. This will be used in the next step.

2.3.3.1.3 Clustering

After performing dimensionality reduction, we apply the HDBSCAN [108] clustering algorithm to group mispredicted inputs that are similar and believed to result from the same causes (faults) in the DNN model. HDBSCAN is a density-based clustering algorithm where each dense region is considered a cluster and low-density regions are considered noise. In other words, it views clusters as areas of high density separated by areas of low density. Clusters found by HDBSCAN can be of any shape, as opposed to other types of clustering algorithms, such as k-means or hierarchical clustering, which assume that clusters are convex-shaped. Each cluster is supposed to correspond to a fault (common problems) in the DNN model because their inputs are similar in terms of extracted features and actual and mispredicted classes.

2.3.3.1.4 Evaluation

As with any clustering algorithm, there are several hyperparameters to fine-tune to obtain the best clustering results. Such hyperparameters include for example, the minimum distance that controls how tightly UMAP is allowed to pack points together, the number of neighbours to consider as locally connected in UMAP and the minimum size of clusters in HDBSCAN. We tried several hyperparameter configurations and selected the best configurations based on both manual and metric-based evaluations. For the latter, we relied on

Dataset	Model	#Train Misp.	#Test Misp.	Silh.	DBCV	#Noisy test	#Clusters
MNIST	LeNet-5	8055	1215	0.64	0.68	58	85
MNIST	LeNet-1	9754	1542	0.71	0.74	72	137
Fashion-MNIST	LeNet-4	5636	1157	0.65	0.53	101	141
Cifar-10	12-layer ConvNet	1173	1707	0.71	0.62	56	187
Cifar-10	ResNet20	2191	1384	0.75	0.63	78	177
SVHN	LeNet-5	151	3009	0.69	0.59	213	147

Table 2.2: Fault estimates across datasets and models

two standard metrics to evaluate the clusters, which are the Silhouette score [129] and the Density-Based Clustering Validation (DBCV) [114] metric.

The Silhouette score is one of the state-of-the-art clustering evaluation metrics that compare inter- and intra-cluster distances. It varies between minus one and one. The closer to one, the better the clustering. A score near zero represents clusters with inputs very close to the decision boundary of the neighboring clusters. A negative score generally indicates that the inputs are assigned to the wrong clusters.

We also relied on the DBCV metric to evaluate the generated clusters. This metric is dedicated to density-based clustering algorithms and assesses clustering quality based on the relative density connection between pairs of inputs. It evaluates the within- and between-cluster density connectedness [53]. Similar to Silhouette, DBCV generates scores between -1 and 1 [114]. High-density within clusters and low-density between clusters lead to high DBCV scores, indicating better clustering results.

We selected the configuration with the best Silhouette and DBCV scores. We further evaluated the generated clusters by performing a manual evaluation. First, we tried to check the content of the clusters to see whether their inputs were similar to or shared some features that might have led to mispredictions by the DNN model. Because of the large number of mispredicted inputs, an exhaustive manual inspection of the clusters was impractical. Therefore, we relied on generating the features' heatmaps related to each cluster to better visualize and assess the quality of the clusters. Figures 2.4, 2.5, 2.6 and 2.7 illustrate four representative examples of heatmaps where rows correspond to the inputs' ids in one cluster, columns refer to their features and colours encode the features' values. As we observe in Figures 2.4, 2.6 and 2.7, within a cluster, well-clustered inputs share common patterns in terms of the features' distribution while ill-clustered inputs

(such as noisy inputs) do not, as is visible in Figure 2.5. Based on our manual analysis of the final selected clusters, we observed that most of them look like the first three figures and share common patterns. We therefore conclude that the mispredicted inputs inside each cluster are similar and share common characteristics (features), potentially causing mispredictions.

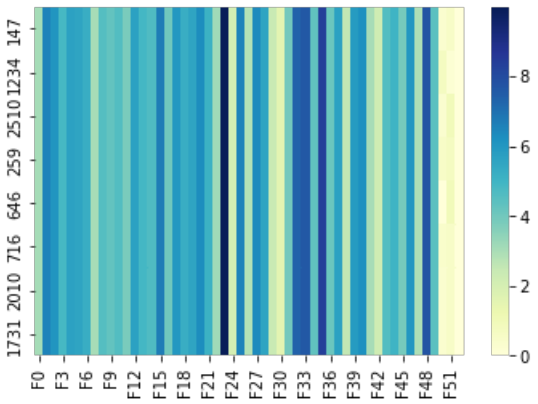


Figure 2.4: Heatmap example 1 related to a final cluster

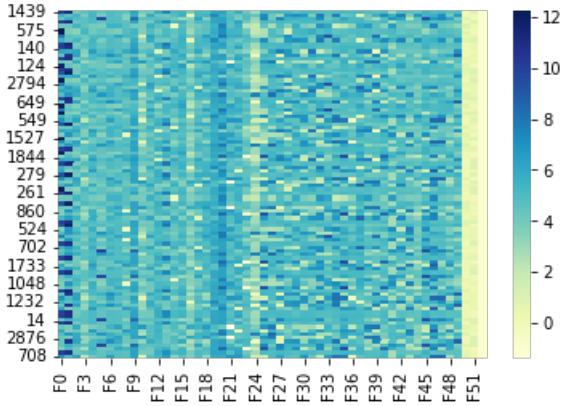


Figure 2.5: Heatmap example 2 related to a noisy cluster

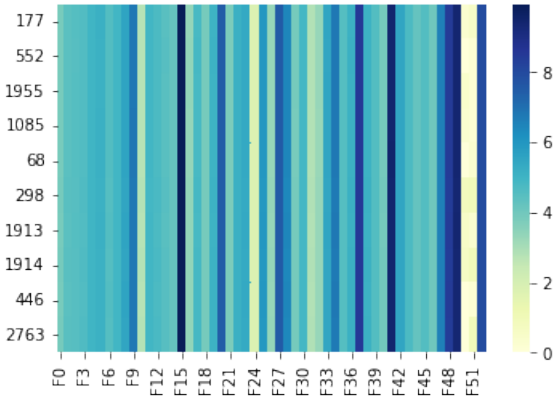


Figure 2.6: Heatmap example 3 related to a final cluster

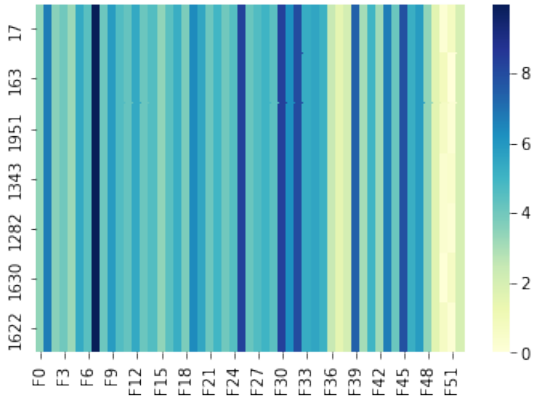


Figure 2.7: Heatmap example 4 related to a final cluster

Table 2.2 describes the final clusters that we generated for the different datasets and models that we used in our experiments.

We observe that the number of noisy inputs (inputs that do not belong to any cluster) is not large compared to the total number of mispredicted inputs. We decided to delete them from the sets of mispredicted inputs in all the following experiments because (1) they do not belong to any cluster and cannot therefore be associated with faults as we defined them, and (2) the minimum number of detected faults in the studied DNN models can thus be assumed to correspond to the number of clusters.

Model	Faults C_i	Accuracy on the retraining cluster C_i	Average Accuracy on the other clusters $C_{j \neq i}$
12-layer ConvNet	Cluster 1	80%	28.23%
	Cluster 2	60%	27.42%
	Cluster 3	55%	27.45%
	Cluster 4	71.40%	27.06%
	Cluster 5	66.70%	27.40%
	Cluster 6	60%	27.95%
	Cluster 7	58.33%	27.76%
	Cluster 8	67.70%	28.60%
	Cluster 9	66.30%	27.74%
	Cluster 10	61%	27.80%
Average		**66.64%**	**27.74%**
ResNet20	Cluster 1	77%	30%
	Cluster 2	75%	29%
	Cluster 3	74%	28%
	Cluster 4	62.5%	31%
	Cluster 5	65%	29%
	Cluster 6	56.5%	28.70%
	Cluster 7	78.40%	30.3%
	Cluster 8	75%	29.9%
	Cluster 9	62.5%	31%
	Cluster 10	64%	31%
Average		**69%**	**29.80%**

Table 2.3: The results of faults validation experiment on 12-layer ConvNet and ResNet20. In each row, we retrained the model under test on 85% of each cluster C_i. We report in the third column the accuracy of the retrained model on the remaining 15% of cluster C_i. The last column refers to the average of accuracies of the retrained models over all other fault clusters C_j $(j \neq i)$

2.3.3.1.5 Validation

As mentioned earlier, we followed an approach similar to existing work [55], [12] to estimate faults in DNNs. The authors conducted an empirical study on six DNNs to validate the clustering-based fault estimation approach. They retrained the DNNs using the original training set and subsets selected from each identified cluster (i.e. fault), which led to a significant improvement in the models' accuracy, thus demonstrating the usefulness of clustering. To further validate the identified faults (clusters) in our work, we followed a finer-grained validation method which aimed to prove that (1) inputs in the same cluster tend to be mispredicted due to the same fault, and (2) inputs belonging to different clusters are mispredicted because of distinct faults.

Inputs that belong to the same cluster are mispredicted due to the same fault in the DNN model. If inputs within one cluster are mispredicted due to the same fault, retraining the model with a subset of the cluster should help fix the model with respect to that fault. We verified this hypothesis for each fault-related cluster C_i where $i \in [1, 2, ..., n]$ and n is the total number of the identified faults (clusters), by retraining the original DNN model under test with a retraining dataset consisting of the original training set and 85% of randomly selected mispredicted inputs inside C_i. The original training set was reused to prevent any reduction in model accuracy [12, 55, 80]. We then tested the retrained model on the remaining 15% of inputs in C_i. We repeated this process five times (as we randomly selected inputs from cluster C_i) and measured the average accuracy of the retrained models when tested with 15% of the remaining inputs from cluster C_i. If clusters included mispredictions caused by the same fault, the retraining process was expected to alleviate the cause of input mispredictions in C_i and thus significantly improve the accuracy of the model for images in that particular cluster.
We did not expect a perfect model with no mispredictions because we did not have a large number of mispredicted inputs in each cluster to retrain the model. Moreover, clustering was not expected to be perfect but only an approximation of faults. Because clusters did not have the same size and were not all large enough to enable this analysis, our analysis focused on the ten largest clusters across datasets and models. Due to the high computational expense of our experiments, we used two models, 12-layer ConvNet and ResNet20, to validate the fault-estimation method. Table 2.3 shows the accuracy of the retrained 12-layer ConvNet and ResNet20 models when tested on the ten largest clusters in each of their corresponding datasets. By definition, the accuracy of the original model on all cluster images was zero because they were all mispredicted. As shown in Table 2.3, there was a significant improvement in the models' accuracy, with an average equal to 66.64% in 12-layer ConvNet and 69% in ResNet20. The results therefore suggested that test inputs belonging to the same cluster are indeed mispredicted due to the same faults, thus supporting the hypothesis underlying our method of counting faults.

Inputs belonging to different clusters are mispredicted due to distinct faults. If the clusters represent distinct faults in the DNN, retraining the model with a subset of a cluster C_i should have a significant effect on the other images in C_i when compared to images in other clusters.
Consequently, to validate that inputs belonging to different clusters are mispredicted due to distinct faults, we test each of the previously retrained DNN models for each cluster C_i on the other clusters C_j where $j \neq i$. We measure the average accuracy of each of the retrained DNN models over all remaining clusters and report the results in Table 2.3. The retrained ResNet20 and 12-layer ConvNet models are significantly more accurate on the clusters for which they were retrained than on other clusters. Indeed, in 12-layer ConvNet, for example, the average accuracy for the latter is only 27.74% compared to 66.64% for the former.
We therefore conclude that inputs belonging to different clusters tend to be mispredicted due to distinct faults. Although they are quite limited, we nevertheless observed improvements in model accuracy on clusters for which the model was not retrained. This can be expected because fixing one fault in the DNN model through retraining may also, to a

more limited extent, fix other related faults, potentially improving the accuracy on other clusters. As acknowledged previously, our clustering is not perfect and although the obtained clusters' Silhouette and DBCV scores are high, they are not equal to one as shown in Table 2.2.

2.3.3.2 RQ1. To what extent do the selected diversity metrics measure actual diversity in input sets?

To directly evaluate the capability of the selected metrics to measure diversity in input sets, we studied how diversity scores changed while varying, in a controlled manner, the number of image classes covered by the input sets. Classes characterize the content of images. For example, a set of images, sampled from the Cifar-10 dataset and containing the two classes *Car* and *Deer* is considered more diverse than a set containing only *cars*. We assumed that diversity scores should increase with the number of classes that are present in an input set.

Algorithm 1 describes at a logical level our experiment's procedure to answer RQ1. This procedure aims to increase the actual diversity of the content of image sets in a controlled manner and observe whether diversity metrics are sensitive to such changes. Given a certain dataset, we started by randomly selecting the first class C_i from the dataset in our experiment (Line 1). Then, we randomly sampled, with replacement, 20 input sets of size 100. Each of these input sets was sampled from the same class C_i (Lines 2-4). We measured the diversity scores for each initial input set (Lines 5-7). For each such input set, we incrementally increased the number of classes it covered by replacing some of its inputs with new ones from a new class $C_{k \neq i}$ while maintaining a uniform distribution across classes inside the samples. To do so, for each initial input set $Fset$, we randomly selected another class C_k that we wanted to add (Lines 8 and 10) and randomly selected new inputs from the class C_k as a $Newset$ (Line 10-11). We randomly kept about $100/k$ inputs (k is the number of selected classes) of each existing class in $Fset$ (Line 12) and merged their inputs in $Fset$ with the newly selected ones in $Newset$ (Line 13). Finally, we measured the diversity scores for each input set (Lines 15-17) and repeated the process until we included images from all classes in the selected dataset (Line 9). The distribution of the diversity scores that are related to each metric using boxplots is depicted in Figure 2.8. Each boxplot illustrates the distribution of the diversity scores of 20 input sets of size 100, each with the same number of classes.

For example, when we evaluated Cifar-10, we selected 20 input sets of size 100. All the selected images inside each input set corresponded to the class *Deer*. For each selected input set, we measured the GD, NCD and STD scores. For each metric, we reported the distribution of the diversity scores related to these samples using boxplots, as depicted in Figures 2.8.a, 2.8.b and 2.8.c. We then increased the number of classes inside each sample by randomly replacing 50 images in *Deer* with new images from the *Truck* class. Each input set contained equal distribution of *Deer* and *Truck* images. We reported again the distribution of the diversity scores using boxplots. We repeated the process until we reached a total number of 10 classes inside the selected samples, while maintaining a uniform distribution across classes inside the input sets at each sampling iteration. As shown in Figure 2.8, we observed that GD outperforms NCD and STD as it exhibits a monotonic

Algorithm 1: Experimental Procedure for RQ1

Input : C: set of n classes in the dataset ($C \leftarrow \{c_1, c_2, \ldots, c_n\}$)
Output: $GDs, STDs, NCDs$

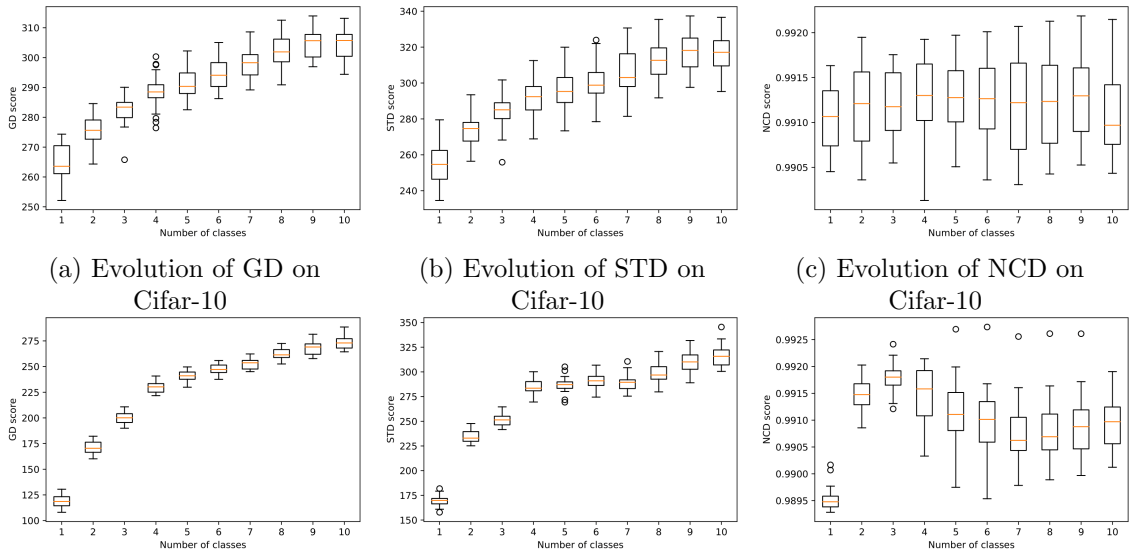
$c_i \leftarrow RandomClassSelect(1, C)$
for j *in* $\{1, 2, ..., 20\}$ **do**
  $k \leftarrow 1$
  $Fset \leftarrow RandomInputSelect(100, c_i)$
  $GDs \leftarrow GD(Fset)$
  $STDs \leftarrow STD(Fset)$
  $NCDs \leftarrow NCD(Fset)$
  $C \leftarrow C \setminus \{c_i\}$
  for k *in* $\{2, ..., c_n\}$ **do**
    $c_k \leftarrow RandomClassSelect(1, C)$
    $NewSet \leftarrow RandomInputSelect(100/k, C_k)$
    $Fset \leftarrow Keep(100/k, Fset)$
    $Fset \leftarrow Merge(Fset, Newset)$
    $C \leftarrow C \setminus \{c_k\}$
    $GDs \leftarrow GD(Fset)$
    $STDs \leftarrow STD(Fset)$
    $NCDs \leftarrow NCD(Fset)$
return $GDs, STDs, NCDs$

increase when increasing the number of classes inside the input sets. As shown in Figures 2.8.a and 2.8.d, the more diverse the input sets, the higher the GD in all datasets and models that we have evaluated. We observed a similar but noisier trend in STD. Using the example of STD scores for MNIST (Cf. Figure 2.8.e), we observe that these scores slightly decrease for samples embedding seven classes. A similar observation can be made in Cifar-10 when going from nine to ten classes (Cf. Figure 2.8.b).

Surprisingly, we found that NCD scores do not increase when input sets become more diverse. We also observed that this diversity metric has low variability in the generated scores. As shown in Figures 2.8.c and 2.8.f, the range of the calculated mean NCD scores for the different input sets in the experiment is between 0.9895 and 0.9913. We should note that we have tried other types of features in our experiments with NCD to further assess the reliability of this metric in evaluating diversity. For this purpose, we followed one of the recommendations of Cohen *et al.* [40] and Cilibrasi *et al.* [38] and calculated the NCD scores of the input sets based on the raw images from MNIST. However, we obtained similarly poor results because the NCD score did not consistently increase when input sets became more diverse.

Besides its poor performance in measuring data diversity, we note that NCD is computationally expensive. It took approximately one hour to calculate the NCD score for one input set of size 100, suggesting another limitation regarding its applicability in testing DNN models. We conclude that, in our context, this metric is neither practical nor reliable in measuring data diversity and is therefore excluded from the rest of our study.

Note that the NCD metric's poor results may be due to the combination of feature inputs



(a) Evolution of GD on Cifar-10

(b) Evolution of STD on Cifar-10

(c) Evolution of NCD on Cifar-10

(d) Evolution of GD on MNIST

(e) Evolution of STD on MNIST

(f) Evolution of NCD on MNIST

Figure 2.8: Evolution of the diversity scores for input sets from Cifar-10 and MNIST. Each boxplot shows the distribution of diversity scores of 20 input sets of size 100.

and compression tools that fail to generate accurate compression distances in our datasets. We, therefore, believe that NCD cannot be applied or function properly without careful selection of image formats and their associated feature representation and the compression tool, which is highly sensitive to these elements. Although we have tried several combinations of the aforementioned configurations based on existing prior works [38, 40], we aim in future work to investigate more combinations of feature images and dedicated compression tools to achieve better results for the NCD metric.

Answer to RQ1: Geometric diversity and STD performed well in measuring actual data diversity in all the studied datasets. This is not the case with NCD, which we excluded from the following experiments.

2.3.3.3 RQ2. How does diversity relate to fault detection?

We aimed to investigate whether higher diversity increases the fault detection capability of test sets. We randomly selected, with replacement, 60 samples of size n $\in$ {100, 200, 300, 400, 1000}. For each sample, we calculated the corresponding diversity scores (GD and STD) and the number of faults(i.e. the number of covered clusters of mispredicted inputs). We calculated the Spearman correlation [21] between the diversity scores and the number of faults. The correlation results are reported in Tables 2.6, 2.5, 2.8, 2.9 and 2.10 for the different datasets and DNN models. The grey boxes in the table refer to statistically significant correlations (p-value <= 0.05). We chose to use the Spearman correlation because

Dataset	Model	Test Set Size	Spearman	P-value
Cifar-10	12-layer ConvNet	100	8%	0.53
		200	17%	0.20
		300	-4%	0.78
		400	-9%	0.50
MNIST	LeNet-5	100	**-27%**	0.04
		200	4%	0.75
		300	-0.2%	0.98
		400	-10%	0.46

Table 2.4: Correlation between faults in subsets and clusters in the entire test set

it measures the strength of a monotonic correlation between two variables, without making assumptions about the form of the relationship or data distributions [21].

Nonetheless, there is a potential confounding factor in the correlation between faults and diversity. If we were to apply clustering in the test dataset, we would expect higher diversity to lead to more clusters. If there is a high similarity in distribution between the entire test dataset and the subset of mispredicted inputs, the correlation between diversity and faults in subsets could be due to the presence of such a confounding factor. To verify this, we analyzed the correlation between the number of clusters in the entire test dataset that were covered and the number of faults (i.e. fault-related clusters) in subsets. A low correlation would indicate that such a threat is unlikely.

We applied the same HDBSCAN clustering process (section 2.3.3.1) to the entire testing dataset to obtain data clusters. We then used the previously selected 60 samples of size n $\in$ {100, 200, 300, 400} and calculated the number of faults (i.e. the number of covered clusters of only mispredicted inputs) and data clusters (i.e. the number of covered clusters of both correctly predicted and mispredicted inputs) inside each sample. Finally, we calculated the Spearman correlation between the number of faults and the number of covered data clusters in subsets. We performed this experiment using 12-layer ConvNet (with Cifar-10) and LeNet-5 (with MNIST). The correlation results are reported in Table 2.4. As shown in the table, we did not find any positive and statistically significant correlation between faults and data clusters. We therefore conclude that there is no confounding factor in our correlation study between diversity and faults, thus giving us more confidence in the cause-effect relationship underlying the observed correlations. These results also suggest that correctly predicted inputs belong to separate clusters from fault-related clusters (i.e. clusters of mispredicted inputs) in general. They provide evidence that mispredicted inputs belong to the same cluster and share common characteristics that are different from the ones shared by correctly predicted inputs.

In our correlation experiment between diversity and faults, we evaluated a total of 60 configurations related to diversity metrics (6 models & datasets x 2 diversity metrics x 5 input sizes). As mentioned in Tables 2.6, 2.5 and 2.7, we found that GD outperforms STD in terms of fault-revealing capabilities as we observed that there was a positive, statistically significant correlation between GD and faults in all configurations (30/30).

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Cifar-10	12-layer ConvNet	GD	100	**29%**	0.02
			200	**32%**	0.01
			300	**25%**	0.05
			400	**31%**	0.02
			1000	**29%**	0.02
		STD	100	**26%**	0.05
			200	**26%**	0.05
			300	19%	0.14
			400	21%	0.11
			1000	**33%**	0.01
		LSC	100	8%	0.53
			200	4%	0.74
			300	0.5%	0.97
			400	5%	0.70
			1000	-5%	0.70
		DSC	100	2%	0.85
			200	18%	0.18
			300	3%	0.80
			400	-8%	0.55
			1000	24%	0.07
		NC	100	-22%	0.10
			200	5.3%	0.69
			300	0.3%	0.98
			400	22%	0.10
			1000	14%	0.28
		KMNC	100	0.51%	0.97
			200	14%	0.29
			300	12%	0.35
			400	-4%	0.76
			1000	19%	0.15
		NBC	100	15%	0.27
			200	5.6%	0.67
			300	7%	0.58
			400	-0.6%	0.96
			1000	11%	0.40
		TKNC	100	15%	0.27
			200	**36%**	0.005
			300	**28%**	0.031
			400	**27%**	0.04
			1000	0.2%	0.98
		SNAC	100	16%	0.22
			200	5%	0.70
			300	8%	0.52
			400	-2%	0.89
			1000	15%	0.24

Table 2.5: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
MNIST	LeNet-5	GD	100	**34%**	0.009
			200	**26%**	0.04
			300	**33%**	0.01
			400	**37%**	0.004
			1000	**35%**	0.005
		STD	100	6%	0.67
			200	**26%**	0.04
			300	**34%**	0.01
			400	23%	0.07
			1000	13%	0.34
		LSC	100	28%	0.83
			200	12%	0.36
			300	24%	0.07
			400	**32%**	0.01
			1000	19%	0.14
		DSC	100	3%	0.80
			200	-10%	0.42
			300	19%	0.16
			400	30%	0.33
			1000	8%	0.53
		NC	100	-7%	0.58
			200	2%	0.90
			300	2%	0.85
			400	**29%**	0.03
			1000	**27%**	0.03
		KMNC	100	-3%	0.83
			200	5%	0.69
			300	-13%	0.34
			400	11%	0.40
			1000	19%	0.14
		NBC	100	2%	0.88
			200	-22%	0.09
			300	9%	0.53
			400	-2%	0.87
			1000	**31%**	0.0006
		TKNC	100	27%	0.06
			200	22%	0.08
			300	17%	0.19
			400	13%	0.31
			1000	**30%**	0.02
		SNAC	100	5%	0.72
			200	**-29%**	0.02
			300	3%	0.80
			400	-10%	0.45
			1000	**32%**	0.0004

Table 2.6: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
MNIST	LeNet-1	GD	100	**33%**	0.01
			200	**39%**	0.002
			300	**28%**	0.04
			400	**33%**	0.01
			1000	**29%**	0.03
		STD	100	6%	0.67
			200	**26%**	0.04
			300	20%	0.15
			400	12%	0.36
			1000	16%	0.21
		LSC	100	-6%	0.62
			200	23%	0.08
			300	18%	0.19
			400	12%	0.35
			1000	16%	0.22
		DSC	100	13%	0.33
			200	-21%	0.10
			300	-25%	0.07
			400	17%	0.19
			1000	6%	0.67
		NC	100	6%	0.67
			200	22%	0.10
			300	24%	0.08
			400	-6%	0.67
			1000	3%	0.82
		KMNC	100	22%	0.10
			200	13%	0.32
			300	2%	0.86
			400	-3%	0.84
			1000	19%	0.15
		NBC	100	**-30%**	0.02
			200	**32%**	0.01
			300	20%	0.15
			400	8%	0.52
			1000	-13%	0.33
		TKNC	100	10%	0.46
			200	2%	0.89
			300	2%	0.88
			400	19%	0.15
			1000	13%	0.31
		SNAC	100	**-28%**	0.03
			200	**27%**	0.03
			300	25%	0.07
			400	4%	0.76
			1000	-13%	0.32

Table 2.7: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Fashion-MNIST	LeNet-4	GD	100	**31%**	0.02
			200	**32%**	0.01
			300	**30%**	0.02
			400	**26%**	0.05
			1000	**28%**	0.03
		STD	100	10%	0.48
			200	10%	0.43
			300	3%	0.82
			400	9%	0.48
			1000	18%	0.18
		LSC	100	14%	0.31
			200	10%	0.44
			300	11%	0.42
			400	9%	0.48
			1000	7%	0.61
		DSC	100	15%	0.28
			200	-19%	0.14
			300	-12%	0.37
			400	-14%	0.28
			1000	8%	0.55
		NC	100	-6%	0.66
			200	25%	0.06
			300	6%	0.63
			400	22%	0.10
			1000	10%	0.46
		KMNC	100	18%	0.18
			200	20%	0.13
			300	**36%**	0.01
			400	**27%**	0.03
			1000	24%	0.07
		NBC	100	-0.2%	0.99
			200	2%	0.88
			300	15%	0.24
			400	1%	0.93
			1000	25%	0.06
		TKNC	100	22%	0.09
			200	13%	0.31
			300	18%	0.18
			400	6%	0.66
			1000	13%	0.32
		SNAC	100	1%	0.96
			200	2%	0.86
			300	16%	0.22
			400	1%	0.95
			1000	25%	0.06

Table 2.8: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Cifar-10	ResNet20	GD	100	**26%**	0.05
			200	**31%**	0.01
			300	**37%**	0.004
			400	**29%**	0.03
			1000	**28%**	0.03
		STD	100	16%	0.22
			200	**28%**	0.03
			300	**34%**	0.01
			400	20%	0.13
			1000	**28%**	0.03
		LSC	100	16%	0.24
			200	**33%**	0.01
			300	-10%	0.43
			400	9%	0.50
			1000	-6%	0.63
		DSC	100	-3%	0.82
			200	-14%	0.28
			300	1%	0.98
			400	9%	0.48
			1000	18%	0.16
		NC	100	12%	0.38
			200	16%	0.23
			300	-6%	0.64
			400	6%	0.66
			1000	7%	0.60
		KMNC	100	4%	0.79
			200	13%	0.32
			300	18%	0.17
			400	**26%**	0.05
			1000	**33%**	0.01
		NBC	100	13%	0.31
			200	13%	0.32
			300	15%	0.25
			400	12%	0.37
			1000	7%	0.61
		TKNC	100	-1%	0.96
			200	1%	0.95
			300	16%	0.23
			400	22%	0.10
			1000	1%	0.93
		SNAC	100	16%	0.24
			200	13%	0.34
			300	19%	0.15
			400	11%	0.40
			1000	11%	0.41

Table 2.9: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Fashion-MNIST	LeNet-4	GD	100	**27%**	0.04
			200	**27%**	0.03
			300	**27%**	0.04
			400	**33%**	0.01
			1000	**30%**	0.02
		STD	100	**27%**	0.04
			200	16%	0.21
			300	20%	0.13
			400	23%	0.08
			1000	6%	0.65
		LSC	100	-5%	0.72
			200	20%	0.12
			300	10%	0.45
			400	**27%**	0.04
			1000	3%	0.83
		DSC	100	-17%	0.20
			200	-9%	0.50
			300	-5%	0.69
			400	4%	0.75
			1000	8%	0.57
		NC	100	9%	0.49
			200	6%	0.64
			300	-2%	0.89
			400	-6%	0.67
			1000	-6%	0.63
		KMNC	100	15%	0.25
			200	22%	0.09
			300	-2%	0.89
			400	**26%**	0.05
			1000	7%	0.61
		NBC	100	-6%	0.64
			200	13%	0.31
			300	16%	0.22
			400	11%	0.41
			1000	-6%	0.64
		TKNC	100	14%	0.29
			200	4%	0.77
			300	10%	0.46
			400	9%	0.50
			1000	-16%	0.23
		SNAC	100	-6%	0.64
			200	13%	0.31
			300	-16%	0.22
			400	11%	0.41
			1000	-6%	0.64

Table 2.10: Correlation results between test criteria and DNN faults. The grey boxes refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	GD	STD	LSC	DSC	NC	KMNC	NBC	TKNC	SNAC
MNIST	LeNet-1	5/5	1/5	0/5	0/5	0/5	0/5	1/5	0/5	1/5
MNIST	LeNet-5	5/5	2/5	1/5	0/5	2/5	0/5	1/5	1/5	1/5
Fashion-MNIST	LeNet-4	5/5	0/5	0/5	0/5	0/5	2/5	0/5	0/5	0/5
Cifar-10	12-layer ConvNet	5/5	3/5	0/5	0/5	0/5	0/5	0/5	3/5	0/5
Cifar-10	ResNet20	5/5	3/5	1/5	0/5	0/5	2/5	0/5	0/5	0/5
SVHN	LeNet-5	5/5	1/5	1/5	0/5	0/5	1/5	0/5	0/5	0/5
Total		**30/30**	**10/30**	**3/30**	**0/30**	**2/30**	**5/30**	**2/30**	**4/30**	**2/30**

Table 2.11: Number of positive statistically significant correlations between testing criteria and faults

Furthermore, they were consistent across all the studied models, datasets and input set sizes. On the other hand, we found that STD had a positive significant correlation with faults in only ten configurations. These results were expected because, in RQ1, GD showed better performance in measuring actual data diversity than STD.
We expected to have a moderate correlation between diversity and faults because we relied on a clustering approach to approximate faults in DNNs (section 2.3.3.1). Such correlation is expected to be higher if we have a more straightforward method to identify faults in DNNs.

Nevertheless, the obtained results clearly indicate that GD can be used to effectively guide DNN testing by devising input sets with maximum diversity to increase their fault-revealing capabilities. Let us recall that GD also has the practical advantage of being black-box, as opposed to state-of-the-art DNN coverage metrics [65, 89, 104, 125], which require access to the internals of DNN models or their training sets.

Answer to RQ2: There is a positive and statistically significant correlation between GD and faults in DNNs. GD is more frequently correlated to faults than STD. Consequently, GD should be used as a black-box approach to guide the testing of DNN models.

2.3.3.4 RQ3. How does coverage relate to fault detection?

Similar to the previous section on diversity, in this research question, we aim to study the correlation between state-of-the-art coverage criteria and faults in DNNs. Our goal is to understand how they compare to diversity in this respect. Based on three factors, we selected the following two coverage criteria: Likelihood-based Surprise Coverage (LSC) and Distance-based Surprise Coverage (DSC) [89]. First, we retained criteria that were recently published in the literature. We also chose those that (1) have been compared to other coverage criteria, and (2) showed better performance in guiding the testing of DNN

models.

We selected coverage metrics that we could apply and replicate on our models and datasets. The first two factors yielded four coverage metrics: Likelihood-based Surprise Coverage, Distance-based Surprise Coverage, Importance-Driven Coverage (IDC) [65] and Sign-Sign Coverage (SSC) [141]. However, we could not apply IDC and SSC on our datasets and models. We got several execution errors[2] when we tried to compute IDC on the 12-layer ConvNet and LeNet models. For SSC, we obtained different results from the original paper [141] when we applied this metric on LeNet-1, and encountered several execution errors in the remaining models. Therefore, we excluded these metrics from our research and only studied LSC and DSC in the correlation between coverage and fault detection in DNNs. In addition to these criteria, we included basic and widely used criteria such as Neuron Coverage (NC) [125] and DeepGauge [104] coverage metrics. We considered the following metrics related to DeepGauge: k-Multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC), Top-K Neuron Coverage (TKNC) and Strong Neuron Activation Coverage (SNAC). We describe these metrics and their limitations in Section 2.6.

To investigate the relationship between coverage and fault detection, we ran the same experiment as in RQ2 and evaluated the same selected subsets. We calculated the different coverage scores for all subsets. For LSC and DSC, we used the same recommended settings for hyperparameters (e.g. upper bound, lower bound, number of buckets) as in the original paper [89] and other existing papers in the literature [159]. We used the same hyperparameters that were recommended in the literature [104] for NC, KMNC and TKNC. We used the activation threshold of 10% for NC and fixed the number of buckets K to three for TKNC and 1,000 for KMNC; this is for the different models and datasets in our experiments. We calculated the Spearman correlation between each coverage criterion and the number of faults. The results are reported in Tables 2.6, 2.5, 2.7 and 2.11.

We considered 30 configurations per metric (6 models & datasets x 5 input sizes). Further, we accounted for a total of 210 configurations related to the coverage criteria (6 models & datasets x 7 criteria x 5 input sizes). As depicted in Table 2.11, out of 30 different configurations per metric, the distributions of positive, statistically significant correlations to faults are as follows: 5 for KMNC, 4 for TKNC, 3 for LSC, and 2 for NC, NBC and SNAC. DSC, however, did not show any statistically significant correlation with faults in any of the datasets and models.

In general, we conclude that significant correlations between coverage and faults are rare in the configurations of models and datasets that we used. None of the studied coverage metrics consistently showed statistically significant correlations across all the models, datasets and input set sizes. For example, we found that LSC is positively correlated to faults in only three out of 30 configurations related to LeNet-5 and ResNet20. However, we did not find any statistically significant correlation for this metric with LeNet-1, LeNet-4 and 12-layer ConvNet.

Our findings raise questions about the usefulness of the selected coverage criteria for enabling effective DNN testing in fault detection. These results confirm, from a different

[2]Authors have been contacted but the execution errors have not been resolved.

angle, many recent studies [34, 71, 100] that questioned the reliability of such coverage criteria to guide the testing of DNN models. A central concern raised by these articles is whether such coverage metrics relate to the model's behaviour and its decision results. Our results suggest that this relationship is, at best, weak.

Answer to RQ3: In general, significant positive correlations between coverage and faults are rarely based on the configurations and datasets used in our experiments. Coverage metrics are not a good indicator of fault detection.

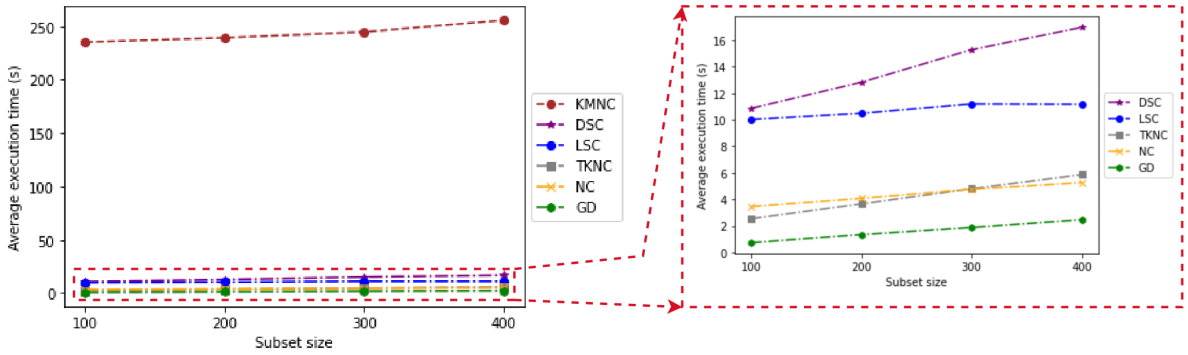


Figure 2.9: Computation time for diversity and coverage metrics with Cifar-10 and ResNet20

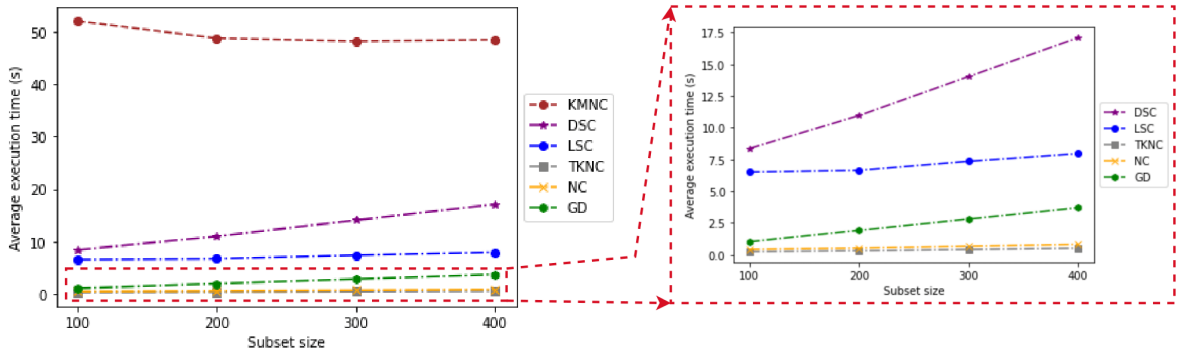


Figure 2.10: Computation time for diversity and coverage metrics with MNIST and LeNet-5

2.3.3.5 RQ4. How do diversity and coverage perform in terms of computation time?

We aimed to compare the computation times of the selected diversity metrics and coverage criteria and assessed how they scaled with the sizes of test sets. For this purpose, we randomly selected, with replacement, 60 samples of size n $\in$ {100, 200, 300, 400}. We

calculated, for each sample, diversity and coverage scores, and measured their respective computation times. Because we found in RQ2 that GD outperforms STD in correlation to faults, in the rest of this work we only used the GD metric to calculate diversity in input sets. For GD, we accounted for the sum of the following two computation times: (1) calculation of diversity based on the extracted features; and (2) the pre-processing time that is required to extract features with the VGG16 model. We report in Figures 2.9 and 2.10 the change in computation times for ResNet20 and LeNet-5 as we increased the sizes of the input sets. We observed that for both diversity and coverage metrics, computation time is linear with test set sizes up to 400. Both types of metrics are not computationally expensive. For example, the computation time related to diversity and coverage metrics in MNIST and LeNet-5 varies between 0.4 and 49 seconds for samples of size 400. We observed that KMNC, SNAC and NBC have the same computation time. Further, they are the most computationally expensive metrics. For example, KMNC, SNAC and NBC are approximately 25 times more computationally expensive than the other metrics in ResNet20. However, we found that GD generally outperforms most of the coverage metrics in computation time, except for NC and TKNC in LeNet-5. We used the Wilcoxon signed-rank test [152] to assess the statistical significance of the difference between GD's computation time and the other testing criteria. We found that GD statistically outperformed all coverage metrics in computation time. For example, it was three-to-five times faster to compute GD than LSC and DSC. Although absolute differences are a matter of seconds, such computations, in the context of test selection or minimization, may be performed thousands of times and thus become practically significant. We studied the distributions of the computation times for diversity and coverage metrics and analyzed their variations. Due to space limitations, we included the related boxplots in our replication package [1]. We found that the distributions of the computations times related to GD showed less variation than the studied coverage metrics for samples of the same size. For example, GD showed less variation than LSC and DSC for samples of the same size because the GD metric depends on the calculation of the determinant of a fixed-size feature matrix, while LSC and DSC depend on a search mechanism for the nearest inputs in the training set. Search time may vary from one sample to another, and therefore leads to increased variance in computation time.

Because GD is black-box, its computation time only depends on the used dataset and is not affected by the complexity of the DNN model (e.g. number of layers and neurons). In contrast, the computation time of white-box coverage metrics is highly sensitive to such complexity.

Answer to RQ4: Both diversity and coverage metrics are not computationally expensive (seconds). However, GD significantly outperforms coverage criteria. In application contexts, such as test case selection and minimization, and based on searches in which we perform many test set evaluations, this difference can become practically significant.

2.3.3.6 RQ5. How does diversity relate to coverage?

We aimed to study the relationship between diversity and coverage to assess if diverse input sets increase the coverage of DNN models. Conversely, increasing coverage should, in theory, increase diversity. Although the results of previous research questions make it unlikely for such correlations to be strong, this needed to be investigated.

We ran the same experiment as in RQ2 and RQ3 and used the same selected subsets. For each subset, we calculated the geometric diversity and coverage scores and measured the Spearman correlation between each pair of diversity and coverage metrics. We evaluated a total of 175 configurations (1 diversity metric x 7 coverage criteria x 5 models & datasets x 5 test set sizes). We include all the results in Appendix 2.8, and they are available online [1]. Out of 175 configurations, only 13 correlations were positive and statistically significant. For example, the only positive correlation (46%) between GD and LSC was for input sets of size 400 from Cifar-10 using ResNet-20. Furthermore, the only statistically significant correlation (-35%) between GD and DSC was for input sets of size 300 from Fashion-MNIST using LeNet-4. For NC, we found only three statistically significant correlations in three different models (LeNet-4, LeNet-5 and ResNet-20). Although we found six statistically significant correlations between KMNC and GD, these results were not consistent across all models and input set sizes. Additionally, the only statistically significant correlation (48%) between GD and NBC was for input sets of size 1,000 from MNIST using LeNet-5. Finally, for TKNC and SNAC, we found only two statistically significant correlations for each metric with GD. To summarize, most configurations (159) show no significant correlations between diversity and coverage metrics, which suggests that, in general, diversity and coverage in DNN models are not correlated. In other words, diverse input sets do not necessarily increase the coverage of DNN models and higher coverage does not systematically lead to higher diversity. These results are also consistent with our previous observations in RQ3 and RQ4, where we found that while geometric diversity is correlated to fault detection, coverage is not.

Answer to RQ5: In general, for most configurations there is no significant correlation between diversity and coverage in DNN models.

2.4 Discussion and Recommendations

We should note that our correlation results between testing criteria (diversity and coverage metrics) and faults are consistent across different datasets and DNN models. Based on our experiments, we show that studying the diversity of the features embedded in test input sets is more reliable as a basis to guide DNN testing than considering the coverage of their hidden neurons. We show that geometric diversity is potentially more effective than existing coverage metrics in guiding the testing of DNN models. This metric requires neither knowledge of the model under test nor access to the training set. Further, it does not require execution of the test input nor reliance on the output of the DNN model

under test. It is therefore a practical, black-box approach that can be used to guide the testing of DNN models. Although the results are encouraging, we only investigated geometric diversity with DNN models using images as input. Further experiments should be conducted to evaluate the performance on other input data types, such as audio and text data. We will therefore explore appropriate feature extraction models to represent new data types with feature vectors used by our diversity metrics (mainly diversity and STD metrics, because NCD supports, by default, any data type). Because diversity metrics are black-box and do not depend on the type of DNN model, we also aim in future work to consider other DNN models for regression and multi-classification tasks to further generalize our results.

In our experiments, we were surprised to see only a few significant correlations between coverage and faults across all the models and datasets we evaluated.

We selected both widely used coverage criteria in the literature and the best coverage metrics in published results and reproducibility (section 2.3.3.4). Nevertheless, coverage showed poor performance as an indicator of detected faults in DNNs. In traditional software, one of the potential reasons for the effectiveness of coverage criteria is that they rely on the logical structure of the system's source code. However, the decision logic in DNNs is not explicit, which makes the definition and usage of coverage criteria more challenging in DNNs. Also, in traditional software, relying on diverse test cases tends to increase code coverage and the fault-detection capabilities of test suites [18, 19]. In contrast, we show that in DNN testing diverse test input sets do not lead to increased DNN coverage but, at least for geometric diversity, lead to more fault detection in the DNN model.

Furthermore, traditional software systems and DNNs are fundamentally different with respect to the notion of fault and their detection. Given a test input, in general, we detect faults in software by comparing the actual test output to the expected output. If there is a mismatch, we consider this to be a failure, and we can debug the system using various fault localization techniques [124, 151, 173] to identify faulty statement(s). However, in DNN models, the notion of fault is elusive because of the black-box nature of DNN models. If the DNN model mispredicts an input, we consider this to be a failure, but debugging and localizing faults in the DNN causing such failure is challenging because there is no explicit and interpretable decision logic. This is also why DNNs are usually fixed through retraining [55]. Because it is common for many mispredicted inputs to be caused by the same problems in the DNN model [55], and because we cannot directly identify root causes, we relied on a clustering-based approach to group similar mispredicted inputs and therefore relied on the number of these clusters to approximate fault-counting in our experiments. Our clustering relied on a density-based clustering algorithm that grouped similar mispredicted inputs based on their (image) features and their misprediction behaviour (pairs of actual and mispredicted classes). Our fault estimation approach is therefore not "*complete*" because we only considered faults with a sufficient number of observed mispredictions to be grouped into a cluster. The others were considered noisy inputs by our clustering approach. In other words, we obtained a good approximation of commonly occurring faults, which underestimates the total number of faults because we do not account for noisy inputs. Because it is more important for testing approaches to detect faults, leading to more frequent mispredictions, this is practically acceptable. As described

in Table 2.2 and to reinforce this point, the number of noisy inputs is very small compared to the total number of mispredicted inputs. We acknowledge that although our retraining-based validation and evaluation results show promise, they only indirectly validate our fault model since there is no direct way to check its fault estimation accuracy. More research is needed to investigate alternative ways to enable fault detection comparisons in experiments involving DNN models.

Our study of the computation time of diversity and coverage metrics was generic and did not target a specific DNN testing scenario (e.g. selection, generation, minimization). However, this was intentional, as we wish to provide general insights into the computational complexity of coverage and diversity metrics. We showed that both types of metrics are not computationally intensive and that GD is generally three-to-five times faster to compute than the studied coverage metrics. However, whether such differences practically matter and to what extent depends on how frequently they are computed in a given application context. Some coverage-based test selection approaches entail computing the coverage score only once for each test input and selecting the test inputs with the highest coverage score. In contrast, in a typical diversity-based test selection approach, where the goal is to select a set of diverse test inputs, the GD score may be computed many times for selected subsets when, for example, the GD metric is used to drive a search algorithm. Finally, we aim in future work to further investigate the computation time of coverage and diversity metrics when used in specific DNN testing scenarios, such as test set selection, generation and minimization.

To summarize, before using any testing criteria to support a particular test scenario for DNNs (e.g. test selection, minimization and generation), one should investigate the correlation between the targeted testing criteria and faults. This is our main motivation in this work, where we investigate the relationship between testing criteria (coverage and diversity metrics) and faults in DNNs. The stronger the correlation between testing criteria and fault detection, the better. The practical implications of our results suggest that one should not rely on coverage, as currently defined, to guide DNN testing if the objective is to detect as many faults as possible. Alternatively, we show that geometric diversity has strong potential as an alternative. It outperforms existing coverage metrics in fault-revealing capability, applicability (as it is black-box) and computation time. We therefore recommend investigating its practical use in testing DNNs to guide the selection, minimization or generation of test input sets.

2.5 Threats to Validity

In this section, we discuss the different threats to the validity of our study and describe how we mitigated them.

Internal threats to validity concern the causal relationship between the treatment and the outcome. We reimplemented three diversity metrics because their source code was unavailable (GD and STD) or not applicable on our datasets (NCD). Consequently, an internal threat to validity might be related to our implementations. To mitigate this threat, we carefully checked our code and its conformance to the original papers in which they

were published. We also verified the correctness of our implementation of the NCD metric by comparing our results with an existing implementation[3] that supported the calculation of the NCD score only for pairs of images or *txt* files. In ***RQ1***, we tested, through a controlled experiment, the reliability of the selected diversity metrics in measuring actual data diversity and excluded the metrics that failed the test.

As we were targeting black-box diversity metrics, we needed to rely on a feature extraction model to build our feature matrix. Therefore, an internal threat to validity might be caused by a low-quality representation of inputs. To mitigate this threat, we relied on VGG16, which is one of the most-used, accurate, state-of-the-art feature extraction models. Furthermore, this DNN model has been pre-trained on the extremely large ImageNet dataset, which contains over 14 million labelled images belonging to 22,000 categories. Further, the configuration of the different hyperparameters in our study may induce additional internal threats to validity. We mitigate this threat in two ways: (1) for coverage metrics hyperparameters, we made use of the original papers' hyperparameter values [89] for each dataset and model that we used; and (2) for fault estimation hyperparameters (clustering), we tried more than 500 configurations related to HDBSCAN and UMAP for each of the datasets and models that we evaluated. To reduce potential bias, we evaluated the configurations' results by using two clustering evaluation metrics (section 2.3.3.1) and by visualizing heatmaps.

A final internal threat to validity is related to randomness when sampling test inputs. We addressed this issue by repeating such sampling multiple times while considering different input set sizes and different datasets and models.

Construct threats to validity concern the relation between the theory and the observations. To study the effectiveness of a given test criterion in guiding DNN testing, we relied on a clustering-based approach to estimate detected faults in DNNs. It is a potential threat to construct validity because this estimate may not be sufficiently accurate. If that is the case, correlations with diversity and coverage might appear weaker than they actually are. Alternatively, relying on misprediction rates is, as previously discussed, misleading, because in practice, many similar mispredicted inputs typically result from the same problems in the DNN model. Accounting for numerous redundant test inputs would blur our correlation analysis, an effect we observed in our study. Further, we relied on a density-based clustering algorithm that is capable of grouping similar inputs in clusters of arbitrary shapes, as opposed to other types of clustering algorithms, such as k-means and hierarchical clustering, which assume that clusters are convex. Next, we clustered similar mispredicted inputs based on their (image) features and misprediction behaviour, thus relying on what semantically distinguishes images. Finally, we quantitatively and qualitatively assessed the obtained clusters to group test inputs with similar characteristics.

Reliability threats to validity concern the replicability of our study results. We relied on publicly available models and datasets and provided all the materials required to replicate our study results online [1]. This includes the set of all selected samples in the experiments and the different configurations we used for all the selected testing criteria.

[3]https://github.com/simonpoulding/DataGenerators.jl

Conclusion threats to validity concern the relation between the treatment and the outcome. We relied on the Spearman correlation because it does not rely on assumptions about the data set distributions or on the shapes of the relationships, except for the latter being monotonic.

External threats to validity concern the generalizability of our study. We mitigated this threat by using four large datasets and five widely used and architecturally different DNN models. Further, in each of our experiments, we evaluated many samples and input set sizes. The selected coverage metrics may not be representative of all existing coverage criteria. However, we selected the best metrics based on their published results and our ability to reproduce their results.

2.6 Related Work

The work presented in this chapter relies on concepts related to test diversity, black-box testing and model coverage in DNNs. In this section, we provide an overview of existing coverage metrics for DNN models. We also describe existing work on black-box DNN testing and studies making use of diversity to guide testing of DNNs and traditional software systems.

2.6.1 Test Coverage Criteria for DNNs

Several coverage metrics have been proposed in the literature. The first attempt was carried out by Pei *et al.* [125] who proposed the Neuron Coverage (NC) metric for test inputs, which is defined as the proportion of activated neurons (neurons whose activation value is above the defined threshold) over all neurons when all available test inputs are supplied to a DNN. However, several studies [130, 141] have shown that 100% neuron coverage is easy to achieve with a small set of inputs, and consequently is going to limit the applicability of such metric when testing DNNs.

Ma *et al.* [104] proposed DeepGauge, a set of DNN coverage metrics. They introduced K-Multisection Neuron Coverage (KMNC), Neuron Boundary Coverage (NBC) and Strong Neuron Activation Coverage (SNAC). KMNC partitions the ranges of neuron activation values into K buckets based on training inputs and counts the number of total covered buckets by a given test input set. NBC measures the ratio of corner-case regions that have been covered. Corner-case regions correspond to the activation values that are beyond the activation ranges observed during training. SNAC measures how many upper corner-cases have been covered. Upper corner-cases correspond to neuron activation values that are greater than the activation ranges observed during training. The authors showed that input tests generated by adversarial methods increase coverage in terms of their metrics. However, they did not study how these metrics relate to DNN mispredictions using natural inputs.

Inspired by the MC/DC test coverage in traditional software testing, Sun *et al.* [141] proposed four coverage metrics that account for the causal relationship between neurons

in neighbouring layers of a DNN model. These metrics were used to guide the generation of test inputs using adversarial methods to test the robustness of DNN models.

Kim *et al.* [89] proposed two coverage criteria called Likelihood-based Surprise Coverage (LSC) and Distance-based Surprise Coverage (DSC). These criteria are based on an analysis of how surprising test sets are given the training set. LSC uses Kernel Density Estimation (KDE) [147] to estimate the likelihood of seeing a test input during the training phase. DSC relies on the calculation of Euclidean distances between vectors that correspond to (1) the neurons' activation values in inputs from the test set, and (2) the neurons' activation values in inputs from the training set. They argue that an input set that covers a wide and diverse range of surprise values is preferable to test and retrain a DNN model. They show that their metrics are correlated with existing coverage criteria [104, 125] when the diversity of inputs is increased. However, our study shows that there is no strong correlation between surprise adequacy coverage and diversity by using only natural inputs. We also show that there is no strong correlation between these coverage metrics and faults in DNNs. Another study conducted by Chen *et al.* [34] showed similar results with respect to DNN misprediction rates when using only natural inputs.

Gerasimou *et al.* [65] proposed the Importance-Driven Coverage (IDC) criterion to focus on the coverage of the most influential neurons in DNN predictions. They argue that IDC is sensitive to adversarial inputs and achieves higher values when applied to input sets that comprise diverse inputs. They also evaluated DeepGauge [89] and surprise adequacy coverage criteria [104] in their experiments and observed that IDC shows a similar increase in these coverage criteria when evaluated with test sets augmented with adversarial inputs.

Byun et al. [29] have recently proposed Manifold Combination Coverage (MCC), a black-box coverage metric for testing DNN models based on projecting test inputs onto a manifold space using a Variational AutoEncoder (VAE). This metric relies on manifold learning [15] that compresses a high-dimensional input space into a lower-dimensional manifold space. It then measures the coverage of test inputs within a subset in the manifold space to assess test thoroughness [29]. The authors compared the misprediction-revealing effectiveness and retraining efficacy of MCC and state-of-the-art white-box coverage metrics. They found that the misprediction-revealing effectiveness of MCC is similar to that of the selected white-box coverage metrics. Further, MCC failed to outperform the white-box coverage metrics in retraining effectiveness. Given the reported performance of MCC compared to white-box coverage metrics, we did not consider this recent work in our empirical analysis.

Despite active research on DNN coverage, several recent articles have questioned the usefulness of coverage criteria to guide the testing of DNN models [34, 49, 100]. For example, Li *et al.* [100] studied a number of structural coverage criteria and discussed their limitations in fault detection capabilities in DNN models. Their experiments found no strong correlation between coverage and the number of misclassified inputs in a natural test set. Furthermore, Dong *et al.* [49] found that retraining DNN models with new input sets that improve coverage does not increase the robustness of the model to adversarial attacks.

Our work on diversity metrics is orthogonal to existing research regarding DNN test

coverage. Most of the state-of-the-art coverage metrics require full access to the internals of DNN state or training data, both of which are often not available to testers in practical contexts. Thus, in our approach we focused on black-box diversity metrics, and aimed to provide guidance to assess test suites or select test cases for DNNs.

State-of-the-art coverage criteria have been largely validated with artificial inputs that have been generated based on adversarial methods [65, 89, 104, 125, 141]. However, their relationship to (often unrealistic) adversarial inputs does not imply they relate to the fault detection capability of natural test input sets. Li *et al.* [100] argue that adversarial inputs are distributed pervasively over the divided space defined by existing coverage criteria. On the other hand, misclassified natural inputs are distributed sparsely, making their detection difficult when using such coverage criteria [100]. Existing studies [34, 100] have failed to find a significant correlation between coverage and the number of misclassified inputs in a test set. Consequently, coverage criteria may be ineffective at guiding DNN testing to increase the fault-detection capability of natural input sets. Furthermore, existing studies [65, 89, 104, 125, 141] have used the number of mispredicted inputs to study the effectiveness of coverage criteria to support DNN testing. However, as previously discussed, simply comparing mispredictions is highly misleading because many test inputs may (and usually do) fail due to the same causes in the DNN model. To address this problem, we approximated faults (i.e. common misprediction causes) by relying on a clustering strategy and by studying the correlation between test criteria (i.e. coverage and diversity) and faults instead of misprediction rates.

2.6.2 Black-box DNN Testing

In this section, we describe black-box testing approaches for DNN models because the focus of this chapter is on black-box metrics and diversity metrics, and on their association with detected faults in DNN models. Feng *et al.* [58] proposed DeepGini, a black-box test selection approach that prioritizes test inputs along with higher uncertainty scores. Intuitively, a test input is likely to be misclassified by a DNN if the model is uncertain about the classification and outputs similar probabilities for each class [58]. They found that DeepGini outperforms random and coverage-based test selection approaches to reveal misclassifications. However, this approach is only applicable to classification problems and cannot be used for regression tasks. A recent study by Gao *et al.* [63], developed concurrently to our work, proposed Adaptive Test Selection (ATS), a method based on uncertainty scores and distribution of the output probability vectors of test inputs in DNN models. The selection is guided by a fault pattern coverage score that is computed using the output probability vectors of the DNN under test. They introduce a mapping from the output domain of the DNN model to a set of intervals in local domains to describe the fault pattern of a given test input or test set. They select test inputs that both cover different fault patterns [63] and have higher uncertainty scores. Similar to DeepGini, this approach can only be used for classification problems. Empirical results show that ATS outperforms coverage and uncertainty-based test selection methods (including DeepGini [58]) in misprediction-revealing capability. They also studied the effectiveness of the proposed approach in finding diverse mispredicted inputs. Consequently, they introduced the concept

of fault types by looking at pairs of actual test input labels and labels predicted by the DNN model under test. Inputs that have different pairs of actual and predicted labels are assumed to correspond to different types of faults. However, as opposed to our work, this proposed method for counting distinct faults was not validated. We relied on a more fine-grained fault estimation approach that is based on clustering mispredicted inputs and accounts for their labels (actual and mispredicted) as well as their features.

Li *et al.* [101] proposed two black-box metrics called Cross Entropy-based Sampling (CES) and Confidence-based Stratified Sampling (CSS) for DNN test set minimization. These metrics are used to guide the selection of a small set of test inputs that can accurately estimate the entire testing dataset. The authors show that their approach outperforms random sampling and requires only about half the labeled test inputs to achieve the same level of accuracy as the whole testing set. The authors also report that CSS does not perform well on poorly trained DNNs because the confidence values it produces cannot be trusted [101].

These black-box testing approaches and their underlying metrics are conceptually different from our black-box diversity metrics. They are model-dependent because they rely on DNN outputs (output diversity) and uncertainty assessments. From a practical standpoint, this implies that all inputs must first be executed to be selected based on their diversity, which is a strong practical impediment in many application contexts, for example when working with large models and large databases of unlabeled inputs. Further, as mentioned above, these diversity metrics cannot be trusted when the model is not accurate. In contrast, our diversity metrics are model-independent and based on an analysis of the diversity of input features, thus requiring no model execution. Finally, these black-box testing approaches focus on specific testing scenarios, while our study is generic and focuses on investigating fundamental and pervasive assumptions related to the relationship between testing criteria and faults in DNNs.

2.6.3 Diversity in Testing

In this section, we describe existing work that relied on diversity to test DNNs and regular software.

Diversity in DNN Testing. A recent study by Langford and Cheng [95] proposed Enki, a DNN input-generation approach based on evolutionary search [51]. Their objective is to diversify image transformation types and to generate new inputs from existing ones to test and retrain DNN models. They started by evolving an archive of image transformation types that have a diverse impact on the DNN model. Given a subset of synthetic inputs generated with a certain image transformation type, the diversity of the impact was evaluated against three elements: (1) the F1-score of the DNN model when applied on the subset; (2) the neuron coverage score [125]; and (3) the neuron's activation pattern [95]. After building the final Enki archive containing the most diverse image transformation types, they (1) tested the DNN models using synthetic inputs generated with the identified image transformation types, and (2) studied the accuracy of the DNNs by retraining

them with such synthetic training data. They also compared their results with random input generation and DeepTest [144]. They concluded that Enki outperformed these two input generation approaches, and reported that testing DNNs with their generated data led to the lowest DNN model accuracy. They also reported that retraining DNNs with their generated data increased the accuracy of DNN models.

What differentiates our work from Enki is that Enki provides a search-based approach to diversify image transformation types. Its goal is to minimize the model accuracy and then use it to guide the generation of synthetic inputs to test and retrain DNNs. In contrast, our approach investigated ways to measure diversity in natural test input sets and compared the best diversity metric with state-of-the-art coverage criteria to guide DNN testing to maximize fault detection. Such diversity metrics can then be used for multiple purposes such as test suite assessment and guidance for selection, minimization and generation. Our focus on faults, as opposed to accuracy, aimed to find test inputs whose mispredictions resulted from distinct root causes. For practical reasons, as already discussed and as opposed to Enki, we intentionally devised an approach that is black-box and did not rely on internal information about the model or its training set.

Diversity in Software Testing. Input and output diversity have been investigated to support different aspects related to traditional software testing. Since executing similar test cases tends to exercise similar parts of the source code, this is likely to lead to revealing the same faults in the system under test. Therefore, relying on diverse test cases should increase the exploration of the fault space and thus increase fault detection rates [30,43,74].

Feldt *et al.* [56] proposed Test Set Diameter (TDSm), a diversity-based test case selection strategy. The approach uses the NCD metric to measure the diversity of test inputs. They applied their approach on four systems and concluded that diverse test input sets increase code coverage. Finally, they show that test sets with larger NCD scores exhibit better fault-detection capabilities.

Hemmati *et al.* [73] conducted an empirical study on similarity-based test selection techniques for test cases generated from state machine models. They studied and compared over 320 variants that relied on different similarity metrics and selection algorithms. Based on their experiments, they found that the best test-selection technique used the Gower-Legendre similarity function [157] and applied a (1+1) Evolutionary Algorithm [50] to select tests with minimum pairwise similarity and thus maximized the diversity of the selected test cases. They further showed that such similarity-based test selection configuration outperformed random selection and coverage-based techniques in fault detection rates and computational cost.

Biagiola *et al.* [19] introduced a web test generation algorithm that produces and selects candidate test cases that are executed in the browser based on their diversity. They showed that their test generation technique achieved higher code coverage and fault detection rates when compared to state-of-the-art, search-based web test generators [18, 110].

Our objectives in this chapter are similar to these studies, but in the context of DNN testing. As several studies have shown the effectiveness of diversity metrics in guiding the testing of software systems, we investigated its usefulness in testing DNN models. We

compared the performance of existing diversity metrics with state-of-the-art DNN coverage criteria in terms of their fault detection capabilities and computational cost.

2.7 Conclusion

In this chapter, we studied the effectiveness of input diversity metrics in guiding the testing of DNN models. We focused on DNN models using images as inputs, as they are common in many systems. Our motivation is to provide a black-box mechanism that does not rely on DNN internal information or training data to assess test sets. This requirement aims to make our approach more applicable in the many practical contexts where such information is not (easily) accessible. We also do not rely on output diversity because this approach would require executing the model with all inputs and would be affected by poor model accuracy. Further, we compare the results achieved by white-box coverage criteria defined for DNNs with those achieved by black-box input diversity.

To this end, we selected and adapted three input diversity metrics and, by means of a controlled experiment, evaluated their capability to measure actual input diversity. We selected the best metrics and analyzed their association with fault detection in DNNs using four datasets and five DNN models. Because simply comparing mispredictions is highly misleading, as many test inputs fail for the same reasons, we relied on a clustering-based approach to group similar mispredicted inputs and thus estimated faults based on the number of such clusters. We further selected the best state-of-the-art coverage criteria based on published results and our ability to reproduce such results. We studied the associations of the selected coverage criteria to both diversity and fault detection.

Based on our experiments, we found that the best diversity metric is geometric diversity and that, though there is still room for improvement (e.g. fault estimation in DNNs, investigating other diversity metrics and feature extraction models), it is a far better surrogate metric than the investigated coverage criteria in terms of their relationship to fault detection. This metric outperforms these coverage criteria in correlation to detected faults and computational time. We, therefore, recommend investigating the use of geometric diversity as a black-box metric to guide the testing of DNN models using images as inputs. We aim to extend our work by studying the application of input diversity in supporting different testing applications such as test set selection, minimization, and generation. We also intend to investigate alternatives for DNN fault estimation.

2.8 Appendix A

Correlation Results Between Geometric Diversity and Coverage Metrics

Note that the grey boxes in all the following tables refer to statistically significant correlations (p-value <= 0.05)

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Cifar-10	12-layer ConvNet	GD & LSC	100	13%	0.45
			200	-2%	0.92
			300	1%	0.97
			400	-10%	0.58
			1000	-12%	0.53
		GD & DSC	100	-20%	0.27
			200	-11%	0.55
			300	11%	0.49
			400	-17%	0.36
			1000	-7%	0.73
		GD & NC	100	-30%	0.09
			200	12%	0.49
			300	10%	0.56
			400	2%	0.90
			1000	2%	0.91
		GD & KMNC	100	**44%**	0.01
			200	15%	0.40
			300	15%	0.37
			400	9%	0.63
			1000	12%	0.54
		GD & NBC	100	31%	0.08
			200	6%	0.75
			300	17%	0.31
			400	-10%	0.62
			1000	-28%	0.14
		GD & TKNC	100	-19%	0.30
			200	17%	0.33
			300	-4%	0.83
			400	7%	0.69
			1000	7%	0.71
		GD & SNAC	100	**34%**	0.05
			200	6%	0.75
			300	13%	0.44
			400	-13%	0.47
			1000	-28%	0.14

Table 2.12: Correlation results between GD and coverage metrics using 12-layer ConvNet and Cifar-10.

Dataset	Model	Metric	Test Set Size	Spearman	P-value
MNIST	LeNet-1	GD & LSC	100	-6%	0.78
			200	-12%	0.57
			300	-6%	0.78
			400	15%	0.30
			1000	-4%	0.80
		GD & DSC	100	-27 %	0.17
			200	-10%	0.64
			300	-27%	0.17
			400	25%	0.09
			1000	-14%	0.41
		GD & NC	100	18%	0.38
			200	-15%	0.47
			300	3%	0.89
			400	-3%	0.85
			1000	-30%	0.08
		GD & KMNC	100	-15%	0.47
			200	-16%	0.45
			300	8%	0.68
			400	28%	0.06
			1000	6%	0.71
		GD & NBC	100	6%	0.78
			200	6%	0.79
			300	-9%	0.65
			400	15%	0.30
			1000	-15%	0.39
		GD & TKNC	100	7%	0.73
			200	11%	0.61
			300	29%	0.14
			400	21%	0.16
			1000	21%	0.23
		GD & SNAC	100	-6%	0.78
			200	7%	0.75
			300	-14%	0.47
			400	17%	0.26
			1000	-13%	0.45

Table 2.13: Correlation results between GD and coverage metrics using LeNet-1 and MNIST.

Dataset	Model	Metric	Test Set Size	Spearman	P-value
MNIST	LeNet-5	GD & LSC	100	3%	0.88
			200	25%	0.17
			300	26%	0.26
			400	-22%	0.23
			1000	-7%	0.74
		GD & DSC	100	13%	0.52
			200	-10%	0.59
			300	-22%	0.34
			400	5%	0.81
			1000	-1%	0.95
		GD & NC	100	-22%	0.28
			200	-11%	0.57
			300	-4%	0.86
			400	-34%	0.06
			1000	**39%**	0.05
		GD & KMNC	100	-1%	0.94
			200	20%	0.29
			300	4%	0.88
			400	5%	0.79
			1000	-31%	0.11
		GD & NBC	100	-27%	0.17
			200	-1%	0.95
			300	-4%	0.86
			400	-14%	0.46
			1000	**48%**	0.01
		GD & TKNC	100	4 %	0.84
			200	24%	0.18
			300	-17%	0.46
			400	-2%	0.90
			1000	3%	0.90
		GD & SNAC	100	-18%	0.38
			200	-1%	0.95
			300	-5%	0.81
			400	-21%	0.25
			1000	**41%**	0.03

Table 2.14: Correlation results between GD and coverage metrics using LeNet-5 and MNIST.

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Fashion-MNIST	LeNet-4	GD & LSC	100	0.2%	0.99
			200	28%	0.17
			300	19%	0.29
			400	21%	0.20
			1000	2%	0.94
		GD & DSC	100	17%	0.38
			200	-7%	0.73
			300	**-35%**	0.05
			400	-2%	0.90
			1000	2%	0.93
		GD & NC	100	**-37%**	0.04
			200	0.3%	0.99
			300	-5%	0.77
			400	23%	0.16
			1000	-14%	0.49
		GD & KMNC	100	23 %	0.22
			200	30%	0.14
			300	33%	0.06
			400	**64%**	0.0001
			1000	6%	0.76
		GD & NBC	100	-25 %	0.19
			200	-31%	0.13
			300	12%	0.51
			400	-22%	0.19
			1000	-21%	0.31
		GD & TKNC	100	-1 %	0.95
			200	**41%**	0.04
			300	-3%	0.85
			400	24%	0.15
			1000	11%	0.61
		GD & SNAC	100	-25%	0.19
			200	-31%	0.13
			300	12%	0.52
			400	-21%	0.20
			1000	-19%	0.35

Table 2.15: Correlation results between GD and coverage metrics using LeNet-4 and Fashion-MNIST.

Dataset	Model	Metric	Test Set Size	Spearman	P-value
Cifar-10	ResNet-20	GD & LSC	100	19 %	0.31
			200	14%	0.55
			300	39%	0.11
			400	**46%**	0.02
			1000	-26%	0.18
		GD & DSC	100	4 %	0.85
			200	-8%	0.74
			300	5%	0.84
			400	-10%	0.65
			1000	-3%	0.86
		GD & NC	100	5 %	0.81
			200	26%	0.25
			300	-2%	0.94
			400	12%	0.58
			1000	**-38%**	0.04
		GD & KMNC	100	19 %	0.3
			200	39%	0.08
			300	17%	0.51
			400	**41%**	0.05
			1000	28%	0.15
		GD & NBC	100	24 %	0.20
			200	12%	0.60
			300	-4%	0.87
			400	-9%	0.67
			1000	-18%	0.35
		GD & TKNC	100	2 %	0.9
			200	18%	0.43
			300	10%	0.70
			400	**58%**	0.003
			1000	-7%	0.73
		GD & SNAC	100	17 %	0.35
			200	11 %	0.62
			300	4 %	0.89
			400	-3 %	0.90
			1000	-14 %	0.47

Table 2.16: Correlation results between GD and coverage metrics using ResNet-20 and Cifar-10.

Dataset	Model	Metric	Test Set Size	Spearman	P-value
SVHN	LeNet-5	GD & LSC	100	15%	0.37
			200	27%	0.09
			300	2%	0.91
			400	15%	0.46
			1000	34%	0.06
		GD & DSC	100	-16 %	0.33
			200	9%	0.58
			300	19%	0.28
			400	14%	0.47
			1000	-14%	0.45
		GD & NC	100	-8%	0.61
			200	4%	0.79
			300	30%	0.08
			400	-28%	0.14
			1000	-12%	0.52
		GD & KMNC	100	**44%**	0.001
			200	**66%**	0.002
			300	23%	0.19
			400	-2%	0.9
			1000	**46%**	0.01
		GD & NBC	100	9%	0.58
			200	5%	0.77
			300	-19%	0.27
			400	26%	0.18
			1000	28%	0.12
		GD & TKNC	100	-3%	0.83
			200	1%	0.95
			300	-22%	0.20
			400	-15%	0.43
			1000	18%	0.34
		GD & SNAC	100	9%	0.58
			200	5%	0.77
			300	-19%	0.27
			400	26%	0.18
			1000	28%	0.12

Table 2.17: Correlation results between GD and coverage metrics using LeNet-5 and SVHN.

Chapter 3

Selecting test inputs with high fault-revealing power

3.1 Overview

This chapter addresses TO_2, i.e., black-box selection of test inputs with high fault-revealing power. The content of this chapter has been published in *ACM Transactions on Software Engineering and Methodology (TOSEM 2024)* [4] under the title of *DeepGD: A Multi-Objective Black-Box Test Selection Approach for Deep Neural Networks*.

As we discussed in chapter 1, for testing and enhancing the performance of DNN-driven applications, a significant amount of labeled data is required. In many real-world scenarios, there are instances where the availability of labeled data for testing is either limited or absent. This situation becomes particularly pertinent when the trained and tested model is deployed in different operational settings and necessitates further evaluation using operational or real data, typically large and unlabeled. The same challenge extends to the testing of third-party pre-trained DNN models for example, where access to internal model information or training data is typically unavailable. In such cases, these models are commonly subjected to further testing using end-users' data, which is, in general, massive and unlabeled.

However, obtaining the correct output labels for a large amount of unlabeled data, referred to as oracle information, can be a challenging and resource-intensive task [33]. This process often requires extensive manual effort by domain experts and can be a significant challenge when the volume of unlabeled data is large and resources are limited. For instance, in many real-world situations, DNN testing is usually bounded by a limited testing budget, especially when testing is performed on costly simulators such as flight simulators or real hardware (e.g., self-driving cars, medical equipment). These execution environments can be expensive and require substantial computing resources, which in turn limits the number of tests that can be performed [172]. This makes it more difficult to effectively test and improve the performance of DNN models. In order to address these challenges and make DNN testing feasible and cost-effective in practice, it is essential to

select a small subset of test inputs that possess a high fault-revealing power. This approach reduces the number of inputs required for DNN testing, making it more cost-effective.

Several test selection approaches for DNN models have been proposed in recent years [63, 80,89,104,105,125,141,144]. Some of them are white-box since they require access to the internals of the DNN models or the training datasets. Similar to code coverage in traditional software systems [72], researchers have proposed several neuron coverage metrics in the context of DNN test selection to prioritize the selection of test inputs with high coverage scores [89, 104, 125, 141, 144]. Studies have shown the effectiveness of these approaches for distinguishing adversarial inputs but failed to demonstrate a positive correlation between coverage and DNN mispredictions [2,34,100,161]. Furthermore, for some coverage metrics, reaching maximum coverage can be easily achieved by selecting a few test inputs [34, 140], thus putting into question the usefulness of such white-box DNN test selection approaches. Last, the use of white-box DNN test selection approaches is hindered by the requirement for access to the internal structure of the DNN model or the training dataset. This can be a significant limitation, particularly when the model is proprietary or provided by a third party [2].

To alleviate this latter problem, black-box test selection approaches have been proposed, which do not require access to the internal information of the DNN model under test or its training set.

They generally rely on DNN output uncertainty metrics to guide the selection of test inputs. It has been observed that a test input is likely to be mispredicted by a DNN if the model is uncertain about its output classification, when similar probabilities are predicted for each class [58]. An empirical experiment conducted by Ma *et al.* [105] revealed that these metrics have medium to strong correlations with mispredicted inputs, while coverage-based metrics have weak or no correlation. One issue though is that if the model is not sufficiently trained, outputs cannot be trusted, thus leading to unreliable uncertainty results and test case selection [101].

Another black-box strategy that has been successfully applied in testing traditional software systems is to maximize diversity among test cases [19, 56, 74]. However, only a few studies have investigated its effectiveness in testing DNN models [2, 63, 167]. Zhao *et al.* [167] studied state-of-the-art (SOTA) test selection approaches for DNNs and found that they do not guarantee the selection of diverse test input sets. Furthermore, in chapter 2, we compared the fault-revealing power of several diversity metrics with coverage criteria [2]. We found that geometric diversity (GD) [94] outperforms coverage criteria in terms of fault-revealing power and computational time. However, we did not investigate how such a metric can be used for DNN test selection.

In this chapter, we propose *DeepGD*(Deep Geometric Diversity), a black-box multi-objective search-based test selection approach. It relies on the Non-dominated Sorting Genetic Algorithm (NSGA-II) [46] to select test inputs that aim at revealing a maximal number of diverse faults in DNN models within a limited testing budget. The latter is mostly determined by test input labeling effort and, for larger models, their execution time. *DeepGD* can therefore be used to select a subset of test inputs to label from a large unlabeled dataset. The strategy of *DeepGD* is twofold: (1) trigger as many mispredicted

inputs as possible by selecting inputs with high uncertainty scores, and (2) maximize the probability of revealing distinct faults in the DNN model by selecting diverse mispredicted inputs.

Similar to failures in traditional software systems, many mispredictions result from the same faults (causes) in the deep learning model and are therefore redundant [2, 55, 171]. Misprediction rates[1] can therefore be misleading when assessing a test selection approach, if a large number of mispredictions stem from the same fault. This is also why, for similar reasons, traditional software testing studies typically focus on faults, not failures [69, 120–122]. Therefore, to validate the effectiveness of *DeepGD*, we use a fault estimation approach proposed in chapter 2, which is based on clustering mispredicted inputs based on their features [2]. Each cluster represents a distinct fault since it contains similar inputs that are mispredicted due to the same root causes.

More specifically, we present an empirical evaluation of the effectiveness of *DeepGD* for selecting test input sets with high fault-revealing power. Since our fault estimation approach is tailored to image datasets and relies on clustering mispredicted inputs according to image features, we conduct our experiments on image classification models. They include five widely-used DNN models and four image recognition datasets. The results are compared with nine SOTA baseline selection approaches, both white-box and black-box, that have been recently published [58, 63, 89, 104, 105, 125].

The experimental results demonstrate that *DeepGD* yields a statistically significant and consistent improvement compared to SOTA approaches in terms of its ability to reveal DNNs faults and optimize the models. Thus, it can be effective in selecting test input sets with high fault-revealing capabilities. Specifically, results show that *DeepGD* is consistently the best approach, up to 4 percentage points (pp) better than the second-best alternative and 14 pp better than the worst black-box alternative, when excluding random selection (RS) since RS showed far inferior results in general. It is important to note that the ranking of alternatives varies across datasets, models, and test set sizes. Consequently, selecting any approach other than DeepGD may end up being the worst choice and lead to significant differences in performance. Further, we also investigate the effectiveness of *DeepGD* in guiding the retraining of DNNs and demonstrate that it consistently provides better results with that respect as well. *DeepGD* is therefore the only technique we can confidently recommend.

To summarize, the key contributions of this chapter are as follows:

- We propose a black-box test selection approach (*DeepGD*) for DNNs that relies on a customized multi-objective genetic search and uses both diversity and uncertainty scores to guide the search toward finding test input sets with high fault-revealing power.
- Unlike existing test selection approaches, we consider in our validation a clustering-based approach to estimate faults in DNN models since test input sets typically contain many similar mispredicted inputs caused by the same problems (faults) in

[1]Misprediction rate = 1 - Accuracy

the model [2, 55]. We thus obtain more meaningful evaluation results, similar to common practice in testing research.

- We conduct a large-scale empirical evaluation to validate *DeepGD* by considering five DNN models, four different datasets, and nine SOTA test selection approaches for DNNs as baselines of comparison. We show that *DeepGD* provides better guidance than baselines for (1) selecting inputs with high fault-revealing power, and (2) improving the performance of the model through retraining based on an augmented training set.

- We study the diversity of inputs selected by *DeepGD* and SOTA baselines to assess whether our approach not only leads to test inputs with higher fault-revealing capabilities but also promotes greater diversity in the selected test input sets. We thus confirm that *DeepGD* indeed selects more diverse input sets compared to baselines.

- We analyze the execution time of *DeepGD* and other baselines and show that they are not computationally expensive. Though *DeepGD* exhibits longer execution times, these remain practical and will have little impact in practice. *DeepGD*'s improved fault detection and retraining guidance more than compensate for them.

Chapter Structure The remainder of the chapter is structured as follows. Section 3.2 presents our test selection approach. Section 3.3 presents our empirical evaluation. Section 3.4 outlines our results. Section 3.5 describes the threats to the validity of our study. Sections 3.6 and 3.7 contrast our work with related work and conclude this chapter, respectively.

3.2 Approach: Reformulation as an NSGA-II Search Problem

A central problem in testing DNNs, especially when the labeling of test data is costly, is the selection of a small set of test inputs with high fault revealing power. In this chapter, we aim to support the testing of DNN models by relying on *DeepGD*, a black-box search-based test selection approach using a genetic algorithm to select small sets of test inputs with high fault-revealing power in DNN models. Intuitively, testers should select a set of diverse test inputs with high failure probabilities in order to be more likely to detect as many diverse faults as possible [170]. Due to the combined high labeling cost of test inputs and large input space, we rely on NSGA-II to select test inputs with high fault-revealing capability. Such inputs are then selected for labeling and will be used to effectively test the DNN model. We choose NSGA-II since it is widely used in the literature and showed its performance in solving many search-based test selection problems [73,149]. We also rely on NSGA-II since it is specifically adapted to our multi-objective search problem. It tries to find solutions with diverse trade-offs between fitness functions instead of covering all fitness functions separately (which is the case of MOSA, the Many Objective Sorting Algorithm [123] for example). Specifically, the search is driven by two objectives: (1) maximizing the

uncertainty score of the test inputs to trigger a maximum number of mispredictions, and (2) maximizing the diversity of the test input set to trigger diverse mispredictions caused by distinct faults.

To properly translate the process into a search problem using NSGA-II, we need to define the following elements.

3.2.1 Individuals and Initial Population

In genetic algorithms, individuals consist of a set of elements called genes. These genes are connected and form an individual that is also called a solution. In our approach, we assign a unique identifier id_i to each input i in the test dataset where $id_{1 \leq i \leq n} \in [1, 2, .., n]$ and n is the size of the test dataset. Our test selection problem has a fixed testing budget $\beta < n$ which corresponds to the total number of inputs selected from the original test dataset to test the DNN model. In our search problem, an individual corresponds to a subset of inputs of size β. Each gene forming an individual corresponds to an id of a test input in the test dataset. In our context, each individual contains β distinct test inputs. We use random selection to build our initial population of individuals.

3.2.2 Fitness Functions

The main objective of our approach is to select unlabeled inputs that possess a strong ability to reveal diverse faults. These faults manifest themselves as diverse mispredicted inputs. To generate the latter, our search process is guided by two fitness functions. The first function aims to maximize the uncertainty score of the selected inputs, as it is moderately to strongly correlated with mispredictions [58, 63, 105, 150]. By maximizing uncertainty scores, we are thus more likely to trigger a larger number of mispredicted inputs [105]. The second function aims to maximize the diversity among the selected inputs. This approach increases the probability of identifying distinct faults. By simultaneously maximizing these two fitness functions, we aim to select a diverse set of mispredicted inputs that are due to distinct faults, thereby maximizing the fault-revealing power of the selected test input set.

We will describe next the two fitness functions and detail how to compute them.

3.2.2.1 Gini Score

We consider the *Gini* score to estimate the likelihood of a test input being mispredicted. Feng *et al.* [58] proposed this metric to measure the classification uncertainty of DNN models and therefore identify potential failing test inputs. Intuitively, a test input is likely to be misclassified by a DNN if the model is uncertain about the classification and outputs similar probabilities for each class [58]. We choose this metric since it has been widely used in the literature, showed good performance in prioritizing test inputs for DNN models, and has a medium to strong correlation to misprediction rates [58, 63, 105, 150]. It is also a black-box metric that only requires the output probabilities of DNN models as

we will describe in the following. Given a test input x and a DNN model that outputs $DNN(x) =< P_{x_1}, P_{x_2}, ..., P_{x_m} >$, where $P_{x_{i\in[1,..,m]}}$ is the probability that input x belongs to class C_i and m is the total number of classes, the *Gini* score of the test input x is defined as:

$$\xi(x) = 1 - \sum_{i=1}^{m} P_{x_i}^2 \tag{3.1}$$

The higher the *Gini* score, the higher the DNN's uncertainty. We compute the *Gini* score of a subset $S = \{s_1, s_2, ..., s_\beta\}$ of size β by computing the average *Gini* score of all inputs in the subset as follow:

$$Gini(S) = \frac{\sum_{i=1}^{\beta} \xi(s_i)}{\beta} \tag{3.2}$$

3.2.2.2 Geometric Diversity

We consider geometric diversity, one of the widely used metrics to measure the diversity of test input sets [2,66,94]. Also, in the previous chapter, chapter 2, several coverage and diversity metrics were investigated and results showed that the GD metric is positively correlated to DNN faults and outperforms SOTA coverage metrics in terms of fault-revealing capabilities [2]. In other words, when the geometric diversity of a test input set increases, its fault-revealing power increases as well since the diverse test input set will cover more faults in the DNN model. Furthermore, GD is a black-box diversity metric that requires neither knowledge about the model under test nor access to the training set. Consequently, we have relied on this metric as a fitness function to guide the search towards finding a diverse test input set with high fault-revealing capability.

Feature Extraction. In order for diversity to account for the content of images, we need to first extract features from each input image and then compute the diversity based on those extracted features. For this purpose, we follow the same feature extraction steps mentioned in *Feature Extraction For Diversity (subsection 2.2.1)*

We extract features of each image in the test input set S using VGG-16 and generate the corresponding feature matrix $F = (f_{ij}) \in R^{n*m}$ where n is the number of input images in S, and m is the number of features. Each row of this matrix represents the feature vector of an image, and each column (F_j) represents a feature. It is worth mentioning that, in order to ensure the compatibility of certain image datasets with VGG-16, specific image pre-processing techniques are applied before feature extraction. These techniques involve operations such as image resizing and color channel duplication. Similar to the process mentioned in *Feature Extraction For Diversity (subsection 2.2.1)*, after extracting the feature matrix, we normalize it as a pre-processing step to eliminate the dominance effect of features with large value ranges and to make the computation of the selected diversity metrics more scalable [2]. For every feature F_j in the feature matrix F where $F_j(i)$ is the value of feature number j for i^{th} input image in S, the normalized feature $F\prime_j$

is calculated as follows:

$$F\prime_j(i) = \frac{F_j(i) - min(F_j)}{max(F_j) - min(F_j)} \tag{3.3}$$

Computation of Geometric Diversity Scores. After extracting the feature vectors, we calculate the geometric diversity of the test inputs as our second fitness function with the goal of selecting as many diverse test input sets as possible. Given a dataset $\mathbb{D}$, the normalized feature matrix $F\prime$ where $F\prime_S$ represents feature vectors of a subset $S \subseteq \mathbb{D}$, the geometric diversity of S is defined as:

$$GD(S) = det(F'_S * F'^T_S) \tag{3.4}$$

which corresponds to the squared volume of the parallelepiped spanned by the rows of F'_S.

3.2.2.3 Multi-Objective Search

We need to maximize in our search the two aforementioned fitness functions to increase the fault-revealing capability of the selected test input set. This is therefore a multi-objective search problem that can be formalized as follows:

$$\max_{S \subset \mathbb{D}} \quad Fitness(S) = (Gini(S), GD(S)) \tag{3.5}$$

where $\mathbb{D}$ is the test dataset, S is a subset of test inputs of size β, function $Fitness : S \rightarrow \mathbf{R}^2$ consists of two real-value objective functions $(Gini(S), GD(S))$, and $\mathbf{R}^2$ is the objective space of our optimization problem. Given our two objectives, the "max" operator in Eq. 3.5 returns non-dominated solutions forming a Pareto front in the space defined by Gini and GD [46], i.e., solutions where at least one of the objective function values is the maximum and the other remains unsurpassed by other solutions.

3.2.3 Genetic Operators

We describe below the two genetic operators in our test selection approach. The first operator is crossover, which generates new offspring by cutting and joining high-fitness parents. The second operator is mutation which introduces small changes in individuals by mutating specific genes to (1) make the search more exploratory and (2) thus attempt to increase the uncertainty score and the diversity among individuals in the population. We provide below a detailed description of how we customized these genetic operators to fit our test selection problem.

3.2.3.1 Crossover

The crossover operator takes as input two parents (i.e., two subsets) and generates new offspring by slicing and joining the different parts of the selected parents. Let S_1 and S_2 be the two selected parents for crossover. We provide next an example of selected parents:

$$S_1 = \{Input_1, Input_2, Input_3, \cdots, Input_i^{s_1}, \cdots, Input_4, Input_5\}$$

$$S_2 = \{Input_6, Input_7, Input_8, \cdots, Input_i^{s_2}, \cdots, Input_9, Input_{10}\}$$

where (1) $Input_i^{s_1}$ and $Input_i^{s_2}$ denote the i^{th} input in S_1 and S_2, respectively and (2) $Input_{1 \leq j \leq 10}$ denote all inputs in each individual. Before applying the crossover operator, we start by sorting the inputs forming each parent according to their *Gini* scores. Inputs with higher *Gini* scores are placed at the beginning of each corresponding parent, as in the example below.

$$S_1 = \{Input_3, Input_1, Input_5, \cdots, Input_i^{s_1}, \cdots, Input_4, Input_2\}$$

$$S_2 = \{Input_7, Input_{10}, Input_6, \cdots, Input_i^{s_2}, \cdots, Input_9, Input_8\}$$

Such reordering will help with the creation of potential high-fitness offspring with high uncertainty scores as we will explain in the following. After such sorting, we randomly select a crossover point using the uniform distribution. A crossover point is randomly selected from the range between the second element and the $n-1$ position of the array (where n is the size of the array). To form the first offspring, we slice and join the first parts of each parent based on the crossover point. Such offspring includes inputs with the highest *Gini* scores thanks to sorting. Finally, the second offspring is formed by joining the remaining parts of the selected parents. Since this offspring has inputs with the lowest *Gini* scores from both parents, it will be potentially discarded later by the selection operator or improved by the mutation operator as we will detail in the next section. Assuming that the crossover point is at position i in the example above, the generated offspring would be:

$$Offspring_1 = \{Input_3, Input_1, \cdots, Input_i^{s_1}, Input_7, Input_{10}, \cdots, Input_{n-i}^{s_2}\}$$

$$Offspring_2 = \{Input_{i+1}^{s_1}, \cdots, Input_4, Input_2, Input_{n-i+1}^{s_2}, \cdots, Input_9, Input_8\}$$

However, applying the crossover on the selected parents may lead to redundant inputs in the created offspring. Since one of our search goals is to maximize the diversity of the selected subsets, we remove such redundant inputs (and therefore increase the diversity of the offspring) by replacing them with random inputs that are not present in the created offspring. We should note that we do not use the GD metric to customize the crossover operator since it is computationally expensive. Instead, as described in the next section, we employ it in the mutation operator since mutations occur less frequently.

3.2.3.2 Mutation

After completing the crossover, the offspring are selected for mutation to randomly change some genes that have (1) a low *Gini* score, and (2) a low contribution to the diversity of the selected offspring. Therefore, the mutation operator is considered in our search approach as a corrective operator with the goal of improving the offspring in terms of both fitness functions. As an example, let us assume that S_m is the test input set to mutate. We select 2% of the inputs from S_m that have the lowest *Gini* score. We mutate half of these inputs that have the least contribution to the diversity of S_m. Consequently, only 1% of its genes are mutated. This choice was inspired by recommendations for multi-point mutations in the literature [106, 146, 164]. We should recall that to achieve our desired final mutation rate of 1%, we initially select 2% of the genes. We compute the contribution to the diversity of the latter and retain the 1% of the genes that show the highest similarity. This allows us to contain the high cost of computing diversity associated with all genes while focusing mutation on genes contributing the least to diversity.

To measure the contribution of an $input_i$ to increasing the diversity in S_m, we measure the difference $GD_{diff}(S_m, i)$ between $GD(S_m)$ and $GD(S_m \setminus \{input_i\})$. The lower the difference, the more similar the input compared to the other ones in S_m. Inputs with low differences should therefore be mutated to accelerate our search process.

An example of offspring generated through mutation is provided below. S_m is the original offspring and $S_{m'}$ is the mutated one. Suppose that S_m has 10 genes. We mutate $Input_1$ since it has a low *Gini* score and a low contribution to the diversity of the selected offspring. We mutate this gene by replacing it with a randomly selected input from the population. In this example, we replace $Input_1$ with $Input_{13}$. To maintain diversity, we make sure that the randomly selected input is distinct from any existing input in the original offspring.

S_m :	$Input_3$	**Input$_1$**	$Input_5$	$\cdots$	$Input_7$	$Input_{10}$
Gini Score:	**70%**	**71%**	75%	$\cdots$	95%	96%
GD_{diff}*:*	60	**10**	$\cdots$			

$S_{m'}$:	$Input_3$	**Input$_{13}$**	$Input_5$	$\cdots$	$Input_7$	$Input_{10}$

It should be noted that each genetic operator has a distinct emphasis on a particular fitness function. The crossover operator primarily aims to generate offspring with high uncertainty scores, while the mutation operator also helps improve the offspring in terms of diversity. Crossover and mutation operators therefore complement each other.

3.2.4 Search Algorithm

As explained earlier, the main objective of our search algorithm is to select from a large unlabeled test dataset, an input set of size β with a high fault-revealing power in order to (1) reduce labeling cost, and (2) effectively test the DNN model. Algorithm 2 describes at

Algorithm 2: High-level NSGA-II algorithm

Input : The initial population (P) that is a set of test subsets, the number of generations (G), the crossover rate ($CrossRate$), the mutation rate ($MutationRate$)

Output: A set of individuals (α) maximizing the fitness function ($Fitness$)

```
1  gen ← 0
2  while gen ≤ G do
3      P_new ← ∅
4      P_Cross ← ∅
5      P_Cross ← Cross(P, CrossRate)
6      P_new ← Mutate(P_Cross, MutationRate)
7      P ← Select(P, P_new, Fitness)
8      Update(α)
9      gen ← gen + 1
10 return α
```

a high level our NSGA-II search process. Assuming P is the initial population, that is a set of $|P|$ randomly selected test input sets, the algorithm starts an iterative process, taking the following actions at each generation until the maximum number of generations (G) is reached (lines 2-10). The search process includes the following steps. First, we create a new empty population P_{new} (line 3). Second, based on predefined crossover and mutation rates, we create offspring using crossover ($Cross$) and mutation ($Mutate$) operators and add newly created individuals to P_{new} (lines 4-6). Third, we calculate the fitness of the new population by computing *Gini* and geometric diversity scores of each subset in the population and select individuals for survival using tournament [112] selection ($Select$) (line 7). Finally, we update the archive α (line 8) by storing the fittest individuals in the population then we move to the next generation (line 9). The non-dominated solutions contained in the population of the last generation α represent the final *Pareto* front of the genetic search. To select the final solution from the Pareto front, we use the knee point technique [25] as recommended by several existing studies [111]. The solution that corresponds to the knee point has been shown to provide the best trade-off between the different objectives in many multi-objective problems [25, 111]. To calculate the knee point, we first identify the *ideal point*, where the coordinates represent the highest fitness scores (GD_{max},$Gini_{max}$) obtained from all solutions on the Pareto front, with each fitness function evaluated independently. Given the solutions at the Pareto front $P = \{S_1, ..., S_p\}$, the knee point $S_k \in P$ is the solution that minimizes the distance $\sqrt{(GD_{max}–GD(S_i))^2 + (Gini_{max}–Gini(S_i))^2}$, for all $S_i \in P$.

We set the NSGA-II parameters in our search algorithm as follows. The crossover rate $CrossRate$ is equal to 75%, given recommended values between 45% and 95% [27, 39, 111]. The mutation rate $MutationRate$ is equal to 70%. Although the recommended mutation rate in the literature is proportional to the length of the individual [27, 39, 111], we have not used such a rate since we used mutation for a different purpose than just exploration, i.e., to help increase both uncertainty and diversity scores. According to our preliminary experiments, we found that smaller mutation rates, consistent with literature recommendations,

led to poorer search results. The population size is set to 700 individuals. Furthermore, the stopping criterion is set to 300 generations. We should note that the number of generations was determined through empirical evaluation. This was done by monitoring the evolution of fitness functions over multiple generations using various datasets and models. The maximum number of generations was carefully selected to ensure the convergence of the search process. Finally, we used Google Colab and the Pymoo library [20] to implement our genetic search.

3.3 Empirical Evaluation

This section describes the empirical evaluation of *DeepGD*, including the research questions we address, the datasets and DNN models on which we perform our assessments, our experiments, and our results.

3.3.1 Research Questions

Our empirical evaluation is designed to answer the following research questions.

RQ1. Do we find more faults than existing test selection approaches with the same testing budget? Similar to traditional software testing, selecting a subset of test inputs with high fault-revealing ability is highly important in DNN testing, as it should increase the effectiveness of the testing process while reducing input labeling costs. Consequently, we aim in this research question to compare the effectiveness of our approach (*DeepGD*) with existing baselines in terms of their ability to reveal faults in DNNs, while considering the same testing budget. Identifying the source of failures in traditional software systems is relatively straightforward due to the clear and explicit decision logic in the code. However, in DNNs, this task is more challenging as the complexity and non-linearity of the decision-making process make it difficult to determine the cause of the failure. Hence, many papers rely on mispredictions [58,89,125] for test selection evaluation. However, similar to failures in traditional software systems, many mispredicted inputs can be due to the same faults in the DNN model and are therefore redundant [12,55]. When selecting inputs on a limited budget, we should therefore avoid similar or redundant mispredictions as they do not help reveal additional root causes or faults in DNN models. To accurately answer this research question, we thus rely on a clustering-based fault estimation approach [2] that we describe in section 2.3.3.1, to investigate the effectiveness of the test selection approaches in detecting faults for a fixed testing budget.

RQ2. Do we more effectively guide the retraining of DNN models with our selected inputs than with baselines? Retraining DNN models with unseen inputs carefully selected based on DNN testing results is expected to enhance the model's performance compared to that of the original training set. More specifically, it is highly recommended to retrain DNNs with inputs that have the potential to lead to mispredictions [12, 58, 63].

Consequently, we aim in this research question to investigate the effectiveness of *DeepGD* in guiding the retraining of DNN models by selecting inputs that will be labeled and used to improve the model through retraining. We will not only measure the accuracy improvement resulting from retraining but also analyze it considering the maximum improvement possible based on the available data for retraining.

RQ3. Can *DeepGD* select more diverse test input sets? We aim to assess whether *DeepGD* selects more diverse test input sets compared to test selection baselines. Diversity in test input selection is important since it is correlated to faults as we reported in a prior study [2]. Specifically, more diverse test input sets reveal more faults in DNN models. Moreover, selecting diverse test inputs mitigates the risk of bias by ensuring that the testing process is not disproportionately skewed towards specific test cases or testing scenarios. It also prevents the detection of redundant faults in DNN models and thus helps optimize the testing budget and effort, since we are more likely to discover unique faults rather than repeatedly identify the same problems in the DNN model under test.

RQ4. How do *DeepGD* and baseline approaches compare in terms of computation time? We aim to compare the computation times of *DeepGD* and baseline approaches. Specifically, we aim to examine how the computational times change as the size of the test input sets increases. It is essential to understand this scalability factor, as excessively long computation times could potentially limit the practicality and real-world applicability of the studied test selection approaches.

3.3.2 Faults Estimation in DNNs

Before addressing our research questions, we will touch upon the issue of counting faults in DNN models. The motivation for counting faults, when evaluating selection techniques, stems from our objective of detecting different root causes for mispredictions. Considering misprediction rates to compare test selection techniques is highly misleading as many test inputs can be mispredicted for the same reasons [2, 55].

According to Chan *et al.* [35], in traditional software testing, failure-causing inputs tend to be dense and close together. The same insight applies to DNN model testing, since similar mispredicted inputs tend to be due to the same fault [2, 12].

The above principles should not be different when testing DNNs, where we aim to identify distinct causes of mispredictions (i.e., faults) [2] and address them. As we discussed in the previous chapter, subsection 2.3.3, a test set that consistently highlights the same faults in a DNN model is a waste of computational resources, particularly when faced with constraints such as limited testing budgets and high labeling costs for testing data [2,170]. Therefore, akin to previous research on test selection in traditional software, our objective is not focused on maximizing the occurrence of failures (i.e., mispredictions) but rather to develop a test selection method that maximizes the identification of faults present in the DNN model.

Given the inherent complexity of detecting faults in DNNs, which is not as straightforward as identifying specific statements responsible for failures in traditional software, we rely on prior work for estimating faults in DNN models in chapter 2, subsection 2.3.3. We cluster similar mispredicted inputs that exhibit common characteristics likely responsible for mispredictions. We then approximate the number of detected faults in the DNN model by counting the total number of clusters. Hence, although it is an approximation, this approach offers a practical and plausible method for estimating and comparing the number of detected faults across various test selection methods.

To investigate the effectiveness of test selection approaches in a DNN and for the reasons mentioned above, we do not rely on misprediction rates as it is highly misleading. We rely instead on the fault detection rate (FDR) which we compute as follows:

$$FDR(S) = \frac{|F_s|}{min(|S|, |F|)} \tag{3.6}$$

where S is the selected test input set, $|F_s|$ is the number of faults revealed by S, $|S|$ is the test input set size, and $|F|$ is the total number of faults revealed by the entire dataset. The goal of our FDR metric is to measure the extent to which a test set can detect the maximum number of faults that can be found given a testing budget. This is why we compute the ratio of the number of faults actually revealed by the selected test set to the maximum potential number of faults that the subset could unveil. This potential is naturally capped by the size of the subset (i.e., testing budget) or the total number of faults in the whole dataset. We should note that testers cannot know in advance the total number of faults in the dataset, and testing budgets are determined by the resources available. Our definition of FDR ensures that its range is always between 0 and 1, enabling us to compare the outcomes of various methods across test subsets. We present two illustrative examples of how we calculate the FDR using the test selection scenario depicted in Figure 2.2. Given a test dataset that reveals a total number of faults $|F|$ equal to four, a test input set size $|S|$ equal to 10, and a selected subset (see *Subset2*) that reveals four mispredicted inputs each belonging to a distinct fault (cluster), in this case the $FDR(S)$ is calculated as follows: $FDR(S) = \frac{4}{min(10,4)} = 100\%$.

Now, considering the same test dataset but selecting only three inputs ($|S| = 3$), which include two mispredicted inputs originating from the same fault ($|F_s| = 1$), the $FDR(S)$ is calculated as: $FDR(S) = \frac{1}{\min(3,4)} = 33\%$. The above examples illustrate that FDR represents the percentage of faults that can be detected in a subset relative to the maximum number of faults that can be detected, accounting for both the subset size and the total number of faults in the entire dataset.

3.3.3 Subject Datasets and Models

Table 3.1 shows the combinations of datasets and models used in our experiments. For our large-scale empirical evaluation on image classification systems, we consider a set of four well-known and publicly available image recognition datasets which are MNIST [47], Cifar-10 [8], Fashion-MNIST [47] and SVHN [119].

We use these datasets with five state-of-the-art DNN models: LeNet1, LeNet4, LeNet5, ResNet20, and a 12-layer Convolutional Neural Network (12-layer ConvNet). Because we compare our results with SOTA baseline approaches [58, 63, 65, 89, 104, 105, 125], we used similar combinations of models and datasets, focusing on the most widely used ones. Further, these models and datasets involve a variety of distinct inputs (in terms of classes and domain concepts) and different internal architectures, and thus constitute a good benchmark for observing key trends in DNN test selection. We provide the accuracy and the number of faults of the used models in Table 3.1. We should note that in our study, we intentionally trained some of our DNNs in a slightly suboptimal manner to make room for improvement in the retraining experiments conducted in RQ2. The accuracy levels achieved in our models are nonetheless consistent with existing studies on test selection for DNN models [63].

Table 3.1: Datasets and models used for evaluation

Dataset	**DNN Model**	**Accuracy**	**# Faults**
MNIST	LeNet5	87.85%	85
	LeNet1	84.5%	137
Fashion-MNIST	LeNet4	88%	141
Cifar-10	12-layer ConvNet	82.93%	187
	ResNet20	86%	177
SVHN	LeNet5	88%	147

3.3.4 Baseline Approaches

We compare *DeepGD* with two categories of baseline approaches: (1) four white-box test selection approaches including three well-known coverage-based approaches [125], along with Likelihood-based Surprise Adequacy (LSA) and Density-based Surprise Adequacy (DSA) [89], and (2) four SOTA black-box prioritization approaches including Maximum Probability (MaxP) [105], DeepGini [58], Adaptive Test Selection (ATS) [63] and Random Selection (RS). We have selected the most prominent and recent baselines that can be applied and replicated on our models and datasets.

A- White-Box Test Selection Baselines

Neuron Coverage (NC). It is the first coverage metric that has been proposed in the literature to test DNN models [125]. It is defined as the ratio of neurons activated by a test input to the total number of neurons in the DNN model. A neuron is activated when its activation value is greater than a predefined threshold.

Neuron Boundary Coverage (NBC). This coverage metric measures the ratio of corner case regions that have been covered by test input(s). Corner case regions are defined as the activation values that are below or higher than the activation ranges observed during the training phase of the DNN model under test.

Strong Neuron Activation Coverage (SNAC). Similar to the NBC metric, SNAC measures how many upper corner cases have been covered by test input(s). Upper corner

cases are defined as neuron activation values that are above the activation ranges observed during the training phase of the DNN model under test.

LSA and DSA. These two metrics have been proposed by Kim *et al.* [89] and are based on the analysis of how surprising test inputs are with respect to the training dataset. LSA uses Kernel Density Estimation (KDE) [147] to estimate the likelihood of seeing a test input during the training phase. According to Kim *et al.* [89], test inputs with higher LSA scores are preferred since they are closer to the classification boundaries. Thus, it could be considered a priority score for DNN test selection. DSA is an alternative to LSA that uses the distance between the activation traces [89] of new test inputs and the activation traces observed during training.

B- Black-Box Test Selection Baselines

DeepGini. It is a test selection approach that prioritizes test inputs with higher uncertainty scores [58]. It relies on the *Gini* metric (Section 3.2.2.1) to estimate the probability of misclassifying a test input.

MaxP. It relies on the maximal prediction probability of the classification task to estimate the prediction confidence of the DNN model for a given input. Such method prioritizes inputs with lower confidence. The maximum probability score of a test input x is defined as $MaxP(x) = 1 - \max_{i=1}^{m} P_{x_i}$, where $P_{x_{1 \leq i \leq m}}$ is the probability that input x belongs to class C_i and m is the total number of classes [105]. Intuitively, higher MaxP scores are more likely to lead to mispredictions [105].

ATS. It is a recent test selection method proposed by Gao *et al.* [63]. The selection is guided by a fitness function based on which test inputs are incrementally added to the final test set. The fitness function measures the difference between a test input and the currently selected test set based on computing a fault pattern coverage score. This score is obtained through analyzing the diversity of the output probability vectors of the DNN under test. They select test inputs with different fault patterns and higher uncertainty scores.

Random Selection (**RS**). It is the most basic and simplest test selection method in the literature. RS consists in randomly selecting (without replacement) β inputs from the test dataset. Each test input has the same probability of being selected.

3.3.5 Choice of Operators

In this work, we made modifications to the traditional NSGA-II algorithm by customizing its fundamental operators as explained in Section 3.2.3. Specifically, we introduced a novel crossover function and mutation operator. To ensure each of our customized operators had a beneficial impact on the search results, we conducted an empirical study by considering four variants of the search algorithm. Each variant focused on changing only one search operator. In the first variant of *DeepGD*, we replaced the customized crossover operator with the standard version, where no reordering of the genes is applied based on their uncertainty score. In the second search variant, we replaced the customized mutation operator

with the standard version, where 1% of the genes in the selected individuals are randomly mutated, regardless of their uncertainty and diversity scores. Finally, we introduced more mutation strategies in the two other variants. Specifically, in the third variant, we mutated the genes with the lowest uncertainty scores, aiming to explore the impact of prioritizing uncertainty in the mutation process. In the fourth variant, we focused on mutating the genes with the lowest diversity score, investigating the effect of emphasizing diversity in the mutation operation.

Table 3.2: Fault Detection Rates of *DeepGD*'s Variants

	Method	FDR	
		Fashion-Mnist & LeNet4	Cifar-10 & 12-layer ConvNet
Subset size=100	Original DeepGD	**39.56%**	**59.67%**
	DeepGD with Simple Crossover	31.00%	54.00%
	DeepGD with Simple Mutation	32.50%	49.00%
	DeepGD with Gini-based Mutation	38.00%	52.00%
	DeepGD with GD-based Mutation	37.00%	52.50%

We evaluated the different variants of *DeepGD* on two models and datasets, considering a testing budget of $\beta = 100$. The search results are reported in Table 3.2. Our findings demonstrate that *DeepGD* consistently outperforms all search variants with its custom operators in terms of fault-revealing power across datasets and models. More specifically, reordering the genes according to their uncertainty scores when applying the crossover, helped the search converge faster and yielded better fault detection results. Moreover, incorporating both diversity and *Gini* scores in the mutation process resulted in better fault detection results compared to using the default version or relying solely on *Gini* or diversity scores for mutations. These results therefore justify our operators' customization choices.

3.4 Evaluation and Results

We describe in this section our experimental evaluation and present in detail the obtained results. Before addressing our research questions, it is important to note that we applied the fault estimation approach described in Section 3.3.2 to all models and datasets. This allowed us to identify the faults in each subject and estimate their number, which we report in Table 3.1.

3.4.1 Research Questions

3.4.1.1 RQ1. Do we find more faults than existing test selection approaches with the same testing budget?

To investigate the effectiveness of test selection approaches in a DNN, existing baselines usually compare the misprediction detection rate [58, 63]. Although the number of mispredicted inputs is a useful metric in some contexts, it is misleading for test selection in

DNN-based systems since many mispredicted inputs may be redundant and caused by the same fault or root cause in the DNN model [2]. Therefore, we compare the effectiveness of *DeepGD* with the existing baselines based on the fault detection rate. However, in order to provide complete information about our analysis, the number of mispredicted inputs revealed by each approach is reported in our replication package [3].

Table 3.3: The fault detection rate for each subject with test subset size $\beta = 100$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	13%	21%	13%	12%	12%	12%	11%	13%	31%	35%	14%	55%	54%	53%	**59%**
	ResNet20	10%	10%	11%	17%	16%	11%	17%	16%	42%	37%	11%	52%	56%	50%	**57%**
MNIST	LeNet1	25%	15%	16%	19%	12%	16%	18%	12%	31%	23%	12%	41%	28%	40%	**42%**
	LeNet5	16%	13%	13%	17%	16%	14%	19%	16%	29%	33%	13%	35%	34%	36%	**40%**
Fashion	LeNet4	5%	18%	14%	15%	14%	14%	15%	14%	17%	34%	10%	**39%**	**39%**	32%	**39%**
SVHN	LeNet5	9%	4%	11%	12%	11%	11%	12%	11%	13%	19%	11%	43%	43%	44%	**47%**

Table 3.4: The fault detection rate for each subject with test subset size $\beta = 300$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	19%	28%	21%	22%	23%	22%	22%	22%	35%	37%	22%	54%	53%	50%	**57%**
	ResNet20	15%	17%	17%	17%	19%	17%	17%	19%	49%	45%	19%	56%	58%	52%	**59%**
MNIST	LeNet1	25%	25%	33%	28%	27%	36%	29%	27%	40%	40%	25%	53%	44%	53%	**54%**
	LeNet5	34%	33%	32%	33%	30%	32%	30%	30%	54%	52%	24%	63%	59%	62%	**64%**
Fashion	LeNet4	8%	25%	19%	23%	23%	19%	23%	23%	35%	45%	17%	51%	47%	49%	**53%**
SVHN	LeNet5	11%	13%	17%	16%	18%	17%	16%	18%	19%	36%	17%	58%	61%	58%	**62%**

Table 3.5: The fault detection rate for each subject with test subset size $\beta = 500$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	30%	34%	32%	33%	30%	30%	31%	29%	45%	51%	33%	67%	65%	64%	**70%**
	ResNet20	28%	28%	28%	29%	30%	28%	29%	30%	61%	63%	27%	68%	71%	63%	**73%**
MNIST	LeNet1	29%	31%	43%	39%	36%	42%	38%	36%	52%	58%	34%	66%	58%	65%	**67%**
	LeNet5	39%	42%	41%	46%	40%	42%	42%	40%	65%	65%	38%	73%	72%	71%	**74%**
Fashion	LeNet4	13%	28%	28%	29%	27%	28%	29%	27%	46%	54%	27%	**70%**	65%	64%	**70%**
SVHN	LeNet5	14%	22%	24%	22%	25%	24%	22%	25%	27%	46%	26%	77%	76%	72%	**78%**

Detailed results of the fault detection rate for six subjects (resulting from the combination of datasets and DNN models), and for different sizes of the test input set $\beta \in \{100, 300, 500\}$, are shown in Table 3.3, Table 3.4, and Table 3.5, respectively. To provide a more comprehensive evaluation of our results, we also report the average number of faults covered by each selection method in Appendix 3.7. The results follow similar trends and the final conclusion is consistent for all test subset sizes. Because of randomness in *DeepGD*, ATS, and random selection, we re-executed each of them 10 times and reported the corresponding average fault detection rates and average number of detected faults (see Appendix 3.7). Our results show that all of the test selection approaches guided by coverage metrics are ineffective in detecting faults. Even after repeating the experiment with various parameters as suggested in their original papers, the results are similar. The fault detection rate of NC, for example, was between 4% and 25% for subset sizes of 100. SNAC and NBC also showed poor fault detection rates, which vary between 11% and 19% for the same subset size. Compared with LSA and DSA, both *DeepGD* and other black-box prioritization selection methods showed a much higher fault detection rate for all subjects. Considering our previous experiments in previous chapter [2], where we investigated white-box coverage metrics and reported that they are not correlated to faults, it is not surprising

that these approaches are unlikely to perform as effectively as black-box approaches in the selection of test inputs with high fault-revealing power. Overall, it can be concluded that black-box approaches are more effective in detecting faults than white-box approaches.

DeepGini and MaxP fare the same as *DeepGD* for two different subjects with a test subset size of 100. However, for other subjects, *DeepGD* achieves a higher fault detection rate than these two approaches. For larger test subset sizes ($\beta \in \{300, 500\}$), *DeepGD* performs better than other black-box baselines in all subjects, with a fault detection rate between 53% and 78%.

We should note that the second-best approach is not consistently the same across subjects and sizes, therefore strengthening our conclusion that *DeepGD* is the only solution we can confidently recommend regardless of the model, dataset, and test set size. For example, with $\beta = 100$, compared to black-box SOTA baselines (ATS, MaxP, Gini), *DeepGD* is, on average, 2 pp better (with a maximum of 4 pp across all models and datasets) than the second-best black-box alternative and 7 pp better (with a maximum of 14 pp across all models and datasets) than the fourth-best black-box alternative (excluding random selection which consistently showed poor performance). Since we cannot predict beforehand how a given alternative will fare compared to the others for a dataset and model, this implies that the effect of selecting another technique than *DeepGD* is potentially significant as we may end up using the worst alternative. For example, if we rely on the inputs selected by *DeepGini* to test LeNet1 on the MNIST dataset ($\beta = 300$), we will only reach a fault detection rate of 44% instead of 54% with *DeepGD*. Further, we also report that *DeepGD* is on average 15 pp better than the best white-box test selection approach and 41 pp better than the worst white-box alternative across all models and datasets.

We performed the statistical analysis using the Wilcoxon signed-rank test [107], with a significance level of $\alpha = 0.05$, to investigate whether *DeepGD* significantly outperforms each SOTA baseline in terms of fault-revealing power. The Wilcoxon signed-rank test, a non-parametric test, is used to compare the medians of continuous variables for paired samples and is employed when the data does not follow a normal distribution, as in our case. We employ a paired test as we compare the fault-revealing power of methods for a given subset size and model. To conduct the statistical tests, we collected data about the performance of each method (DeepGD, ATS, GINI, and MaxP) across the 18 different combinations of datasets, models, and subset sizes. For each SOTA baseline, we thus compared pairs of fault detection rates reported for DeepDG and the selected baseline across 18 different combinations. Results showed that all p-values are lower than 0.05, indicating that *DeepGD* significantly surpasses all SOTA baselines in finding more faults in DNNs.

Answer to RQ1: *DeepGD* outperforms both white-box and black-box test selection approaches for DNNs, in terms of detecting distinct faults given the same testing budget. Further, the second-best black-box approach after *DeepGD* is not consistently the same.

3.4.1.2 RQ2. Do we more effectively guide the retraining of DNN models with our selected inputs than with baselines?

Having examined the effectiveness of *DeepGD* and other baselines in selecting test input sets with high fault-revealing power, we next focus on the extent to which the test selection approaches can help select data to effectively retrain the DNN models under test. We only consider black-box test selection approaches in this research question, as they showed much better performance than white-box test selection baselines. We consider the same models and datasets as in the previous experiment. We augment the original training dataset with the test input selected in RQ1 with the size of 300 by *DeepGD*, ATS, DeepGini, and MaxP, respectively, to retrain the DNN model. We measure the accuracy of the retrained model on both the whole test dataset and a newly generated dataset that is obtained by applying five realistic image transformations to the original test dataset. The latter allows for a fairer and more complete comparison between our proposed method and the other baseline approaches since none of the inputs in the generated dataset were used for retraining the model. We leveraged the datasets generated by Gao *et al.* [63], a recent work in test selection, that we consider as one of our baselines. The authors applied various transformations to the MNIST, Fashion-MNIST, and Cifar-10 test datasets to generate more valid test inputs. However, since the SVHN dataset was not included in their experiments, to ensure consistency, we generated new test inputs for this dataset as well by applying their transformations with identical settings, ensuring the validity of generated inputs. These transformations include sheering, rotating, zooming, and changing the brightness and the blurriness of all images in each dataset. We should note that the size of the newly generated test dataset is five times larger than the original test dataset since we apply five image transformations on each original test input. For each black-box test selection approach, we measure the accuracy improvement of the retrained models on both the original test dataset and the generated one. We report the results in Table 3.6. We acknowledge that studying the accuracy improvement of the retrained models on the generated datasets is more suitable in our context since it does not contain any of the inputs used for retraining. But we choose to also report improvements with the original test dataset to verify that the retraining process did improve model accuracy. Note that we only select a small part of the original test dataset to retrain the model (only 300 out of 10,000 to 26,000 inputs). Because of the randomness in *DeepGD* and ATS, we re-executed each of them five times on the different subjects. For each execution, we retrained each subject with the selected test input set and reported the corresponding average model accuracy improvements.

As described in Table 3.6, we found that *DeepGD* provides better guidance for retraining DNN models than the other black-box test selection approaches. It consistently outperforms ATS, DeepGini, and MaxP in terms of accuracy improvements across all models and datasets. Similar to the previously reported results in RQ1, we found that the second-best approach for guiding retraining is not consistently the same across all subjects. For example, ATS was the second-best approach for retraining ResNet20 with Cifar-10, while it was the third-best approach for retraining LeNet5 with SVHN.

To further investigate the effectiveness of *DeepGD* and the other black-box baselines in

Table 3.6: DNNs accuracy improvements after retraining with the selected test input sets.

		Accuracy imp. on orig. test dataset				Accuracy imp. on generated test dataset			
Data	Model	MaxP	Gini	ATS	**DeepGD**	MaxP	Gini	ATS	**DeepGD**
Cifar-10	12 ConvNet	3.52%	2.09%	2.12%	**3.53%**	5.35%	5.12%	4.62%	**6.85%**
	ResNet20	1.66%	0.74%	2.03%	**2.50%**	4.12%	3.45%	5.97%	**6.18%**
MNIST	LeNet1	10.39%	10.40%	10.40%	**11.10%**	13.18%	13.24%	13.19%	**14.76%**
	LeNet5	7.94%	7.91%	7.92%	**9.26%**	13.02%	12.92%	13.08%	**15.82%**
Fashion	LeNet4	3.94%	3.94%	3.91%	**4.17%**	7.33%	7.41%	7.62%	**7.69%**
SVHN	LeNet5	0.96%	0.99%	0.93%	**1.24%**	0.61%	0.54%	0.63%	**1.26%**

Table 3.7: Optimization effectiveness compared to retraining with all candidate tests

			Opt effectiveness on orig. test dataset (T)					Opt effectiveness on generated dataset (G)			
Data	Model	Max imp. (100% T)	MaxP	Gini	ATS	**DeepGD**	Max imp. (100% G)	MaxP	Gini	ATS	**DeepGD**
Cifar-10	12 ConvNet	16.36%	21.51%	12.78%	12.96%	**21.58%**	28.98%	18.47%	17.65%	15.94%	**23.65%**
	ResNet20	9.99%	16.62%	7.41%	20.32%	**25.03%**	23.33%	17.65%	14.80%	25.59%	**26.47%**
MNIST	LeNet1	11.10%	93.60%	93.72%	93.69%	**100%**	23.80%	55.39%	55.62%	55.43%	**62.04%**
	LeNet5	9.52%	83.40%	83.08%	83.19%	**97.27%**	22.78%	57.15%	56.72%	57.41%	**69.46%**
Fashion	LeNet4	12.00%	32.83%	32.83%	32.58%	**34.75%**	23.72%	30.91%	31.23%	32.10%	**32.41%**
SVHN	LeNet5	1.25%	77.01%	79.79%	74.84%	**99.90%**	8.92%	6.87%	6.06%	7.04%	**14.15%**
Average		10.04%	54.16%	51.60%	52.93%	**63.09%**	21.92%	31.07%	30.35%	32.25%	**38.03%**

retraining DNN models, we also computed their *optimization effectiveness* by accounting for the maximum possible accuracy improvement. We therefore retrain the model with the entire original test dataset and report the best accuracy that can be achieved by retraining. Then, for each test selection method, we calculate its optimization effectiveness, defined as the ratio of (1) the accuracy improvement when retaining the model with only 300 selected inputs from the original test dataset (Table 3.6) to (2) the accuracy improvement when retaining the model with the entire original test dataset. We also repeat the above experiment for the generated test data set to once again get a fairer comparison of the test selection approaches and to obtain better generalizability for our results. More specifically, we retrain the original model by adding all generated inputs to the training dataset to achieve the highest accuracy possible. Then, we calculate the corresponding optimization effectiveness which is the ratio of (1) the accuracy improvement when retaining the model with only 300 inputs from the original test dataset (Table 3.6) to (2) the accuracy improvement when retaining the model with the entire generated test dataset. Because of the randomness in *DeepGD* and ATS, we re-executed each of them five times on the different subjects and reported the corresponding average results in Table 3.7.

We found that *DeepGD* consistently outperforms other black-box test selection approaches in terms of optimization effectiveness. Compared to black-box SOTA baselines, *DeepGD* is, on average, 8.93 pp better (with a maximum of 20.11 pp) than the second-best black-box alternative and 11.49 pp better (with a maximum of 25.06 pp) than the worst black-box alternative. As for RQ1, it is worth noting that since the performance of alternatives is not consistent across datasets and models, selecting one of them may yield the worst results. We also report the average optimization effectiveness on the generated test datasets. Gini, MaxP and ATS yield 30.35%, 31.07%, and 32.25% respectively, while *DeepGD* achieves 38.03%. In other words, with only 300 test inputs, from the original test dataset and selected by *DeepGD* for retraining, we were able to reach 38.03% of the

maximum achievable accuracy by retraining with all generated test dataset. Similar to RQ1, we performed a statistical analysis using Wilcoxon signed-rank tests, with a significance level of 0.05, to investigate whether *DeepGD* significantly outperforms the selected black-box test selection approaches in terms of optimization effectiveness across all subjects. We found all p-values to be less than 0.05, indicating that *DeepGD* is significantly better than the selected baselines in retraining DNN models. In other words, results show that *DeepGD* can select a small, more informative subset from a large unlabeled dataset to effectively retrain DNN models and minimize the labeling costs. Selecting diverse inputs with high uncertainty scores not only helps at detecting more faults in the DNN model, but also significantly improves the accuracy of the model through retraining.

Answer to RQ2: *DeepGD* provides better guidance than black-box alternatives for retraining DNN models. It consistently and statistically outperforms other black-box test selection approaches in terms of accuracy improvement, in absolute terms and relatively to the maximum achievable improvement.

3.4.1.3 RQ3. Can DeepGD select more diverse test input sets?

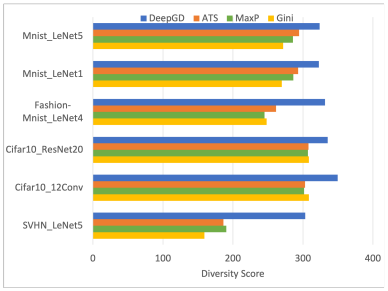


Figure 3.1: GD scores for subsets size=100

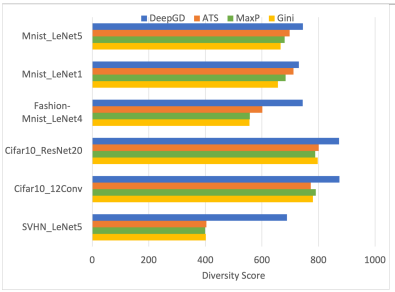


Figure 3.2: GD scores for subsets size=300

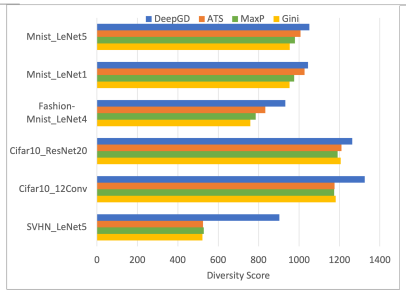


Figure 3.3: GD scores for subsets size=500

To answer this research question, we measured the diversity score of the selected test input sets for *DeepDG* and the other black-box baselines. We relied on the geometric diversity metric since we demonstrated in our prior work [2] its construct validity in measuring the actual diversity of input sets. We used the same subjects, subset sizes and test input sets that were selected in RQ1. Given the inherent randomness in *DeepGD* and ATS, we reported the average diversity scores based on 10 executions. The results, presented in Figures 3.1, 3.2 and 3.3, consistently demonstrate that *DeepGD* outperforms the baseline approaches in selecting diverse input sets. Across all subjects and subset sizes, *DeepGD* consistently obtained the highest diversity scores. Additionally, ATS emerged as the second-best approach for selecting diverse input sets in 12 out of 18 configurations, while *Gini* performed in general the poorest, ranking last in 11 out of 18 configurations.

We also performed a statistical test analysis using the Wilcoxon signed-rank test, with a significance level of 0.05, to investigate whether *DeepGD* significantly outperforms each baseline approach in selecting diverse test input sets. Similarly to RQ1 and RQ2, we

performed a paired test across the different combinations of datasets, models, and subset sizes. For each baseline approach, we compared pairs of diversity scores reported for *DeepGD* and the selected baseline across the 18 different configurations. Results show that all p-values are below 0.05, indicating that *DeepGD* significantly outperforms all SOTA baselines in selecting more diverse test input sets.

Answer to RQ3: *DeepGD* provides better guidance than black-box alternatives for selecting diverse test input sets. It consistently and statistically outperforms other black-box test selection baselines in terms of diversity.

3.4.1.4 RQ4. How do DeepGD and baseline approaches compare in terms of computation time?

To address this research question, we compared the execution time of *DeepGD* with baseline approaches and analyzed its evolution with different subset sizes. Similar to RQ2, we focused on black-box test selection approaches (excluding random selection) as they demonstrated better performance compared to white-box test selection baselines. The subjects and subset sizes used were consistent with our previous experiments. For each subject and subset size, we executed *DeepGD* and the selected baseline approaches 10 times and measured the computation times. This repetition allowed us to account for random variations in execution time.

The results, presented in Figure 3.4, illustrate the change in computation times as the size of the input sets increased. Our findings indicate that all selected approaches were computationally efficient. *Gini* and MaxP exhibited similar execution times, approximately five seconds, and demonstrated the best performance in terms of computation time. In contrast, *DeepGD*, since it relies on a search algorithm, had higher execution times ranging from 200s to 850s across all subjects and subset sizes. Furthermore, we observed that, unlike the other baseline approaches, the execution time of *DeepGD* increases with subset size. However, it is important to note that such execution times remain practically acceptable since, in DNN testing, (1) the labeling cost of test inputs is far more expensive than the computation cost of the search, and (2) test selection is neither frequent nor a real-time task. Despite the longer execution times, the better fault detection performance and retraining guidance provided by *DeepGD* therefore outweigh the increase in computation time in comparison to the baselines.

Answer to RQ4: None of the black-box test selection approaches are computationally expensive. Though *DeepGD* yields longer execution times (a matter of minutes), these remain practical, even for large test subsets, given that test selection is not a frequent task performed under tight time constraints.

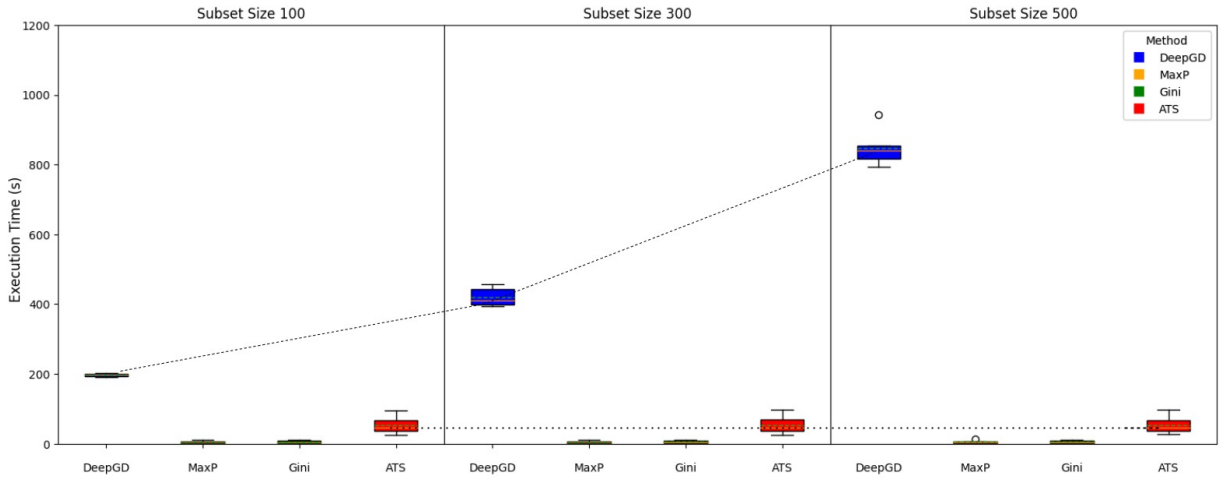


Figure 3.4: Evolution of the execution time over the subset sizes

3.4.2 Discussions

Based on our experimental results, we show that *DeepGD* provides better guidance than existing baselines for selecting test inputs with high-fault revealing power. The second-best approaches for test selection and model retraining are not consistently the same across all models and datasets. This reinforces our conclusion that *DeepGD* is the only solution that we can confidently recommend regardless of the model, dataset, and test set size. Indeed, selecting diverse test inputs with high uncertainty scores not only enables higher fault detection, but also provides more effective guidance for retraining DNN models. Although results are encouraging, and though this remains to be further investigated, we conjecture that *DeepGD* is particularly useful when datasets are more redundant, a situation that is often observed in real-world DNN testing scenarios, especially with massive generated datasets. Selecting diverse test input sets with *DeepGD* is then expected to help reduce redundancy by selecting more informative inputs to be labeled in order to test and retrain DNN models.

Since *DeepGD* relies on genetic search, it entails some degree of randomness in the final selected test sets. We, therefore, studied the stability performance of *DeepGD* by running the tool 10 times with each combination of dataset, model, and subset size that we have previously used in our experiments. We report the minimum, maximum, and standard deviation of FDR results in Table 3.8. As shown in the table, the standard deviations of the FDRs reported by *DeepGD* are consistently low, ranging from 0.008 to 0.021. This indicates that *DeepGD* consistently delivers stable and thus reliable results across subjects and subset sizes over multiple runs. We also found that the faults revealed by *DeepGD* are largely consistent across multiple runs. This can be attributed to the use of the *Gini* score as one of the fitness functions to guide the search towards test inputs with high uncertainty scores.

Table 3.8: Stability of *DeepDG's* fault detection rates.

		Size =100				Size =300			
Data	Model	Ave. FDR	Min. FDR	Max. FDR	Std.	Ave.	Min. FDR	Max. FDR	Std.
Cifar-10	12 ConvNet	59%	57%	62%	0.019	57%	55%	58%	0.009
	ResNet20	57%	55%	60%	0.013	59%	57%	60%	0.009
MNIST	LeNet1	42%	40%	45%	0.017	54%	51%	54%	0.009
	LeNet5	40%	38%	42%	0.016	64%	60%	68%	0.021
Fashion	LeNet4	39%	36%	43%	0.019	53%	52%	55%	0.008
SVHN	LeNet5	47%	44%	49%	0.017	62%	61%	64%	0.012

Note: The values of average, minimum, and maximum FDRs reported in this table have been rounded up to two decimal digits and are represented as percentages. The standard deviation (std) of FDRs has been rounded up to three decimal digits.

3.5 Threats to Validity

We discuss in this section the different threats to the validity of our study in this chapter and describe how we mitigated them.

Internal threats to validity concern the causal relationship between the treatment and the outcome. Since *DeepGD* is black-box and relies on extracting feature matrices to measure the diversity of the selected test input sets, an internal threat to validity might be caused by the poor quality representation of inputs. To mitigate this threat, we have relied on VGG-16, one of the most accurate feature extraction models in the literature. This model has been pre-trained on the very large ImageNet dataset that contains more than 14 million labeled images belonging to 22,000 categories. Also, in our prior work [2], we assessed the use of the GD metric with the VGG-16 model to measure diversity in input sets. Our results showed that the GD metric, in conjunction with our selected feature extraction model, performed well in measuring actual data diversity across all the studied datasets. Moreover, *DeepGD* relies on the specification of a few hyperparameters related to NSGA-II. This also applies to several white-box and black-box test selection approaches that we considered in our study. The configuration of the different hyperparameters in our work may induce additional threats. To mitigate them, we have relied on the NSGA-II hyperparameters recommended in the literature [27, 39, 111] except for the mutation rate, which was intentionally set higher and experimentally tuned since we customized the mutation operator to take into account both fitness functions, as described in section 3.2.3.2. We have also considered different configurations of the baselines-related hyperparameters according to their original papers. A last internal threat to validity would be related to randomness when selecting test input sets with *DeepGD*, ATS, and random selection. We addressed this issue by repeating such selection multiple times while considering different input set sizes and different datasets and models. We also reported on the stability results of *DeepGD*, which indicated that it is stable in this context.

Construct threats to validity concern the relation between the theory and the observations made. A construct threat to validity might be due to inaccuracies in estimating DNN faults since detecting faults in DNNs is not as straightforward as in traditional software. To mitigate this threat, we have relied on a SOTA clustering-based fault estimation

approach [2] that has been thoroughly validated on several models and datasets. In our previous work, we conducted various validation experiments demonstrating that mispredicted inputs within the same cluster are typically mispredicted due to the same fault in the DNN model, whereas inputs from different clusters are mispredicted due to distinct faults (root causes). Additionally, our prior work established that clusters of mispredicted inputs are very different from those in the entire test dataset despite the use of the same feature extraction model (i.e., VGG-16). Such difference further reinforces the validity of our chosen fault estimation approach. We have reused this publicly available approach to obtain accurate fault estimates. Nonetheless, relying on a such fault estimation approach is still far better than just considering mispredicted inputs that are redundant and due to the same root cause in the DNN model.

External threats to validity concern the generalizability of our study. We mitigate this threat by considering six different combinations of widely used and architecturally distinct models and datasets. We also considered many testing budgets in our experiments and compared our results with nine SOTA baselines for DNN test selection.

3.6 Related Work

In this section, we introduce existing work related to our proposed approach from two aspects: test selection and test diversity in the context of DNN models.

Test Selection for DNNs. Test selection approaches proposed for DNN models can be characterized as black-box or white-box, depending on their access requirements to the internals of the DNN model. A few black-box test selection approaches for DNNs have been introduced in the literature. They generally rely on the uncertainty of model classifications. For example, Feng *et al.* [58] proposed DeepGini, a test selection approach that prioritizes the selection of inputs with higher *Gini* scores [58]. They conjecture that if a DNN model is unsure about a classification and outputs similar probabilities for each class, it is more likely to mispredict the test input. Compared to random and coverage-based selection methods, DeepGini was shown to be more effective in uncovering mispredictions [58, 150]. Similar to their work, we rely on *Gini* scores as one of the fitness functions in our approach. However, we also consider the diversity of the selected test set and rely on a multi-objective genetic search to guide the search toward finding test inputs with high fault-revealing power. Li *et al.* [101] introduced Cross Entropy-based Sampling (CES) and Confidence-based Stratified Sampling (CSS), for black-box DNN test selection. These metrics are used to select a small subset of test inputs that accurately reflect the accuracy of the whole test dataset. They show that compared to random sampling, their approach could achieve the same level of precision with about half of the labeled data. Their goal is clearly different from ours. While our focus is to prioritize the selection of test inputs with high fault-revealing capability, their goal is to minimize test sets. Also, though Arrieta [10] relied on NSGA-II and uncertainty scores for selecting metamorphic follow-up test cases, our goal with *DeepGD* is also different. We use both diversity and DNN uncertainty to guide the search toward test inputs with high fault-revealing power, for a fixed budget.

Several white-box test selection approaches have been proposed as well. Such approaches generally rely on coverage [65, 104, 125] or surprise metrics [89] to select inputs that will be labeled and used for testing DNN models. For example, Pei *et al.* [125] proposed Neuron Coverage (NC), which measures neurons activation rates in DNN models. Ma *et al.* [104] proposed *DeepGauge*, a set of coverage metrics for DNN models that consider neurons activation ranges. Kim *et al.* [89] proposed surprise coverage metrics which measure how surprising test sets are given the training set. The selection of test inputs for all these approaches is based on maximizing coverage or surprise scores. However, several studies [2, 34, 49, 71, 100, 105, 159, 161], have shown that these white-box metrics are not always effective for guiding DNNs test selection. For example, Ma *et al.* [105] performed an empirical study and compared the effectiveness of white-box metrics (coverage and surprise metrics) with black-box uncertainty in guiding test input selection. Results showed that the former have a weak or no correlation with classification accuracy while the latter had a medium to strong correlation. Uncertainty-based metrics not only outperform coverage-based metrics but also lead to faster improvements in retraining. In our work, as mentioned above, we also consider maximizing the uncertainty score of the test inputs as one of our two fitness objectives.

Diversity in DNN Testing. Many works have studied diversity-based test selection and generation for traditional software [19, 56, 73]. The underlying assumption is that there is a strong correlation between test case diversity and fault-revealing power [73]. Their results confirmed this assumption and showed that diversity-based metrics are effective in revealing faults. Inspired by these encouraging results, researchers devised diversity-based approaches for DNN testing [2, 63, 167, 170]. Zhao *et al.* [167] conducted an empirical study of SOTA test input selection approaches for DNNs [33, 101, 168] and concluded that they have a negative impact on test diversity and suggest that more research is warranted on designing more effective test selection approaches that guarantee test diversity.

In a recent study, Gao *et al.* [63] proposed the adaptive test selection method (ATS) for DNN models that use the differences between model outputs as a behavior diversity metric. Although ATS aims to cover more diverse faults, its test selection is guided only by model outputs. *DeepGD*, however, considers both the uncertainty of model output probabilities and input features' diversity. Moreover, our results show that *DeepGD* outperforms ATS in test input selection for different combinations of models and datasets and for different test subset sizes. In our prior work [2], we studied three black-box input diversity metrics for testing DNNs, including geometric diversity [67], normalized compression [40], and standard deviation. We investigated the capacity of these metrics to measure actual diversity in input sets and analyzed their fault-revealing power. Our experiments on image datasets showed that geometric diversity outperforms SOTA white-box coverage criteria in terms of fault detection and computational time. However, in that work, we did not study how the GD metric can be used in practice to guide the selection of test inputs for DNN models.

Input Selection for Deep Active Learning. Deep Active Learning (DAL) involves the incremental selection of inputs to be labeled and used for (re)training DNN models. Active learning typically starts with a model that has been trained using a small, randomly selected set of inputs. Then, in each (re)training iteration, a subset of unlabeled inputs is selected based on a query strategy, manually labeled, and then used to (re)train the model

and improve its performance. In particular, the objective of each iteration is to select the most informative inputs for labeling that enable reducing the labeling cost while improving the accuracy of the model. DAL methods can be categorized into three classes based on the used input selection strategy: uncertainty-based, diversity-based, and combined approaches that leverage both diversity and uncertainty to select inputs for (re)training DNN models effectively [11, 136, 165]. Zhan *et al.* [165] conducted a comparative study including 19 DAL approaches from the three categories. They concluded that the combined DAL approaches such as Wasserstein Adversarial Active Learning (WAAL) [136] and Batch Active learning by Diverse Gradient Embeddings (BADGE) [11], provide in general better guidance for model retraining compared to uncertainty and diversity-based DAL techniques. Both BADGE and WAAL propose a combined approach for input selection based on uncertainty and diversity.

Although BADGE, WAAL, and *DeepGD* appear to share similarities in terms of selection metrics, they diverge in how they measure uncertainty and diversity. For instance, BADGE adopts a white-box approach, relying on the last-layer loss gradients of the model during (re)training to compute uncertainty and construct input feature vectors. It then employs K-means clustering on these feature vectors to select diverse inputs from each cluster. Notably, BADGE primarily operates with DNN models trained using commonly used gradient-based optimizers, unlike *DeepGD* which is agnostic to the model's optimizer.

In addition, WAAL requires the initial labeling of 1% to 5% of the (re)training dataset before performing additional and incremental input selection. These initially labeled input sets serve as a basis for selecting diverse inputs from the unlabeled dataset, with a focus on those displaying the largest Wasserstein distance w.r.t. the labeled input set [136].

Since active learning techniques require model training-specific data such as gradient loss, uncertainty estimates, or prediction confidence scores during the training epochs [165], they are only applicable during the (re)training phase of DNN models. In contrast, test selection techniques are applicable in both the training and testing phases. It is also worth noting that active learning techniques require human-in-the-loop labeling throughout the entire (re)training process. This iterative selection of inputs during training iterations makes such approaches impractical in many situations. On the other hand, test selection techniques, including *DeepGD*, only require labeling once, within a given testing budget, before testing and retraining the model.

Based on the above points, active learning techniques are therefore not adapted to the objectives of automatically and optimally supporting test selection and using the selected test inputs to support retraining, without requiring iterative human intervention for labeling inputs through the retraining process.

3.7 Conclusion

In this chapter, we propose *DeepGD*, a multi-objective search-based test input selection approach for DNN models. Our motivation is to provide a black-box test selection mechanism that reduces labeling costs by effectively selecting unlabeled test inputs with high

fault-revealing power. We rely on both diversity and uncertainty scores to guide the search toward test inputs that reveal diverse faults in DNNs. We conduct an extensive empirical study on six different subjects and compare *DeepGD* with nine state-of-the-art test selection approaches. Our results show that *DeepGD* statistically and consistently outperforms baseline approaches in terms of its ability to reveal faults in DNN models. Selecting diverse inputs with high uncertainty scores with *DeepGD* not only helps detecting more faults in the DNN model for a given test budget, but also significantly improves the accuracy of the model through retraining with an augmented training set. Our results also indicate that the second-best approach for testing or retraining is not consistent across all models and datasets, further supporting the choice of *DeepGD*. We aim to extend our work by studying the application of diversity and uncertainty metrics for other DNN testing purposes such as test minimization and generation. Finally, we need to investigate alternative ways to estimate faults in DNNs and study the relationship between the fault detection rate and the model repair capability of test selection methods.

Appendix 2

Table 3.9: The average number of detected faults in each subject with test subset size $\beta = 100$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	13	21	13	12	12	12	11	13	31	35	14	55	54	53	**59**
	ResNet20	10	10	11	17	16	11	17	16	42	37	11	52	56	50	**57**
MNIST	LeNet1	25	15	16	19	12	16	18	12	31	23	12	41	28	40	**42**
	LeNet5	14	11	11	15	14	12	16	14	25	28	11	30	29	30.6	**34**
Fashion	LeNet4	5	18	14	15	14	14	15	14	17	34	10	**39**	**39**	32	**39**
SVHN	LeNet5	9	4	11	12	11	11	12	11	13	19	11	43	43	44	**47**

Table 3.10: The average number of detected faults in each subject with test subset size $\beta = 300$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	35	52	39	41	43	41	41	41	65	69	41.1	101	99	93.5	**109.6**
	ResNet20	26	30	30	30	33	30	30	33	87	80	33.6	99	102	92	**104.4**
MNIST	LeNet1	34	34	45	38	37	49	40	37	55	55	34.3	73	60	72.6	**74**
	LeNet5	29	28	27	28	25	27	25	25	46	44	20.4	54	50	52.7	**54.4**
Fashion	LeNet4	11	35	27	32	32	27	32	32	49	63	24	72	66	69	**74.7**
SVHN	LeNet5	16	19	25	23	26	25	23	26	28	53	25	85	90	85.3	**91.1**

Table 3.11: The average number of detected faults in each subject with test subset size $\beta = 500$

Data	Model	White-box										Black-box				
		NC		NBC			SNAC			LSA	DSA	RS	MaxP	Gini	ATS	**DeepGD**
		0	0.75	0	0.5	1	0	0.5	1							
Cifar-10	12 ConvNet	56	64	60	62	56	56	58	54	84	95	61.7	125	122	119.8	**131.5**
	ResNet20	50	50	50	51	53	50	51	53	108	112	47.8	120	126	111.5	**129.5**
MNIST	LeNet1	40	42	59	53	49	58	52	49	71	79	46.6	90	79	89.1	**92**
	LeNet5	33	36	35	39	34	36	36	34	55	55	32.3	62	61	60.3	**63**
Fashion	LeNet4	18	39	39	41	38	39	41	38	65	76	38	**99**	92	90	**99**
SVHN	LeNet5	21	32	35	32	37	35	32	37	40	68	38.2	113	112	105.8	**115.1**

Chapter 4

Test generation for the differential testing of DNN models

This chapter addresses TO3, which focuses on developing a differential testing method capable of generating images that reveal important differences in models' behavior. This approach aims to support better-informed decisions regarding model selection, optimization, and deployment in real-world environments. The content of this chapter is currently under review at the *IEEE Transactions on Software Engineering.*

4.1 Overview

As discussed in chapter 1, comparing multiple DNN models is crucial, particularly in tasks like model selection, compression, and regression testing. Relying solely on the original test dataset is often insufficient to uncover subtle behavioral differences between models. Even when models exhibit similar accuracy on standard test sets, they may behave differently under real-world conditions. For example, models with similar accuracy and functionality might perform very differently in specific scenarios, such as varying environmental conditions or operational constraints. This makes selecting the best-performing model or optimally combining models particularly challenging, especially when testing datasets are limited.

These challenges underscore the importance of differential testing, which focuses on generating triggering inputs—test inputs specifically designed to expose behavioral discrepancies between DNN models. Differential testing is essential in scenarios such as model selection, where organizations must often choose between models trained for the same task but with different architectures, training datasets, or optimization techniques. It is also critical for model compression, ensuring that compressed models retain essential functionality while improving efficiency. Similarly, in regression testing, differential testing helps detect unintended performance regressions when models are updated.

Although these models may achieve similar accuracy on a given test dataset, their performance can vary significantly under different operational conditions. Differential testing

helps identify these variations by generating inputs that expose where one model might outperform the others, providing deeper insights that go beyond simple accuracy metrics. Similarly, in model compression, large DNN models are often compressed using techniques such as quantization, pruning, or knowledge distillation to make them suitable for deployment in resource-constrained environments, such as mobile devices or embedded systems. Differential testing is essential here to ensure that the compressed models retain the accuracy of the original uncompressed versions. By generating inputs that highlight behavioral differences between the compressed and original models, differential testing allows developers to better understand the trade-offs involved in compression and to ensure that the model's critical performance characteristics are preserved. For regression analysis, when DNN models are updated or re-trained, differential testing identifies and generates inputs where the new version performs worse than the original or vice versa, ensuring that updates do not compromise the accuracy of the model. Last, when using ensembles of models, a voting mechanism is required and one needs to be able to learn when to trust which model's output.

However, the differential testing process presents several challenges. First, the input subspace where highly accurate models mispredict is often extremely limited, within an extremely large input space. This makes it difficult to find the specific inputs that will trigger behavioral differences, especially in models designed to perform well across a wide range of scenarios. Efficient and effective techniques are thus required to reveal meaningful such discrepancies. Second, many models are treated as black-boxes, where only the inputs and outputs are visible without access to the internal workings of the model. This lack of access complicates the generation of test inputs since the internal decision-making processes remain hidden. Third, limited testing budgets constitute another constraint, particularly in scenarios where collecting, labeling, and validating test input is time-consuming and resource-intensive. In cases where models are deployed in environments that require expensive simulators or manual validation, testing costs can quickly escalate, further limiting the resources available for comprehensive testing.

Finally, ensuring the validity of the generated inputs in differential testing poses a significant challenge. Invalid inputs can lead to mispredictions and skew models' performance assessments. Human feedback is often required to verify the contextual relevance and validity of test inputs, adding complexity and making the process more resource-intensive. This manual validation step is difficult to scale, further complicating efforts to evaluate true performance discrepancies between models effectively.

Several test generation approaches for the differential testing of DNN models have been proposed in the literature [24, 125, 143, 153, 154, 158, 162], but they often come with key limitations. Many of these methods are white-box [24, 125, 153, 158], requiring access to the internal structure and parameters of the DNN models under test. This restricts their applicability to proprietary models, where such internal access is typically unavailable due to privacy or intellectual property concerns. Some other approaches [24, 158] assume structural similarities between the models under test, limiting their applicability on DNN models of different architecture. Black-box differential testing methods [143, 154, 162], on the other hand, often rely on customized mutation operators applied to seed inputs to generate triggering inputs. However, these methods are highly dependent on the availability

and quality of the seed data, restricting the generation process to variations of existing inputs.

In response to these challenges, we propose *DiffGAN*, a black-box test generation approach tailored for the differential testing of DNN models with similar accuracy levels on a given test dataset. By leveraging Generative Adversarial Networks (GANs) and the Non-dominated Sorting Genetic Algorithm II (NSGA-II), *DiffGAN* generates diverse and valid triggering inputs that effectively reveal behavioral disagreements between the models under test. Since *DiffGAN* operates as a black-box method, it does not require access to the internal structure of the models or assume structural similarities between them, making it suitable for testing any pair of DNN models.

To evaluate *DiffGAN*, we constructed a benchmark consisting of eight pairs of models (16 DNN models) trained on two widely used image classification datasets. We compared its performance to *DRfuzz*, a SOTA differential testing baseline assuming the same testing budget. Our experimental results demonstrate that *DiffGAN* significantly outperforms the baseline, generating on average four times more triggering inputs. Moreover, these inputs include a greater number of valid images and exhibit higher diversity, enabling a more comprehensive assessment of the models' behaviors.

We further demonstrate an application of the triggering inputs generated by *DiffGAN* through the development of a machine learning-based selection mechanism. This system uses a machine learning model to dynamically select between two classifiers, predicting which one will perform better based on the input characteristics. Our results show that training the selection model with inputs generated by *DiffGAN* leads to significantly more accurate selections compared to those trained with inputs from the baseline approach.

The main contributions of chapter 4 are as follows:

1. We propose *DiffGAN*, a novel, black-box test generation approach that leverages GANs and NSGA-II to find triggering inputs for differential testing of DNN models, enabling an exhaustive analysis of behavioral disagreements between models with similar accuracy levels.

2. We perform a comprehensive evaluation on a benchmark of eight image classification model pairs, showing that *DiffGAN* significantly outperforms the state-of-the-art baseline in generating more diverse and valid triggering inputs, given the same testing budget.

3. We show that the triggering inputs generated by *DiffGAN* can be used to train a machine learning-based mechanism to select the most accurate model based on input characteristics, leading to improved results when compared to models trained on data generated by the baseline. Model selection constitutes a very important application of differential testing.

4. To support future research and reproducibility, we have made our source code and benchmark publicly available[1].

[1] `https://github.com/zohreh-aaa/Diff_testing_GAN`

Chapter Structure: The rest of this chapter is structured as follows. In Section 4.2, we define the research problem and outline the key assumptions underlying our approach. Section 4.3 details the methodology and design of our proposed solution. In Section 4.4, we present the empirical evaluation and corresponding results. Section 4.6 discusses potential threats to the validity of our findings. Section 4.7 reviews related works. Finally, Section 4.8 concludes this chapter.

4.2 Problem Definition

In this chapter, we propose a test generation approach for the differential testing of DNN models to reveal behavioral disagreements between DNN models that exhibit similar accuracy levels.

4.2.1 Motivations

Differential analysis is crucial across a variety of practical applications such as model selection (before and during inference time), model compression [143, 148, 154], model regression [162], and detecting model reuse [99]. For example, consider the context of model selection. In practice, organizations often face the challenge of choosing the most suitable model during the training phase, prior to inference time. This decision process involves choosing from multiple DNN models, each one possibly trained by different providers for the same task, using different model architectures or training datasets, and demonstrating similar accuracy on a testing dataset. In such scenario, the ability to generate inputs that clearly indicate differences in model behavior is invaluable to guide model selection.

Differential testing is also crucial for model output selection at inference time, in the context of ensemble learning or voting systems, where multiple DNN models are combined to obtain a better prediction than with individual models. This type of testing involves revealing triggering inputs that can not only help characterize the conditions under which one model may outperform the others but also enhance our understanding of each model's operational boundaries. By understanding the conditions under which one model fares better, one can determine at inference time which model's output to trust.

Differential testing is also crucial in the context of model compression, a process where large DNN models are transformed into more compact versions through techniques such as quantization, pruning, and knowledge distillation [5]. This transformation is aimed at deploying these models in resource-constrained environments while preserving as much of their original accuracy as possible. In these scenarios, differential testing is invaluable as it helps identify how the behavior of the compressed model deviates from the original. By generating triggering inputs that expose these differences, developers can better understand the trade-offs involved in the compression process. This is crucial for ensuring that the compressed models still meet the required accuracy levels before they are deployed in real-world applications.

Differential testing also plays a pivotal role in identifying model regression [162], which entails detecting discrepancies between iterations of a deep learning model (original versus updated versions). For instance, an update aimed at enhancing the overall accuracy of a DNN model might inadvertently introduce inaccuracies under specific input conditions. To mitigate this, triggering test inputs are essential, to systematically assess changes in accuracy across updates and thus ensure they do not lead to poorer accuracy in critical conditions.

Additionally, differential testing is useful for detecting model reuse [99]. In industries where proprietary models are crucial assets, it is important to verify that deployed models are original. Differential testing can identify if a model presented as new is actually derived from or modified from an existing model, thus protecting intellectual property and ensuring the authenticity of models.

In conclusion, while differential testing has broad applications, our focus in this chapter is specifically on model selection. Unlike other contexts such as model compression, regression detection, or model reuse, the model selection scenario presents unique challenges. Here, the goal is not to evaluate how a single model evolves or changes, but rather to compare multiple models developed independently for the same task. These models may exhibit similar accuracy levels but differ in training data, structure, and optimization methods. Focusing on model selection allows us to explore how differential testing can expose behavioral differences between DNN models. This focus is critical because, in practice, selecting the most appropriate model, whether after training or at inference time, can directly impact the systems using these models.

4.2.2 Challenges

Generating triggering inputs to reveal behavioral disagreements between DNN models under test introduces three main challenges. First, uncovering discrepancies in highly-accurate models is inherently difficult due to the very small input subspace leading to mispredictions. Because they are designed to perform well under a wide range of conditions, we require effective and efficient techniques to identify behavioral disagreements. Second, in many real-world scenarios, we do not have access to the internal information of DNN models. Indeed, such models are often proprietary and deployed on servers, accessible only through cloud APIs. This limitation constrains us to using black-box techniques to generate triggering inputs as we only have access to the inputs and outputs of the DNN models. Finally, the testing budget is often limited, particularly when models (1) require extensive manual labor to label and validate triggering inputs, or (2) operate on expensive simulators or physical devices. These simulators usually provide a controlled environment that replicates true operational scenarios, allowing for rigorous testing without the costs and risks associated with testing models deployed on expensive physical devices. Within a realistic testing budget, we must effectively generate triggering inputs that reveal diverse behavioral disagreements between the models under test.

4.2.3 Assumptions

In this study, we propose a differential testing approach for DNN models considering the following assumptions:

Black-Box Models: We assume that our DNN models are black-box. The internal DNN workings, such as intermediate computations and gradients, are inaccessible. This assumption is crucial, especially in scenarios involving models received from third-party providers or models trained via federated learning, where internal model details are naturally obscured to protect privacy and intellectual property. Moreover, frameworks such as TensorFlow Lite and ONNX Inference, commonly used for model quantization, usually offer only end-to-end inference APIs. These frameworks do not provide access to intermediate model data, emphasizing efficiency in inference [13].

Pairwise Differential Testing: Our test generation approach primarily focuses on the pairwise comparison of DNN models. This assumption means that our analysis of behavioral differences is based on comparisons between two DNN models at a time, allowing for the generation of triggering inputs for a pair of DNN models.

No Access to the Training Data: We assume that we do not have access to the original training datasets of the models under test. This is a common scenario in many real-world situations where models are proprietary or when model testing is provided as a service for example. Consequently, our ability to generate inputs and interpret model behavior is based solely on observations from models' outputs and the currently available testing data, rather than insights from the training process. **Similar Accuracy Levels:** We assume that each pair of models under test yields similar and high accuracy on the currently available testing dataset. Such generation warrants the effective generation of triggering inputs to reveal behavioral disagreement between the DNN models, to better assess their respective capabilities and thus enable their use in the various practical applications mentioned above.

Limited Testing Budget: We assume that our test generation approach is constrained by a limited testing budget, a valid assumption in many real-world situations when (1) DNNs are deployed on physical devices or costly simulators in the loop, or (2) when the cost of the manual labeling or the validation of triggering inputs is high. In this study, we determine the testing budget based on the feasible execution time for test generation. We thus restrict the number of inputs to generate and analyze. Furthermore, a fixed and identical testing budget allows for a fair evaluation of alternative differential test generation approaches.

4.2.4 Differences from Adversarial Inputs

Adversarial samples are inputs subtly modified to mislead a specific single model by incorporating nearly imperceptible changes that may lead to incorrect predictions. Conversely, triggering inputs are designed to expose discrepancies between two versions of a model—for example the original and its compressed counterpart—by demonstrating that they produce divergent outcomes when fed with the same input. This distinction is crucial

for understanding how models behave after modification or for comparing the behavior of two distinct models designed for the same task. Tian *et al.* [143] demonstrated that, on the MNIST dataset, only 18 out of 10,000 adversarial samples generated using various adversarial attacks served as triggering inputs for differential testing. Similarly, prior studies [37, 87] indicate that adversarial inputs can be limited in their ability to expose differences between compressed models. Given that adversarial inputs target single model weaknesses and are not suitable for revealing differences in models' behavior, they are excluded from our differential testing analysis.

4.3 Approach: Reformulation as an NSGA-II Search Problem

A central problem in differential testing, particularly when evaluating DNN models that exhibit similar accuracy levels, is to generate triggering inputs that reveal behavioral disagreements between models. The generation and analysis of such inputs are crucial, as they not only reveal the specific conditions under which one model may outperform another but also enhance our understanding of the models' operational boundaries. This insight is important for extracting the precise operational contexts in which each model is likely to fare best, thereby informing more targeted and effective application of these models in real-world scenarios. Traditional approaches, such as transformation filters and adversarial techniques, often fall short in producing inputs that introduce novel characteristics or maintain realism, thus limiting their effectiveness in the comprehensive DNN assessment [23, 134] and differential testing of DNNs. To address this challenge, we introduce *DiffGAN*, a differential testing approach for DNN models that leverages the capabilities of GANs and NSGA-II to drive the generation of diverse triggering inputs. The use of a GAN model for this purpose results from its proven efficacy in generating realistic and new test inputs across different DNN testing scenarios [61, 70, 81, 166]. When adequately trained, GANs can produce novel inputs that are distinct from the training data and belong to the same input domain—inputs that can be classified within the existing training labels.

In a typical GAN architecture, two deep neural networks operate in tandem: a generator, with the goal of creating new images from a random latent vector exploring the GAN's latent space, and a discriminator, whose purpose is to determine whether a given image is produced by the generator or a direct sample from the training dataset. The ultimate goal is for the generator to produce new inputs to ensure realism and compliance with the training distribution.

Note that the latent space in the context of GANs refers to a high-dimensional space that encodes a compressed representation of the data on which the GAN is trained [126, 132].

Since the models under test often exhibit similar high accuracy levels, GANs alone might not be effective in generating diverse triggering inputs that reveal behavioural disagreements between highly accurate models. To overcome these limitations, we leverage

NSGA-II to effectively explore the large GAN's latent space, thereby enhancing the generation of diverse triggering inputs. More specifically, our genetic search aims at finding latent vectors that maximize the generation of inputs produced by the GAN that are both diverse and reveal behavioral disagreements between pairs of DNN models under test.

We also rely on NSGA-II since it is specifically adapted to our multi-objective search problem. It is widely used in the literature and has shown good performance in solving many search-based problems in DNN testing [4, 73, 149]. Many existing works in the literature have shown the effectiveness of NSGA-II in navigating the complex latent space of GANs and solving multi-objective optimization problems [133]. Unlike algorithms such as MOSA, the Many Objective Sorting Algorithm [123], which approaches each fitness function in isolation, NSGA-II tries to find solutions that offer diverse trade-offs among multiple objectives. More specifically, our search is driven by two objectives: (1) maximizing the difference between the output probability vectors of the inputs to generate triggering inputs, and (2) maximizing the diversity of generated inputs by the GAN model.

We illustrate in Figure 4.1 our test generation approach. We describe in the following the different steps of our approach along with the different components of our search algorithm.

4.3.1 GAN Training Dataset

To effectively train our GAN model, we use the testing dataset available for the models under test. Indeed, access to the original training datasets of these models is often restricted in many real-world scenarios, particularly with proprietary models, hence our reliance on testing datasets.

To enrich the diversity of the GAN's training data, we enhance the testing dataset of the models under test with five realistic image transformations, encompassing operations such as rotation, shifting, shearing, and adjustments to the images' brightness and blurriness. These transformations are applied to every image in the dataset, following the same procedures and parameters established by Shorten *et al.* [135], which serve to ensure (1) the validity of the generated inputs and (2) labels preservation by these transformations. Consequently, the augmented dataset feeding into the GAN's training process is expanded to five times its initial size, due to the application of five distinct image transformations on each original test input. Finally, we merge the original testing dataset with the augmented data to build the final GAN training dataset.

4.3.2 GAN Model Training

We use a GAN model [126] in our approach which encompasses a generator to produce new images and a discriminator whose role is to distinguish between the generated images and real images from the GAN's training dataset.

There are unique challenges in evaluating generators in GANs, which differ fundamentally from standard DNNs. In the realm of traditional DNNs, models' predictions across

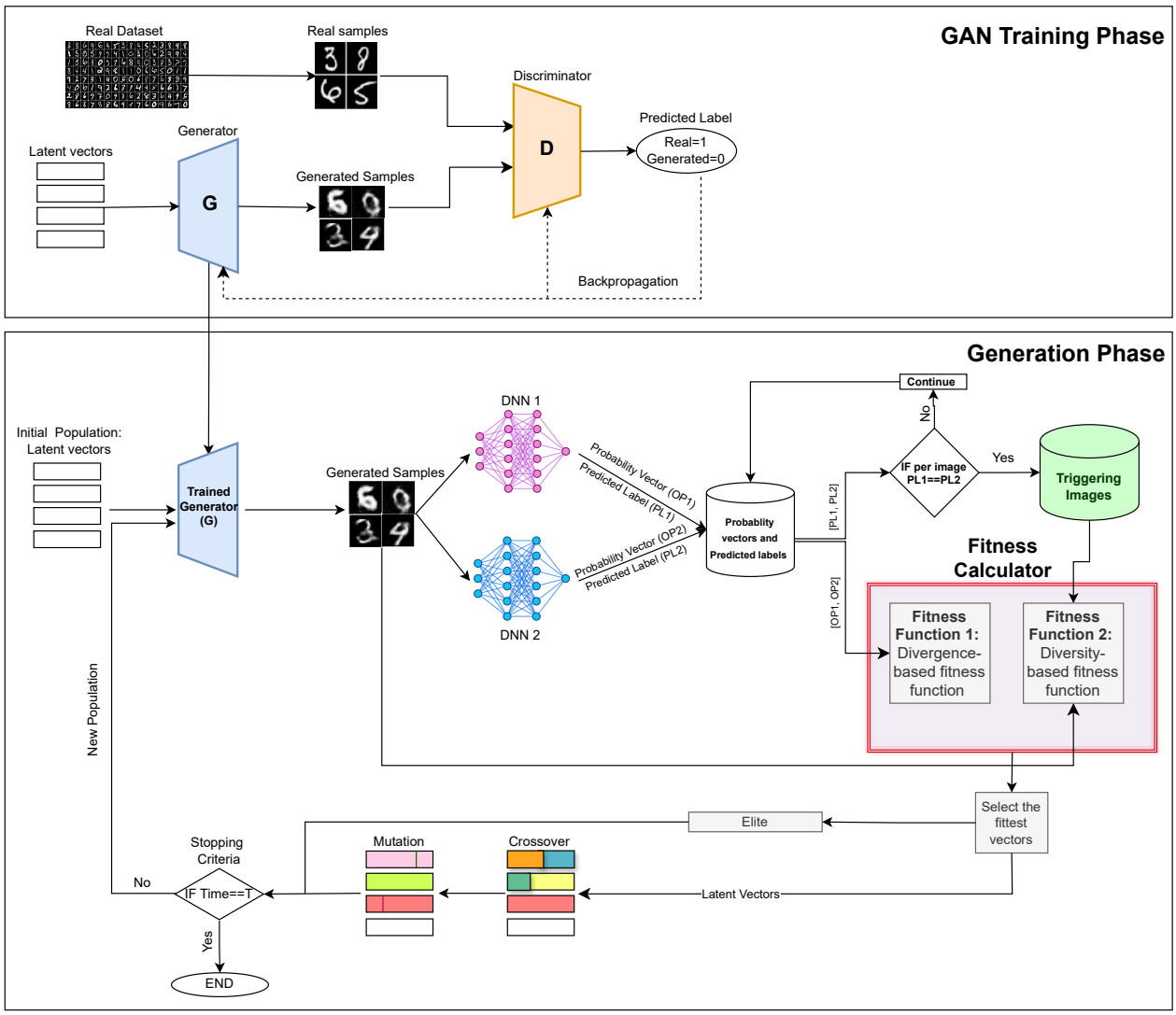


Figure 4.1: Overview of DiffGAN

various tasks are compared against a ground truth, which consists of the actual outcomes or labels. However, generative models, such as the generator component of a GAN, follow a different paradigm. Their output is not a label but a new, synthetic image that did not previously exist in the dataset. As a result, traditional metrics based on accuracy against a ground truth are not applicable.

To evaluate the generator's performance effectively, we employed two SOTA metrics designed for evaluating GAN's performance: the *Fréchet Inception Distance* (FID) and the *Kernel Inception Distance* (KID) [156].

The FID metric quantifies the difference between the distribution of GAN-generated images and the distribution of actual images in the training dataset. Note that in the literature [77], FID scores typically range from zero to a few hundred, with lower scores indicating higher quality in image generation, suggesting better GAN performance.

The KID measures the similarity between the distributions of real and generated images by computing the maximum mean discrepancy between their feature representations [156]. Unlike FID, KID is calculated using a kernel function, making it more robust to variations in sample size. KID scores often range from zero to one. Scores nearing zero for KID similarly reflect superior quality in the images generated by the model.

These metrics were chosen for their strengths in assessing the quality and diversity of the images generated by GANs. FID is sensitive to differences in image distributions, making it effective for evaluating the overall alignment between generated and real data distributions. On the other hand, KID is robust against varying sample sizes, providing accurate results even when only a few images are available [156]. Together, FID ensures that the generated images closely match the real distribution, while KID maintains robustness across different dataset sizes, making the two metrics complementary in their ability to provide a thorough evaluation of GAN-generated images [156].

We train the GAN model using our training dataset and perform hyperparameter tuning using the Optuna framework [2], which is known for its efficiency in automating the optimization process for machine learning models [6]. Optuna was selected for its user-friendly interface, capability to efficiently search ML models hyperparameter spaces, and support for a variety of search strategies. We also employ Optuna's pruner feature for stopping the evaluation of unpromising trials, which allows for expediting the optimization process. Our tuning process targets hyperparameters, learning rates, epochs, and batch sizes, and aims to minimize the FID and KID scores.

During each iteration of the Optuna optimization process, a GAN model is trained, and 10,000 images are generated by feeding 10,000 randomly sampled latent vectors into the trained GAN. These generated images are compared with a randomly selected subset of 10,000 real images from the GAN's training dataset. To ensure the reliability of our results and reduce the impact of randomness, this process is repeated ten times per iteration. The average FID and KID scores across these ten runs are computed for the currently trained GAN model, and Optuna uses these scores to guide the search for the next set of hyperparameters, aiming to minimize FID and KID. Finally, the GAN model with the lowest average scores is selected as the optimal model.

4.3.3 Generation of Triggering Inputs

After training the GAN, we employ the NSGA-II search algorithm to effectively explore the GAN's latent space, guiding the search towards the generation of diverse triggering input images. In this approach, a genetic algorithm manipulates and evolves a population of latent vectors through successive crossover and mutation operations. This evolutionary procedure iteratively refines these latent vectors towards the objectives. The selection process is guided by specific fitness functions, which, in our context, prioritize the selection of latent vectors that maximize the generation of diverse triggering input images. In other words, through successive iterations, we expect to increasingly produce test inputs that

[2]https://optuna.readthedocs.io/en/stable/

reveal a high number of diverse behavioral disagreements between the DNN models under test. To properly translate the generation process into a search problem using NSGA-II, we need to define the following elements.

4.3.3.1 Individuals and Initial Population

In genetic search, the initial population contains individuals consisting of a set of elements called genes. These genes are connected and form an individual also referred to as a solution. In our context, the initial population contains 100 latent vectors, each of size 100, as recommended in the literature [59]. We use random selection to build our initial population of individuals, thereby ensuring a diverse starting point for our genetic search.

4.3.3.2 Fitness Functions

The main objective of our approach is to generate test inputs that reveal diverse behavioral disagreements between the DNN models under test. To achieve this, our search process is guided by two fitness functions. The first function aims to maximize the difference between the output probability vectors of the DNN models under test when fed with the generated input images. By maximizing the difference between the output probability vectors, we are more likely to trigger more pronounced behavioral disagreements between the DNN models under test. The second function aims to maximize the diversity of the generated inputs since generating redundant inputs is a waste of computational resources, particularly when faced with constraints such as limited testing budgets and high labeling costs for testing data.

This strategy ensures that each input contributes to assessing the models' accuracy and identifying conditions where one model outperforms another. By combining both fitness functions, we aim to generate diverse test input images that maximize the chances of revealing distinct behavioral disagreements between the DNN models under test.

4.3.3.2.1 Divergence-based Fitness Function

This fitness function estimates the divergence between the output probability vectors of the models under test when fed with the same input image generated by the trained GAN. More specifically, the fitness function takes as input a GAN latent vector Z responsible for generating the candidate test input image. We feed the models under test with such image and compute the Euclidean distance between the resulting output probability vectors Y_{m1} and Y_{m2}. The goal of maximizing this distance is to increase the probability of selecting latent vectors that lead to the generation of input images revealing significant behavioral disagreements between the models under test.

The Euclidean distance serves as a straightforward measure of the magnitude of discrepancy between the model predictions, capturing how far apart the models' outputs are in the probability space. The choice of Euclidean distance over other metrics, such as KL divergence [76], is motivated by its efficiency in terms of computation cost.

However, the use of Euclidean distance or any other distance metrics alone has a limitation: they do not account for the differences in the predicted labels by the models. This means that while a high Euclidean distance indicates a discrepancy in the output probabilities, it does not necessarily reflect a disagreement in the models' final predicted labels. This is particularly critical when the primary objective lies in searching for triggering inputs that lead to different label predictions by the models under test.

We add the following example to illustrate the above limitation. Let us consider that for a generated image (image 1), the Euclidean distance calculated between the output probability vectors P1 = [0.05,0.05,0.05,0.85] from the first model under test (Model 1) and Q1 = [0.15,0.15,0.15,0.55] from the second model under test (Model 2) is 0.3. Further, both models predict the same class for image 1. Then, let us consider another image (image 2), where the calculated Euclidean distance between the output probability vectors P2 = [0.1,0.2,0.3,0.4] from Model 1 and Q2 = [0.15,0.25,0.35,0.05] from Model 2 is also 0.3, despite these models predicting different classes. We obtain the same Euclidean distances for both images though clearly image 2 is more interesting given our objectives.

Using a standard Euclidean distance in the context of differential testing is therefore not optimal and indicates the need to refine the fitness function, to ensure it not only considers the magnitude of input differences but also the impact of changes in the predicted labels, indicating different models predictions.

To address this limitation, we modified the fitness function to reflect differences in model predictions. More specifically, given a GAN latent vector, if the models output different labels for a GAN-generated image, the fitness function increments the Euclidean distance by one to assign a higher score for triggering input images. Otherwise, it returns the Euclidean distance between the two output probability vectors as illustrated in Equation 4.1:

$$D(z) = \begin{cases} \sqrt{\sum_{i=1}^{n} (Y(i)_{m1} - Y(i)_{m2})^2} + 1 & \text{if the models output} \\ & \text{different labels} \\ \sqrt{\sum_{i=1}^{n} (Y(i)_{m1} - Y(i)_{m2})^2} & \text{otherwise} \end{cases} \tag{4.1}$$

Where z represents a GAN latent vector, $D(z)$ denotes the divergence of z, and Y_{m1} and Y_{m2} are the output probability vectors of the models under test.

To illustrate the practical impact of our modified fitness function, reusing the examples above, Image 1, a triggering input, receives a fitness score of 1.3, whereas Image 2, a non-triggering input, is assigned a score of 0.3. This adjusted fitness function more accurately characterizes inputs that are likely to reveal significant behavioral differences between models, prioritizing inputs that not only produce different probability vectors but also result in distinct classifications by the models.

4.3.3.2.2 Diversity-based Fitness Function

Given a latent vector, this fitness function computes the contribution of the corresponding GAN-generated image to the diversity of the collected triggering inputs during the search

process. To do so, the diversity-based fitness function computes the minimum cosine distance score between the pixel values of this generated image and those of the set of collected triggering inputs during the search.

To reduce the computational complexity of comparing each generated image against all collected triggering inputs at each iteration of the search process, we approximate such distance by applying a clustering algorithm to group similar inputs. Then, we only return the minimum cosine distance between the generated image and the representative triggering inputs for all clusters. Preliminary experiments confirmed that, despite the cost of clustering, significant time could be saved in computing similarity scores of new images, thus leading to generating more triggering inputs. More specifically, at each iteration of the search process, we apply the HDBSCAN [108] clustering algorithm on the set of the collected triggering input images to group similar inputs in one cluster. We select HDBSCAN for its ability to identify clusters of varying densities without requiring the prior specification of the number of clusters. This adaptability makes HDBSCAN particularly suited in our context.

More specifically, at each iteration of the search process, we apply the HDBSCAN [108] clustering algorithm on the set of the collected triggering input images to group similar inputs in one cluster. We select HDBSCAN for its ability to identify clusters of varying densities without requiring the prior specification of the number of clusters. This adaptability makes HDBSCAN particularly suited in our context.

Prior to clustering, we implement UMAP for dimensionality reduction, which assists in forming more distinct clusters [4, 9] at a lower cost. Subsequently, within each cluster, we identify the representative input—specifically, the input that has the maximum probability of association with its corresponding cluster[3]. The fitness function then calculates the diversity of the GAN-generated image by measuring the cosine distances between this image and the representatives of all clusters. Finally, the diversity-based fitness function returns the minimum of these cosine distances to quantify the diversity contribution of the image as illustrated in Equation 4.2:

$$Diversity(z) = \min_{i=1}^{m} CosineDistance(image_z, image_i) \tag{4.2}$$

where z is the latent vector, $image_z$ is the generated image by GAN given z, $image_i$ is the representative image of the i^{th} cluster formed by HDBSCAN, and m is the total number of clusters.

4.3.3.2.3 Multi-Objective Search

We need to maximize in our search the two aforementioned fitness functions to drive the generation of diverse triggering inputs. This is therefore a multi-objective search problem that can be formalized as follows:

[3]HDBSCAN Documentation—retrieved October 5, 2024, from `https://hdbscan.readthedocs.io/en/latest/soft_clustering_explanation.html`

$$\max_{z \subset \mathbb{P}} Fitness(z) = (Divergence(z), Diversity(z)) \tag{4.3}$$

where $\mathbb{P}$ is the population of latent vectors, z is a latent vector, function $Fitness : z \rightarrow \mathbf{R}^2$ consists of two real-value objective functions $(Divergence(z), Diversity(z))$, and $\mathbf{R}^2$ is the objective space of our optimization problem.

4.3.4 Genetic Operators

We describe below the two genetic operators in our input generation approach.

4.3.4.1 Crossover

The crossover operator takes as input two parents (i.e., two latent vectors) and generates new offspring by combining the selected parents. Similar to existing works that employ NSGA-II on GAN latent vectors [133, 163], we use the simulated binary crossover (SBX) in our approach as it is suitable for real-coded individuals (i.e., continuous optimization problem) [45, 133, 163]. Unlike traditional binary crossover operators, which are designed for discrete or binary representations, SBX operates directly on the real values of the latent vectors. Let V_1 and V_2 be the two selected parents for crossover. We provide next an example of the resulting offspring after applying crossover:

$$Offspring_1(i) = 0.5[(1+\beta)V_1(i) + (1-\beta)V_2(i)] \tag{4.4}$$

$$Offspring_2(i) = 0.5[(1-\beta)V_1(i) + (1+\beta)V_2(i)]$$

where β is a random variable controlling the extent of the crossover and i denotes the index of the latent vectors' components.

4.3.4.2 Mutation

Our genetic search also employs a mutation operator to further explore the search space.

Following existing works that use genetic algorithms in GAN latent space [133, 163], we also adopt polynomial mutation [44] to address the continuous nature of the latent space.

Polynomial mutation applies polynomial-distributed noise to the genes forming the selected individuals in our search. By applying such noise to the genes, it allows for the exploration of nearby solutions while still maintaining proximity to the original solution. This is also essential for efficiently navigating the complex and high-dimensional search space of GAN latent vectors. Polynomial mutation helps generate diverse solutions by introducing variability in the population of the search. This diversity is crucial for escaping local optima and finding globally optimal or near-optimal solutions.

Algorithm 3: High-level NSGA-II algorithm

Input : Initial population of individuals ($ind \in P$), maximum running time (T_{max}), trained GAN, two DNNs under test

Output: triggering inputs α

1 $\alpha \leftarrow \emptyset$
2 $T_{start} \leftarrow$ current time ;
3 **while** *current time* - $T_{start} \leq T_{max}$ **do**
4 $\quad P_{new} \leftarrow \emptyset$
5 $\quad P_{Cross} \leftarrow \emptyset$
6 $\quad P_{Cross} \leftarrow Cross(P, CrossRate)$
7 $\quad P_{new} \leftarrow Mutate(P_{Cross}, MutationRate)$
8 $\quad$ **foreach** $ind \in P$ **do**
9 $\quad\quad \text{image}_{\text{gen}} \leftarrow \text{GAN}(ind)$;
10 $\quad\quad [OP1, OP2] \leftarrow \text{Model1}(\text{image}_{\text{gen}}), \text{Model2}(\text{image}_{\text{gen}})$
11 $\quad\quad$ **if** *generated image is triggering* **then**
12 $\quad\quad\quad \alpha \leftarrow \alpha \cup \{ind\}$
13 $\quad P \leftarrow Select(P, P_{new}, Fitness)$
14 $\quad gen = gen + 1$;
15 **return** α

4.3.5 Search Algorithm

The main objective of our search algorithm is to maximize the generation of diverse triggering inputs to effectively identify the discrepancies between the DNN models under test. A high-level overview of our NSGA-II search process is outlined in Algorithm 3. Assuming P represents the initial population, consisting of a set of $|P|$ random latent vectors, the algorithm initiates an iterative procedure, executing the following steps in each generation until the maximum execution time (T) is reached (lines 3-14). The search process includes the following steps. First, we create a new empty population P_{new} (line 4). Second, based on predefined crossover and mutation rates, we create offspring using crossover ($Cross$) and mutation ($Mutate$) operators, incorporating newly created individuals to P_{new} (lines 5-7). Then, for each individual, we generate the corresponding image with the trained GAN and store the triggering images in an archive (lines 8-12). Next, we calculate the fitness scores of the new population by computing the diversity- and divergence-based fitness scores of each individual in the population. We employ tournament [112] selection ($Select$) to choose individuals for survival (line 13). Lastly, we update the population by incorporating the fittest individuals, proceeding to the next generation (line 14). The search terminates when the maximum execution time (T) is reached and returns the archive of generated triggering inputs (line 15).

We set the NSGA-II parameters in our search algorithm as follows. The crossover rate $CrossRate$ is equal to 90% and the mutation rate $MutationRate$ is equal to 10%, given recommended values in the literature [78, 111, 139]. Finally, we used Google Colab and the Pymoo library [20] to implement our genetic search.

4.3.6 Filtering Invalid Inputs

Despite performing well on standard metrics such as FID and KID, *DiffGAN*, like any input generation process, might still produce some invalid inputs. To address this issue, we developed a two-stage filtration process for invalid triggering inputs, applied after the final generation step. This process incorporates two state-of-the-art filtration techniques to ensure the quality and validity of the generated images. Due to the computational cost associated with the filtration process, it is applied only after the final generation step rather than throughout the generation iterations with NSGA-II, optimizing both the testing budget and output quality.

4.3.6.1 Discriminator-Based Filtering

The first stage of our filtration process leverages the trained discriminator from our GAN model. We continuously evaluate our GAN's performance during training to ensure it reaches a high level of performance (in terms of GAN's evaluation metrics). After training, we use the discriminator to filter out unrealistic or invalid images. We should recall that, when properly trained, the discriminator can be highly effective at distinguishing and removing invalid images that do not align with the desired output distribution. Since the discriminator is trained to differentiate between real and generated images, it serves as a straightforward filter, ensuring that only images deemed valid are retained for further analysis.

This approach aligns with findings from the literature [82,160,162], which demonstrate the utility of using discriminators not only during the training process but also for post-generation validation. By leveraging the capabilities of our trained discriminator, we ensure a context-specific and reliable validation process that is tailored to the dataset on which the GAN was originally trained.

4.3.6.2 SSIM-Based Invalidity Detection

As a complementary validation method, we incorporated Structural Similarity Index (SSIM)-based detection. SSIM is a metric that evaluates the similarity between two images by comparing their luminosity, contrast, and structure. In our method, each generated image is compared with its most similar real counterpart from the original training set. If the SSIM score is low, it indicates that the generated image is likely to be invalid due to its significant divergence from real data. A low similarity score indicates significant visual differences from its most similar real image, making it likely to be unrealistic or invalid.

To determine the SSIM threshold for filtering invalid inputs, we generated 1,000 test inputs using the trained GAN. For each generated input, we calculated its SSIM score by comparing it to the most similar image from the original training dataset. After manually labeling each input as either valid or invalid, we analyzed the distribution of SSIM scores for both categories. This procedure was applied to both datasets used in our experiments.

We consistently observed that invalid inputs exhibited SSIM scores below 40%. Based on these findings, we selected 40% as the threshold for filtering out invalid inputs.

By integrating discriminator-based filtering with SSIM-based invalidity detection, we developed a two-stage filtration system for enhancing the quality and validity of GAN-generated images. The discriminator provides the first layer of defense against invalid images, while SSIM-based filtration, further refines this process by evaluating the structural similarity between each generated image and its closest counterpart in the original training data. This dual filtration approach significantly reduces the risk of maintaining invalid triggering inputs, leading to more useful *DiffGAN* outputs.

4.4 Empirical Evaluation: Questions and Design

4.4.1 Research Questions

Our empirical evaluation is designed to answer the following research questions, covering the effectiveness of *DiffGAN* in terms of finding triggering inputs, their diversity and validity, and our ability to guide the training of a model selection mechanism that better learns the conditions under which a model fares best.

RQ1. Do we generate more triggering inputs than *DRfuzz* with the same testing budget?

We aim to study in this research question the effectiveness of *DiffGAN* in producing more triggering inputs compared to a baseline, namely *DRfuzz*, within the same testing budget. Our choice of *DRfuzz* is carefully justified below, along with a description of the technique. The testing budget in our context is defined by a predetermined and fixed execution time. Generating large numbers of triggering inputs that reveal behavioral disagreement between DNN models is essential to accurately learn when each of the models can be expected to fare better. Relying on too few triggering images would not enable *DiffGAN* to identify the subtle differences between models, especially those with similar, high accuracy. Therefore, our research question aims to evaluate and contrast the efficiency of *DiffGAN* against *DRfuzz* in terms of their ability to generate more triggering inputs within time constraints.

RQ2. How do *DiffGAN* and *DRfuzz* compare in terms of generating valid triggering inputs within the same testing budget?

Generating large numbers of triggering inputs is not sufficient. To the largest extent possible, they also need to be valid. We, therefore, aim to compare the validity of the triggering inputs produced by *DiffGAN* and those generated by *DRfuzz*. The concept of input validity is crucial for DNN testing in general and differential testing in particular since it ensures that the generated inputs are both relevant and meaningful. More specifically, they should be valid and belong to the targeted input domain. A valid input for a DNN model is defined as one that human experts within the domain can recognize and categorize accurately, using labels from the domain [128]. For example, in the context of handwritten

digit recognition, an image is considered valid if it can be recognized as a digit by a human expert, despite potential imperfections. We, therefore, aim to compare in this research question the proportions of valid inputs that *DiffGAN* and *DRfuzz* generates. This comparison is crucial because generating a large number of triggering inputs is only beneficial if these inputs are to a large extent valid, as defined above.

RQ3. Does *DiffGAN* find more diverse triggering inputs than *DRfuzz*?

This research question centers on the diversity of the triggering inputs generated by *DiffGAN*, compared to those produced by *DRfuzz*. Diversity in this context refers to the range of scenarios and conditions under which the inputs cause the DNN models to exhibit behavioral differences. Further, a diverse set of triggering inputs is indicative of the approach's ability to explore and expose a broad spectrum of potential weaknesses across the models under test. Hence, we aim to assess not only the quantity and validity of the triggering inputs but also the breadth of scenarios these inputs represent. **RQ4. Can training of an ML-based model selection mechanism be guided more effectively using the triggering inputs generated by *DiffGAN* compared to those produced by *DRfuzz*?**

We aim in this research question to determine whether the triggering inputs generated by *DiffGAN* can more effectively guide the training of a model selection mechanism, allowing it to learn the conditions under which one DNN model outperforms another. In many real-world scenarios, it is common to encounter multiple DNNs that perform similarly in terms of overall accuracy, but exhibit significant variations in their performance under specific input conditions. This presents a challenge for developing a model selection mechanism that can dynamically choose the most suitable model based on the input characteristics without having to run all models in real-time. An effective solution to this problem is to use an ML-based selection mechanism, which can predict which of the available models will perform better for a given input. However, training such an ML-based model selection mechanism requires diverse and informative training data that captures and reveals a wide range of behavioural differences between the models. This is where the generation of triggering inputs becomes critical for the selection mechanism to more effectively learn under which condition one model is preferable over the other. Indeed, the effectiveness of the selection mechanism heavily depends on the quality of the triggering inputs used for training. Therefore, we aim to explore in this RQ whether *DiffGAN*, can better guide the training process compared to *DRfuzz*. It is important to clarify that we do not intend to propose a novel ensemble method or voting mechanism in this thesis. Instead, this research question focuses on investigating an important application scenario, that is studying whether *DiffGAN* helps learn which model fares better under various input conditions and thus train a model selection mechanism.

4.4.2 Baseline Approach

Since our main focus is to generate triggering inputs that reveal behavioral disagreements between pairs of DNN models, we selected the baseline approaches against which to compare our approach according to the following criteria.

First, the baselines should involve the generation of test inputs for more than one DNN model to reveal differences in DNNs behaviors. This criterion ensures that the baselines under comparison are aligned with our core objective of identifying divergences in DNNs' behavior.

Second, the baseline approaches should be generally applicable and should not make any assumption on the structural similarity between the DNN models under test (e.g., the approach should not only be applicable to retrained models or quantized ones).

Third, the baselines should be peer reviewed, and open-source implementations of the proposed solutions should be provided. This requirement enables rigorous replicability, thus allowing for a fair comparison of *DiffGAN*'s effectiveness against established benchmarks.

Finally, we exclude baselines that limit the generation of triggering inputs based on the number of initial test inputs available. Specifically, some approaches cap the maximum number of triggering images to match the size of a seed set—a small subset selected from the initial test dataset. This constraint can introduce bias into comparative evaluations. When methods are assessed within a fixed testing budget, often measured in terms of execution time, such limitations can lead to unfair performance comparisons. If one method is restricted by the number of initial inputs while others are not, it does not provide a level playing field for evaluation. Furthermore, in practical scenarios, the availability of initial test inputs might be limited. This scarcity makes it challenging to gauge a baseline method's ability to generate additional triggering inputs beyond the initial set. From a practical standpoint, it is also difficult to accurately predict the maximum number of triggering inputs required to uncover all possible behavioral disagreements between the DNNs under test. Without knowing this number in advance, setting a cap based on initial seed inputs could hinder the thoroughness of the testing process.

Based on the first criterion we identified *DiffChaser* [154], *BET* [148] *Diverget* [158], *DRfuzz* [162], and *DFlare* [143], as candidates. However, *BET* and *Diverget* were excluded due to their lack of open-source implementations. Additionally, *DFlare* was not selected because its approach to generating triggering inputs is constrained by the number of initial inputs in the testing dataset. The method applies multiple mutation rules to each initial test input to find a triggering input. Once a triggering input is found for a particular initial input, the method stops mutating that input and proceeds to the next one. This process limits the maximum number of possible triggering inputs to the number of initial inputs in the testing dataset, as it generates at most one triggering input per initial input.

We also decided not to include *DiffChaser* in our experiments, as *DRfuzz* demonstrated superior performance in generating a greater number of trigger inputs [162]. Consequently, we only selected *DRfuzz* as a baseline to evaluate the effectiveness of our proposed approach.

DRfuzz is an approach designed for generating test inputs that uncover diverse regression faults in updated versions of DNN models. In the context of DNNs, a regression fault occurs when an input, correctly predicted by a previous model version, leads to incorrect predictions in the updated version. *DRfuzz* initiates this process with seed inputs, upon which it applies various mutation rules to produce new inputs that trigger regression

Table 4.1: MNIST Models' Scenario

Pair	Model	Architecture	Training Data	Optimizer	Parameters	Accuracy
Pair 1	Model 1	**4 Conv, 20 layers**	MNIST Org (60,000 images)	Adam (lr=0.001)	#Param: 330,730, Batch: 128, Epoch: 10	**99.26%**
	Model 2	**7 Conv, 19 layers**	MNIST Org (60,000 images)	Adam (lr=0.001)	#Param:: 327,242, Batch: 128, Epoch: 10	**99.50%**
Pair 2	Model 3	7 Conv, 33 layers	**40000 Org + (10% shifts, 15% zooms/rotations) on 20,000 MNIST**	RMSprop (lr=0.001)	#Param: 696,402, Batch: 128, Epoch: 10	**99.09%**
	Model 4	7 Conv, 33 layers	**40000 Org+ (5% shifts, 10% zooms/rotations) on 20,000 MNIST**	RMSprop (lr=0.001)	#Param: 696,402, Batch: 128, Epoch: 10	**99.55%**
Pair 3	Model 5	LeNet-5	MNIST Org (60,000 images)	**SGD (lr=0.01)**	#Param: 44,426, Batch: 32, Epoch: 10	**97.04%**
	Model 6	LeNet-5	MNIST Org (60,000 images)	**Adam (lr=0.001)**	#Param: 44,426, Batch: 32, Epoch: 10	**98.93%**
Pair 4	Model 7	AlexNet	MNIST Org (60,000 images)	Adam (lr=0.00001)	**#Param: 24,733,898, Batch: 128, Epoch: 9**	**98.85%**
	Model 8	AlexNet	MNIST Org (60,000 images)	Adam (lr=0.0001)	**#Param: 24,733,898, Batch: 256, Epoch: 14**	**99.33%**

Table 4.2: CIFAR-10 Models' Scenario

Pair	Model	Architecture	Training Data	Optimizer	Parameters	Accuracy
Pair 1	Model 1	**ResNet18**	Cifar10 Org (60,000 images)	SGD (lr = 1e-1)	#Param: 11183562, Batch=128, Epoch=200	**83.38%**
	Model 2	**ResNet20**	Cifar10 Org (60,000 images)	SGD (lr = 1e-1)	#Param: 274442, Batch=128, Epoch=200	**85.71%**
Pair 2	Model 3	Inception	**40000 Org + (15% shifts, 5%zooms, 10%roations) on 20,000 CIFAR-10**	SGD (lr = 1e-1)	#Param: 5984858, Batch=128 , Epoch=120	**93.02%**
	Model 4	Inception	**40000 Org + (10% shifts, 10%zooms, 15%rotations) on 20,000 CIFAR-10**	SGD (lr = 1e-1)	#Param: 5984858, Batch=128, Epoch=120	**93.27%**
Pair 3	Model 5	Inception	Cifar10 Org (60,000 images)	**SGD (Nesterov=True)**	#Param: 5984858, Batch=128, Epoch=150	**89.10%**
	Model 6	Inception	Cifar10 Org (60,000 images)	**Adam (lr = 1e-3)**	#Param: 5984858, Batch=128, Epoch=150	**92.78%**
Pair 4	Model 7	ResNet18	Cifar10 Org (60,000 images)	SGD (lr = 1e-1)	**#Param: 11183562, Batch=250, Epoch=150**	**80.85%**
	Model 8	ResNet18	Cifar10 Org (60,000 images)	SGD (lr = 1e-1)	**#Param: 11183562, Batch=128, Epoch=200**	**83.38%**

faults. To ensure the validity and relevance of these inputs, *DRfuzz* employs the discriminator component of a trained GAN to filter invalid inputs. A customized reward function in *DRfuzz* assigns priority scores to each mutation rule based on their effectiveness in generating diverse triggering inputs from the seed inputs. This selection strategy emphasizes mutation rules that are more likely to uncover behavioral disagreements between model versions. To align *DRfuzz* with our specific research objectives, we have customized the source code of *DRfuzz* accordingly. We refined the approach to keep not only the inputs that one model predicts correctly while being mispredicted by the other model, but also any inputs that show behavioral disagreements between the models under test.

4.4.3 Datasets and Models

We consider different combinations of datasets and models in our empirical evaluation. More specifically, we consider MNIST [47] and Cifar-10 [8], two SOTA image recognition datasets. As shown in Figure 4.2, the MNIST dataset consists of 70,000 black-and-white images depicting handwritten digits, split into a training set of 60,000 images and a test set of 10,000 images. The Cifar-10 dataset comprises 60,000 color images distributed across ten categories (Figure 4.3), including 50,000 images designated for training and 10,000 images for testing. These datasets were employed in conjunction with 16 different DNN models, which included both custom-developed and widely used architectures such as ResNet, LeNet-5, AlexNet, and inception models [93, 96, 142]. Comprehensive details about the combinations of datasets and models we used, along with their accuracy, are outlined Tables 4.1 and 4.2.

As shown in the tables, our experimental study spanned four distinct and common sources of variation per dataset across DNN models, including the model structures, training data, optimizers and hyperparameters. Our experiments thus cover 16 different DNN models, each selected to evaluate the effectiveness of our test generation approach. For each

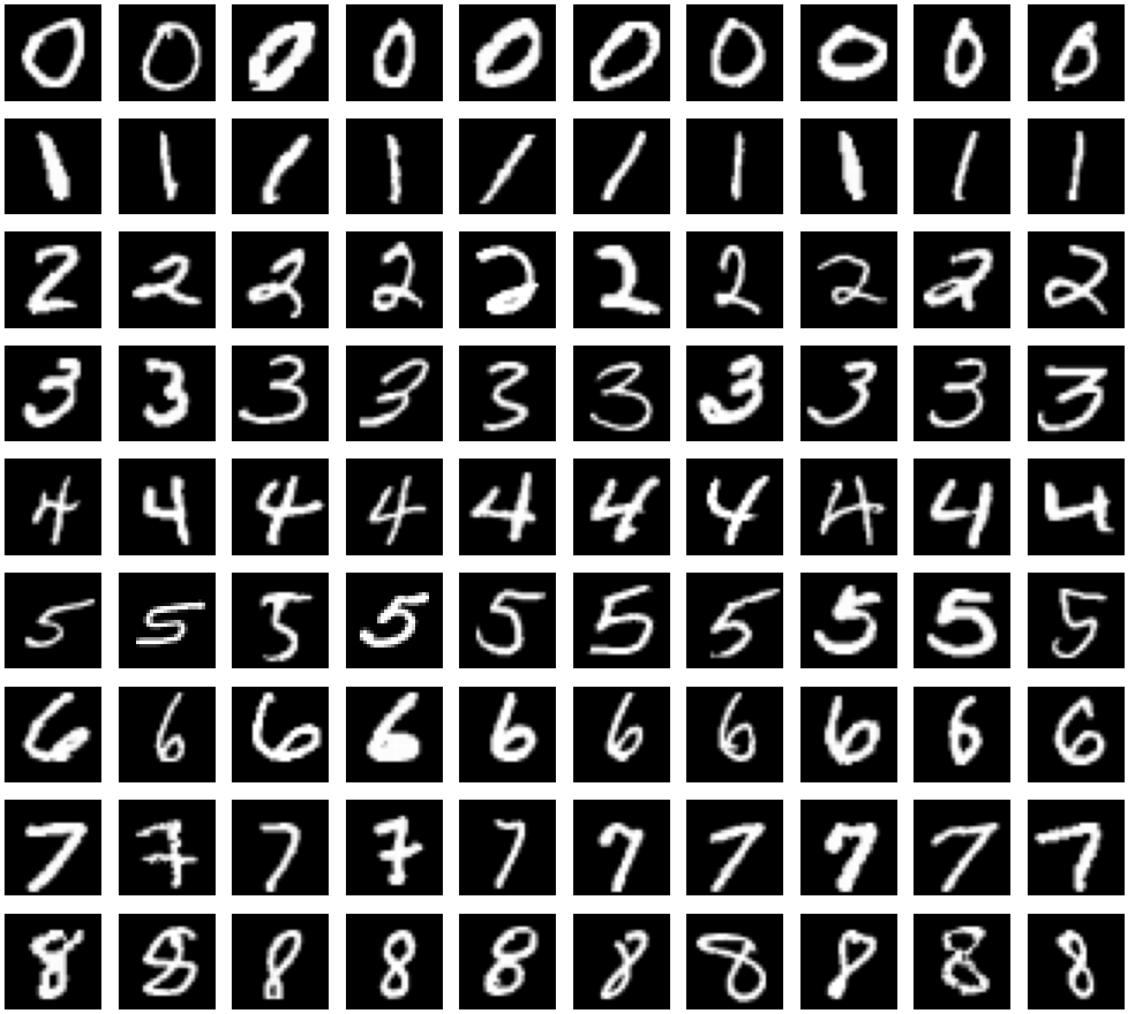


Figure 4.2: Examples of original MNIST images.

model pair, the differences in DNN models are highlighted in bold in Tables 4.1 and 4.2. Typical differences across models include:

1. **Structural Variations:** We considered models with different architectures. For example, in the first configuration in Table 4.1, the two models, although both trained on the MNIST dataset, have a different architecture. More specifically, the first model featuring four convolutional layers among a total of 20 layers, achieved a 99.26% accuracy on the MNIST test dataset. In contrast, the second model, with seven convolutional layers out of a total of 19 layers, achieved a 99.50% accuracy.

2. **Training Data Variations:** Our analysis also covered models that are architecturally identical but trained on different training datasets. For example, as depicted in the second configuration in Table 4.1, the first DNN model was trained using 40,000 images from the MNIST training dataset and 20,000 MNIST images modified by 10% shifts and 15% zooms/rotations, achieving 99.09% accuracy. Conversely, the

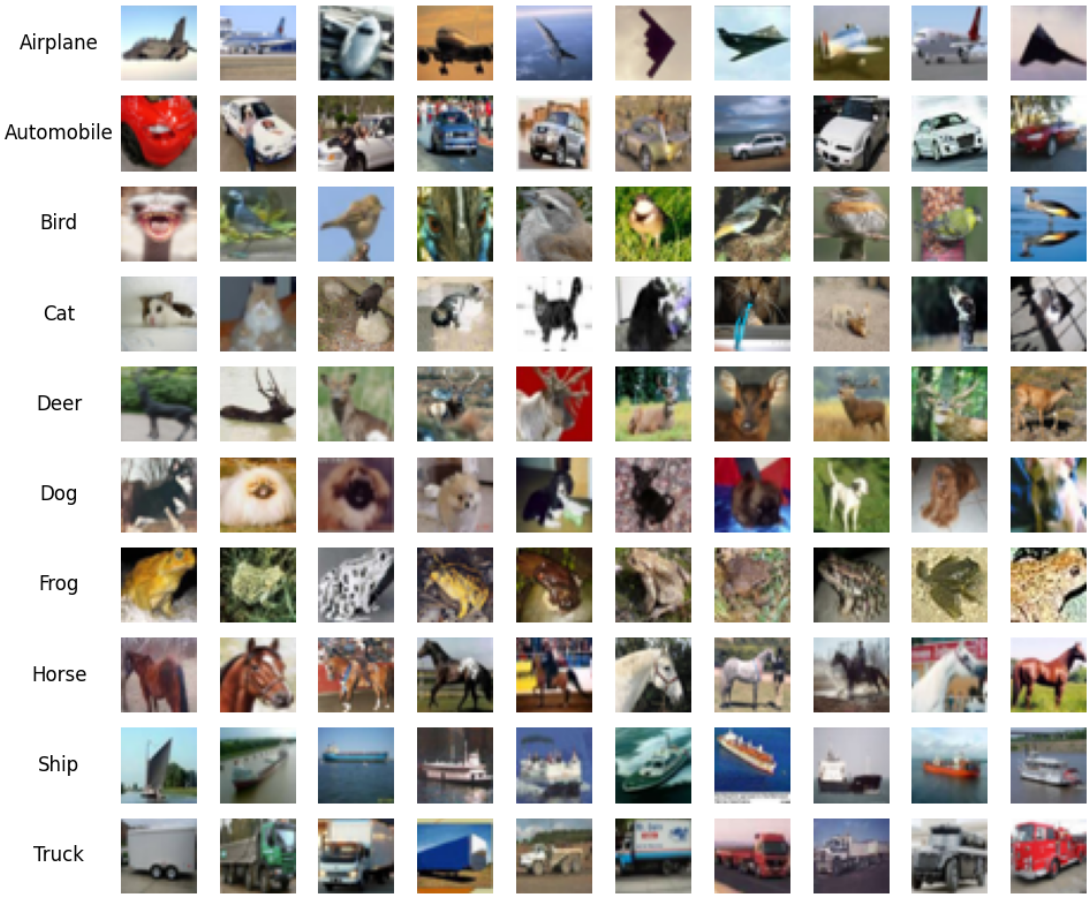


Figure 4.3: Examples of original Cifar-10 images.

second model was trained on 40,000 images from the MNIST training dataset and 20,000 MNIST images adjusted by 5% shifts and 10% zooms/rotations, yielding a slightly higher accuracy of 99.55%.

3. **Optimizer Variations:** We also considered the scenario where two models are trained with different optimizers. For instance, in a case study involving the Cifar-10 dataset (see Pair 3 in Table 4.2), two inception models were trained using distinct optimizers. Specifically, one model was trained with the SGD optimizer and reached an accuracy of 89.10%, whereas the other model was trained with the Adam optimizer, reaching an accuracy of 92.78%.

4. **Hyperparameter Differences:** Finally, we considered the scenario where two models are different in terms of hyperparameter values. For instance, we trained two AlexNet models on the MNIST dataset (see Pair 4 in Table 4.1) with varying hyperparameters such as learning rate, epochs, and batch sizes. The first model, set with a learning rate of 0.00001, 9 epochs, and a batch size of 128, reached a 98.85% accuracy. Conversely, the second model, with a learning rate of 0.0001, 14 epochs, and a batch size of 256, demonstrated a 99.33% accuracy.

These particular variations were chosen because they represent key aspects of model development and testing that are frequently encountered in practice, for example in frameworks like MLflow[4] to support machine learning development, which is widely used for managing machine learning experiments. MLflow tracks models across these four dimensions, emphasizing their importance in both research and real-world applications.

Further, we selected models ranging from simple architectures, such as LeNet, to more complex ones like AlexNet and ResNet, particularly for the Cifar-10 dataset. This choice allowed us to evaluate our framework across a broad spectrum of models, from smaller, less complex architectures to larger, more sophisticated ones. We could thus assess how well our framework performs under different computational and architectural constraints.

Additionally, we specifically focused on model pairs where the accuracy difference between models was minimal. It ensures that our evaluation captures subtle differences in model performance. Detecting such small variations is crucial for demonstrating that our framework can reliably identify differences even when models achieve nearly identical, high accuracy.

4.5 Empirical Results

We describe in this section our experimental evaluation and report on the results. We first investigate whether we generate more triggering inputs than the baseline with the same testing budget. We then study the validity and diversity of the generated triggering inputs by both approaches. Finally, we present an important application scenario of the generated triggering inputs, namely the development of a machine learning-based mechanism to predict the best model's output based on input characteristics, and report the corresponding results.

4.5.1 RQ1. Do we generate more triggering inputs than the existing baseline approach with the same testing budget?

To answer this research question, we executed *DiffGAN* and *DRfuzz* considering the same execution time $T \in [1h, 2h, 3h]$, across four pairs of DNN models and two distinct datasets, resulting in eight unique combinations (Section 4.4.3). Because of the randomness in *DiffGAN* and *DRfuzz*, we re-executed each of them three times and reported the corresponding number of triggering inputs in Tables 4.3 and 4.4. This decision to limit the execution of the approaches to three iterations was initially due to the considerable execution time associated with the experiments. More specifically, our experiments take 96h (8 model pairs x 6h x 2 approaches) if we execute each approach once. As shown in the tables, *DiffGAN* consistently outperforms *DRfuzz* in terms of generating more triggering inputs across all models and datasets. More specifically, our results show that *DiffGAN* generates on average four times more triggering inputs than *DRfuzz* when considering the same testing

[4] `https://mlflow.org`

Table 4.3: RQ1 results: Number of generated triggering images for models trained on MNIST dataset using DiffGAN and DRfuzz

Models	**Configs**		**DRfuzz**	**#TI (Initial)**	**DiffGAN**
M1, M2	1h	run1	1229	82	**4932**
		run2	1237	82	**5188**
		run3	1228	82	**4963**
	2h	run1	2724	82	**8091**
		run2	2525	82	**7539**
		run3	3427	82	**6853**
	3h	run1	3745	82	**10077**
		run2	5056	82	**10638**
		run3	4888	82	**10839**
M3, M4	1h	run1	1368	95	**4327**
		run2	1517	95	**4531**
		run3	1702	95	**4161**
	2h	run1	3221	95	**5607**
		run2	3227	95	**6980**
		run3	2953	95	**5308**
	3h	run1	4493	95	**7143**
		run2	4737	95	**7243**
		run3	5089	95	**6880**
M5, M6	1h	run1	3743	268	**6399**
		run2	3279	268	**8191**
		run3	3044	268	**7600**
	2h	run1	7837	268	**12560**
		run2	7023	268	**11911**
		run3	7161	268	**10866**
	3h	run1	10766	268	**14927**
		run2	9601	268	**16271**
		run3	10226	268	**15430**
M7, M8	1h	run1	1478	112	**6349**
		run2	1198	112	**5196**
		run3	865	112	**5385**
	2h	run1	2360	112	**8195**
		run2	2088	112	**7933**
		run3	2686	112	**8046**
	3h	run1	3246	112	**11538**
		run2	4400	112	**13622**
		run3	3141	112	**15003**

Table 4.4: RQ1 results: Number of generated triggering images for models trained on Cifar10 dataset using DiffGAN and DRfuzz

Models	**Configs**		**DRfuzz**	**#TI (Initial)**	**DiffGAN**
M1, M2	1h	run1	36329	1993	**80700**
		run2	36916	1993	**80875**
		run3	30284	1993	**80952**
	2h	run1	41345	1993	**156002**
		run2	40009	1993	**154942**
		run3	46268	1993	**154489**
	3h	run1	62148	1993	**200924**
		run2	62456	1993	**195712**
		run3	66384	1993	**199785**
M3, M4	1h	run1	7040	618	**62596**
		run2	7203	618	**61854**
		run3	6913	618	**62609**
	2h	run1	14725	618	**103446**
		run2	14159	618	**104297**
		run3	12262	618	**119261**
	3h	run1	18717	618	**156179**
		run2	20832	618	**160324**
		run3	19915	618	**178621**
M5, M6	1h	run1	9553	959	**65544**
		run2	9851	959	**67152**
		run3	9605	959	**65544**
	2h	run1	19248	959	**124322**
		run2	18352	959	**126532**
		run3	17120	959	**122833**
	3h	run1	33369	959	**224578**
		run2	34321	959	**208223**
		run3	32499	959	**208522**
M7, M8	1h	run1	26068	1750	**74039**
		run2	26737	1750	**73702**
		run3	17765	1750	**75850**
	2h	run1	46005	1750	**156716**
		run2	45730	1750	**133703**
		run3	45363	1750	**133569**
	3h	run1	63434	1750	**243193**
		run2	63879	1750	**221499**
		run3	62148	1750	**218304**

Table 4.5: MNIST Improvement

Models	Time	DRfuzz Imp (%)	DiffGAN Imp (%)
M1, M2	1h	1401.62	6030.08
	2h	3425.61	9029.67
	3h	5465.85	12722
M3, M4	1h	1610.87	4462.81
	2h	3103.16	6466.66
	3h	4655.79	7095.44
M5, M6	1h	1679.29	3037.69
	2h	3494.03	5388.33
	3h	2770	4508.1
M7, M8	1h	898.21	4871.43
	2h	2291.07	8100.3
	3h	3080.65	12990.18

budget. We also calculated the Coefficient of Variation (CV) [54], a measure of relative variability that compares the standard deviation to the mean, independent of the data's scale. We computed the CV score for each model pair and testing budget across the three runs. A lower CV indicates higher consistency across different runs. For example, in our analysis of the MNIST dataset, *DRfuzz* showed a CV between 0.33% and 21.23%, while *DiffGAN* exhibited a CV between 1.33% and 12.20%, suggesting less variability in the number of generated triggering inputs across the different runs. These CV values demonstrate the relative stability and reliability of *DiffGAN* in generating triggering images, with less variability compared to *DRfuzz*. These observations are also consistent for the Cifar-10 dataset. These CV values demonstrate the relative stability and reliability of *DiffGAN* in generating triggering images, with less variability compared to *DRfuzz*.

We should note that *DRfuzz* generates inputs by applying a sequence of mutation rules to seed images, optimizing the order of these transformations to increase the likelihood of finding triggering inputs. However, this approach is by definition limited by the seed images and the specific transformations applied to them, which significantly restricts the potential triggering inputs that can be generated. In contrast, *DiffGAN* explores a much larger input space by optimizing latent vectors within the GAN's input space through genetic search. The GAN's input space inherently offers a broader range of possible image variations than the transformations applied in *DRfuzz*. By optimizing these latent vectors, *DiffGAN* can generate novel combinations of inputs that may not be generated through mutation-based methods alone. This broader exploration capability thus allows *DiffGAN* to produce more triggering inputs, as it can effectively explore the GAN's latent space to uncover a wider range of inputs that may reveal more behavioral disagreements between the DNN models under test.

Moreover, when comparing the results to the initial set of triggering inputs in the original testing datasets, both *DiffGAN* and *DRfuzz* demonstrate significant improvements. The improvement is measured as the ratio of the difference between the number of triggering inputs generated through the differential testing approach and the size of the initial set

Table 4.6: Cifar10 Improvement

Models	Time	DRfuzz Imp (%)	DiffGAN Imp (%)
M1, M2	1h	1631.62	3959.07
	2h	2031.51	7559.31
	3h	3053.63	10094.88
M3, M4	1h	1054.86	10055.92
	2h	2120.68	17201.02
	3h	3106.35	23296.37
M5, M6	1h	914.39	6959.54
	2h	2409.05	16357.15
	3h	3390.65	23611.05
M7, M8	1h	1254.76	4195.18
	2h	35699.33	147930
	3h	63013.5	219901.5

Table 4.7: Validity Comparison

Models	Method	Set Size	Av. V ratio	V range (95% conf.)	#Est. Valid
M1, M2	DiffGAN	379	0.9125	[0.8819, 0.9396]	**8292**
	DRfuzz	365	0.8246	[0.7848, 0.8635]	1999
M3, M4	DiffGAN	377	0.80	[0.76, 0.8329]	**6555**
	DRfuzz	364	0.79	[0.7582, 0.82]	1945
M5, M6	DiffGAN	381	0.8973	[0.8687, 0.9265]	**10286**
	DRfuzz	377	0.8094	[0.7692, 0.8488]	4882
M7, M8	DiffGAN	381	0.811	[0.7717, 0.8478]	**6535**
	DRfuzz	378	0.7469	[0.7037, 0.7910]	1430

of triggering inputs, divided by the size of this initial set. Specifically, *DiffGAN* shows an improvement ranging from 2,660% to 12,727% in the MNIST dataset and from 3,956% to 26,606% in the Cifar-10 dataset. Moreover, *DRfuzz* achieves an improvement ranging from approximately 954% to 5,465% in configurations with MNIST and from 908% to 3,509% in configurations with Cifar-10. Though both approaches are effective in enhancing the generation of triggering inputs, *DiffGAN* is significantly more so.

Answer to RQ1: *DiffGAN* significantly and consistently outperforms *DRfuzz* in generating many more triggering inputs within a given time budget, effectively revealing about four times more behavioral disagreements between DNN models than *DRfuzz*.

4.5.2 RQ2. How do *DiffGAN* and *DRfuzz* compare in terms of generating valid triggering inputs within the same testing budget?

To systematically compare *DiffGAN* and *DRfuzz* in terms of generating valid trigger inputs within a given time budget, we considered the inputs generated in RQ1 by both methods. More specifically, we considered all four model pairs using the MNIST dataset and ran both approaches for two hours on these models. As mentioned in RQ1, we ran each method three times to minimize randomness and recorded all images generated by each approach during each run. After completing the runs, for each approach and model pair, we aggregated the images generated from all three runs into a comprehensive dataset. By combining the results from three separate runs for each configuration, we were able to mitigate the effects of randomness in the generation process, thereby increasing the reliability of our comparisons.

Removing redundant inputs. To eliminate redundancy from the aggregated dataset, we employed a simple hashing-based technique that focused on the exact byte-level representation of each image. Specifically, we converted each image into its byte form and computed a hash value using Python's built-in hash() function[5] . By comparing these hash values, we identified and removed duplicate images that were identical at the byte level. This method ensured that only unique images—in terms of their pixel data—were retained for further analysis.

Selecting representative samples. Due to the large number of triggering inputs generated by *DiffGAN* and *DRfuzz*, we selected a sample of triggering inputs from each aggregated dataset. For each setting, we determined the sample size that can confidently represent the whole corresponding dataset with a 95% confidence level and a margin of error of 5%[6]. As a result, in our experiments, the sample size varied from 361 to 381 across the four settings that we considered, as shown in Table 4.7. This approach helped choose a minimal subset that was large enough to provide results that accurately reflected the overall distribution of inputs generated by *DiffGAN* and *DRfuzz* [138].

Manual validity checking. From the non-redundant set, we randomly sampled images for validation, according to the sample size determined the previous step. To ensure unbiased evaluation, we shuffled images generated by both *DiffGAN* and *DRfuzz* and presented them anonymously to three independent expert evaluators. These experts were tasked with assessing the validity of each image based on predefined criteria. Specifically, they labeled images as either valid or invalid. Invalid images were defined as those that displayed two numbers, no number, or other issues such as (1) Images that were a 50/50 blend of two digits and could not be clearly identified as a specific digit, (2) Incomplete or blended digits that could not be classified into one category by consensus, (3) Images containing random shapes or non-digit drawings, or (4) Images that were completely white or dark, with no discernible content. Examples of valid and invalid inputs generated by both approaches

[5]Python documentation, `hash()` function, available at: `https://docs.python.org/3/library/functions.html#hash`

[6]As computed from https://www.calculator.net/sample-size-calculator.html

are depicted in Figure 4.4. To determine the validity of each image, we considered the agreement among the evaluators. If at least two out of three evaluators labeled an image as valid, we considered it valid. To quantify the level of agreement between the evaluators, we employ Cohen's kappa [14]—a statistical measure that accounts for agreement occurring by chance. In our study, the value of Cohen's kappa is 70%, indicating a high level of agreement among the evaluators and underscoring the reliability of the manual labeling process.

Bootstrapping. After completing the labeling process, we applied bootstrapping to estimate the probability distribution of the validity ratio of the entire generated inputs. Bootstrapping is a nonparametric, simulation-based technique that allows statistical inference from small samples [113]. By resampling the labeled subset (with replacement) multiple times, this method estimates the probability distribution of the validity ratio of the entire dataset of generated inputs. More specifically, we resampled the labeled subset 1000 times with replacement, calculating the validity ratio for each resample. The resulting distribution of validity ratios provided a more reliable estimate of the dataset's overall validity, accounting for variability and reducing potential bias from relying on a single sample. This resampling approach allowed us to estimate the confidence interval for the percentage of valid inputs across the entire dataset.

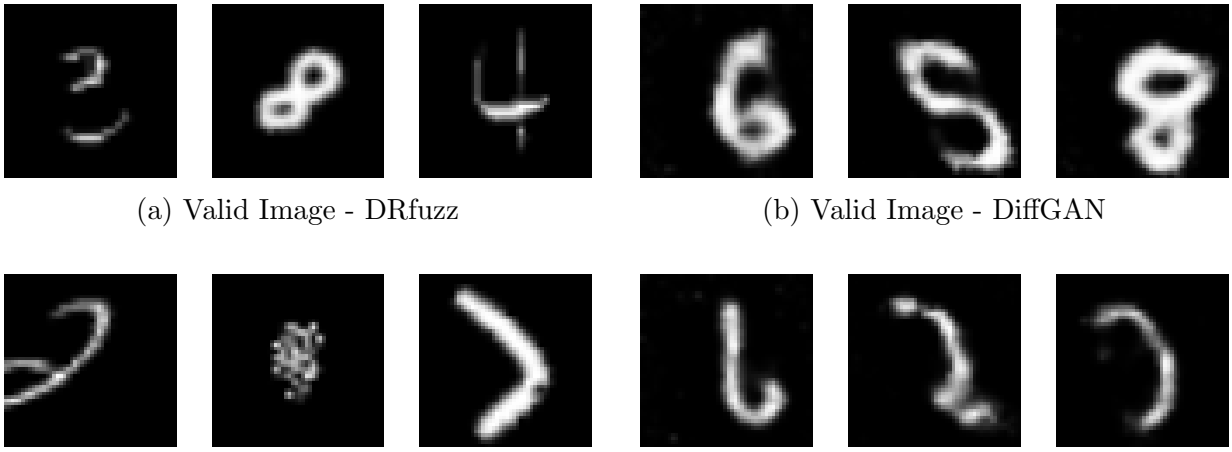
Reporting the total number of valid inputs. We calculated the proportion of valid images for each method by dividing the number of valid images by the total number of sampled images. We then multiplied the validity ratio, calculated from the subset, by the average number of images generated in the runs to estimate the expected number of valid images in each run. Additionally, we determined the 95% confidence interval as reported in Table 4.7

Finally, we compared the expected number of valid inputs generated by *DiffGAN* and *DRfuzz* to assess which method was more effective at producing valid triggering inputs within the same time budget of two hours. The results presented in Table 4.7, clearly suggest that *DiffGAN* significantly outperforms *DRfuzz* in generating more valid triggering inputs. For instance, in the first setting, it is estimated that *DiffGAN* generated 8,292 valid triggering inputs, whereas *DRfuzz* produced only 1,999 such inputs. This corresponds to a validity ratio of 91.25% for *DiffGAN* compared to 82.46% for *DRfuzz*. Importantly, those results were consistent across all the settings we considered. To further analyze the statistical significance of our results we performed a statistical analysis using the Wilcoxon signed-rank test [107], with a significance level of $\alpha = 0.05$, to assess whether *DiffGAN* significantly outperforms *DRfuzz* in generating valid triggering inputs. The Wilcoxon signed-rank test, a non-parametric method, is used to compare the medians of continuous variables for paired samples and is ideal for our case since it does not require any assumptions about the data distribution.

Instead of considering the number of valid inputs, we relied on the validity ratio due to differences in sample sizes. We evaluated the validity ratios from 4000 samples for *DiffGAN* and 4000 samples for *DRfuzz*, selected via bootstrapping across four different scenarios (1000 samples per scenario). The Wilcoxon signed-rank test, with *w-statistic* = 343465.5, *p-value* of $p < 0.000001$, confirmed that *DiffGAN* significantly outperformed

DRfuzz at generating valid triggering inputs across all scenarios.

It is important to note that *DiffGAN* outperforms *DRfuzz* in generating valid triggering inputs primarily due to the validation mechanisms it employs throughout the input generation process. While *DRfuzz* relies solely on the GAN discriminator to assess the validity of generated inputs, *DiffGAN* incorporates both the generator and the SSIM score, enabling a more effective and comprehensive filtering of invalid inputs. Furthermore, *DiffGAN* leverages the generative capabilities of GANs, which are specifically designed to produce realistic and diverse inputs, enhancing its ability to generate valid samples. In contrast, *DRfuzz* applies multiple mutation rules to the same seed image, which can often lead to the generation of more invalid inputs.



(a) Valid Image - DRfuzz

(b) Valid Image - DiffGAN

(c) Invalid Image - DRfuzz

(d) Invalid Image - DiffGAN

Figure 4.4: Examples of Valid and Invalid Images for DRfuzz and DiffGAN

Answer to RQ2: *DiffGAN* consistently and significantly outperforms *DRfuzz* in generating more valid triggering inputs within the same time budget across all experimental settings.

4.5.2.1 RQ3. Does *DiffGAN* find more diverse triggering inputs than *DRfuzz*?

To answer this research question, we measured the diversity scores of the generated test input sets for *DiffGAN* and *DRfuzz*. We selected well-established diversity metrics that are directly applicable to input sets to measure their diversity scores. Specifically, we used two SOTA metrics: *Geometric Diversity* [2] and *Shannon* diversity [41, 131], each leveraging different types of input features.

Shannon Diversity, also known as the *Shannon-Wiener Index* or *Shannon Entropy*, is a SOTA metric used to quantify diversity in a dataset [41, 64, 88, 127]. It is calculated as follows:

Table 4.8: Diversity scores of MNIST generated images

		DRfuzz		**DiffGAN**	
Models	Configs	GD	Exponential Shannon	GD	Exponential Shannon
M1, M2	1h	422.03	$e^{11.36} = 86,361.86$	**530.62**	$e^{12.10} = 178,632.90$
	2h	481.24	$e^{11.34} = 84,643.49$	**536.05**	$e^{12.16} = 191,548.60$
	3h	318.04	$e^{11.32} = 82,829.03$	**366.21**	$e^{12.10} = 178,497.79$
M3, M4	1h	438.42	$e^{11.39} = 89,882.98$	**727.20**	$e^{12.21} = 200,195.46$
	2h	592.18	$e^{11.39} = 89,238.14$	**665.31**	$e^{12.27} = 213,472.83$
	3h	645.69	$e^{11.38} = 88,908.85$	**661.64**	$e^{12.23} = 204,636.72$
M5, M6	1h	410.67	$e^{11.50} = 98,610.81$	**712.76**	$e^{12.27} = 213,402.74$
	2h	536.47	$e^{11.43} = 92,631.60$	**649.41**	$e^{12.23} = 204,753.74$
	3h	551.06	$e^{11.42} = 91,634.01$	**578.24**	$e^{12.26} = 211,800.43$
M7, M8	1h	116.72	$e^{11.40} = 90,159.89$	**481.94**	$e^{12.10} = 178,751.64$
	2h	244.65	$e^{11.40} = 90,792.41$	**551.06**	$e^{12.12} = 183,319.27$
	3h	231.87	$e^{11.38} = 87,505.95$	**666.11**	$e^{12.14} = 187,734.83$

Table 4.9: Diversity scores of Cifar-10 generated images

		DRfuzz		**DiffGAN**	
Models	Configs	GD	Exponential Shannon	GD	Exponential Shannon
M1, M2	1h	1790.80	$e^{14.71} = 2,435,217.87$	**2396.99**	$e^{14.84} = 2,774,122.83$
	2h	2108.90	$e^{14.71} = 2,432,731.99$	**2333.51**	$e^{14.83} = 2,772,404.84$
	3h	2318.33	$e^{14.70} = 2,430,965.85$	**2457.05**	$e^{14.94} = 3,073,279.80$
M3, M4	1h	976.64	$e^{14.71} = 2,434,515.58$	**2395.81**	$e^{14.84} = 2,774,819.14$
	2h	1117.67	$e^{14.71} = 2,432,366.04$	**2339.19**	$e^{14.83} = 2,772,679.02$
	3h	1466.08	$e^{14.70} = 2,429,166.95$	**2284.04**	$e^{14.83} = 2,772,395.01$
M5, M6	1h	711.47	$e^{14.73} = 2,453,279.49$	**2387.32**	$e^{14.84} = 2,774,233.41$
	2h	1463.54	$e^{14.71} = 2,432,146.96$	**2334.41**	$e^{14.83} = 2,772,466.97$
	3h	2045.84	$e^{14.70} = 2,430,669.35$	**2283.65**	$e^{14.83} = 2,772,148.43$
M7, M8	1h	1893.78	$e^{14.71} = 2,432,859.74$	**2378.10**	$e^{14.84} = 2,774,307.47$
	2h	2199.47	$e^{14.69} = 2,426,034.24$	**2331.51**	$e^{14.83} = 2,772,278.57$
	3h	2183.13	$e^{14.69} = 2,425,208.35$	**2295.15**	$e^{14.83} = 2,771,688.20$

$$H = -\sum_{i=1}^{S} p_i \log_b(p_i) \tag{4.5}$$

where:

- S is the total number of distinct categories (taxa, species, or classes).
- p_i is the proportion of the dataset represented by the i-th category.
- b is the base of the logarithm, which is e (natural logarithm).

The metric computes the sum of the proportions of each category multiplied by the logarithm of those proportions. The negative sign ensures that the index is positive and higher diversity translates into higher values of H. To calculate the Shannon diversity for a set of images, each image is first flattened into a one-dimensional array of pixel values. These flattened arrays can then be combined into a single array that represents the distribution of pixel values across the entire image set. The Shannon score, or entropy, is then calculated using this distribution, quantifying the diversity of pixel intensities within the dataset. A higher Shannon score indicates greater diversity within the set of images.

In some contexts, computing the exponential of the Shannon diversity score is used to provide a more intuitive interpretation of the index. Specifically, by calculating e^H, where H is the Shannon diversity of a set of images, we can derive the *effective number of image categories* within the set. This value represents the number of equally likely categories that would produce the same level of entropy as observed in the dataset. For example, if H increases from 11 to 12, the exponential value e^{12} is substantially larger (approximately 162,754.79) when compared to $e^{11} = 59{,}874.14$, indicating that the dataset is considerably more diverse. This approach allows to translate entropy into a more intuitive measure of diversity, making it easier to understand and compare the diversity of different datasets.

Geometric Diversity is a widely used metric for assessing the diversity of test input sets [2, 66, 94]. To calculate the *Geometric Diversity* score of an input set, we use features extracted by the VGG16 model. In the chapter 2 [2, 4], we have shown that combining the *Geometric Diversity* metric with VGG16 features is highly effective in accurately capturing and quantifying the diversity of input sets.

The *Geometric Diversity* metric, denoted as $G(S)$, is computed as follows: Given a dataset X and its feature vectors V, the geometric diversity of a subset $S \subseteq X$ is defined by:

$$G(S) = \det(V_s \times V_s^T) \tag{4.6}$$

This formula represents the squared volume of the parallelepiped formed by the rows of V_s, where V_s consists of the feature vectors corresponding to the subset S. A larger volume indicates greater diversity within the subset S in the feature space.

In this research question, we employed both the Shannon and Geometric diversity metrics to analyze the dataset, aiming to evaluate image diversity from two complementary perspectives. The Shannon diversity metric offers a probabilistic view, emphasizing the distribution of pixel values, while the Geometric diversity metric assesses diversity based on spatial relationships and geometric patterns within the image features extracted by VGG16. By integrating these two metrics, we achieve a more comprehensive understanding of the datasets' diversity.

To measure the diversity scores of the generated test input sets for *DiffGAN* and *DRfuzz*, we used the same subjects and generated triggering inputs as in RQ1. Given that *Geometric Diversity* scores cannot be normalized based on subset size [2], we ensured a fair comparison by randomly selecting subsets of equal size for both metrics. Specifically, for each approach, subject, and testing budget, we randomly selected 1,000 unique, generated triggering inputs and calculated their *Shannon* and *Geometric Diversity* scores. To account for the inherent randomness in *DiffGAN* and *DRfuzz*, we repeated this process 30 times and reported the average diversity scores.

The results, as shown in Tables 4.8 and 4.9, consistently demonstrate that *DiffGAN* outperforms *DRfuzz* in generating more diverse triggering inputs, when considering both metrics, highlighting its effectiveness in exploring a broader range of input variations.

Similarly to RQ1, we analysed the variability of our diversity results using the CV score. Our CV analysis shows that both *DiffGAN* and *DRfuzz* exhibit low variability across different runs, with *DiffGAN* demonstrating greater consistency. *DRfuzz*'s GD diversity has a CV range of 6.76% to 18.44%, while *DiffGAN* is more stable with a CV range of 2.24% to 14.23%. In terms of Shannon metrics, *DRfuzz* shows a CV range between 0.09% and 0.70%, whereas *DiffGAN* is even more consistent, with a CV range of 0.02% to 0.33%. These results underscore the superior stability of *DiffGAN*'s outputs compared to *DRfuzz*.

Furthermore, the superior performance of *DiffGAN* in generating more diverse triggering inputs can be attributed to its large latent space, which allows for a greater variety of generated inputs, and its diversity-based fitness function specifically designed to maximize the diversity of generated inputs, a feature that is limited in *DRfuzz*. Indeed, *DRfuzz* focuses on triggering behavioral disagreements through limited mutation rules, which, while effective in some contexts, do not inherently prioritize or ensure diversity. As a result, the input sets generated by *DRfuzz* tend to be less diverse compared to *DiffGAN*.

Answer to RQ3: *DiffGAN* consistently outperforms *DRfuzz* in generating more diverse triggering inputs across all DNN models and datasets.

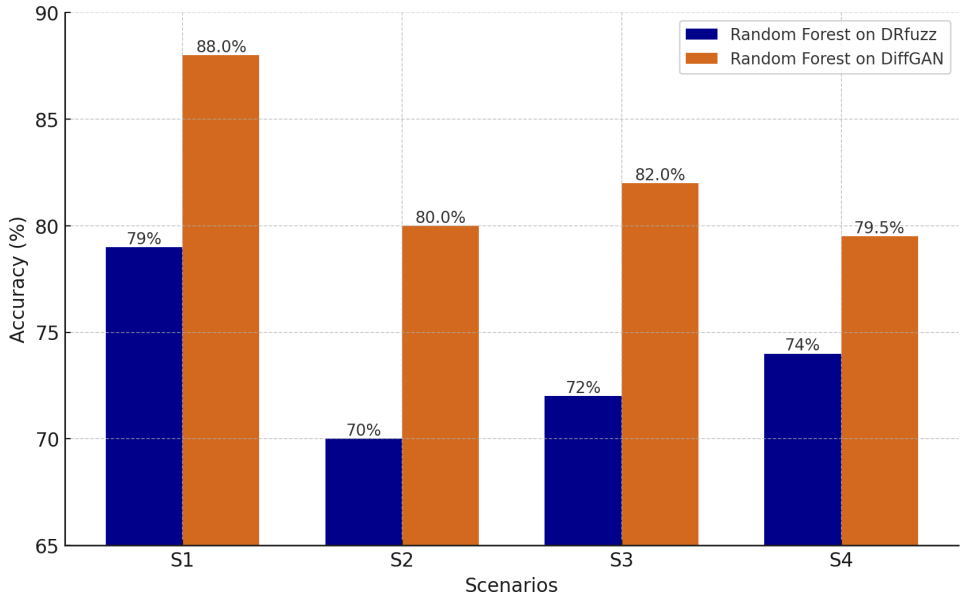


Figure 4.5: Accuracy of Random Forest in Selecting the Best Model

4.5.3 RQ4. Can training of an ML-based model selection mechanism be guided more effectively using the triggering inputs generated by *DiffGAN* compared to those produced by *DRfuzz*?

We aim in this question to study whether we can better guide the training of an ML-based model selection mechanism that consists of (1) two classifiers (model 1 and model 2) responsible for image classification, and (2) a third model that dynamically selects which of the two classifiers to use based on the input characteristics, which is based on Random Forest (RF). It predicts, based on the characteristics of the incoming input, which of the two classifiers (model 1 or model 2) will produce a more accurate output, without the need to run both models in an online setting. When properly trained with triggering inputs, the RF model should learn when each classifier is more accurate, effectively serving as a decision mechanism for selecting the optimal classifier for a given input. We choose RF because: (1) it scales effectively to handle a large number of features, and (2) its robustness against overfitting [26, 60, 62]. We do not intend to provide a novel voting mechanism here but, instead, provide an example of an important application scenario for the triggering inputs we generate. It also serves as a basis to compare the value of the inputs generated by *DiffGAN* to enable such model selection and compare it with that of *DRfuzz*.

To achieve this, we trained the model selection system using two distinct training datasets. The first dataset consists of triggering inputs generated by *DiffGAN*, while the second contains triggering inputs produced by *DRfuzz*. To create these datasets, we leveraged the inputs generated by *DiffGAN* and *DRfuzz* in RQ1 (with an execution time budget of one hour), considering the four model pairs from the MNIST dataset. For each method and model pair, we randomly sampled 1,200 triggering images, resulting in a total

of 9,600 images (1,200 images × 4 model pairs × 2 methods). Given the large volume of unlabeled triggering inputs, we outsourced the classification to a professional labeling company. Each input was labeled with a value from 0 to 9, with an additional label for "*invalid*" inputs. The number of inputs removed ranged from 21 to 121 for *DRfuzz*, and from 3 to 34 for *DiffGAN*. Invalid images were subsequently filtered out from each of the 1,200-image sets. It is important to note that we relied on a single labeling company due to cost and time constraints. This company was chosen for its strong reputation and track record of high-quality labeling. To further ensure the accuracy of the labels, we manually validated the labels of a random sample of 1,000 images. We found that all images were correctly labeled, confirming the reliability of the labeling process.

To compare the accuracy of the RF-based model selection systems, we categorized the images into three groups based on whether one model predicted correctly or if both models failed. We excluded images where both models failed from further analysis due to their low occurrence and lack of relevance to the comparison (i.e., if both models fail, it does not matter which classification model is selected). We trained the RF classifiers using features extracted from a pre-trained VGG16 model, rather than raw pixel data, in an attempt to improve accuracy. The VGG16 model provides feature vectors that capture more abstract and informative patterns than pixel-based features, which is expected to improve the RF model's ability to distinguish between correct and incorrect predictions. For each method and model pair, we selected 700 valid labeled images and used their VGG16 features to train eight RF-based selection models. To ensure a fair comparison between the RF models trained on different datasets in each scenario, we need a common test dataset. For each scenario, we selected 100 images from the *DiffGAN*-generated test set and 100 from the *DRfuzz*-generated test set, creating a combined test set of 200 images. This test set allowed us to evaluate the performance of the RF models using the same test set in an unbiased manner. Additionally, we applied the same feature extraction process as for the training sets.

We report the accuracy levels of each of the four selection model we trained in Figure 4.5. As clearly visible, the RF-based selection models trained using triggering inputs generated with *DiffGAN* are far more accurate than those trained with inputs from *DRfuzz*. For instance, in the first scenario involving two RF-based selection models built using classifiers Model 1 and Model 2 from Table 4.1, the RF model trained with *DRfuzz* achieved an accuracy of 78%, whereas the one trained with *DiffGAN* reached a much higher accuracy of 87%. Overall, the accuracy levels for the RF models trained with *DiffGAN* range between 79% and 87% compared to a range of 70% to 78% for those with *DRfuzz* inputs. This highlights that *DiffGAN* produces more effective triggering inputs for training, allowing the RF model to better capture the conditions under which one classifier outperforms the other. As a result, RF models trained using *DiffGAN* inputs are more accurate in selecting the best-performing classifier, leading to a better model selection mechanism.

This improved accuracy can be attributed to several factors. *DiffGAN* leverages the combination of GAN and NSGA-II to generate highly diverse triggering inputs. Unlike *DRfuzz*, which applies mutation rules to existing seed inputs, *DiffGAN* explores a significantly broader latent input space, enabling the generation of more diverse inputs that expose more pronounced behavioral differences between the models. These behavioral dif-

ferences are crucial for the RF model to learn the conditions when each classifier performs better. Additionally, *DiffGAN* optimizes its inputs to both maximize diversity and reveal discrepancies, which also enables the RF model to develop a more accurate understanding of the conditions under which one classifier outperforms the other.

Though the accuracy of RF models can still be improved, our main goal here was to show the usefulness of *DiffGAN* for one important application, not to optimize model selection. The latter would require the labeling of a much larger dataset and possibly the fine-tuning of the RF models or experimenting with other machine-learning techniques, which is out of the scope of this thesis.

Answer to RQ4: The ML-based model selection mechanisms trained with *DiffGAN* triggering inputs are more accurate in selecting the best-performing classifier among alternative models, leading to better model selection mechanisms. This highlights that *DiffGAN* produces more triggering inputs that are more effective for training, allowing the RF model to better capture the conditions under which one classifier outperforms the other. In turn, this can lead to more accurate voting mechanisms at inference time and thus higher classification accuracy.

4.6 Threats to Validity

In this section, we discuss the threats to the validity of our study and describe how we mitigated them.

Internal threats to validity concern the causal relationship between the treatment and the outcome. One such threat in our study is the inherent randomness in the experimental setup (RQ1-3), particularly the stochastic nature of *DiffGAN* and *DRfuzz*. To mitigate this, we executed each experiment three times, and we observed that the variance across these runs was minimal, leading to consistent conclusions in our RQs. As a result, we concluded that further repetitions, which would come at a high experimental cost, were unnecessary to draw reliable conclusions.

Another potential internal threat is related to the bias in the estimation of the validity of the generated inputs. To address this, we relied on three independent evaluators to assess the validity of the generated inputs manually. By requiring consensus among at least two of the three evaluators for determining validity, we minimized the influence of subjective bias, ensuring a reliable and consistent evaluation process. Additionally, the high level of agreement among evaluators, as measured by Cohen's Kappa, further confirms the reliability of the validity assessments.

Construct threats to validity concern the relationship between the theory and the observations made. A potential threat to construct validity in this chapter could arise from the generation of invalid inputs by *DiffGAN*. To mitigate this, we implemented a filtration process using two SOTA techniques: the SSIM score and the GAN's discriminator. We experimented with different SSIM score thresholds to find a near-optimal value for filtering

out invalid images, manually reviewing all images in a set to verify the adequacy of this threshold. While we acknowledge that this threshold might potentially filter out a few valid images, it effectively removes the majority of invalid inputs. We also recognize that while this process significantly reduces the occurrence of invalid inputs, it does not entirely eliminate them. However, it minimizes their impact, enhancing the overall validity of our results. Another potential threat relates to the construct validity of diversity measurements for the generated input sets. To mitigate this, we used two SOTA diversity metrics: Shannon Diversity and Geometric Diversity, each capturing different aspects of inputs' variability. The consistent results across both metrics further reinforce our confidence in our diversity assessment of the generated inputs.

External threats to validity concern the generalizability of our study. We mitigate this threat by considering eight different combinations of widely used models and datasets. These combinations feature diverse model pairs varying in architecture, training data, optimizers, and hyperparameters. Additionally, we explored a wide range of testing budgets (execution times) in our experiments and compared our results against a SOTA baseline for the differential testing of DNN models.

4.7 Related Work

Differential testing approaches for DNN models can be characterized as black-box or white-box, depending on their access requirements to the internals of the DNN models under test. A few black-box differential testing approaches for DNNs have been proposed in the literature.

Xie *et al.* [154] proposed *DiffChaser*, a black-box, search-based test generation approach aiming to find behavioral disagreements between DNN models and their quantized versions. They initiate the generation process by selecting seed images. Through iterative mutations and crossovers of these seed images, *DiffChaser* applies a k-Uncertainty [154] fitness function to guide the generation process, aiming to find discrepancies between a model and its quantized version during testing.

Tian *et al.* [143] proposed *Dflare*, a search-based test generation approach that iteratively applies a series of mutation operations to a given seed image until a triggering input is found. They rely on *Markov Chains* and divergence-based fitness function to guide the selection of mutation operators [143]. The fitness function is designed to prioritize mutated inputs that (1) maximize differences between the output probability vectors of two models or (2) trigger new, previously unobserved probability vectors in the models under test.

You *et al.* [162] introduced *DRfuzz*, a differential testing method that detects regression faults in updated versions of DNN models. A regression fault occurs when an input correctly predicted by an earlier model version results in an incorrect prediction post-update. Similar to previous approaches, *DRfuzz* uses seed inputs and mutation rules to generate new inputs that trigger regression faults. It employs a customized reward function to select mutation rules, favoring those that produce more fault-triggering inputs. *DRfuzz* also leverages a GAN's discriminator to filter invalid inputs.

The above test generation approaches heavily depend on the availability and quality of seed data, limiting the process to mutated versions of existing inputs. This dependence significantly narrows the range of test scenarios, as it prevents the introduction of entirely new features (i.e, features that are not present in the initial testing dataset) into the testing process—a capability that *DiffGAN* notably possesses, enhancing its effectiveness in differential testing by generating diverse and new test input scenarios.

A few white-box differential testing approaches have been proposed as well. *DeepExplore* [125] is the first white-box differential testing approach for DNN models. It formulates the image generation problem as an optimization problem, which uses gradient-based search techniques to find images that maximize neuron coverage and the number of behavioral disagreements between the DNN models under test.

DeepHunter [153] is a white-box test generation approach for DNNs that uses mutation operators on seed inputs along with coverage metrics to guide the generation of triggering inputs. The approach explores different seed selection strategies, incorporating both diversity-based and recency-based methods, to optimize the seed selection process. *DeepHunter* supports five different coverage metrics to guide the generation process. However, evaluations on quantized DNNs indicate that no single coverage metric consistently outperforms the others across different quantized DNN models, in terms of generating more triggering inputs.

DiverGet [158] and *DeepEvolution* [24] are two white-box differential testing approaches for quantized DNN models. They leverage multi-objective search algorithms and domain-specific mutation operators to generate triggering inputs. The generation is guided by two fitness functions. The former fitness function maximizes the divergence in output probability vectors between the models under test, while the latter aims to maximize the coverage of neuron activations within those models. While *DeepEvolution* is designed to test both original and quantized DNN models for image classification tasks, *DiverGet* specifically targets DNN models used for hyperspectral image classification [158].

The use of white-box differential testing approaches is hindered by the required access to the internal structure of the DNN models. This can be a significant limitation in practice, particularly when the models under test are proprietary or provided by a third party [2]. Additionally, several white-box approaches [24, 158] rely on comparing neuron activation values or model gradients of DNN models, thus assuming structural similarity among the models being tested. These requirements do not apply to *DiffGAN*, which, unlike existing white-box differential testing approaches, supports testing across any pair of DNN models without requiring structural similarity or the need for internal access.

4.8 Conclusion

We presented *DiffGAN*, a novel black-box test generation framework that leverages GANs and NSGA-II for differential testing of deep neural networks. *DiffGAN* uses customized fitness functions in NSGA-II to guide the search process, generating diverse and valid triggering inputs that expose behavioral differences between models. To the best of our

knowledge, this is the first work addressing black-box differential testing aimed at model selection. We show that *DiffGAN* is capable of revealing behavioural differences across models by exploring large input spaces through GAN and NSGA-II's diversity-driven fitness functions. Our empirical evaluation on eight pairs of image classification models trained on two widely-used datasets demonstrated that *DiffGAN* significantly outperforms the SOTA differential testing baseline, *DRfuzz*, by generating on average four times more triggering inputs. These inputs are not only more diverse but also more likely to be valid, allowing for a deeper analysis of the models' behavior. Furthermore, we showed how these inputs can be applied in a machine learning-based voting mechanism, which dynamically selects the most accurate model based on input characteristics, resulting in improved classification accuracy compared to the baseline approach.

For future work, we aim to explore *DiffGAN* in dynamic environments to assess the side effects of updating a single ML component on the overall system performance. Rather than comparing pairs of DNN models, we will compare the original system with the updated version. This approach will provide valuable insights into maintaining system reliability and adaptability after updates, particularly in critical areas like autonomous driving, and healthcare.

Chapter 5

Discussion

This thesis focuses on selecting or generating test inputs and performing differential testing of deep neural networks (DNNs) for image data. To reach the objectives (TO1, TO2, TO3), we developed methods for evaluating test adequacy, improving fault detection, and enhancing model selection using black-box testing. While these methods have been effective for image data, adapting them to text data, particularly in the context of large language models (LLMs), presents unique opportunities and challenges. In this section, we discuss how these techniques can be adapted for LLMs, identify key challenges, and propose necessary modifications.

- TO1: Test Adequacy Metrics for Text Data: For image data, test adequacy was measured using neuron coverage and GD diversity metric which leverages feature extraction models like VGG16. In the context of text, features would need to be extracted using LLMs like BERT or GPT, which capture the semantic and contextual relationships in language. To adapt these test adequacy metrics for text datasets and models, we propose the following steps
  - Feature Extraction for Text: LLMs such as BERT or GPT can be used to extract semantically rich embeddings that capture contextual meaning, allowing for more accurate test input evaluation.
  - Textual Neuron Coverage: Neuron coverage could be applied to the transformer layers of LLMs to measure the extent to which diverse linguistic structures activate different parts of the model, analogous to how it is used for image data.
- TO2: Fault Detection in Text-Based Models: Our fault detection approach for image data used feature extraction with VGG16 and NSGA-II to generate diverse, fault-triggering inputs. For text data, this must be adapted to linguistic features and text classification models. Key considerations for adapting fault detection methods to text data include:
  - Feature Extraction: Text embeddings from models like BERT or GPT encode semantic and syntactic information, allowing us to evaluate how linguistic variations trigger faults in text models.

  - NSGA-II Adaptation: The fitness functions used in NSGA-II should be adapted to maximize diversity in syntactic and semantic variations while measuring output divergence between different models.

- TO3: DiffGAN for Text Data: Since it is targeting images, DiffGAN employed a GAN to generate diverse inputs for differential testing between DNN models. To adapt this for text data, we propose using an LLM, such as GPT or BERT, to generate diverse, linguistically rich textual inputs. Key components of this adaptation include:
  - Generative Language Model: An LLM would generate linguistically diverse text samples to explore the input space effectively.
  - Text-Based DNNs: These text inputs can be passed through two DNN classifiers to detect differences in predictions, revealing subtle behavioral variations between models.
  - Fitness Functions: Diversity-based fitness functions would measure syntactic and semantic variability, while divergence-based functions would assess model output differences.

General Challenges: While these adaptations hold promise, transitioning from image-based models to text introduces several challenges that must be addressed to ensure effective testing:

- Evaluation Complexity: Text data introduces more complexity in defining correctness, as there may be multiple valid interpretations for a given input.
- Computational Cost: LLMs are more computationally expensive than CNNs, increasing the resources needed to generate and evaluate large volumes of test inputs.

Summary: In summary, the methods developed in this thesis for image data can be adapted for linguistic data by leveraging LLMs for feature extraction and modifying differential testing techniques accordingly. While there are challenges related to the complex structure of language, ensuring linguistic validity, and handling increased computational costs, these adaptations hold significant potential for improving the accuracy of text models in tasks like text classifications and question answering, especially in critical applications where robust model evaluation is crucial.

In case you want to apply the DiffGAN approach to other types of models, such as generative models, you first need to address the question: What does 'triggering input' mean in that context? The definition of triggering input will vary depending on the model's purpose. For generative models, it may involve identifying inputs that lead to significant deviations in generated outputs of two models or assessing how input changes affect the quality and diversity of the generated content. Adapting the method requires redefining the fitness functions and selecting metrics that align with the objectives of the new model type.

Chapter 6

Thesis Conclusion

This thesis presents a comprehensive study on the testing of Deep Neural Networks (DNNs), focusing on improving fault detection, optimizing test selection, and enhancing differential testing methods. Given the increasing reliance on DNNs in critical applications such as autonomous driving, medical diagnostics, and manufacturing, ensuring the accuracy and safety of these models is crucial. This research has addressed key challenges, including the inefficacy of traditional adequacy metrics, the high cost of labeling, and the difficulty of selecting effective test cases for real-world applications.

The primary contributions of this thesis are as follows:

- **Empirical Evaluation of Adequacy Metrics:** This research conducted an extensive empirical analysis of current DNN adequacy metrics, demonstrating that traditional metrics like neuron coverage have limited utility in real-world fault detection. The findings showed that high neuron coverage and its variants do not guarantee improved fault detection, highlighting the need for alternative approaches that better correlate with real-world fault identification. This insight formed the foundation for the development of more effective testing criteria and better guidance for test image selection or generation.

- **Proposing a Novel Fault Estimation Approach:** The thesis introduced a clustering-based fault estimation approach, which significantly enhances fault estimation by grouping similar mispredicted inputs that share underlying failure causes. This method aims to improve the assessment of DNN reliability by focusing on the root causes of mispredictions, providing a more meaningful evaluation of model performance.

- **Development and Evaluation of DeepGD for Test Case Selection:** This research also proposes DeepGD, a black-box, multi-objective test selection framework that optimizes for both diversity and uncertainty. DeepGD efficiently selects test inputs with a high likelihood of revealing distinct faults while minimizing labeling costs, addressing the practical constraints of real-world DNN testing. Through extensive empirical evaluations on multiple datasets and models, DeepGD was shown

to outperform existing test selection methods, improving fault detection rates and enhancing the effectiveness of DNN retraining.

- **Introduction of DiffGAN for Differential Testing:** A major contribution of this research is DiffGAN, a GAN-based differential testing framework designed to generate diverse and valid inputs that expose behavioral differences between accurate DNN models. DiffGAN was evaluated based on its ability to generate diverse test inputs and its effectiveness in tasks such as dynamic model selection, model compression, and regression testing. This framework offers a robust method for comparing and optimizing DNN models, particularly in safety-critical applications where behavioral discrepancies between models can lead to severe consequences.

6.1 Implications of the Research:

The findings of this thesis have significant implications for both the theoretical and practical aspects of DNN testing. On a theoretical level, the introduction of diversity metrics and the fault estimation method shifts the paradigm from traditional neuron coverage metrics to more effective approaches for guiding DNN testing. These methods not only provide a deeper understanding of fault detection but also challenge the current reliance on adversarial inputs for testing, advocating for the use of more realistic, natural input sets.

Practically, the DeepGD framework offers a scalable, cost-effective solution for selecting fault-revealing test inputs, which is particularly valuable in contexts where labeling costs are high and the test input space is vast.

Additionally, DiffGAN provides a practical and efficient solution for differential testing in real-world applications, particularly because it operates as a black-box approach, making it highly applicable where internal model details are unavailable. Instead of repeatedly running simulators or searching through large datasets to find rare inputs that trigger different behaviors in similar models, DiffGAN efficiently generates novel and realistic inputs that expose behavioral discrepancies. This not only saves substantial time but is crucial, especially in high-accuracy models like the ones used in our study, where finding inputs that produce different results between models with accuracies of 99.5% and 99.25% on 10,000 images is highly challenging. Manually discovering such inputs through traditional data collection or testing would be impractical.

Limitations Despite the significant contributions of this research, certain limitations should be acknowledged. First, while DeepGD and DiffGAN were tested on a variety of datasets and models, their applicability to other domains, such as text-based DNN models or more complex real-world systems, has yet to be fully explored. However, we have thoroughly discussed the necessary modifications for applying this work to text data in the Discussion section.

Although DiffGAN was evaluated on standard image classification datasets, the methodology demonstrates strong potential for real-world applications, particularly in safety-critical domains where rare but critical faults—such as those that could lead to accidents in

autonomous driving—must be identified. While we did not test DiffGAN in these specific environments, its ability to generate rare, fault-triggering inputs underscores its value in enhancing model accuracy across diverse and complex contexts.

6.2 Future Research Directions:

The results of this thesis open several avenues for future research:

- Expanding the Application of DeepGD and DiffGAN: Future research could explore the application of these frameworks to a wider range of DNN models, including those used for text processing, speech recognition, and other modalities. Additionally, investigating the use of DeepGD in other testing applications, such as test set minimization or generation, could further enhance its utility.
- Enhancing Fault Estimation Methods: Improving the fault estimation process by refining clustering techniques or incorporating advanced feature extraction methods could lead to even more accurate fault characterization and estimation. Furthermore, exploring automated approaches for identifying the root causes of faults could reduce reliance on manual intervention in fault diagnosis.
- Alternative Diversity Metrics: Future studies could explore additional diversity metrics that might offer further improvements in fault detection. Investigating the combination of multiple metrics to create a more holistic approach to test input selection is another promising direction.
- Real-time Testing and Dynamic Environments: The application of these methods in real-time or dynamic environments, such as continuously evolving DNNs or models that require frequent updates, could significantly enhance their relevance in real-world scenarios. Future research could explore how these frameworks perform in live environments where models need to adapt to changing conditions.

6.3 Final Remarks

In conclusion, this thesis makes substantial contributions to the field of DNN testing by developing new methods that address critical challenges in fault detection, test case selection, and differential testing. By advancing both the theoretical understanding of DNN testing and offering practical solutions, this research provides a strong foundation for future work aimed at improving the accuracy, safety, and robustness of DNN models. As DNNs continue to play an increasingly important role in various domains, the approaches developed in this thesis will be instrumental in ensuring their successful and trustworthy deployment in real-world systems.

References

[1] Zohreh Aghababaeyan, Manel Abdellatif, Lionel Briand, Ramesh S, and Mojtaba Bagherzadeh. Dnn testing replication package.

[2] Zohreh Aghababaeyan, Manel Abdellatif, Lionel Briand, Ramesh S, and Mojtaba Bagherzadeh. Black-box testing of deep neural networks through test case diversity. *IEEE Transactions on Software Engineering*, 49(5):3182–3204, 2023.

[3] Zohreh Aghababaeyan, Manel Abdellatif, Mahboubeh Dadkhah, and Lionel Briand. Replication package of deepgd. `https://github.com/ZOE-CA/DeepGD`, 2023.

[4] Zohreh Aghababaeyan, Manel Abdellatif, Mahboubeh Dadkhah, and Lionel Briand. Deepgd: A multi-objective black-box test selection approach for deep neural networks. *ACM Trans. Softw. Eng. Methodol.*, feb 2024. Just Accepted.

[5] Nima Aghli and Eraldo Ribeiro. Combining weight pruning and knowledge distillation for cnn compression. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 3185–3192, 2021.

[6] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631, 2019.

[7] Mahbubul Alam, Manar D Samad, Lasitha Vidyaratne, Alexander Glandon, and Khan M Iftekharuddin. Survey on deep neural networks in speech and vision systems. *Neurocomputing*, 417:302–321, 2020.

[8] Krizhevsky Alex, Nair Vinod, and Hinton Geoffrey. The cifar-10 dataset.

[9] Mebarka Allaoui, Mohammed Lamine Kherfi, and Abdelhakim Cheriet. Considerably improving clustering algorithms using umap dimensionality reduction technique: a comparative study. In *International conference on image and signal processing*, pages 317–325. Springer, 2020.

[10] Aitor Arrieta. Multi-objective metamorphic follow-up test case selection for deep learning systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1327–1335, 2022.

[11] Jordan T Ash, Chicheng Zhang, Akshay Krishnamurthy, John Langford, and Alekh Agarwal. Deep batch active learning by diverse, uncertain gradient lower bounds. *arXiv preprint arXiv:1906.03671*, 2019.

[12] Mohammed Oualid Attaoui, Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. Black-box safety analysis and retraining of dnns based on feature extraction and clustering. *arXiv preprint arXiv:2201.05077*, 2022.

[13] TensorFlow Authors. Tensorflow lite guide. `https://www.tensorflow.org/lite/guide`, 2024. Accessed: 2024-07-08.

[14] Mousumi Banerjee, Michelle Capozzoli, Laura McSweeney, and Debajyoti Sinha. Beyond kappa: A review of interrater agreement measures. *Canadian journal of statistics*, 27(1):3–23, 1999.

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

[16] Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *International conference on machine learning*, pages 552–560. PMLR, 2013.

[17] Charles H Bennett, Péter Gács, Ming Li, Paul MB Vitányi, and Wojciech H Zurek. Information distance. *IEEE Transactions on information theory*, 44(4):1407–1423, 1998.

[18] Matteo Biagiola, Filippo Ricca, and Paolo Tonella. Search based path and input data generation for web application testing. In *International Symposium on Search Based Software Engineering*, pages 18–32. Springer, 2017.

[19] Matteo Biagiola, Andrea Stocco, Filippo Ricca, and Paolo Tonella. Diversity-based web test generation. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 142–153, 2019.

[20] Julian Blank and Kalyanmoy Deb. Pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020.

[21] Douglas G Bonett and Thomas A Wright. Sample size requirements for estimating pearson, kendall and spearman correlations. *Psychometrika*, 65(1):23–28, 2000.

[22] Rebecca Schuller Borbely. On normalized compression distance and large malware. *Journal of Computer Virology and Hacking Techniques*, 12(4):235–242, 2016.

[23] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger N. Gunn, Alexander Hammers, David Alexander Dickie, Maria Valdés Hernández, Joanna Marguerite Wardlaw, and Daniel Rueckert. Gan augmentation: Augmenting training data using generative adversarial networks. *ArXiv*, abs/1810.10863, 2018.

[24] Houssem Ben Braiek and Foutse Khomh. Deepevolution: A search-based testing approach for deep neural networks. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 454–458. IEEE, 2019.

[25] Jürgen Branke, Kalyanmoy Deb, Henning Dierolf, and Matthias Osswald. Finding knees in multi-objective optimization. In *Parallel Problem Solving from Nature-PPSN VIII: 8th International Conference, Birmingham, UK, September 18-22, 2004. Proceedings 8*, pages 722–731. Springer, 2004.

[26] Leo Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

[27] Lionel C Briand, Yvan Labiche, and Marwa Shousha. Using genetic algorithms for early schedulability analysis and stress testing in real-time systems. *Genetic Programming and Evolvable Machines*, 7(2):145–170, 2006.

[28] Paulo MS Bueno, W Eric Wong, and Mario Jino. Improving random test sets using the diversity oriented test data generation. In *Proceedings of the 2nd international workshop on Random testing: co-located with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007)*, pages 10–17, 2007.

[29] Taejoon Byun, Sanjai Rayadurgam, and Mats PE Heimdahl. Black-box testing of deep neural networks. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 309–320. IEEE, 2021.

[30] Emanuela G Cartaxo, Patrícia DL Machado, and Francisco G Oliveira Neto. On the use of a similarity function for test case selection in the context of model-based testing. *Software Testing, Verification and Reliability*, 21(2):75–100, 2011.

[31] Elisa Celis, Vijay Keswani, Damian Straszak, Amit Deshpande, Tarun Kathuria, and Nisheeth Vishnoi. Fair and diverse dpp-based data summarization. In *International Conference on Machine Learning*, pages 716–725. PMLR, 2018.

[32] Biyuan Chen, Xueyi He, Bangquan Pan, Xiaobing Zou, and Na You. Comparison of beta diversity measures in clustering the high-dimensional microbial data. *PloS one*, 16(2):e0246893, 2021.

[33] Junjie Chen, Zhuo Wu, Zan Wang, Hanmo You, Lingming Zhang, and Ming Yan. Practical accuracy estimation for efficient deep neural network testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 29(4):1–35, 2020.

[34] Junjie Chen, Ming Yan, Zan Wang, Yuning Kang, and Zhuo Wu. Deep neural network test coverage: How far are we? *arXiv preprint arXiv:2010.04946*, 2020.

[35] Tsong Yueh Chen, Fei-Ching Kuo, Robert G Merkel, and TH Tse. Adaptive random testing: The art of test case diversity. *Journal of Systems and Software*, 83(1):60–66, 2010.

[36] Tsong Yueh Chen, R Merkel, PK Wong, and G Eddy. Adaptive random testing through dynamic partitioning. In *Fourth International Conference onQuality Software, 2004. QSIC 2004. Proceedings.*, pages 79–86. IEEE, 2004.

[37] Zuohui Chen, RenXuan Wang, Yao Lu, Jingyang Xiang, and Qi Xuan. Adversarial sample detection via channel pruning. In *ICML 2021 Workshop on Adversarial Machine Learning*, 2021.

[38] Rudi Cilibrasi and Paul MB Vitányi. Clustering by compression. *IEEE Transactions on Information theory*, 51(4):1523–1545, 2005.

[39] Helen G Cobb and John J Grefenstette. Genetic algorithms for tracking changing environments. Technical report, Naval Research Lab Washington DC, 1993.

[40] Andrew R Cohen and Paul MB Vitányi. Normalized compression distance of multisets with applications. *IEEE transactions on pattern analysis and machine intelligence*, 37(8):1602–1614, 2014.

[41] R.R. Coifman and M.V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Transactions on Information Theory*, 38(2):713–718, 1992.

[42] Dinu Coltuc, Mihai Datcu, and Daniela Coltuc. On the use of normalized compression distances for image similarity detection. *Entropy*, 20(2):99, 2018.

[43] Francisco G de Oliveira Neto, Azeem Ahmad, Ola Leifler, Kristian Sandahl, and Eduard Enoiu. Improving continuous integration with similarity-based test case selection. In *Proceedings of the 13th International Workshop on Automation of Software Test*, pages 39–45, 2018.

[44] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[45] Kalyanmoy Deb, Ram Bhushan Agrawal, et al. Simulated binary crossover for continuous search space. *Complex systems*, 9(2):115–148, 1995.

[46] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[47] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[48] Alex Diaz-Papkovich, Luke Anderson-Trocmé, and Simon Gravel. A review of umap in population genetics. *Journal of Human Genetics*, 66(1):85–91, 2021.

[49] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jinsong Dong, and Ting Dai. An empirical study on correlation between coverage and robustness for deep neural networks. In *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 73–82. IEEE, 2020.

[50] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the analysis of the (1+ 1) evolutionary algorithm. *Theoretical Computer Science*, 276(1-2):51–81, 2002.

[51] Agoston E Eiben, James E Smith, et al. *Introduction to evolutionary computing*, volume 53. Springer, 2003.

[52] Mohamed Elfeki, Camille Couprie, Morgane Riviere, and Mohamed Elhoseiny. Gdpp: Learning diverse generations using determinantal point processes. In *International Conference on Machine Learning*, pages 1774–1783. PMLR, 2019.

[53] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. Density-connected sets and their application for trend detection in spatial databases. In *KDD*, volume 97, pages 10–15, 1997.

[54] Brian Everitt. *The Cambridge Dictionary of Statistics*. Cambridge University Press, Cambridge, U.K., repr. with corrections edition, 1998.

[55] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. Supporting deep neural network safety analysis and retraining through heatmap-based unsupervised learning. *IEEE Transactions on Reliability*, 2021.

[56] Robert Feldt, Simon Poulding, David Clark, and Shin Yoo. Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*, pages 223–233. IEEE, 2016.

[57] Robert Feldt, Richard Torkar, Tony Gorschek, and Wasif Afzal. Searching for cognitively diverse tests: Towards universal test diversity metrics. In *2008 IEEE International Conference on Software Testing Verification and Validation Workshop*, pages 178–186. IEEE, 2008.

[58] 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.

[59] Paulo Fernandes, Joao Correia, and Penousal Machado. Evolutionary latent space exploration of generative adversarial networks. In *Applications of Evolutionary Computation: 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Seville, Spain, April 15–17, 2020, Proceedings 23*, pages 595–609. Springer, 2020.

[60] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *The journal of machine learning research*, 15(1):3133–3181, 2014.

[61] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[62] Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pages 1189–1232, 2001.

[63] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. Adaptive test selection for deep neural networks. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 73–85. IEEE, 2022.

[64] Nuno M Garcia, Micael Santos, Nuno Pombo, João Redol, and Susanna Spinsante. Differential image analysis using shannon's entropy: preliminary results. 2015.

[65] 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.

[66] Boqing Gong, Wei-Lun Chao, Kristen Grauman, and Fei Sha. Diverse sequential subset selection for supervised video summarization. *Advances in neural information processing systems*, 27:2069–2077, 2014.

[67] Zhiqiang Gong, Ping Zhong, and Weidong Hu. Diversity in machine learning. *IEEE Access*, 7:64323–64350, 2019.

[68] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 37(3):362–386, 2020.

[69] Tracy Hall, Sarah Beecham, David Bowes, David Gray, and Steve Counsell. A systematic literature review on fault prediction performance in software engineering. *IEEE Transactions on Software Engineering*, 38(6):1276–1304, 2011.

[70] Ge Han, Zheng Li, Peng Tang, Chengyu Hu, and Shanqing Guo. Fuzzgan: A generation-based fuzzing framework for testing deep neural networks. pages 1601–1608, 2022.

[71] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. Is neuron coverage a meaningful measure for testing deep neural networks? In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 851–862, 2020.

[72] Hadi Hemmati. How effective are code coverage criteria? In *2015 IEEE International Conference on Software Quality, Reliability and Security*, pages 151–156. IEEE, 2015.

[73] Hadi Hemmati, Andrea Arcuri, and Lionel Briand. Achieving scalable model-based testing through test case diversity. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 22(1):1–42, 2013.

[74] Hadi Hemmati, Zhihan Fang, and Mika V Mantyla. Prioritizing manual test cases in traditional and rapid release environments. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10. IEEE, 2015.

[75] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. Comparing white-box and black-box test prioritization. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pages 523–534. IEEE, 2016.

[76] John R Hershey and Peder A Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07*, volume 4, pages IV–317. IEEE, 2007.

[77] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6629–6640, Red Hook, NY, USA, 2017. Curran Associates Inc.

[78] John H. Holland. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. 1992.

[79] Yuta Hozumi, Rui Wang, Changchuan Yin, and Guo-Wei Wei. Umap-assisted k-means clustering of large-scale sars-cov-2 mutation datasets. *Computers in biology and medicine*, 131:104264, 2021.

[80] Qiang Hu, Yuejun Guo*, Maxime Cordy, Xiaofei Xie, Lei Ma, Mike Papadakis, and Yves Le Traon. An empirical study on data distribution-aware test selection for deep learning enhancement. *ACM Transactions on Software Engineering and Methodology*, 2022.

[81] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5967–5976, 2017.

[82] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5967–5976, 2017.

[83] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[84] Mark Joswiak, You Peng, Ivan Castillo, and Leo H Chiang. Dimensionality reduction for visualizing industrial chemical process data. *Control Engineering Practice*, 93:104189, 2019.

[85] Byungkon Kang. Fast determinantal point process sampling with application to clustering. *Advances in Neural Information Processing Systems*, 26, 2013.

[86] Taranjit Kaur and Tapan Kumar Gandhi. Automated brain image classification based on vgg-16 and transfer learning. In *2019 International Conference on Information Technology (ICIT)*, pages 94–98. IEEE, 2019.

[87] Faiq Khalid, Hassan Ali, Hammad Tariq, Muhammad Abdullah Hanif, Semeen Rehman, Rehan Ahmed, and Muhammad Shafique. Qusecnets: Quantization-based defense mechanism for securing deep neural network against adversarial attacks. In *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pages 182–187, 2019.

[88] Pirazh Khorramshahi, Hossein Souri, Rama Chellappa, and Soheil Feizi. Gans with variational entropy regularizers: Applications in mitigating the mode-collapse issue, 2020.

[89] 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.

[90] András Kocsor, Attila Kertész-Farkas, László Kaján, and Sándor Pongor. Application of compression-based distance measures to protein sequence classification: a methodological study. *Bioinformatics*, 22(4):407–412, 2006.

[91] Andrei N Kolmogorov. Three approaches to the quantitative definition ofinformation'. *Problems of information transmission*, 1(1):1–7, 1965.

[92] Andreas Krause, Ajit Singh, and Carlos Guestrin. Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(2), 2008.

[93] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

[94] Alex Kulesza and Ben Taskar. Determinantal point processes for machine learning. *arXiv preprint arXiv:1207.6083*, 2012.

[95] Michael Austin Langford and Betty HC Cheng. Enki: A diversity-driven approach to test and train robust learning-enabled systems. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 15(2):1–32, 2021.

[96] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[97] David Leon and Andy Podgurski. A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases. In *14th International Symposium on Software Reliability Engineering, 2003. ISSRE 2003.*, pages 442–453. IEEE, 2003.

[98] Ming Li, Xin Chen, Xin Li, Bin Ma, and Paul MB Vitányi. The similarity metric. *IEEE transactions on Information Theory*, 50(12):3250–3264, 2004.

[99] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. Modeldiff: Testing-based dnn similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 139–151, 2021.

[100] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. Structural coverage criteria for neural networks could be misleading. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 89–92. IEEE, 2019.

[101] Zenan Li, Xiaoxing Ma, Chang Xu, Chun Cao, Jingwei Xu, and Jian Lü. Boosting operational dnn testing efficiency through conditioning. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 499–509, 2019.

[102] Hui Lin and Jeff A Bilmes. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871*, 2012.

[103] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[104] L. Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, C. Chen, Ting Su, L. Li, Y. Liu, Jianjun Zhao, and Yadong Wang. Deepgauge: Multi-granularity testing criteria for deep learning systems. *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 120–131, 2018.

[105] Wei Ma, Mike Papadakis, Anestis Tsakmalis, Maxime Cordy, and Yves Le Traon. Test selection for deep learning systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 30(2):1–22, 2021.

[106] Yue Ma, Bo Li, Wentao Huang, and Qinqin Fan. An improved nsga-ii based on multi-task optimization for multi-uav maritime search and rescue under severe weather. *Journal of Marine Science and Engineering*, 11(4):781, 2023.

[107] Thomas W MacFarland, Jan M Yates, Thomas W MacFarland, and Jan M Yates. Wilcoxon matched-pairs signed-ranks test. *Introduction to Nonparametric statistics for the biological sciences using R*, pages 133–175, 2016.

[108] Leland McInnes, John Healy, and Steve Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, 2017.

[109] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[110] Ali Mesbah, Arie Van Deursen, and Danny Roest. Invariant-based automatic testing of modern web applications. *IEEE Transactions on Software Engineering*, 38(1):35–53, 2011.

[111] Salma Messaoudi, Annibale Panichella, Domenico Bianculli, Lionel Briand, and Raimondas Sasnauskas. A search-based approach for accurate identification of log message formats. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pages 167–16710. IEEE, 2018.

[112] Brad L. Miller and David E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evol. Comput.*, 4(2):113–131, jun 1996.

[113] Christopher Z. Mooney and Robert D. Duval. *Bootstrapping: A Nonparametric Approach to Statistical Inference*. Sage Publications, Newbury Park, CA, USA, 1993.

[114] Davoud Moulavi, Pablo A Jaskowiak, Ricardo JGB Campello, Arthur Zimek, and Jörg Sander. Density-based clustering validation. In *Proceedings of the 2014 SIAM international conference on data mining*, pages 839–847. SIAM, 2014.

[115] Wafa Mousser and Salima Ouadfel. Deep feature extraction for pap-smear image classification: A comparative study. In *Proceedings of the 2019 5th International Conference on Computer and Technology Applications*, pages 6–10, 2019.

[116] Hossein Ziaei Nafchi, Atena Shahkolaei, Rachid Hedjam, and Mohamed Cheriet. Mean deviation similarity index: Efficient and reliable full-reference image quality evaluator. *Ieee Access*, 4:5579–5590, 2016.

[117] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7:19143–19165, 2019.

[118] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[119] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. 2011.

[120] Jalaj Pachouly, Swati Ahirrao, Ketan Kotecha, Ganeshsree Selvachandran, and Ajith Abraham. A systematic literature review on software defect prediction using artificial intelligence: Datasets, data validation methods, approaches, and tools. *Engineering Applications of Artificial Intelligence*, 111:104773, 2022.

[121] Rongqi Pan, Mojtaba Bagherzadeh, Taher A Ghaleb, and Lionel Briand. Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering*, 27(2):29, 2022.

[122] Rongqi Pan, Taher A Ghaleb, and Lionel Briand. Atm: Black-box test case minimization based on test code similarity and evolutionary search. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1700–1711. IEEE, 2023.

[123] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. Reformulating branch coverage as a many-objective optimization problem. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*, pages 1–10, 2015.

[124] Spencer Pearson, José Campos, René Just, Gordon Fraser, Rui Abreu, Michael D Ernst, Deric Pang, and Benjamin Keller. Evaluating and improving fault localization. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, pages 609–620. IEEE, 2017.

[125] 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.

[126] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[127] Ameet Annasaheb Rahane and Anbumani Subramanian. Measures of complexity for large scale image datasets. In *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pages 282–287, 2020.

[128] Vincenzo Riccio and Paolo Tonella. When and why test generators for deep learning produce invalid inputs: an empirical study. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 1161–1173, 2023.

[129] 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.

[130] Jasmine Sekhon and Cody Fleming. Towards improved testing for deep learning. In *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pages 85–88. IEEE, 2019.

[131] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 1948.

[132] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9243–9252, 2020.

[133] Yuki Shimizu, Shigeo Morimoto, Masayuki Sanada, and Yukinori Inoue. Automatic design system with generative adversarial network and convolutional neural network

for optimization design of interior permanent magnet synchronous motor. *IEEE Transactions on Energy Conversion*, 38(1):724–734, 2023.

[134] Connor Shorten and Taghi Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6, 07 2019.

[135] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[136] Changjian Shui, Fan Zhou, Christian Gagné, and Boyu Wang. Deep active learning: Unified and principled method for query and training. In *International Conference on Artificial Intelligence and Statistics*, pages 1308–1318. PMLR, 2020.

[137] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[138] Ajay S Singh and Micah B Masuku. Sampling techniques & determination of sample size in applied statistics research: An overview.

[139] M. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, June 1994.

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

[141] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Structural test coverage criteria for deep neural networks. *ACM Transactions on Embedded Computing Systems (TECS)*, 18(5s):1–23, 2019.

[142] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

[143] Yongqiang Tian, Wuqi Zhang, Ming Wen, Shing-Chi Cheung, Chengnian Sun, Shiqing Ma, and Yu Jiang. Finding deviated behaviors of the compressed dnn models for image classifications. *ACM Transactions on Software Engineering and Methodology*, 32(5):1–32, 2023.

[144] 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*, ICSE '18, page 303–314, New York, NY, USA, 2018. Association for Computing Machinery.

[145] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[146] Shanu Verma, Millie Pant, and Vaclav Snasel. A comprehensive review on nsga-ii for multi-objective combinatorial optimization problems. *Ieee Access*, 9:57757–57791, 2021.

[147] Matt P Wand and M Chris Jones. *Kernel smoothing*. CRC press, 1994.

[148] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. Bet: black-box efficient testing for convolutional neural networks. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 164–175, 2022.

[149] Shuai Wang, Shaukat Ali, and Arnaud Gotlieb. Cost-effective test suite minimization in product lines using search techniques. *Journal of Systems and Software*, 103:370–391, 2015.

[150] Michael Weiss and Paolo Tonella. Simple techniques work surprisingly well for neural network test prioritization and active learning (replicability study). In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 139–150, 2022.

[151] W Eric Wong, Vidroha Debroy, Ruizhi Gao, and Yihao Li. The dstar method for effective software fault localization. *IEEE Transactions on Reliability*, 63(1):290–308, 2013.

[152] Robert F Woolson. Wilcoxon signed-rank test. *Wiley encyclopedia of clinical trials*, pages 1–3, 2007.

[153] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, pages 146–157, 2019.

[154] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. Diffchaser: Detecting disagreements for deep neural networks. International Joint Conferences on Artificial Intelligence Organization, 2019.

[155] Haotian Xu and Zhijian Ou. Scalable discovery of audio fingerprint motifs in broadcast streams with determinantal point process based motif clustering. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 24(5):978–989, 2016.

[156] Qiantong Xu, Gao Huang, Yang Yuan, Chuan Guo, Yu Sun, Felix Wu, and Kilian Weinberger. An empirical study on evaluation metrics of generative adversarial networks. *arXiv preprint arXiv:1806.07755*, 2018.

[157] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[158] Ahmed Haj Yahmed, Houssem Ben Braiek, Foutse Khomh, Sonia Bouzidi, and Rania Zaatour. Diverget: a search-based software testing approach for deep neural network quantization assessment. *Empirical Software Engineering*, 27(7):193, 2022.

[159] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. Correlations between deep neural network model coverage criteria and model quality. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 775–787, 2020.

[160] Ceyuan Yang, Yujun Shen, Yinghao Xu, Deli Zhao, Bo Dai, and Bolei Zhou. Improving gans with a dynamic discriminator. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, *Advances in Neural Information Processing Systems*, volume 35, pages 15093–15104. Curran Associates, Inc., 2022.

[161] Zhou Yang, Jieke Shi, Muhammad Hilmi Asyrofi, and David Lo. Revisiting neuron coverage metrics and quality of deep neural networks. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 408–419. IEEE, 2022.

[162] Hanmo You, Zan Wang, Junjie Chen, Shuang Liu, and Shuochuan Li. Regression fuzzing for deep learning systems. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*, pages 82–94. IEEE, 2023.

[163] Ran Yuan, Bo Wang, Zhixin Mao, and Junzo Watada. Multi-objective wind power scenario forecasting based on pg-gan. *Energy*, 226:120379, 2021.

[164] Qunli Yuchi, Nengmin Wang, Shuang Li, Zhen Yang, and Bin Jiang. A bi-objective reverse logistics network design under the emission trading scheme. *IEEE Access*, 7:105072–105085, 2019.

[165] Xueying Zhan, Qingzhong Wang, Kuan-hao Huang, Haoyi Xiong, Dejing Dou, and Antoni B Chan. A comparative survey of deep active learning. *arXiv preprint arXiv:2203.13450*, 2022.

[166] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 132–142, 2018.

[167] Chunyu Zhao, Yanzhou Mu, Xiang Chen, Jingke Zhao, Xiaolin Ju, and Gan Wang. Can test input selection methods for deep neural network guarantee test diversity? a large-scale empirical study. *Information and Software Technology*, 150:106982, 2022.

[168] Jianyi Zhou, Feng Li, Jinhao Dong, Hongyu Zhang, and Dan Hao. Cost-effective testing of a deep learning model through input reduction. In *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pages 289–300. IEEE, 2020.

[169] Tao Zhou, Zoltán Kuscsik, Jian-Guo Liu, Matúš Medo, Joseph Rushton Wakeling, and Yi-Cheng Zhang. Solving the apparent diversity-accuracy dilemma of recommender systems. *Proceedings of the National Academy of Sciences*, 107(10):4511–4515, 2010.

[170] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 79–90, 2021.

[171] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, Mojtaba Bagherzadeh, and Ramesh S. A search-based testing approach for deep reinforcement learning agents. *IEEE Transactions on Software Engineering*, 49(7):3715–3735, 2023.

[172] Amirhossein Zolfagharian, Manel Abdellatif, Lionel C. Briand, and Ramesh S. Smarla: A safety monitoring approach for deep reinforcement learning agents, 2023.

[173] Daming Zou, Jingjing Liang, Yingfei Xiong, Michael D Ernst, and Lu Zhang. An empirical study of fault localization families and their combinations. *IEEE Transactions on Software Engineering*, 2019.