ENHANCING DNN TEST DATA SELECTION THROUGH UNCERTAINTY-BASED AND DATA DISTRIBUTION-AWARE APPROACHES

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF INFORMATICS OF
THE MIDDLE EAST TECHNICAL UNIVERSITY
BY

DEMET DEMIR

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN
THE DEPARTMENT OF INFORMATION SYSTEMS

JULY 2024

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Demet Demir

Signature :

ABSTRACT

ENHANCING DNN TEST DATA SELECTION THROUGH UNCERTAINTY-BASED AND DATA DISTRIBUTION-AWARE APPROACHES

Demir, Demet

Ph.D., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Elif Sürer

Co-Supervisor: Assoc. Prof. Dr. Aysu Betin Can

July 2024, 131 pages

In this thesis, we introduce a testing framework designed to identify fault-revealing data in Deep Neural Network (DNN) models and determine the causes of these failures.

Given the data-driven nature of DNNs, the effectiveness of testing depends on the adequacy of labeled test data. We perform test data selection with the goal of identifying and prioritizing test data that will cause failures in the DNN. To achieve this, we leveraged the degree of uncertainty of the model for inputs. Initially, we employed state-of-the-art uncertainty estimation methods and metrics, then proposed new ones. Lastly, we developed a novel approach using a meta-model that integrates multiple uncertainty metrics, overcoming the limitations of individual metrics and enhancing effectiveness in various scenarios.

The test data distribution significantly impacts DNN performance and is critical in assessing test results. Therefore, we generated test datasets with a distribution-aware perspective. We propose to first focus on in-distribution data for which the DNN model is expected to make accurate predictions and then include out-of-distribution (OOD) data. Furthermore, we investigated post-hoc explainability methods to identify the causes of incorrect predictions. Visualization explanation techniques provide insights into the reasons for incorrect decision-making by DNNs, however they require detailed manual assessment.

We evaluated the proposed methodologies using image classification DNNs and datasets. The results show that uncertainty-based test selection effectively identifies fault-revealing inputs. Specifically, test data prioritization using the meta-model approach outperforms state-of-the-art methods. Consequently,

we conclude that using prioritized data in tests significantly increases the detection rate of DNN model failures.

Keywords: Deep Neural Network Testing, Test Data Selection and Prioritization, Data Distribution, Deep Learning Explainability, Deep Learning Uncertainty

ÖZ

BELİRSİZLİĞE DAYALI VE VERİ DAĞILIMINA DUYARLI YAKLAŞIMLAR YOLUYLA DNN TESTI VERİ SEÇİMİNİN GELİŞTİRİLMESİ

Demir, Demet

Doktora, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Elif Sürer

Ortak Tez Yöneticisi: Doç. Dr. Aysu Betin Can

Temmuz 2024, 131 sayfa

Bu tezde, Derin Sinir Ağı (DNN) modellerindeki hataları ortaya çıkaran verileri belirlemek ve bu hataların nedenlerini tespit etmek için tasarlanmış bir test çerçevesi sunulmaktadır.

DNNlerin veri odaklı doğası göz önüne alındığında, testlerin etkinliği etiketlenmiş test verilerinin yeterliliğine bağlıdır. Test verisi seçimi, test edilen DNN modelinde hatalara neden olacak test girdilerini belirleme ve önceliklendirme hedefiyle gerçekleştirildi. Bunu başarmak amacıyla modelin bir girdi için belirsizlik derecesinden faydalanıldı. İlk olarak, en ileri belirsizlik tahmin yöntemleri ve metrikleri kullanıldı, ardından yenileri önerildi. Son olarak, birden fazla belirsizlik metriğini birleştiren bir meta-model kullanan, metriklerin tek başına kullanımlarının sınırlamalarını aşan ve çeşitli senaryolarda etkinliği artıran yenilikçi bir yaklaşım geliştirildi.

Test verisi dağılımı, DNN performansını önemli ölçüde etkilemekte ve test sonuçlarının değerlendirilmesinde kritik bir rol oynamaktadır. Bu nedenle, dağılım farkındalığına sahip bir perspektifle test veri setleri oluşturuldu. İlk olarak, DNN modelinin doğru tahmin yapması beklenen dağılım içi veriye odaklanılması ve ardından dağılım dışı (OOD) verinin dahil edilmesi önerildi. Ayrıca, yanlış tahminlerin nedenlerini belirlemek için sonradan açıklanabilirlik metodları incelendi. Görsel açıklama teknikleri, DNN'lerin yanlış karar verme nedenleri hakkında içgörüler sağlamaktadır ancak, bu detaylı bir manuel değerlendirme gerektirir.

Önerilen metodolojiler, görüntü sınıflandırma DNN modelleri ve veri setleri kullanılarak değerlendirildi. Sonuçlar, belirsizliğe dayalı test seçiminin, hatayı ortaya çıkaran girdileri belirlemede etkili olduğunu göstermektedir. Özellikle, meta-model yaklaşımı ile yapılan önceliklendirme, en ileri yöntemler-

den daha iyi performans sergilemektedir. Sonuç olarak, testlerde önceliklendirilmiş veri kullanmanın, DNN model hatalarını tespit etme oranını önemli ölçüde artırdığı sonucuna varılmıştır.

Anahtar Kelimeler: Derin Sinir Ağı Testi, Test Veri Seçimi ve Önceliklendirmesi, Veri Dağılımı, Derin Öğrenmede Açıklanabilirlik, Derin Öğrenmede Belirsizlik



To my family

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to my supervisors, Assoc. Prof. Dr. Elif Surer and Assoc. Prof. Dr. Aysu Betin Can, whose guidance, support, and encouragement have been invaluable throughout my study. Your expertise, patience, and continuous feedback have been fundamental to the completion of this thesis.

I would also like to extend my sincere thanks to the members of my dissertation committee, Prof. Dr. Altan Koçyiğit and Assoc. Prof. Dr. Ayça Kolukısa, for their insightful comments, constructive criticisms, and valuable suggestions, which have greatly improved the quality of this work. Additionally, I am grateful to Prof. Dr. Banu Günel Kılıç and Assoc. Prof. Dr. Nurcan Alkış Bayhan for their valuable comments and constructive feedback during my thesis defense.

I would also like to acknowledge the support of TUBITAK ULAKBIM, High Performance and Grid Computing Center (TRUBA resources), for providing the infrastructure necessary to conduct our experiments.

I express my heartfelt appreciation to my loving husband, Hüsnü, for his love, sacrifices, and encouragement during the most challenging moments of this study. I am also grateful to my son Kerem and my daughter Zeynep for their understanding throughout my studies. Special thanks to my best friend Zuhal, for her support and belief in me. This journey would not have been possible without them.

TABLE OF CONTENTS

ABSTRACT iv

ÖZ vi

DEDICATION viii

ACKNOWLEDGMENTS ix

TABLE OF CONTENTS x

LIST OF TABLES xiv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Challenges in DNN Testing 4

1.1.1 Test Data Selection 4

1.1.2 Distribution Shift 5

1.1.3 Analyses of the Test Results 8

1.2 Proposed Solution 8

1.3 Research Questions 11

1.4 Contributions of the Study 11

1.5 Organization of the Thesis 12

2 RESEARCH METHODOLOGY 13

2.1 Design Science Research 13

2.1.1 Problem Conceptualization 14

2.1.2 Solution Design, Instantiation, and Validation 16

2.1.3 Communication of Results 18

3 BACKGROUND AND RELATED WORK 19

3.1 DNN Testing 19

3.1.1 Test Oracle 19

3.1.2 Test Adequacy 20

3.1.3 Test Data Selection 21

3.1.4 Distribution-Aware Testing 23

3.2 Uncertainty 24

3.3 Explainability 24

4 METHODOLOGY-1: DNN MODEL TESTING FRAMEWORK 29

4.1 Test Data Generation Implementation 31

4.2 Test Data Selection Implementation 32

4.2.1 Out of Distribution Detection 32

4.2.2 Test Data Prioritization 34

4.2.3 Evaluation of Test Data Prioritization Approaches 36

4.3 Test Results Interpretation Implementation 39

5 METHODOLOGY 2 : IMPROVEMENTS IN TEST DATA SELECTION 41

5.1 Alternative Uncertainty-Based Test Prioritization Metrics 41

5.1.1 Test Data Prioritization with Mahalanobis Uncertainty Score 41
5.1.2 Test Data Prioritization with Deep Ensemble Variation Score 43
5.2 Test Selection using Meta-Models with Uncertainty Metrics 44
5.3 OOD Detection with Vision Transformer Models 45
6 EXPERIMENTS 49
6.1 Test Datasets and DL Models 49
6.2 Test Data Generation Methods 53
6.3 Experiments for DNN Testing Framework 54
6.3.1 Test Setup 54
6.3.2 Experiment-1 55
6.3.3 Experiment-2 59
6.4 Experiments for Alternative Test Data Prioritization Approaches 59
6.4.1 Experiment-3 59
6.4.2 Experiment-4 60
6.5 Experiments for Test Selection with Meta-Models 60
6.5.1 Test Setup 60
6.5.2 Experiment-5 62
7 EXPERIMENT RESULTS 65
7.1 Results for DNN Testing Framework 65
7.1.1 Experiment-1 Results 65
7.1.2 Experiment-2 Results 79
7.2 Results for Alternative Test Data Prioritization Approaches 82
7.2.1 Experiment-3 Results 83

7.2.2 Experiment-4 Results 84
7.3 Results for Test Selection with Meta-Models 85
7.3.1 Experiment-5 Results 88
8 DISCUSSION 103
8.1 Test Data Selection 103
8.1.1 Uncertainty Based Test Data Prioritization 103
8.1.2 Test Data Prioritization with Mahalanobis Uncertainty Score 106
8.1.3 Test Data Prioritization with DE Variation Score 106
8.1.4 Meta-model Based Test Data Prioritization 106
8.2 OOD Data Detection 109
8.2.1 OOD Detection with Variational Autoencoders 109
8.2.2 OOD Detection with Visual Transformer Models 110
8.3 Test Results Interpretation with Post-hoc Explainability Methods 110
8.4 Limitations 111
9 CONCLUSION AND FUTURE WORK 113
REFERENCES 119
CURRICULUM VITAE 131

LIST OF TABLES

Table 1 Datasets and DNN Models used in experiments. 53

Table 2 Details of Generated Datasets for Experiment-1. 55

Table 3 Parameters of Adversarial Attack methods employed to generate test data. 56

Table 4 Degree of image transformations applied to original test data. 56

Table 5 Architecture of Variational Autoencoder Models used for OOD Data Detection. 56

Table 6 Uncertainty-based Test Data Prioritization Strategies. 58

Table 7 Details of Test Datasets Employed in the Evaluation of Test Selection using Meta-Models. 61

Table 8 Meta-Models and their input features 63

Table 9 F1-Score Values for OOD detection performed with VAEs using selected threshold values. 65

Table 10 Experiment-1 - Accuracy values of DNN models for Generated Test Datasets without OOD data. 66

Table 11 Experiment-1 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD. 68

Table 12 Experiment-1 - Ratio of Area Under Curve values for Generated Test Datasets (MNIST). 72

Table 13 Experiment-1 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-10). 73

Table 14 Experiment-1 - Correlation Between Test Prioritization Strategies and Misclassifications (MNIST). 75

Table 15 Experiment-1 - Correlation Between Test Prioritization Strategies and Misclassifications (CIFAR-10). 76

Table 16 Experiment-1 - Accuracy Values for Generated Test Dataset including OOD Data. . . . 77

Table 17 Experiment-1 - Average Percentage of Fault Detected values for Generated Test Dataset including OOD Data. 77

Table 18 Experiment-3 - Average Percentage of Fault Detected (APFD) values for test data prioritization with Mahalonobis Uncertainty Scores. 83

Table 19 Experiment-3 - Correlation Between Mahalonobis Uncertainty Scores and Misclassifications (CIFAR-10). 83

Table 20 Experiment-4 - Average Percentage of Fault Detected (APFD) values for test data prioritization with DE Variation Scores. 84

Table 21 Experiment-4 - Correlation Between DE Variation Scores and Misclassifications (CIFAR-10). 85

Table 22 Performance results of OOD Detection performed using Visual Transformer (ViT) models with selected OOD datasets. 85

Table 23 Coefficient Values of Input Features in Logistic Regression Models used as Meta-Models. 88

Table 24 Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (MNIST). 90

Table 25 Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (CIFAR-10). 90

Table 26 Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (CIFAR-100). 91

Table 27 Experiment-5 - Ratio of Area Under Curve values for Original Test Datasets. 92

Table 28 Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (MNIST). 93

Table 29 Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-10). 94

Table 30 Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-100). 95

Table 31 Alternative Meta-Models and their input features. 98

Table 32 Experiment-5 - Accuracy of models for Generated Test Data. 98

Table 33 Experiment-5 - APFD Values of Test Data Prioritization Strategies for Generated Test Datasets with OOD Data 99

Table 34 Experiment-5 - Ratio of Area Under Curve values for Test Datasets with OOD data. . 100

LIST OF FIGURES

Figure 1 DL system development life cycle 1

Figure 2 Input data categorization according to data distribution. 6

Figure 3 Proposed DNN Test Framework. 10

Figure 4 Design Science Research Main Activities. 14

Figure 5 Applied Design Science Research Process. 15

Figure 6 Visual abstract of this study with Design Science Research constructs. 16

Figure 7 DNN Testing Framework with a distribution aware and uncertainty-based approach. 29

Figure 8 Architecture of Variational Auto Encoder (VAE). 32

Figure 9 Illustration of Uncertainty Estimation Methods. 35

Figure 10 Sample Ratio of Area Under Curve (RAUC) Graph. 37

Figure 11 Sample Fault Detection Ratio Graph. 38

Figure 12 Meta-Model approach. 44

Figure 13 Meta-Model Training Process. 45

Figure 14 OOD Data Detection Threshold Value Selection Process. 46

Figure 15 Sample images from MNIST dataset. 50

Figure 16 Architecture of LeNet-5 model. 50

Figure 17 Sample images from CIFAR-10 and CIFAR-100 datasets. 51

Figure 18 Architecture of ResNet-32 and VGG-19 DNN models. 52

Figure 19 Percentages of OOD Data in Generated Test Datasets using OOD detection with Variational Autoencoders (VAEs). 66

Figure 20 Experiment-1 - Fault Detection Ratios according to the size of Generated Test Datasets (MNIST). 69

Figure 21 Experiment-1 - Fault Detection Ratios according to the size of Generated Test Datasets (CIFAR-10). 70

Figure 22 Experiment-1 - RAUC Values of Test Prioritization Strategies for Generated Test Datasets without OOD Data. 74

Figure 23 Experiment-1 - Fault Detection Ratios according to the size of Generated Test Dataset including OOD Data. 78

Figure 24 Experiment-2 - Total Number of Misclassified Test Data According to Classes (Generative Models Test Dataset). 79

Figure 25 Experiment-2 - Total Number of Misclassified Test Data According to Classes (Image Transformation Test Dataset). 80

Figure 26 Experiment-2 - Total Number of Misclassified Test Data According to Classes (Adversarial Attacks Test Dataset. 80

Figure 27 Experiment-2 - Correctly classified Cat images from the original CIFAR-10 test dataset 81

Figure 28 Experiment-2 - Correctly classified Dog images from the original CIFAR-10 test dataset. 81

Figure 29 Experiment-2 - Dog images misclassified as Cat in CIFAR-10 Image Transformation Test Dataset. 82

Figure 30 OOD Scores of MNIST vs Selected OOD Datasets (Fashion MNIST/K-MNIST). 86

Figure 31 OOD Scores of CIFAR-10 vs Selected OOD Datasets(SVHN/CIFAR-100). 86

Figure 32 OOD Scores of CIFAR-100 vs Selected OOD Datasets (SVHN/CIFAR-10). 87

Figure 33 Percentages of OOD Data in Generated Test Datasets using OOD detection with ViT Models. 87

Figure 34 Experiment-5 - RAUC Values of Test Prioritization Strategies for Original and Generated Test Datasets without OOD Data. 96

Figure 35 Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Original Test Dataset 97

Figure 36 Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Generated Test Datasets without OOD data. 97

Figure 37 Experiment-5 - RAUC Values of Test Data Prioritization Strategies for Generated Test Datasets with OOD Data. 99

Figure 38 Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Generated Test Datasets with OOD data. 101

LIST OF ABBREVIATIONS

AI	Artifical Intelligence
APFD	Average Percent of Fault Detected
AUROC	Area Under the Receiver Operating Characteristic Curve
BNN	Bayesian Neural Network
CAM	Class Activation Map
CNN	Convolutional Neural Network
C&W	Carlini & Wagner
DE	Deep Ensemble
DL	Deep Learning
DNN	Deep Neural Network
DSA	Distance-based Surprise Adequacy
DSR	Design science research
FGSM	Fast Gradient Sign Method
FPR	False Positive Rate
GAN	Generative Adversarial Network
LSA	Likelihood-based Surprise Adequacy (LSA)
MCP	Multiple Boundary Clustering and Prioritization
MUS	Mahalanobis Uncertainty Score
MUT	Model Under Test
OOD	Out-of-Distribution
PGD	Projected Gradient Descent
RAUC	Ratio of Area Under Curve

RQ	Research Question
SOTA	State-of-the-art
TPR	True Positive Rate
VAE	Variational Autoencoder
VI-F	Variational Inference with Flipout
ViT	Visual Transformer
XAI	Explainable Artifical Intelligence

CHAPTER 1

INTRODUCTION

In the last decade, Deep Learning (DL) [1] has made significant advancements, which can be mainly attributed to the emergence of new and improved learning algorithms, enhanced hardware capability and capacity, the increase in the volume of data, and its ease of access [2]. As a result, DL has been widely used in real-world applications due to its ability to process complex and high amounts of input data and to solve challenging problems, such as image recognition, natural language processing, and speech recognition.

The development life cycle of DL systems can be broadly categorized into three key stages, which are Data Management, Model Development, and Operations, as depicted in Figure 1. The fundamental building block of DL systems is the Deep Neural Network (DNN). DNNs are a type of artificial neural networks that have multiple hidden layers between the input and output layers, which enable them to learn complex patterns in large datasets. They are inherently different from programmed software due to their unique development process. DNN model development is a data-driven approach that is different from conventional software development.

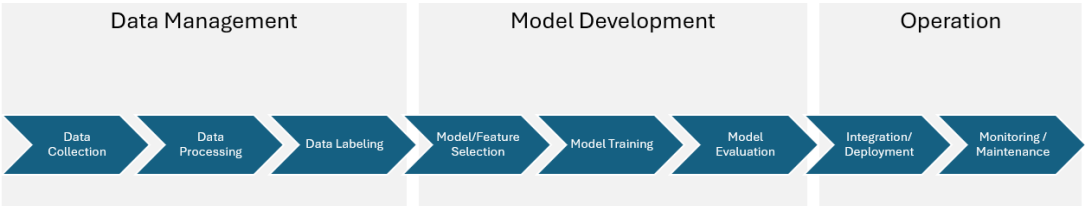


Figure 1: DL system development life cycle, based on [3, 4]. There are three main stages—Data Management, Model Development, and Operations—where the first two stages include three phases each, and the final stage has two phases.

Data collection is the first phase of the data management phase in developing a DL system, followed by data processing and labeling phases to prepare the data that will be used in model development. These labeled data are typically partitioned into training, validation, and test datasets. Training and validation datasets are used in the model training phase of the DNN model. During the model training, the training dataset is employed to adjust the model's parameters, whereas the validation dataset is used to gauge the model's performance and to mitigate overfitting to the training data. The training of the model continues for a predetermined number of iterations or until the model attains the targeted performance on the validation dataset. The trained DNN is expected to be able to make correct predictions for new input that has not been encountered during the training. In the model evaluation phase, the test dataset is used to evaluate the performance of the trained DNN model, with test accuracy serving as a quality

metric. In the next stage, the DNN model is incorporated into the system in which it will function, followed by an evaluation of the DL system as a whole with all its components. Subsequently, the DL system is deployed for real-world use. Finally, ongoing monitoring is performed to ensure necessary maintenance.

The goal of the DL development life cycle is to provide a systematic and structured approach for creating a product intended to solve a problem considered to be addressable through DL techniques. Although DL systems can achieve remarkable success, matching or even exceeding human capabilities in complex tasks, they may exhibit incorrect behaviors in real-world scenarios. The vulnerabilities of DL systems may arise due to various factors like limited training data, inadequate training processes, and the inability to generalize to the deployment environment. Testing is the primary instrument to evaluate and identify their weaknesses before deployment to the real world.

Different levels of testing are carried out to systematically evaluate individual components, their integration, and the overall application. The primary testing levels in software engineering are Unit Testing, Integration Testing, and System Testing. The testing level is related to the granularity of the tested software. Definition of Integration and System Testing in software engineering can also apply to DL testing by considering a DNN model as a software component [5]. In Integration Testing, software components, hardware components, or both are brought together and tested to assess how they interact with each other. On the other hand, system testing is performed on a fully integrated system to assess its adherence to the requirements specified. In contrast, unit testing is defined as testing individual hardware or software units or groups of related units [6]. A specific part of a software program is tested in isolation from the rest of the software during unit testing. Similarly, in DL systems, *model testing* can be considered a form of unit testing where the DNN model is tested in isolation from the rest of the system. In the development workflow of DL, Figure 1, model testing is performed in the Model Evaluation phase, while integration and system testing are conducted in the Deployment phase.

Model testing is typically performed with the original test data. We use the term *original test data* for the test dataset used in the model development. Datasets used in DNN model development originate from the same data distribution. They are separated into train, validation, and test datasets. Although testing performed with original test data provides an objective evaluation of model performance, its capacity to comprehensively and adequately assess model quality remains uncertain.

In the field of software engineering, it is expected that software run in a deterministic and correct manner [7], where correctness is the adherence of software to its specified requirements [6]. Unlike conventional software that strives for deterministic correctness, DL systems are expected to operate within a reported accuracy range, which means that they are inherently "defective" [7]. For example, a DNN model with 99% accuracy might still incorrectly process one out of every hundred inputs.

As the DL systems are adopted in safety and security-critical systems like autonomous driving vehicles [8], medical treatments [9], and robotics [10], failures of these systems may lead to catastrophic results, including loss of human life. Therefore, ensuring the trustworthiness of DL systems, like conventional software systems, is crucial in these applications. As a result, testing of these systems has gained more attraction in the last decade, and the adequacy of typical model testing performed is questioned, especially compared to the extensive testing processes used in software development for safety-critical systems. A growing number of studies have been conducted in the field of DL testing with the objective of establishing a more systematic testing process and increasing trust in DL systems [11–46]. In this

regard, research is being carried out to test DNN models comprehensively beyond the tests performed with *original test data*.

The evaluation of the DNN model should be performed according to the quality requirements. Testing properties address why the test is performed and the guarantee that needs to be met by the DNN model. These properties may be functional or non-functional and include, but are not limited to, correctness, robustness, security, data privacy, fairness, efficiency, stability, and interpretability [21]. The field of quality assessment for artificial intelligence (AI)-based software is an active research domain, with ongoing efforts to refine the definitions and evaluation metrics of quality attributes [47]. While these attributes are widely recognized, the specific definitions and metrics used to evaluate them can differ. There are continuing studies for standardization and providing guidelines and best practices to assess and report the quality properties of these systems.

Correctness and robustness are the most frequently studied properties in DNN model testing. The correctness measures the probability that a model's predictions align with the actual labels of test data [21]. On the other hand, robustness measures the impact on the behavior of the model when small manipulations are introduced in the inputs [21,48,49], and these manipulated inputs do not necessarily need to be invalid. The definition of robustness for DNNs differs from the definition of robustness for software in the IEEE standard glossary of software engineering terminology [6]. Robustness for software is defined as "The degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions". However, DNN robustness is the capability of DNNs to tolerate perturbations in the input. This means that if the input undergoes a slight change, the DNN model's output should not be significantly affected. Furthermore, DNN robustness is categorized into various subcategories such as perturbation robustness, corruption robustness, and adversarial robustness [50].

In DNN model testing, black-box testing is performed without prior knowledge of the model's architecture or parameters. Black-box testing is a widely preferred approach as it simplifies the testing process by focusing on input-output relationships and assessing the functionality from an end-user's perspective. It is conducted by providing input to the system and checking the correctness of the output against the expected output, where it does not require internal knowledge or the implementation details of the tested system. However, the main challenge in this approach for DNN testing is identifying a test oracle to decide whether the actual output is the expected one. The challenge of distinguishing the correct behavior from potentially incorrect behavior is called the "test oracle problem" [51]. We need a ground truth to decide the correctness of the behavior of the model. Labeling of test data is often employed as a solution to the test oracle problem in DNN model testing, and it is generally performed manually by humans. Testing DNN models with every possible input requires substantial labeling and testing effort; besides, not all test inputs have the same value for testing. Consequently, this highlights the importance of the test data selection process, which aims to identify the most impactful test data from a vast pool of unlabeled data for testing purposes.

In this study, we focus on strategic test data selection in DNN model testing that is performed to evaluate the correctness and robustness properties of the model. Our objective in test data selection is to identify and prioritize fault-revealing test inputs from an extensive unlabeled test dataset beyond the *original test data*. We perform test data selection with a black-box approach without requiring access to the model's internal mechanisms. Furthermore, we explore identifying the underlying causes of test failures by enhancing our understanding of the model's behavior and potential weaknesses using DL explainability methods.

1.1 Challenges in DNN Testing

The data-driven development process makes DNNs different from programmed software. DNNs rely on training data to learn how to behave, rather than coding performed by humans according to well-defined specifications. Therefore, the behavior of a DNN is encoded in the parameters of the model architecture by training. Additionally, DNNs lack a clear decision path, making it harder to follow the execution flow compared to programmed software.

As a result of these differences, conventional software testing techniques and metrics can not be directly applied in DNN testing. Several studies have been conducted to transfer the knowledge/experience gained in software testing to DNN testing. In this context, in addition to developing novel DNN-specific testing methods, it is also studied how to adapt the methods used in software testing to DNN testing, such as fuzz testing, concolic testing, property-based testing, and mutation testing [13, 14, 16, 52].

Furthermore, to gain insight into the problems of the ML developers working in the industry, researchers have explored the challenges that large technology companies face in developing DL systems, even if they have advanced hardware, software, and skilled personnel resources [53]. However, numerous smaller companies also engage in ML system development despite lacking the extensive resources of their larger counterparts. Unfortunately, their models suffer from undetected failures resulting from missing or biased data and inadequate testing. According to these smaller companies, the primary cause of these failures is subjectivity in data collection and labeling. Furthermore, these companies often rely on labor-intensive testing using manually selected test cases, and they struggle due to the black-box nature of ML models [54].

These studies highlight a variety of challenges faced in the development of machine learning applications in different categories, such as issues in communication with stakeholders, the necessity for an end-to-end pipeline, and the quality of data. In the subsequent sections, we elaborate in detail on specific challenges that impact the effective evaluation of DNNs, which are the subjects of our research. Initially, we address the challenges posed by the test data selection from an unlabeled data pool. Subsequently, we discuss the influence of test data distribution on DNN testing. Lastly, we explore the challenges associated with analyzing the cause of failures encountered in tests.

1.1.1 Test Data Selection

Test data is vital in DNN model testing. A wide, diverse, and comprehensive test dataset is necessary to test DNN models adequately. To enhance the test dataset, additional real-world data can be collected. Although there are still fields where data is scarce and difficult to access, such as radiology images of a specific disease used in medical diagnosis, in general, digital systems gather enormous volumes of data daily. Consequently, DL systems also have an increasing number of data sources, and accessing large amounts of test data is no longer difficult. Furthermore, synthetic data generation techniques are frequently employed in DL testing to boost test data quantity and diversity [17–20].

However, before using the generated or collected data in testing, they need to be labeled, which is a time-consuming task typically done manually by humans. In some fields, expert human knowledge is also required for labeling. Therefore, there is a significant human labor requirement for labeling these

data. Let us suppose that DNN testers have access to a substantial pool of unlabeled data and can only afford to label a limited number of data. This challenge of choosing the most valuable inputs that will yield the most effective results upon labeling is referred to as *test data selection*. In this study, the most valuable inputs are those that lead to incorrect predictions by the DNN model.

Similar to test data selection, active learning was proposed for selecting the most valuable data to be used in the DNN training process. Active learning is a type of learning process that involves selecting a small subset of data that is most helpful for training a DNN model [55]. In active learning, training is conducted in multiple steps, where a subset of data is chosen for each step in an iterative manner. In test data selection, some of the methods and metrics used in active learning are also used for prioritizing test inputs. However, the difference between these two approaches lies in the state of the model and the aim of data selection. Active learning involves using an intermediate model and data acquisition techniques to identify a small subset of data that will be most helpful in training the model. The ultimate objective of this training is to produce a model with the same performance as when trained on the entire dataset. In this context, the selected subset of data is expected to be representative of the whole data. On the other hand, test data selection uses a fully trained model and test data selection is performed for different goals, such as detecting weaknesses of the trained model by identifying test inputs that will result in incorrect predictions.

Several test data selection approaches have been proposed in recent years [23–30]. Test data selection can be performed for different objectives. The objective of test data selection defines the goal of testing. A widely used objective of test data selection is prioritizing the fault-revealing test data. Test inputs that cause the tested DNN model to make incorrect predictions are referred to as *fault-revealing* inputs. In a different category, test data selection is performed in order to estimate the model's performance on all the data, using only a selected subset of it [56, 57]. Additionally, test data selection is performed with the aim of increasing the model's performance (accuracy, robustness, etc.) after retraining the model with the selected test inputs added to the training dataset [24, 26]. In this study, test data selection is performed with the objective of identifying and prioritizing the fault-revealing test data.

1.1.2 Distribution Shift

The data space of a DNN is represented with input and label space. DNNs are trained to optimize their parameters for solving a problem within a specific data distribution represented by the training dataset. During training, a DNN learns the relationship between input and label spaces. When there are differences between the statistical properties of training and test datasets, it is referred to as a data distribution shift. Distribution shift can occur in either input space distribution, label space distribution, or both. A distribution shift can be a covariate or semantic shift [58]. In covariate shift, the label space remains the same across two datasets. Examples of a covariate shift include style changes, domain changes, or perturbations due to adversarial attacks. A dataset with cartoon cat images is a domain change for an image classification DNN model trained for categorizing the cats from real cat images. On the other hand, in the case of a semantic shift, the label spaces can be different between training and test datasets, or a semantic shift may cause a change of label.

DNN models are expected to handle covariate data shifts as part of their generalization range. Model generalization is the capability of the DNN model to make correct predictions for data that it has not encountered in the training dataset but has the common input features and label space with the training dataset. However, the extent of the shift is important for the model to handle it effectively. As the

degree of the shift increases, it becomes more challenging for the model to make correct predictions as the new data significantly differs from the training data.

Data originating from a data distribution different from the training data distribution is called out-of-distribution (OOD) data [59]. Conversely, data that aligns with the training distribution is referred to as in-distribution data. The training and validation datasets used in model development are in-distribution data as well as the *original test data*. When perturbations are applied to in-distribution data, the resulting data can be either in-distribution or OOD. It depends on the degree of perturbation and also the properties of the data. Natural perturbations are applied to simulate real-world corruptions, such as camera blurring and changes in brightness. On the other hand, adversarial perturbations are small manipulations performed to deceive the DNN model and produce incorrect output. In this study, we use the input data categorization presented in Figure 2.

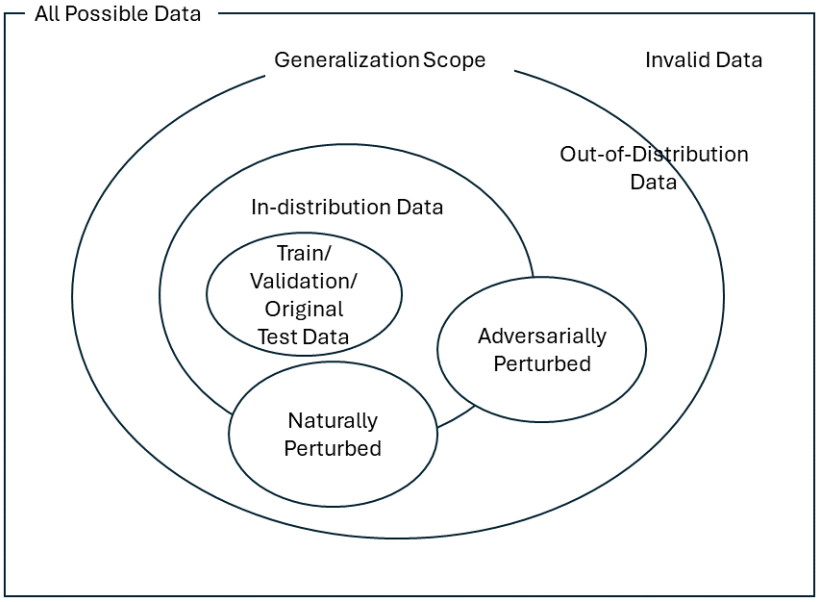


Figure 2: Input data categorization according to data distribution.

Detecting OOD data is a difficult task for which several methods have been proposed [58]. There is no ground truth to validate whether a sample is OOD or not unless semantically separate datasets are used. Semantic data shifts in which datasets have distinct data label spaces have been the primary focus of OOD detection research [60].

There has been limited research on DNN testing that pays attention to test data distribution [19, 20, 24, 43, 46]. The difference between the training data and test data distributions affects the performance of the DNN model in making accurate decisions. This effect should be considered while constructing the test datasets and assessing the test results. When a tested DNN model fails to make a correct prediction for a test input, it is hard to decide whether the misprediction is due to the improper model structure, inadequate training process, etc., or the DNN model training dataset does not contain data similar to this test input.

The recent DNN testing studies demonstrate the need for additional research on distribution-aware testing. These studies mainly fall into two categories. The first category of studies evaluates the

impact of OOD data on the testing of DNNs [24, 46]. OOD detection techniques are employed to identify OOD data in test datasets that have been generated by transformations or adversarial attacks instead of treating all transformed data as OOD. The second category involves studies focused on generating solely in-distribution test data using synthetic data generation methods. This approach is quite different from the first category and is seen in the researches [19,20,43] where generative models were used to create test data.

Berend et al. [46] compared the ratios of OOD data in test datasets generated using different test criteria. They also revealed that the misclassified in-distribution test data were correctly classified by DNN models with a different architecture from the tested model but were trained with the same training dataset, indicating that the misclassifications were model-related. However, changing the model did not result in the same level of improvement for the misclassified OOD data, which is mainly related to the difference between the training and test datasets. Furthermore, they investigated the robustness and accuracy improvement in models after retraining the model with additional in-distribution and OOD data to the training dataset. Their results showed that although incorporating OOD data into the training dataset enhances the model's performance, it can eventually degrade the overall performance if the OOD data significantly deviates from the training data distribution.

The distribution of the test dataset is a critical property that should be aligned with the specific objectives of the DNN testing process. By carefully selecting and tailoring the dataset, it can be ensured that the evaluation is meaningful, relevant, and aligned with the intended testing goal. Software testing is a common practice for verifying and validating software. During software verification, testing is conducted to ensure that the software meets the specified requirements. Verification answers the question, "Was the system built correctly according to the specification?". If any failure is detected during the verification process, the software is updated to fix it. Once the software is verified, validation testing is performed to demonstrate that the software functions according to the user's needs in real-world scenarios. Validation answers, "Was the right system built?" and ensures that the system meets the expectations of the end users. If any feature is missing in the system, a new requirement is defined to ensure the software meets the user's expectations.

When we adopt these definitions to DNN development, verification testing for DNN models primarily focuses on in-distribution test data. In software verification testing, the software is tested against the requirements specifications. There are no explicit written requirements for DNN models as opposed to conventional programmed software. DNN models learn how to behave from the training data. Consequently, the requirements of DNNs are defined by the training dataset, and requirements change if the training data changes. DNN model testing with a test dataset that is coming from a similar data distribution to the training dataset can be considered verification testing since the behavior of the DNN model is based on training data, and they can be thought of as specifications defining model behavior. Validation testing, on the other hand, involves assessing the model in its operational environment with real-world data. Model validation testing evaluates the model's performance using data that tests its ability to generalize beyond the training dataset distribution. If the model fails to make correct predictions due to insufficient training data, this is considered a deficiency in requirements. Addressing this involves adding missing data to the training dataset and retraining the model, thereby enhancing its functionality.

In this study, we propose testing the DNN model first with data on which it is expected to make accurate predictions. This involves testing the model with in-distribution data, which shares a similar data distribution to the training dataset. After the model demonstrates satisfactory results with in-

distribution test datasets, its performance can be tested using test datasets containing OOD data to assess its ability to generalize. This approach can be considered analogous, starting with verification testing and proceeding to validation testing.

1.1.3 Analyses of the Test Results

In the conventional software engineering development process, after identifying the test cases that resulted in failure, which means that the expected output and the actual output of the software do not match, software developers analyze the test results and debug the software. However, identifying the root cause of incorrect prediction of DNN models is challenging. In most of the DNN testing studies, after determining the test data that result in incorrect predictions, they are added to the training dataset, and the model is retrained to improve its accuracy [23–26, 29].

In this way, the model might be trained to accurately categorize similar cases in the future. Furthermore, the inclusion of rare data instances or underrepresented classes in the training dataset might enhance the generalization capability of the model. However, if the test inputs are not representative of the distribution as a whole or if the model begins to memorize specific cases instead of learning generalizable patterns, there may be a risk of overfitting to the newly included specific data. Besides, adding incorrectly predicted data to the training dataset alone does not solve possible underlying problems like improper model architecture and training process.

1.2 Proposed Solution

This study aims to improve fault-revealing test data selection and debugging for DNN models by designing a testing framework to help DNN developers detect weaknesses in the DNN models and determine their root causes.

It may not be possible to test the DNN model with every possible test input due to limited resources and time available. The selection of the data to be labeled and used in the test is crucial. In test data selection, we prioritize the test inputs that will cause failures in the tested DNN model. When a DL component fails to perform its intended task due to human mistakes during DL development, it is referred to as a DL *fault* occurrence [61]. Incorrect configuration of learning parameters or the use of an unbalanced training set can be examples of a DL fault. Additionally, we use the term *failure* for incorrect prediction of the model for a test data input. Conventional software is programmed to output error codes or messages in case of failures. However, DNN models do not have the capability to output error codes. For instance, when we consider a classification DNN model designed to classify data into ten different categories, it will always produce an output belonging to one of these ten categories. This remains true even if the input to the model does not belong to any of these categories. The model will still output a result, classifying the input into one of the predefined categories, regardless of its actual relevance.

There is a question of how we can choose fault-revealing data from an unlabeled data pool to use in DNN testing automatically. This can be achieved by identifying the most challenging data for the model. In the literature, for identifying fault-revealing test data inputs, several test data selec-

tion methods are used, including confidence-based [25], mutation-based [28], and neuron activation frequency-based [36].

In this study, we prioritize test data based on the model's degree of uncertainty. When a model is uncertain, it is more likely to make an incorrect prediction, and that is where human expertise in labeling can have the most value. The degree of uncertainty in DNN refers to how much the model is unsure of its prediction. There may be various reasons for the uncertainty of the model, such as inadequate training data, improper model architecture or training procedures, model complexity, and overfitting. DNN models can yield incorrect predictions with high confidence. To mitigate this issue, uncertainty estimation is utilized to gain deeper insights into the model's predictions. We used the model's uncertainty to prioritize the test data by employing uncertainty quantification methods. In uncertainty-based test data selection, the test input that creates the highest uncertainty in the model is given the highest priority.

Prioritizing test data based on uncertainty metrics leads to earlier detection of fault-revealing inputs compared to random data selection. For example, assuming that the tested model will incorrectly predict 20 out of 100 unlabeled test data. It may be necessary to label 50 data using the random selection method to detect 10 data that will be mispredicted, while this number will be much lower with prioritized data. This means that the same number of fault-revealing data can be detected by labeling fewer data than randomly selected data.

We proposed a DNN testing framework that emphasizes uncertainty-based test data selection to identify fault-revealing data. As shown in Figure 3, this framework has three main phases: Test Data Generation, Test Data Selection, and Test Results Interpretation. Our framework separates the evaluation of the model's performance in settings not included in the training dataset (i.e., the model's generalization scope) from the evaluation of its performance in the context it is anticipated to handle (i.e., in-distribution test data). To achieve this, OOD data detection is incorporated into the test data selection process. This approach involves identifying data that exhibit significant covariate shifts, exceeding a pre-established threshold, as OOD. Consequently, test datasets are organized into those containing OOD data and those comprising solely in-distribution data. Testing is conducted first with in-distribution datasets, followed by testing with datasets that include OOD data.

After conducting tests and determining the test data that the model fails to predict correctly, we propose performing an analysis with these test data to provide an explanation for the model's decision. The concept of explanation in machine learning (ML) refers to the information or justifications provided by the model to facilitate the comprehension of its functioning for its intended audience [62]. Machine learning models can be inherently interpretable (transparent models) or explained through external techniques. DNNs, which are regarded as black-box models [63], necessitate the latter approach. Post-hoc explainability techniques are utilized to clarify the model's reasoning for specific predictions without altering the model's architecture or training process. Hence, our framework employs post-hoc explainability methods to gain insights into the decision-making process behind the incorrect prediction of the model. If the model's reasoning for making an incorrect prediction can be understood, informed decisions can be taken to improve the model's performance.

We evaluated the proposed framework by conducting case studies in the image classification domain with three different datasets and several DNN models. We instantiated each phase of the proposed framework with carefully selected methods.

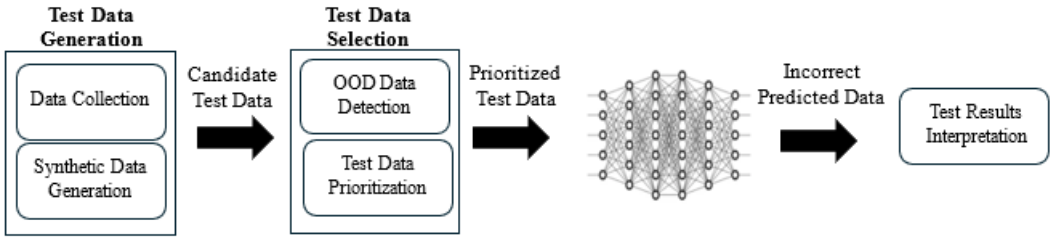


Figure 3: Proposed DNN Test Framework.

For OOD data detection, we evaluated two different methods. We used OOD detection methods that identify test inputs that exhibit covariate distribution shifts; they differ from the training data while retaining similar semantic properties and label space. The first method is based on the use of a generative model, namely Variational Autoencoder (VAE), and its reconstruction probabilities to determine whether a test input is OOD or not. The second method is based on the use of a pre-trained Visual Transformer (ViT) model and its embeddings. This method uses the distance between embeddings of the test input and training dataset to differentiate OOD and the in-distribution data.

State-of-the-art methods for OOD detection have already achieved significant success in detecting far OOD data, but identifying near OOD data is a challenging problem. The test datasets generated using test image generation methods (such as image transformation, adversarial attacks, and generative models) create data that are semantically similar to the original dataset but have a covariate shift. We specifically chose the OOD detection method that employs the ViT models due to its high detection performance, even with near OOD datasets. We evaluated the success of the selected methods with OOD datasets, and the results demonstrated that these methods achieved successful discrimination of the OOD data.

In the test prioritization phase, we employed several uncertainty quantification methods, including the Deep Ensemble (DE) method [64], Variational Inference with the Flipout (VI-F) method [65], with widely recognized uncertainty metrics Margin, Entropy, and Least Confidence. Then, to reach measurable results, we assessed the success of test data prioritization methods in terms of identifying fault-revealing inputs by using several evaluation metrics recognized in the DNN testing literature. These metrics are the Average Percentage of Fault Detected (APFD) [25, 37, 66, 67], Fault Detection Ratio [27,29], and Ratio of Area Under Curve (RAUC) [28,29,34,68]. Also, statistical correlation tests (Point-Biserial and Spearman) are performed to evaluate the correlation between the mispredictions of the model and test selection metrics.

Based on the results we achieved with these test data selection methods and metrics, we proposed a novel test data prioritization approach based on using a meta-model. We employ a meta-modeling approach where a meta-model is positioned on top of base models and learns to make predictions based on the outputs or decisions of underlying base models [69]. The meta-model is referred to as a model that acts as an additional layer of learning that utilizes the outputs of the base models to improve overall performance, adapt to different tasks or datasets, or provide more reliable predictions. In our approach, the meta-model gets uncertainty metrics as input, which are derived from the outputs of other base models. Integrating different uncertainty metrics helps overcome individual limitations of these metrics and be effective in a broader range of scenarios. We trained the meta-models with the objective of predicting whether a test input would lead the tested model to make an incorrect prediction or not. Then, we used the outputs of these meta-models to prioritize the test data.

Following the identification of test data that led to incorrect predictions by the DNN model, we employed visualization techniques with these instances. We explored the effectiveness of Grad-Cam [33], Grad-Cam++ [34], and Score-Cam [35] for test data visualization with the goal of understanding the cause of incorrect predictions.

1.3 Research Questions

The research approach of this thesis is based on the Design Science Research (DSR) method. Design science is defined as the "design and investigation of artifacts in context" [70]. It iterates on designing an artifact to solve a problem and creating knowledge by answering questions through the evaluation of the artifact. Conceptualizing problems, designing solutions, and validating them are the three main components of the design science paradigm. In this study, we started with defining the problem by performing a literature survey on DNN testing. Then, we proposed a solution for the identified problems and improved the solution by implement-evaluate cycles. Finally, we performed validation of the solution through multiple-case experiments.

In this thesis, we aim to answer the following research questions:

- RQ1: What is the effectiveness of uncertainty-based test selection methods in identifying and prioritizing fault-revealing data instances within datasets generated through different techniques, and how does this effectiveness vary under different conditions?

  RQ1.1: How effective are the test selection methods when the datasets exclusively consist of in-distribution data?

  RQ1.2: What variations in effectiveness are observed when the datasets contain both OOD and in-distribution data?

- RQ2: How effective is the test prioritization approach proposed in this study, which is based on a meta-model that combines different uncertainty metrics, in identifying and prioritizing fault-revealing data instances within datasets, both include and exclude OOD data?

  RQ2.1: How does the effectiveness of the proposed method change according to different labeling budgets (i.e., the number of test data to be selected and prioritized)?

- RQ3: Is it possible to use post-hoc explainability methods in DNN testing to understand the root cause of test failures (i.e., incorrect predictions of the tested DNN model)?

1.4 Contributions of the Study

Our primary focus is test data selection with the aim of identifying fault-revealing test inputs in DNN model testing. The key contribution of our study is the notion of distribution-aware testing with uncertainty-based test prioritization. Our goal is to identify and prioritize test inputs from an extensive unlabeled dataset that will be most useful in exposing the weaknesses of a DNN model, considering a limited labeling budget imposed by a fixed number of test data.

We proposed a novel test data prioritization method that improves the identification of fault-revealing test inputs in DNN testing. It achieves this by effectively combining multiple uncertainty metrics with

meta-models and using the output of this model for test data prioritization. The proposed method demonstrated superior performance over state-of-the-art test input prioritization techniques. This improvement is a valuable contribution to the domain, optimizing the efficiency of DNN testing processes.

This study explored the use of out-of-distribution detection methods for creating test datasets that take into account the distribution of data. This approach handles out-of-distribution data in test datasets, which is a challenging aspect of DNN testing. We introduced the idea of conducting DNN testing in two phases: one with test datasets comprising exclusively in-distribution data and another with datasets that include both in-distribution and out-of-distribution (OOD) data.

We assessed the efficacy of test input prioritization based on the model's uncertainty in detecting inputs that reveal faults. We compared several state-of-the-art uncertainty quantification methods and metrics. To the best of our knowledge, our study is the first to investigate the Deep Ensemble and Variational Inference with Flipout uncertainty quantification methods for test data selection.

We performed empirical evaluations across different types of datatypes and various DNN architectures to evaluate the effectiveness of different test data selection techniques, comparing them using several metrics recognized in the DNN testing literature.

Besides, this study also proposes the idea of debugging through posterior model explanations. We investigated the use of visual post-hoc explanation methods to enhance understanding of the model's decision-making logic for incorrectly predicted test data and make suggestions to improve model performance.

We published the following studies from this thesis:

- D. Demir, A. Betin Can, and E. Surer, "Distribution Aware Testing Framework for Deep Neural Networks," *IEEE Access*, vol. 11, pp. 119481–119505, 2023.
- D. Demir, A. Betin Can, and E. Surer, "Test Selection for Deep Neural Networks using Meta-Models with Uncertainty Metrics," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ACM, 2024.

1.5 Organization of the Thesis

The remainder of this dissertation is structured as follows: Chapter 2 outlines the research methodology used in this study. Chapter 3 includes background information and a literature review on DNN Testing, DL Uncertainty, and DL Explainability. Chapters 4 and 5 provide detailed descriptions of the proposed methodologies, with Chapter 4 introducing a DNN testing framework and Chapter 5 proposing improvements in the test data selection phase of this framework. Chapter 6 describes the experiments conducted to evaluate the proposed methodologies, while Chapter 7 presents the results of these experimental evaluations. Chapter 8 provides a thorough discussion of our findings, accompanied by limitations of our research. Chapter 9 concludes this dissertation and offers recommendations for future research directions.

CHAPTER 2

RESEARCH METHODOLOGY

In this chapter, we provide information on how we applied the Design Science Research (DSR) methodology in this thesis. First, in Section 2.1, design science research methodology is introduced. Then, the subsequent subsections of this section provide details of the steps we followed, starting with problem identification, continuing with the development and evaluation of the solution, and concluding with the dissemination of our study's findings.

2.1 Design Science Research

Design science research (DSR) is a research methodology focused on creating and evaluating artifacts within a specific context in order to address identified problems. The process of design science involves iterating over developing solutions to design problems and providing answers to knowledge questions [70]. A design problem in DSR is a real-world problem that requires creating or improving an artifact, while a knowledge question is an inquiry made with the goal of contributing to the body of knowledge related to the design problem. With knowledge questions, the researcher explores new insights during the development and evaluation of the artifact. Knowledge questions can be empirical or analytical. Empirical questions require data to be collected during the evaluation of the artifact, whereas answers to analytical questions can be generated through conceptual analysis.

The term "artifact" is crucial to DSR and refers to something that is created by humans. The main types of artifacts recognized in DSR are constructs (vocabulary and symbols), models, methods, prototypes, systems, algorithms, guidelines, and design theories (enhanced models of design or design processes) [71].

Offerman et al. [72] defined Design Science Research (DSR) as consisting of three main steps: "Problem Identification", "Solution Design", and "Evaluation." Feedback loops are commonly observed between these steps. It may be necessary to re-investigate the problem during solution design, or new problems may emerge after the solution is evaluated. This loop typically occurs between the "Solution Design" and "Evaluation" encompassing ongoing improvement across several cycles of development and evaluation steps until the necessary solution is achieved. Each cycle contributes to a better understanding of the problem and enhanced artifact, resulting in more significant contributions, Figure 4.

Runeson et al. [73] elaborated the constructs of design science for software engineering and framed the DSR for empirical software engineering with five main activities: problem conceptualization, solution



Figure 4: Design Science Research Main Activities.

design, abstraction, instantiation, and empirical validation. In the problem conceptualization step, the problem is identified and described from the problem instances. Next, a general solution is developed in the solution design step, and subsequently, this general solution is implemented in a context in the instantiation step. Finally, empirical validation assesses how effectively the implemented solution resolves the problem. On the other hand, abstraction identifies the critical design choices within the solution's defined scope of applicability.

We followed their approach with the addition of the "Communication of Results" step as a final step in the same way as "Summarize Results" in [72]. Furthermore, we employed Literature Research in Problem Conceptualization and Solution Design steps during the identification of the problem and designing artifact as outlined in [72].

A single-case mechanism experiment involves testing a mechanism in a single object of study to observe and explain the cause-and-effect behavior of a specific study subject, [70]. On the other hand, multiple case studies involve a thorough analysis and comparison of various instances or cases. This approach helps in understanding how the artifact performs in different contexts and in identifying patterns and commonalities that support the generalizability and reliability of research findings. In this study, we employed multiple-case experiments to validate the proposed solutions.

The research process we followed is presented in Figure 5. Moreover, we illustrated our study with the visual abstract template proposed by Storey et al. [74] to describe the design science constructs in our research in Figure 6

2.1.1 Problem Conceptualization

The first step of design science research is the identification of the research problem. Our research started with the goal of understanding why some of the faults that are causing failures of DL systems, particularly those deployed in safety-critical environments, go undetected during the development phase. Traditional software development for safety critical systems is subject to rigorous certi-

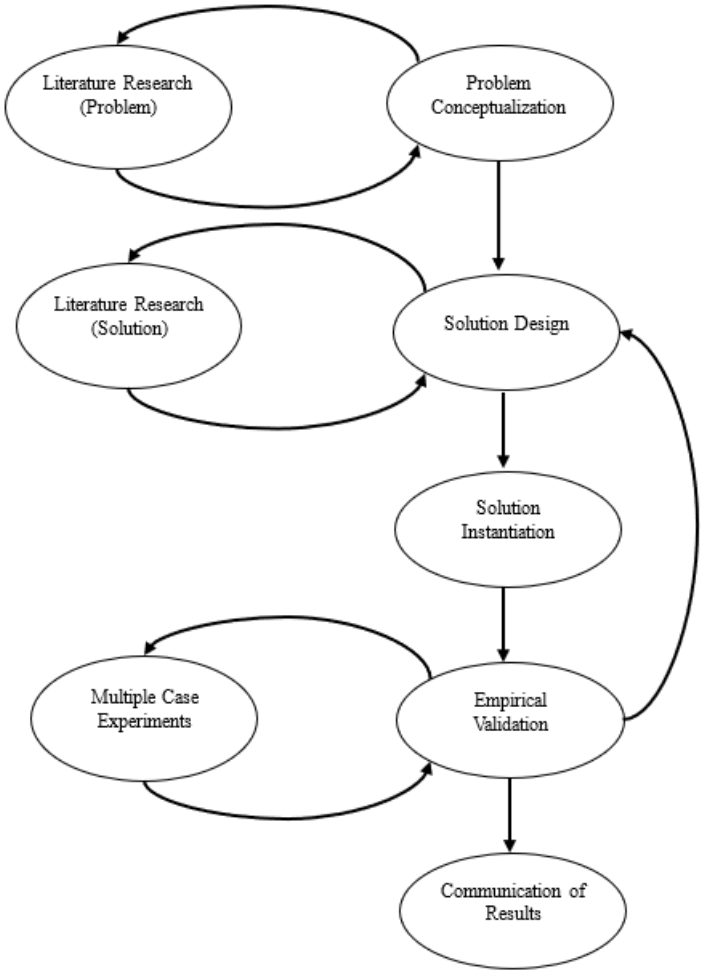


Figure 5: Applied Design Science Research Process.

fication standards that must be met before deployment, including specific activities during the testing phase. Based on this idea, we conducted a comprehensive literature review of testing methodologies applied to DL systems and specifically to DNN models. Our review revealed ongoing studies focused on adapting established software engineering principles to the development of DL systems and the challenges faced in this context [75–80]. We also gained insight into the challenges encountered in DNN testing [4, 5, 21, 81–84]. From this review, we identified three distinct problem instances and committed to exploring potential solutions for these issues:

- Effective testing of a DNN model requires diverse and comprehensive test datasets. Test data need to be labeled, which requires manual human effort and sometimes expert knowledge.
- The distribution of the test data highly affects the performance of the DNN model and is an important factor when assessing the results of tests.
- The nature of DNN models hinders the traditional debugging processes for incorrect predictions.

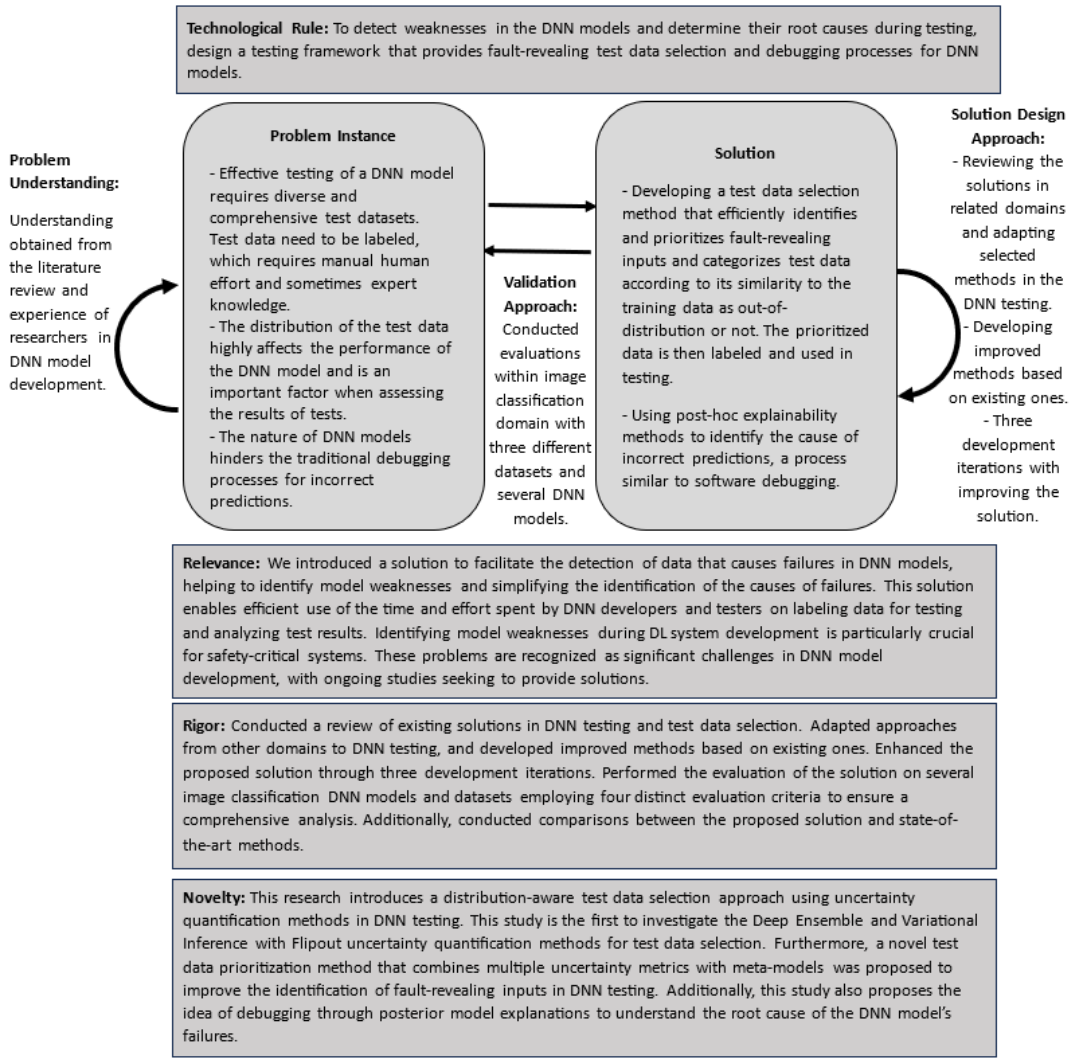


Figure 6: Visual abstract of this study with Design Science Research constructs.

We reviewed the DNN testing literature and focused on a detailed examination of existing research related to test data selection for DNN testing. This review enabled us to gain insights into the metrics and methods employed in this area.

2.1.2 Solution Design, Instantiation, and Validation

Runeson et al. defined the *technological rule* as generalized knowledge about the relationships between problem instances and their solutions, expressing it in the following form:

To achieve <Effect > in <Context > apply <Intervention>

The applicability of the solution is determined by the intended impact of the proposed intervention in a specific context. We expressed the technological rule of this study as follows:

To detect weaknesses in the DNN models and determine their root causes during testing, design a testing framework that provides fault-revealing test data selection and debugging processes for DNN models.

The solution design phase primarily involves exploring potential solutions and developing the artifacts. During this phase, we delved into the existing knowledge base to review state-of-the-art solutions for the identified problems and to investigate how solutions to other problems might shed light on our research problems. Offerman et al. [72] emphasize that artifact design is not a standardized process but rather a creative engineering process that often lacks detailed guidance.

For the problems we identified for our research, we proposed the following solutions:

- Developing a test data selection method that efficiently identifies and prioritizes fault-revealing inputs and categorizes test data according to its similarity to the training data as out-of-distribution or not. The prioritized data is then labeled and used in testing.
- Using post-hoc explainability methods to identify the cause of incorrect predictions, a process similar to software debugging.

We formulated this solution as a DNN testing framework with Test Input Generation, Test Data Selection, and Test Results Interpretation phases. We then refined the Test Data Selection phase through iterations to improve our solution. During each iteration, we instantiated the solution and conducted empirical validations.

We chose the image classification problem to implement our solution. We instantiated the proposed DNN testing framework by selecting specific methods for each phase of it. These methods include test input generation, out-of-distribution (OOD) detection, test data prioritization, and visual explainability methods. After implementing this framework with the selected methods, we proceeded to Empirical Validation.

Validation of the proposed solution was performed through multiple-case mechanism experiments. These studies involved two different datasets, each with differing image properties, alongside selected two DNN models appropriate for each dataset. We conducted empirical evaluations using these datasets and DNN models. Based on the results of this evaluation, we developed and proposed two new test data selection metrics. Subsequently, we performed empirical evaluations for these metrics using the same experiment setup. In the third iteration, we introduced a novel test data prioritization strategy and implemented an improved OOD detection method. In the evaluation of this approach, we incorporated an additional and more complex dataset and a new DNN model into the experiment setup. The solutions proposed in three iterations are detailed in Chapters 4 and 5, while the empirical validations conducted for each iteration are discussed in Chapters 6 and 7.

In summary, we presented our solution through two methodologies and conducted five distinct experiments for empirical validation of these methodologies. The results of these experiments enabled us to answer the three research questions outlined in Section 1.3 of the Introduction chapter. The following list provides a mapping between these methodologies, experiments, and research questions.

- Methodology - 1
  - Experiment - 1 (RQ-1)
  - Experiment - 2 (RQ-3)
- Methodology - 2
  - Experiment - 3 (RQ-1)
  - Experiment - 4 (RQ-1)
  - Experiment - 5 (RQ-2)

2.1.3 Communication of Results

We prepared two papers documenting the results of our design science research to contribute to the body of knowledge. The first paper details the research conducted in the scope of the methodology presented in Chapter 4, providing insights into its effectiveness. The results of the experiments performed in the scope of this methodology were published in this paper. The second paper focuses on the meta-model based test data selection methodology described in Chapter 5 and the corresponding results, offering a detailed analysis of its application and outcomes. These papers ensure that our research findings are effectively communicated to both academic and practitioner communities to advance the field and support future research endeavors.

- D. Demir, A. Betin Can, and E. Surer, "Distribution Aware Testing Framework for Deep Neural Networks," *IEEE Access*, vol. 11, pp. 119481–119505, 2023.
- D. Demir, A. Betin Can, and E. Surer, "Test Selection for Deep Neural Networks using Meta-Models with Uncertainty Metrics," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ACM, 2024.

CHAPTER 3

BACKGROUND AND RELATED WORK

3.1 DNN Testing

Deep Learning (DL) is a data-centric approach that leverages deep neural networks (DNNs) to extract decision-making patterns from data. Over the last ten years, there has been a growing interest in DNN testing, leading to a rapid increase in proposed testing techniques. These techniques primarily address (i) test data generation, (ii) test adequacy evaluation, (iii) test data selection, and (iv) test oracle determination [11–21].

3.1.1 Test Oracle

Several studies have been conducted to adapt traditional software testing knowledge and methodologies to DNN testing while minimizing the need for labeled data. Notably, metamorphic relations and cross-referencing have emerged as preferred test oracles since they reduce the necessity for manual labeling. Metamorphic testing involves analyzing software functions to identify inherent metamorphic relations, which describe how changes in input correspond to changes in output. In metamorphic testing, it is a common approach to transform the input data so that it does not change the expected output.

Applying this concept to DL systems, metamorphic testing is similarly defined. For instance, in metamorphic testing of an image classification DNN, transformations like blurring or rotation can be applied to input images. These transformed images are then tested using the same labels as the original images since the expected output remains unchanged despite the alterations in the input data. Metamorphic transformations are commonly used in DNN testing [11,12,15,17]. When using metamorphic relationships as a test oracle without altering the output, the variations between the inputs should not cause semantic differences to prevent a change in the output. However, this constraint limits the diversity of the test dataset.

Cross-referencing is another method for creating test oracles, incorporating techniques like differential testing and N-version programming. Differential testing, a traditional software testing technique, aims to identify software faults by checking if similar applications produce different results for the same inputs. In the context of DL systems, differential testing involves training multiple DNN models for the same task and comparing their outputs. If the outputs differ, the input is considered a fault-revealing test input for the system. However, differential testing in DL is a costly approach as it requires training

multiple models and running them during inference. Additionally, there is a risk that all DNNs may produce the same erroneous output, potentially masking faults.

3.1.2 Test Adequacy

Test adequacy criteria are used to quantitatively measure whether existing test cases are satisfactory to stop the testing process and whether there is sufficient confidence that the software under test is working as intended. One of the main test adequacy criteria in software testing is code coverage. Code coverage is used to determine the extent to which software source code has been tested. It is measured according to covered (which means executed with test cases) source code branches/statements. The belief is that software with high code coverage value is less likely to contain unidentified software faults. There are different code coverage types widely used in the industry: Statement Coverage, Decision Coverage, Condition Coverage, and Modified Condition/Decision Coverage [85]. These coverage criteria are required to be satisfied in some of the certification standards for the safety-critical software [86].

Since the decision logic of DNN systems is different from that of programmed software, existing coverage criteria cannot be directly applied to DNN testing. From a similar perspective, neuron coverage (NC) was proposed as a DNN test coverage criterion by Pei et al. in the study of DeepXplore [12]. Neurons whose activation outputs are greater than a specific threshold value are expressed as active, and the ratio of active neurons to total neurons in DNN is defined as NC. Several variants of DNN coverage criteria are proposed based on the NC definition [33,87]. Extensive research has been conducted on coverage-guided test data generation [13, 14, 41, 42], in which test data that increase coverage criteria are synthetically generated. However, the effectiveness of neuron coverage criteria in generating test data is challenged by many researchers concerning the naturalness of generated test data and their contribution to fault detection [88, 89]. Dong et al. argue that there is a limited correlation between the neuron coverage criteria and the robustness of the model, and increasing the coverage does not necessarily increase the robustness of the model [90]. Furthermore, even with a limited number of test inputs, it is possible to achieve extensive neuron coverage. However, increasing neuron coverage does not necessarily lead to greater diversity in test inputs.

Surprise Coverage (SC) is another coverage criterion proposed for the adequacy of DNN tests [22]. It measures the novelty (surprise) of a new input relative to the training inputs by discretizing input ranges. Neurons in the DNN activate in response to inputs, forming an Activation Trace. These activation values are recorded for the training dataset. Then, the activation trace of each new test input is calculated and compared with the activation trace values of the training dataset to determine its Surprise Adequacy (SA). There are two variants of surprise adequacy: Likelihood-based Surprise Adequacy (LSA) and Distance-based Surprise Adequacy (DSA). LSA uses Kernel Density Estimation to calculate the probability density of the values in the activation trace and compares this value with the new input to compute the surprise adequacy. DSA uses Euclidean distance to compare the activation trace of the new input with the activation trace of the training set.

The SC is calculated using the computed SA values. Since SA values are defined in continuous space, they are discretized using bucketing. Test data is distributed into these buckets based on their SA values. The SC value is defined as the ratio of the number of buckets containing test data to the total number of buckets. A high SC indicates a diverse set of test inputs, ranging from ones similar to the training data to ones very different from the training data.

3.1.3 Test Data Selection

Several test data selection techniques have been proposed in the literature, which can be broadly classified into three categories: uncertainty-based, coverage-based, and mutation-based techniques [23–30]. Later on, test data selection methods by considering multiple concerns together to improve the test dataset were also proposed [30, 31].

Coverage-based methods aim to increase the neuron coverage [12] by constructing a test dataset that activates the maximum number of neurons possible [11, 12, 32, 33]. Therefore, at each test input selection, the goal is to find data that activates neurons that have not been previously activated. However, it has been shown that higher neuron coverage does not always correlate with test data, which leads to incorrect predictions by the DNN model [88, 91]. On the other hand, mutation-based methods involve applying mutation operators to the DNN model to create mutant models and selecting test inputs likely to cause incorrect predictions of the mutated models [28, 34]. The core principle in mutation-based test data selection is that test inputs causing different outputs between the mutant and the original models are more valuable for identifying faults. However, these methods tend to have a higher computational load compared to other techniques.

Furthermore, uncertainty-based methods prioritize test inputs by estimating the DNN model's uncertainty for a given input based on its classification probability output [25, 35]. Test data with higher uncertainty are prioritized as the model is considered less confident in its predictions. Studies have demonstrated that, compared to neuron coverage-based test data selection methods, uncertainty-based methods are more effective at identifying inputs that reveal faults in the model [23, 25].

The DeepGini [25] is introduced as a statistical technique for prioritizing test data. In classification problems, the model's uncertainty about its predictions increases when the predicted probabilities for different classes are closer, resulting in a higher risk of misclassification. DeepGini proposes a metric similar to entropy but simpler to calculate based on the statistical concept of impurity.

$$\text{DeepGini} = 1 - \sum_i {p_i}^2 \quad p_i : \text{predicted probability for class } i \tag{1}$$

DeepGini outperformed neuron coverage and surprise coverage guided test data selection in terms of identifying fault-revealing test data and improving accuracy after retraining the model with the selected test data. Afterward, DeepGini has become a benchmark in test data selection studies in the literature.

Shen et al. proposed the multiple-boundary clustering and prioritization (MCP) approach, which focuses on defining class boundaries and prioritizing test data from these boundary areas [26]. The MCP clusters test data into boundary regions based on the first and second most probable classes. It selects samples from these boundary areas by calculating the priority as the ratio of the probability of the first class to that of the second class. Test data are evenly chosen from each non-empty cluster with higher priorities. The underlying idea of MCP is that if the top two probabilities are close to each other for a test input, it may be on the decision boundary between these two classes. DNN models are likely to make incorrect predictions for these data at the boundary areas.

Byun et al. [29] used Likelihood-based Surprise Adequacy (LSA) and Distance-based Surprise Adequacy (DSA) definitions and proposed a test data prioritizing technique based on the surprising degree of test inputs. This method assigns higher priority to test inputs that exhibit high surprise values.

TestRank method, which is proposed by Li et al. [30], is a test selection method that combines intrinsic and contextual attributes of the test data and uses it for test data prioritization. Intrinsic attribute values are calculated from the probabilistic output of the model for the test inputs with metrics such as entropy, confidence, etc. A multi-step process is employed to determine contextual attributes. A feature extractor is used to extract the features of each test input, and using these features, a k-nearest neighbor graph graph is formed between them. Then, a Graph Neural Network (GNN) model is used to extract the contextual attributes from the similarity graph. Finally, the Multi-Layer Perceptron model is used to predict the fault-revealing capability of input from the intrinsic and contextual attributes. This study is different from the previous ones in terms of combining metrics with different objectives into a single selection metric.

Zhang et al. [36] and Yan et al. [37] proposed alternative prioritization methods that use neuron activation values for test data selection. Zhang et al. [36] categorized neurons as either active or inactive based on their activation values during training and calculated the activation frequency of all neurons for each class. They proposed that if neurons frequently activated for a particular class are also activated by a test input, the input likely belongs to the same class. They introduced a familiarity score based on neuron activation, suggesting that test inputs with lower familiarity scores for the class predicted by the DNN model may induce incorrect predictions.

In a follow-up study, Yan et al. [37] refined this approach by using neuron output values before the activation function for pattern extraction. They classified neurons as active, inactive, or noisy for each class based on neuron valuations collected during training. For new test inputs, they assessed the familiarity of neuron patterns against those from the training data and assigned a familiarity score similar to Zhang et al.'s method. This score is then used to prioritize the test data. These two approaches aimed to improve the identification of potential DNN failures by closely examining neuron activation patterns.

Furthermore, empirical studies have been conducted to compare the effectiveness of various test selection metrics [23, 24, 27]. Hue et al. [24] compared test prioritization metrics on datasets created with varying ratios of out-of-distribution (OOD) and original test data. They retrained the DNN model using the selected test data and evaluated the accuracy improvement. While no single metric consistently outperformed the others across all test data sets, uncertainty methods (such as entropy, MCP, and DeepGini) were more effective when the test dataset contained more original data than OOD data. Conversely, when the test dataset had a higher proportion of OOD data, simple random selection outperformed all other metrics. Consequently, they recommended using different uncertainty metrics based on the distribution of the test dataset.

Ma et al. [23] compared various uncertainty metrics with Neuron Coverage and Surprise Coverage metrics. In their evaluation, uncertainty metrics were more effective in prioritizing test inputs that are likely to be misclassified by the DNN model. When the test dataset was augmented with adversarial data, surprise metrics performed comparably to uncertainty metrics, but neuron coverage metrics did not yield promising results under any experimental conditions. They employed the Monte Carlo (MC) Dropout Bayesian [92] approximation to estimate uncertainty.

Shi et al. [27] carried out the most extensive empirical analysis so far, assessing the effectiveness of various test selection metrics in identifying fault-revealing inputs. In their study, they evaluated eleven metrics and analyzed the correlation of these metrics with the model's misclassifications. Overall, uncertainty metrics were the best and most strongly correlated metrics with misclassifications in most

cases, and they also achieved the highest failure detection ratio. It has been demonstrated that the most effective uncertainty metric varies depending on the dataset and its size.

Moreover, a comprehensive survey on test data optimization in DNN testing was conducted by Hu. et al. [38]. The optimization in this study's context refers to testing with minimal data labeling effort. They conducted a comparative analysis of the research focuses between the software engineering and machine learning communities.

In recent research, the meta-modeling approach has been used to quantify uncertainty [69,93,94]. This method includes training a meta-model to predict the uncertainty of the base model. Moreover, this approach is used in test data selection by using the meta-model as a predictor for the performance of the base model. Bernhard et al. [39] proposed a meta-model based test data selection strategy for the automotive domain. A meta-model is trained to predict the tested DNN model's performance on the unseen test cases based on the results of the already tested subset of test data. Linear regression model and neural networks were used as meta-models in their study. Simulation software is used to generate synthetic test images, and the proposed method is used to select the images that may cause incorrect predictions in the tested DNN model. Similarly, Taskazan et al. [40] employed a meta-modeling approach to reduce labeling effort by test data selection. They used a meta-model to predict the performance of the base model. The confidence scores of the labeled test dataset were separated into bins, with the mean and standard deviation calculated for each bin. For an unlabeled test input, they estimated the base model's uncertainty by comparing the DNN model's confidence score with the corresponding bin's mean and standard deviation. They then prioritized the test data based on the highest estimated uncertainty.

3.1.4 Distribution-Aware Testing

The modeling of data distribution is crucial for testing DNNs, as the similarity of the test dataset's distribution to the training dataset directly impacts the performance of the DNN models. Two popular approaches are commonly used in distribution-aware testing: OOD data detection and test data creation with generative models. OOD detection methods are employed to determine whether the test input comes from a similar distribution as the training data. Generative models, on the other hand, are used to create new test inputs based on the training data distribution.

OOD data detection is an active area of research in deep learning. Several methods have been proposed to distinguish OOD data from in-distribution data. A recent review study categorized these methods into three groups: classification-based, density-based, and distance-based [58]. Within the density-based category, generative models are employed to learn the features of the training data distribution and to evaluate the probability that the new input aligns with the training data distribution.

To date, only a limited number of studies have been conducted on distribution-sensitive DL testing, which only started to be researched in recent years. The studies can mainly be categorized into two groups: evaluation of OOD data impact on DNN testing [24,46] and application of distribution awareness in test data generation [19,20,43–45]. These studies demonstrate the need for additional research on this topic.

Byun et al. [20] introduced a method for generating and selecting test cases that are similar to the training data distribution by using a Conditional Variational Auto Encoder (CVAE). Similarly, Kang et

al. [19] incorporated VAEs into a search-based test data generation approach to create new test images. They implemented a differential testing strategy across several DNN models, focusing on generating test data that results in distinct outputs from the models.

Alternatively, Huang et al. [45] proposed a two-stage DL testing method that incorporates employing feature-level information when choosing test data seeds while using pixel-level details when creating test data from these seeds. Their goal is to produce adversarial data with a distribution-aware perspective and with natural and realistic modifications.

Dola et al. [43] performed an evaluation of test data generated by existing DNN testing methods (DeepXplore [12], DLFuzz [13], and DeepConcolic [16]) and demonstrated that a high ratio of the test data generated by these methods is categorized as OOD data by OOD detection methods. They used VAEs to detect OOD data and then employed the decoder network of the same VAE to generate new test data resembling the training data. Furthermore, Berend et al. [46] conducted a study to assess the relationship between testing criteria, test data generation methods, and OOD data. They highlighted the significance of being aware of the data distribution in DL testing, which can lead to more effective testing.

VAEs are used for both OOD data detection and generating new test inputs in DNN testing. In this study, we employed VAEs to identify OOD data within the generated test data instead of producing new test data instances.

3.2 Uncertainty

DL uncertainty is categorized as aleatoric uncertainty and epistemic uncertainty [95]. Aleatoric uncertainty is driven by data due to reasons such as the structure and nature of the data or the way it was collected. For example, for DL systems that take images as inputs, the camera's properties, such as sensor range, resolution, calibration, and contamination, can affect the information provided by the samples to the model during training. These issues limit the capability of the model. Aleatoric uncertainty cannot be prevented by supplying additional training data since the same issues will also exist in the new data. Epistemic uncertainty is defined as model-based uncertainty. Epistemic uncertainty arises from the inappropriate model architecture or insufficient training due to a lack of data representing all possible situations. This type of uncertainty can be reduced by providing training data for missing or underrepresented cases.

Accurately measuring the uncertainty of DNN models is crucial for deciding whether to trust the prediction of the model [96]. In classification models, a DNN's confidence is indicated by the highest class probability in its output. However, this confidence measure alone is not always adequate for assessing uncertainty. DNN models can produce highly confident but incorrect predictions, such as in the case of adversarial data [97]. To address this, various techniques and metrics have been proposed to more accurately measure the uncertainty of DNN models [98–101].

3.3 Explainability

With the increasing use of DL in the last decade, artificial intelligence models have become more complex, making their decision-making processes less understandable. In this context, Explainable

Artificial Intelligence (XAI) has emerged as a field that aims to explain the decisions of AI models in a human-understandable way. XAI is developed to understand the intrinsic decision processes of AI models and to explain their decisions to users, thereby increasing the trust in AI systems.

DL models have commonly used quantitative performance metrics to evaluate prediction accuracy and computational complexity, but criteria such as interpretability or explainability are not easily quantifiable. This challenge leads to the variation in explanation techniques. The most appropriate explanation method depends on the application area, the type of explanation, the input data type, the user's background knowledge, and the specific question being addressed.

Explainable AI methods are subject to different groupings in terms of their technical characteristics, such as Intrinsic Explainability/Post-hoc Explainability, Model Agnostic/Model Specific, and Global Explanation/Local Explanation.

Intrinsic Explainability refers to AI models characterized by their straightforward structures, for instance, short decision trees or lean linear models, and those that inherently offer explanations for their decisions within the output of the model. Post-hoc Explainability refers to explanation methods applied after model training. Black-box models are inherently opaque, requiring post-hoc explainability methods, while white-box models are transparent and easier to understand, with the applicability of Intrinsic Explainability methods.

In XAI, the concepts of Model Agnostic and Model Specific are used to clarify the applicability range of the methods. The term Model Agnostic indicates that an explainability method can be applied to all model types. That is, this type of method can work compatibly with various models, such as from decision trees to deep neural networks. On the other hand, the term Model Specific means that an explainability method is designed specifically for certain model types. These concepts provide an important distinction in explainability research when focusing on specific model types.

Furthermore, the terms Global and Local are used to indicate the scope of explainability methods. The term Global indicates that an explainability method provides a general explanation covering all data instances. That is, it provides information about the general behavior of the model. For example, an explainability method enables one to understand the general characteristics of a classification model that addresses the entire dataset.

On the other hand, the term Local indicates that the explanation is limited to a specific data instance or a few ones. In this case, the explainability method specifically focuses on particular data instances to explain the model's decisions at those points. For example, a Local explainability method can be used to understand a classification model's decision on a specific input. This property represents an important distinction in explainability analysis for understanding general model behavior or focusing on specific data instances.

The XAI methods can be mainly categorized as backpropagation-based, feature-based, rule-based, and example-based methods.

Feature-based methods aim to explain the model's decisions based on the input's features. These methods provide insights into the input features considered most important by the model and their impact on the model's decisions. Methods like SHAP [102] and Integrated Gradients [103] are well-known feature-based approaches. SHAP is designed for datasets with multiple features and calculates

the influence of each feature on the prediction. By computing feature weights across the entire dataset, SHAP also considers the interactions between features.

Backpropagation-based methods use the advantage of information processing through a series of connected nodes or neurons. By tracing the information flow through the network, these methods identify the most influencing nodes or neurons on the model's output and consequently determine the features of input most effective on the outputs of these neurons. In computer vision, backpropagation-based methods explain the model's decisions by tracking the propagation of neuron activations to the specific image features/regions and highlighting the most influential image regions. Examples of such methods include Layer-wise Relevance Propagation (LRP) [104], Class Activation Mapping (CAM) [105], and saliency maps [106]. LRP determines the influence of each input feature on the output by calculating feature weights at each layer of the model and assessing the impact at each stage. CAM uses the final layer's weights in a deep learning model to identify the regions of an image that most influence the model's decision. Grad-CAM [107], Grad-CAM++ [108], and Score-CAM [109] are enhanced versions of the CAM approach.

Rule-based methods express the model's behavior as a series of rules that produce specific outcomes under certain conditions. These methods aim to explain the predictions of the model using logical statements that can be easily understood and interpreted by humans. Rule-based methods are employed in inherently rule-based models, such as decision trees, or through the simplification of rules extracted from complex models. For instance, the Achor [110] is a rule extraction technique, which is a local, post hoc, and model-agnostic method. It determines the restricting and sufficient conditions for features where the model provides a high-precision prediction. It creates if-then rules, which are called anchors. This approach generates an explanation by assigning certain value ranges of specific attributes to particular predictions.

Example-based methods involve selecting specific data instances to explain the behavior of models. These methods can operate independently from the model. They perform better when the feature values of the data instances carry more context information and the data exhibits a clear structure. These methods are most beneficial for visual and text-based models. Examples of these methods include Contrastive Explanations [111], and Prototype-Based Explanations [112]. Contrastive explanation is a local method that reveals which minimal features are sufficient to obtain a certain outcome and which should be absent. On the other hand, Prototype-Based Explanations involve selecting representative examples from the dataset that capture the foundation of the model's predictions instead of individual examples.

Advancements in explainability methods have led to the exploration of using these methods to understand the reasons behind failures in DL models. This is particularly evident in the natural language processing domain, where studies have employed post-hoc explainability methods or self-explainable models [113]. Most of these studies were conducted using Naive Bayes or Logistic Regression models.

In DL visual models, post-hoc explainability methods are widely studied using back-propagation techniques or applying perturbations to input images to understand and interpret model predictions. The applicability of visual post-hoc model explanation methods to understand the failures of DNN models was investigated by Adebayo et al. [114]. Their research focused on examining the use of explainability methods for debugging purposes in the image classification problem. They employed several post-hoc methods on a convolutional neural network model. They categorized the bugs that were evaluated using explainability methods in their study into three groups: data contamination bugs, model

contamination bugs, and test-time contamination bugs. Their results suggested that feature attribution methods are successful in detecting spurious correlations but do not definitively assist in distinguishing incorrectly labeled inputs from correct ones in the training dataset, which is investigated in the context of data contamination bugs. In the case of model contamination, they noted that certain feature attributes remain unaffected by changes in the parameters of the higher layers in DL models, indicating limited effectiveness in explaining these bugs. Additionally, they revealed that feature attributions for out-of-distribution inputs closely resemble attributions obtained from an in-distribution input. In contrast to this study, our focus is the failures of the DNN model due to insufficient training data containing certain features. This approach emphasizes understanding the limitations posed by the lack of relevant data in the training set.

CHAPTER 4

METHODOLOGY-1: DNN MODEL TESTING FRAMEWORK

In this chapter, we propose a three-phase Deep Neural Network (DNN) testing framework: Test Data Generation, Test Data Selection, and Test Results Interpretation, Figure 7. This framework aims to subject the DNN model to a wide variety of test inputs by prioritizing the ones that will be most helpful in identifying the weaknesses of the DNN model.

Manual effort in labeling test data and analyzing test results is a significant aspect of DNN testing. We aim to concentrate the manual effort on test inputs that are most likely to reveal insights and lead to meaningful improvements in the model. To achieve this, we prioritize test data that trigger incorrect predictions by the DNN, which are important for identifying vulnerabilities. During this process, we create test datasets by considering the data distribution. Finally, we emphasize that understanding the causes of model failures observed in the tests allows for more informed decisions on the necessary actions for model improvement.

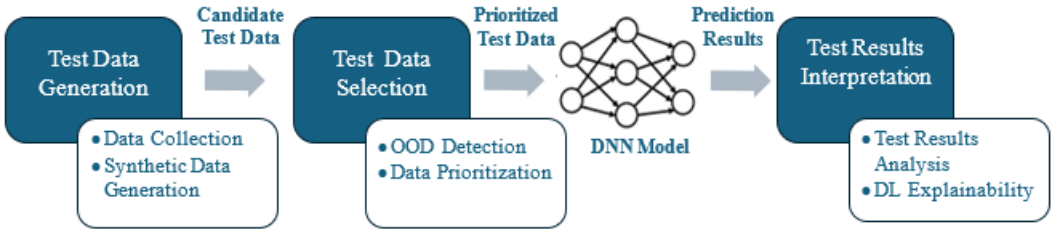


Figure 7: DNN Testing Framework with a distribution aware and uncertainty-based approach.

First, we will explain each phase of the proposed framework. Then, we will provide details for the implementation of this framework in the image classification context.

Test Data Generation Phase

This initial phase expands the limited *original test dataset* typically used in model development by generating a diverse and comprehensive test dataset. This is accomplished by either collecting real-world data or synthesizing it. Data generation techniques might use the original test data as seed inputs to generate new data. Despite the challenges in creating realistic synthetic data, especially for complex data types like images, their automated generation is valuable for testing. The goal of this phase is to enhance the dataset's variety to represent various real-world scenarios and potential adversarial

conditions. The variety of datasets is a desired property for evaluating the performance of test data selection methods in different settings.

Test Data Selection Phase

Given the constraints of time and resources, it is impractical to test all generated inputs, especially when considering the high cost of data labeling. This phase focuses on selecting the most critical test inputs that reveal the model's vulnerabilities. Test selection is carried out by identifying and prioritizing a subset of data that is likely to provide the most insight into the model's weaknesses from a pool of unlabeled data.

It is important to distinguish the sources of incorrect predictions—whether they stem from a model's weakness or from data shifts such as out-of-distribution (OOD) inputs. The variance in the distribution of training and test datasets has a significant impact on the DNN's ability to accurately predict the test data. By understanding the distribution of test data, developers can differentiate between failures originating from in-distribution and OOD data. It is expected that the model will correctly predict new input data that closely follow the distribution of its training data. In contrast, OOD data originate from distributions that are not represented in the training dataset, presenting potential challenges in model prediction. When the DNN model makes an incorrect prediction, it is important to determine whether the model was expected to make a successful prediction or if the input was significantly different from the training dataset. By categorizing test datasets, developers can identify the underlying reasons for incorrect predictions made by the DNN model in a more focused manner. Hence, gaining deeper insights into the DNN's performance can help pinpoint specific areas for model improvement and make more informed decisions about how to enhance the accuracy and reliability of DNN models.

In our proposed framework, we first focus on test data that resemble the data that the model was trained on, known as in-distribution data. This testing with in-distribution data ensures the model's reliability before it is subjected to data with distribution shifts. Testing a DNN model with a dataset that does not match the training data without first confirming its performance on similar data with training data can lead to misleading results about the model's performance.

We apply OOD detection techniques to classify the test data into two distinct categories: one comprising solely in-distribution data and the other containing a mix of both in-distribution and OOD data. Upon generating these categorized test datasets, we prioritize the test inputs based on their likelihood of leading to incorrect predictions by the DNN model. This prioritization strategy is designed to identify the test data deemed most critical, potentially revealing significant insights into the model's vulnerabilities. Subsequently, if necessary, test data with lower priority are examined. This approach not only streamlines the testing process but also improves the overall efficiency of the testing framework by focusing on the test data that will be most helpful to identify the model's weaknesses.

Test Results Interpretation Phase

The final phase involves analyzing the incorrect predictions to understand why the DNN failed. Given the opaque nature of DNNs, interpreting their behavior is challenging. We employ post-hoc deep learning explainability methods to uncover the reasons behind specific predictions, enhancing our understanding of the model's decision-making processes.

The methods used in each phase of the framework can be chosen based on the properties of the DNN model and its input data, and this approach may be applied to any kind of DNN model.

Implementation of The Framework

For our research, we focused on an image classification problem and implemented the proposed framework for this context. We involved methods and strategies carefully selected by taking into account the unique properties of DNN models and datasets used in image classification. Some of the methods we employed have been used for different purposes in the literature but have not been applied to DNN testing, such as the use of Deep Ensemble and Variational Inference uncertainty estimation methods to prioritize test data.

We decided to focus on the image classification problem for our study because of its wide applicability in DNN testing research, which facilitates conducting comparative analyses [5]. There are various publicly accessible datasets, models, and benchmarks available for image classification problems, making it suitable for experimentation. Additionally, the strong support infrastructure, including pre-trained models and an active research community, provides extensive resources. The popularity and proven relevance of image classification make it a good choice for our research.

4.1 Test Data Generation Implementation

In the Test Data Generation phase, we used three different image generation techniques to create a diverse and extensive test dataset. The three methods we employed were Image Transformations, Adversarial Attacks, and Generative Models, all of which are widely used in the literature and have unique properties that make them appropriate for generating test data. Throughout the test dataset generation process, we aimed to create test data images that could represent a variety of different conditions under different scenarios.

In the first approach, we employed two categories of image transformations: pixel transformations and affine transformations. These transformations are used to simulate real-life scenarios that a DNN model may face, such as images from a rotated camera in an autonomous vehicle or changed brightness conditions. Pixel transformations involve tweaking the values of pixels present in the image, while affine transformations involve moving the pixels of the image. We selected random seed inputs from the original test dataset and performed the transformations on them to generate new test data.

The second approach, known as Adversarial Attacks, entails adding minor, deliberately designed perturbations to images with the goal of deceiving DNN models. These perturbations are designed to be almost unnoticeable by humans, but they can cause failures in DNN models. The test inputs generated with these techniques are notably useful in assessing the resilience of DNN models against harmful attacks. This type of test data reveals vulnerabilities of the models that might not be identified with test data collected from the real world. We generated adversarial data by applying adversarial attack methods to randomly selected samples from the original test dataset, using the model under test as the basis for these manipulations.

Finally, we used Generative Adversarial Networks (GANs) [115] to create test data that had similar features to the training dataset and originated from a similar distribution. GANs are a type of DNN model that consists of a generator and a discriminator. By training a GAN with the same data used to train the tested DNN model, we aimed to enable the GAN to learn the features present in the training data and subsequently generate synthetic data resembling the distribution of the training dataset. In this context, with the deliberate use of GAN, we generated data similar to the training data.

4.2 Test Data Selection Implementation

Test data selection is performed in two stages. At first, OOD data detection is performed on the generated test datasets, and OOD data is identified. Subsequently, test datasets are formed based on the categorization of test data produced by OOD detection methods, and they are prioritized according to the DNN model's uncertainty. In Section 4.2.1, we elaborate on how the OOD detection is implemented, and in Section 4.2.2, we outline the prioritization of test data using uncertainty estimation methods and metrics.

4.2.1 Out of Distribution Detection

To detect OOD data, we used Variational Autoencoders (VAEs) [116], a class of generative neural networks, following the methodology outlined by An et al. [117]. An autoencoder is an unsupervised neural network trained to reproduce the input at the network's output. It has an encoder and a decoder network. The encoder network encodes the high-dimensional input data into a lower-dimensional latent space representation. The decoder network produces outputs using the latent space representation of inputs. The difference between the input and output is called reconstruction error. The autoencoder is trained with the objective of minimizing reconstruction errors. VAEs encode input not as a single point in the latent space but as a data distribution. Sampling is made from this latent data distribution, and the sampled value is fed into the decoder network to get an output value. The decoder network reconstructs the parameters of the input's distribution rather than the input itself. The likelihood of reconstructing input data with the distribution parameters generated by the decoder network for that input is known as the reconstruction probability. The reconstruction probability for each input is calculated by taking several sampling values from the latent data distribution, then using the decoder network to generate output values for these sampled values and averaging the reconstruction probability for each sample.

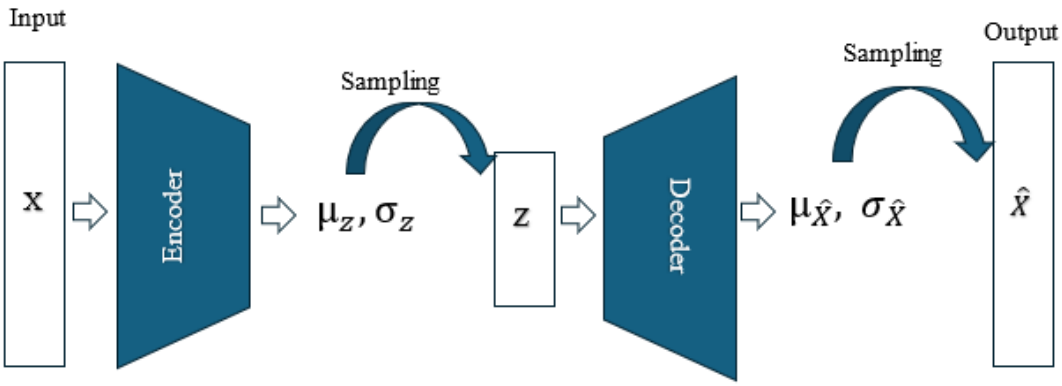


Figure 8: Architecture of Variational Auto Encoder (VAE).

We employed a VAE with probabilistic encoder and decoder networks for OOD detection, Figure 8. We train the VAE with the training dataset of the tested DNN model in an unsupervised manner, enabling it to learn the intrinsic characteristics of the data. This unsupervised training of VAEs is particularly advantageous as it does not require labeled data.

Data that deviate from the training data distribution produce lower reconstruction probabilities when fed into the trained VAE, and in this way, they are identified as OOD data. We need to define a threshold value to identify whether data is OOD or not based on the reconstruction probability of the data. Data that has a reconstruction probability higher than the threshold value is classified as in-distribution, and conversely, data that has a reconstruction probability less than the threshold value is classified as OOD data. To determine the threshold value, we require an in-distribution dataset and an OOD dataset. For the in-distribution dataset, the original test dataset is used, while a distinct dataset is selected to be used as the OOD dataset. We calculate the reconstruction probabilities for the original test dataset using trained VAE. Subsequently, we compute the reconstruction probabilities of the dataset, which is selected as the OOD dataset. We establish an empirical threshold to differentiate between OOD and in-distribution data based on these reconstruction probabilities. Then, we use this threshold to classify data from both the original test dataset and the OOD dataset according to their reconstruction probabilities. We calculate the ratio of correct and incorrect classifications across various threshold values. To determine the optimal threshold value that distinguishes between OOD and in-distribution data, we employ the F1-score — a metric that provides a harmonic mean of precision and recall. This process of selecting a threshold value is crucial for ensuring the accuracy of OOD detection. The algorithm we applied for OOD data detection is given in Algorithm 1.

Algorithm 1 Out-of-Distribution Detection with VAE (adapted from [118])

$f_\phi, g_\theta \leftarrow$ Encoder, Decoder of trained VAE
threshold_list $\leftarrow$ List of possible N threshold values
ID Dataset $\leftarrow$ Original test dataset of DNN model
OOD Dataset $\leftarrow$ Selected dataset as OOD dataset
for $i = 1$ to N **do**
  $threshold_i \in$ *threshold_list*
  for all x in *ID Dataset* and *OOD Dataset* **do**
    $\mu_z, \sigma_z \leftarrow f_\phi(z|x)$
    for $j = 1$ to L **do**
      Draw sample from $z \sim \mathcal{N}(\mu_z, \sigma_z)$
      $\mu_{\hat{x}}, \sigma_{\hat{x}} \leftarrow g_\theta(x|z)$
      $rp_j = ReconstructionProbability(\mu_{\hat{x}}, \sigma_{\hat{x}}, x)$
    end for
    reconstruction_probability $= \frac{1}{L}\sum_{j=1}^{L} rp_j$
    if *reconstruction_probability* $\geq threshold_i$ **then**
      x is In-Distribution data
    else
      x is Out-of-Distribution data
    end if
  end for
  Calculate F1-Score for OOD/In-distribution categorization with $threshold_i$
end for
Selected Threshold $\leftarrow$ threshold with maximum F1-Score
Repeat steps 7-20 for Generated Test Datasets using Selected Threshold value

After determining the threshold value that yields the highest F1-score, we classify the generated test data as either in-distribution or OOD data. We accomplish this by using the reconstruction probabilities of these data, which are obtained using VAE and comparing the resulting probabilities with the selected threshold value.

4.2.2 Test Data Prioritization

In our study, we incorporated uncertainty-based test data prioritization to identify fault-revealing inputs. This approach is grounded in the principle that inputs generating higher uncertainty for the DNN model are more likely to lead to incorrect predictions.

To assess the uncertainty of the DNN model on its predictions, we implemented three different methods, each using the same uncertainty metrics to quantify uncertainty. Uncertainty metric values are computed for each test dataset through these three distinct methods. The test data that yields higher uncertainty values are prioritized over the others. Consequently, we obtained datasets ordered according to nine different test prioritization strategies, as we used three distinct uncertainty metrics with three distinct methods.

A. Uncertainty Estimation Methods

Firstly, we used a straightforward approach where uncertainty metrics are calculated directly from the outputs of the model under test (MUT). The term "Model Under Test" is analogous to "System Under Test" in traditional software engineering, denoting the specific DNN being tested.

For more sophisticated uncertainty assessments, we employed two additional methods: Deep Ensemble (DE) and Variational Inference (VI) using the Flipout techniques. In these methods, metric values are derived by aggregating several output values associated with the same test data, contrasting with the MUT approach, where metric values are determined using a single output value per test data. These approaches are recognized as state-of-the-art methods for quantifying uncertainty [96], [100].

Deep Ensemble (DE) [64] is a sampling-based approach that has been proposed as an alternative to Bayesian Neural Networks (BNN) for estimating uncertainty. It is relatively easy to implement and requires less hyperparameter tuning than BNNs. This approach involves training multiple instances of the same DNN model and using their outputs to assess the uncertainty. Each model shares the same architecture with the MUT. They are initialized with different initial values at the start of training and then trained with the same dataset used to train the MUT. In this way, each DNN learns a distinct set of hyperparameters. Hence, by ensuring diversity among the ensemble models, each model can capture different aspects of the data. By combining their outputs, a better uncertainty estimation performance can be achieved compared to a single model. Following training, by evaluating the same test data across these models, we obtain a probability distribution for each output, from which we compute the uncertainty metrics. The average of these metrics across DE models provides an estimate of uncertainty, which is used in the prioritization of test inputs.

Variational Inference with Flipout (VI-F) is a Bayesian approximation method. In this method, standard layers in a DNN model are replaced by layers that implement the Flipout process, improving the model's ability to estimate uncertainty. Flipout enhances the sampling process by generating distinct weight perturbations for each input data in a training minibatch. This method involves a stochastic de-

cision mechanism, similar to flipping a coin, to determine the sign (+ or -) of the perturbations applied to the weights. This is mathematically represented as $w = \mu \pm \sigma \odot \epsilon$, where μ, σ, ϵ denote the mean, standard deviation, and a perturbation value, respectively. The Flipout process, with its stochastic nature, helps in producing gradients that are less correlated. By reducing the variance of gradients during the optimization process, Flipout leads to faster convergence during training.

Upon completion of training with Flipout layers, a Monte Carlo approach is conducted. This involves multiple forward passes of the same input data through the model, and the results are cumulatively evaluated. The uncertainty of predictions for each input data in the test dataset is calculated with these passes. The uncertainty metrics are computed from the output of every pass, and then the average of these metrics is calculated, similar to the Deep Ensembles approach. Afterward, this average value is utilized to prioritize the test data, identifying inputs with high levels of prediction uncertainty.

Figure 9 illustrates the three methods employed for estimating uncertainty.

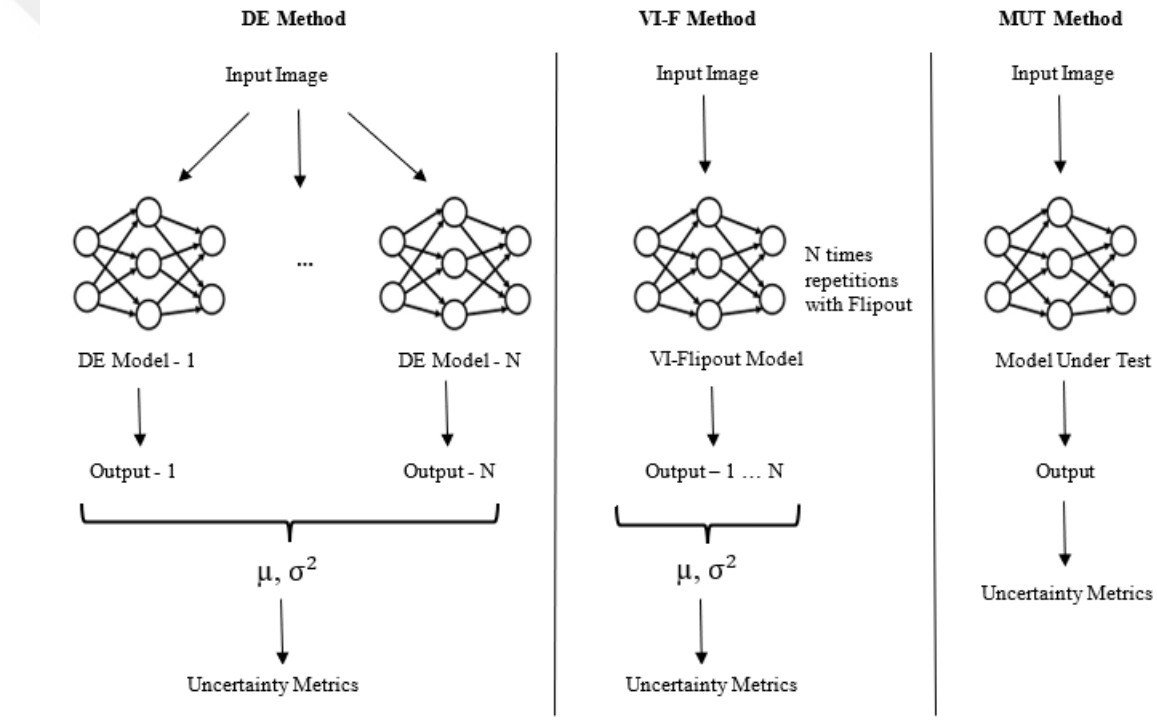


Figure 9: Illustration of Uncertainty Estimation Methods.

B. Uncertainty Metrics

Uncertainty metrics provide a quantitative assessment of the uncertainty a model experiences regarding its predictions. We employ metrics calculated from the prediction probabilities produced by DNN models for different classes: Least Confidence, Margin, and Entropy. We selected these metrics because they can be directly computed from the class probability outputs of image classification DNN models. Their implementation is straightforward and does not require access to the internal structure

of the DNN models. They can be applied to any type of classification model. This makes them versatile and easy to integrate into various frameworks. Each of these metrics offers a unique perspective. Least Confidence uses only the highest class probability and discards the other class probabilities, while Margin considers the two highest probabilities. On the other hand Entropy evaluates the entire distribution of probabilities for estimating the uncertainty of the model. These metrics are well-known uncertainty metrics for probabilistic classification tasks [119].

Least Confidence: This metric assesses the difficulty of an input for the model based on the confidence of its prediction. A classification DNN model's final layer outputs a probability distribution across the classes, where each class has a score between 0 and 1.0, summing to 1.0 across all classes. High confidence in a prediction corresponds to a probability close to 1 for one class and close to 0 for others. Conversely, if the probability scores across different classes are almost equal, the model exhibits low confidence. The Least Confidence metric is calculated by subtracting the maximum probability value across classes from 1.0, Equation 2. Therefore, a higher value of Least Confidence indicates greater uncertainty.

$$\text{Least Confidence} = 1 - max(p_i) \qquad p_i : \text{predicted probability for class } i \tag{2}$$

Entropy: Entropy is a well-known measure used in information theory and is also applied to determine the uncertainty of DNN models. Entropy quantifies the average amount of information required to encode the probability distribution of a random variable in information theory. For DNNs, to estimate the uncertainty, entropy is calculated from the class probability output of the last layer, Equation 3. Higher entropy values indicate greater uncertainty, occurring when the output probability distribution is even across all classes. Entropy is minimized when the model's predictions are more certain, reflected in a probability distribution where one class has a probability close to 1.0.

$$\text{Entropy} = -\sum_i p_i log p_i \qquad p_i : \text{predicted probability for class } i \tag{3}$$

Margin: Margin calculates the difference between the top two class probabilities, Equation 4. A large margin indicates a certain prediction, with a significant difference between the highest and second-highest class probabilities. A small margin, where the top two class probabilities are similar, suggests the model struggles to distinguish between these classes, reflecting higher uncertainty.

$$\begin{aligned} \text{Margin} = p_k - p_j \qquad & k : \text{class having maximum probability,} \\ & j : \text{second most probable class} \end{aligned} \tag{4}$$

4.2.3 Evaluation of Test Data Prioritization Approaches

We evaluated test data prioritization strategies to assess their effectiveness in identifying fault-revealing test data. Specifically, we measured the success of these strategies by evaluating their ability to identify test data that cause misprediction on the tested DNN model. We employed four different evaluation methods for assessing the effectiveness of the test prioritization strategies. By employing multiple evaluation methods, we aimed to comprehensively evaluate the performance of each prioritization approach and minimize the risk of potential biases that could result from relying on a single method. We

chose to use the Ratio of Area Under Curve (RAUC), Average Percentage of Fault Detected (APFD), and Fault Detection Ratio metrics. Additionally, we analyzed the correlation between the uncertainty scores computed by each test data prioritization strategy and incorrect predictions of the DNN model.

Average Percentage of Fault Detected Evaluation (APFD):

In software testing, APFD serves as a measure to gauge the effectiveness of a test suite. This metric has been adapted and used in DNN testing [25, 37, 66] to assess the efficacy of test selection methods in identifying fault-revealing inputs. Consider a test dataset T consisting of n test inputs. Suppose the model produces m incorrect predictions for this dataset, with w_i representing the order of test input in the test dataset, causing the i^{th} incorrect prediction. The APFD score is calculated as given in Equation 5. The APFD value ranges from 0 to 1, where higher scores indicate a higher fault detection speed.

$$\text{APFD} = 1 - \frac{\sum_{i=1}^{m} w_i}{nm} + \frac{1}{2n} \tag{5}$$

Ratio of Area Under Curve (RAUC): Ideally, test data prioritization techniques should assign higher scores to incorrectly predicted test data than to correctly predicted ones. This ideal scenario would create a linear relationship on a graph where the x-axis shows the number of prioritized test data, and the y-axis shows the number of incorrectly predicted test data among those prioritized data up to the total count of incorrectly predicted test data. However, in practice, the relationship depicted on this graph often deviates from linearity as in Figure 10.

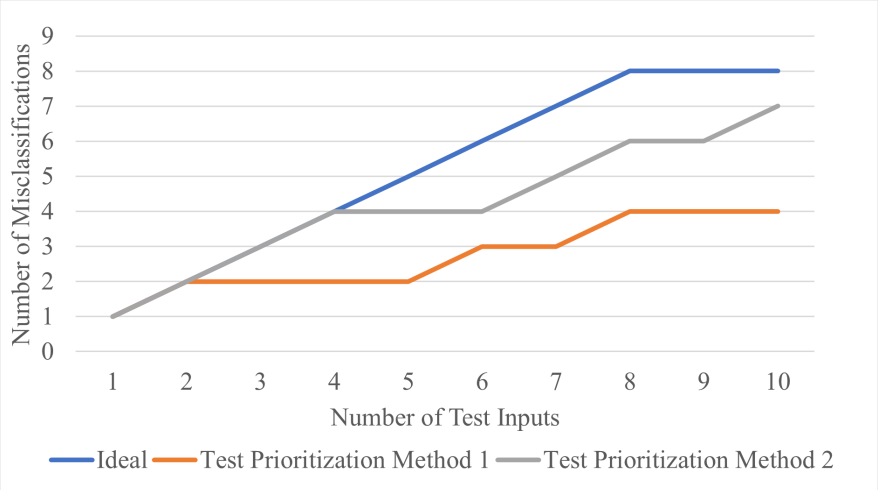


Figure 10: Sample Ratio of Area Under Curve (RAUC) Graph.

To quantify the effectiveness of the prioritization techniques, RAUC measures the ratio of the area under the curve generated by a prioritization technique to the area under the ideal linear curve [29] [34] [68]. It is calculated based on a budget for the total number of prioritized test data. Specific values such as $RAUC_{100}$, $RAUC_{300}$, $RAUC_{500}$ and $RAUC_{1000}$ indicate the RAUC scores when the first 100, 300, 500, 1000 prioritized test data are considered, respectively. When all of the test data is used, the RAUC value is denoted by $RAUC_{all}$. The RAUC values range between 0 and 1, with higher values indicating better prioritization performance.

Suppose that we have a test dataset of 50 inputs, out of which 8 are incorrectly predicted by the tested DNN model. We want to compare the prioritization methods with a budget of 10 test data. In Figure 10, we see the graph resulting from the prioritization performed by an ideal test data prioritization and two different test data prioritization methods. Ideally, the first 8 prioritized data should be those incorrectly predicted by the model, as shown with the blue line. The RAUC values of the other two methods are calculated as the ratio of the area under the curve of these methods to the area under the ideal curve as given in Equation 6.

$$RAUC_n = \frac{\text{AUC(method,n)}}{\text{AUC(ideal,n)}} \tag{6}$$

Fault Detection Ratio Evaluation: The fault detection ratio evaluation measures the number of faults detected as a percentage of the total number of faults for different percentages of prioritized test data. When test data is randomly ordered, the percentage of mispredictions is expected to be linear with the percentage of test data selected. A successful test data prioritization is indicated by a higher ratio of faults detected than the percentage of test data used. This results in steeper curves when depicted on a graph where the x-axis represents the percentage of test data, and the y-axis represents the ratio of faults detected. In Figure 11 given as an example, it is shown that Test Prioritization Method-4 is more successful in prioritizing data that causes incorrect predictions and has a steeper curve than Random selection. When 20% of the test data is used, Random selection identifies 20% of data that is mispredicted by the DNN model, whereas Test Prioritization Method-4 identifies 50% of the incorrectly predicted data.

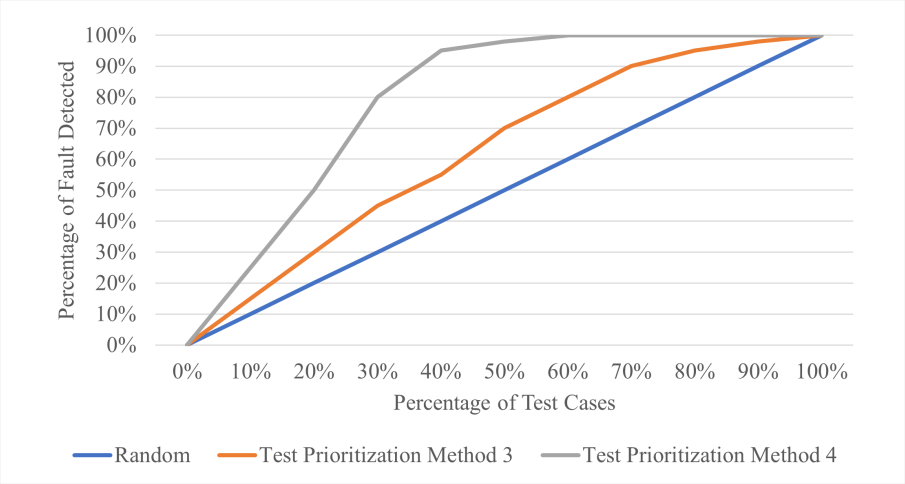


Figure 11: Sample Fault Detection Ratio Graph.

Statistical Correlation between Misprediction and Metrics: We evaluated the success of test data prioritization strategies by analyzing the statistical correlation between incorrect predictions made by the tested DNN model and the uncertainty score values calculated by each prioritization strategy. We employed two correlation tests, namely the Point-Biserial correlation and the Spearman correlation tests. We analyzed the correlation between mispredictions and uncertainty scores by the binary encoding of model predictions, where 0 represents incorrect predictions, and 1 represents correct predictions. A higher correlation value indicates a stronger relationship between incorrect predictions and uncertainty scores.

4.3 Test Results Interpretation Implementation

We used visualization techniques as a post-hoc analysis method to understand the decision-making process behind the incorrect predictions of the tested DNN model. One of the key advantages of post-hoc explainability methods is their independence from the training data, as they do not require access to this data nor any retraining of the model. They also do not necessitate any modifications to the existing model architecture. Furthermore, we chose visualization techniques that can be applied across various types of pre-trained models, subject to some restrictions only. All these properties make them highly practical for analyzing the behavior of a DNN model during inference. Visualization techniques highlight the input regions that significantly impact the model's predictions, and generated visual representations are easily interpretable by humans. We employed Grad-Cam [107], Grad-Cam++ [108], and Score-Cam [109] visualization techniques in this study. These are local explainability methods and focus on individual data instances and explain the model's decisions for this instance. Additionally, they are in the group of backpropagation-based explainability methods.

Grad-Cam [107]: This technique is specifically designed for use with Convolutional Neural Networks (CNNs), and the activation function used in the final layer of the model should be differentiable. This technique uses the last convolutional layer's outputs of the DNN model as feature maps that represent high-level abstractions of the input image. Grad-Cam computes the gradients of the predicted class probability score with respect to these feature maps. These gradients are essential as they reveal how variations in the feature maps influence the predicted class's probability score, providing insights into the sensitivity of the output to different features within the input.

Following the computation of the gradients, the Grad-Cam assesses the relative importance of each feature map. This is achieved through the global average pooling of the gradients. Using these pooled values as weights, Grad-CAM then constructs a Class Activation Map (CAM) by performing a weighted sum of the feature maps. This CAM is used as a heatmap that highlights the critical regions of the input image that most significantly impact the model's prediction.

Grad-Cam++ [108]: The Grad-CAM++ method represents an advancement over the original Grad-CAM technique, offering improved localization of objects in input images. Unlike Grad-CAM, which relies on the global average pooling of gradients to compute the CAM, Grad-CAM++ employs a weighted average of positive gradients. This modification enhances the method's ability to pinpoint relevant regions more precisely, especially when dealing with images that contain multiple instances of objects from the same class.

Score-Cam [109]: The Score-CAM method starts similarly to the Grad-CAM method by generating the feature maps from the last convolutional layer's outputs of the DNN model, but it has a significantly different process for constructing the CAM. Unlike Grad-CAM, Score-CAM does not rely on gradient calculations to determine the significance of each feature map. Instead, it calculates the CAM through a weighted sum of the feature maps, where the weights are derived in a unique and gradient-independent manner. In Score-CAM, each feature map is used as a mask on the original input image to create perturbed versions of the image. These perturbations involve selectively obscuring parts of the image corresponding to each feature map.

The algorithm then measures the impact of each perturbed image on the model's output. Specifically, it computes the change in the model's confidence in the predicted class, defined as the difference between the probability score of the perturbed image and the original image. The weight assigned to

each feature map is based on this change in confidence score. This approach provides a measure of how much each part of the input image contributes to the final prediction when considered in isolation.

Finally, Score-CAM uses these weights to generate a heatmap, which is then overlaid on the original input image. By eliminating the need for gradient calculations, Score-CAM reduces the computational complexity. However, a separate forward pass is required for each feature map to calculate its weight, which increases the computational demand depending on the number of feature maps and the complexity of the model.

CHAPTER 5

METHODOLOGY 2 : IMPROVEMENTS IN TEST DATA SELECTION

In this chapter, we focus on the Test Data Selection phase of the DNN testing framework proposed in Chapter 4 and introduce new methods to enhance this process. With the aim of identifying and prioritizing the fault-revealing inputs, we proposed two new uncertainty-based test data prioritization metrics, detailed in Section 5.1. Furthermore, in Section 5.2, we present a data prioritization approach that combines the strength of multiple uncertainty metrics through a meta-model to achieve more effective prioritization across various data types. Finally, we integrate a new out-of-distribution (OOD) data detection method into our framework, which is more effective at distinguishing near-OOD data, in Section 5.3. This adaptation aims to better categorize OOD data resulting from data shifts within the same dataset.

5.1 Alternative Uncertainty-Based Test Prioritization Metrics

In this section, we introduce two novel uncertainty-based metrics for test data prioritization: the Mahalanobis Uncertainty Score (MUS) and the Deep Ensemble (DE) Variation Score. We calculate these metric values and use them to prioritize the test data.

The first metric, MUS, quantifies the uncertainty by measuring the similarity between a test input and the model's training data. The second metric, the DE Variation Score, derives an uncertainty value from the differences between the predictions of the DE models and the MUT.

In the following sections, we provide a detailed outline of these two metrics and their use in test data prioritization.

5.1.1 Test Data Prioritization with Mahalanobis Uncertainty Score

When a DNN model has two or more class probability scores that are close to each other, it means that the model is not certain about classifying the data instance. This uncertainty may occur when the test data instance resides near the boundary between two classes. The DNN models struggle to correctly classify the data at the boundaries of classes.

The probability scores of a classification DNN model are based on the logits of the DNN model. If two input data have similar logits, the DNN model produces similar probabilities, leading to the higher

possibility of the same class prediction for both inputs. We decided to use the output of the layer preceding the logits layer as the representation of inputs. It is possible for two inputs to have the same prediction if their representations are similar, like with logits. If an input has a representation that is similar to two data belonging to different classes, it indicates that this data point may lie on the boundary between these two classes.

We chose to use the output of the layer preceding the logits layer because the layers closer to the output layer are able to capture more abstract and high-level information about the input data. These layers are generally better at learning complex features of the input and are therefore considered to be a good choice for representing an input. Moreover, this layer provides a more fine-grained representation of the input compared to the logits layer, which may be too high-level to capture detailed information about the input.

To identify the data with high uncertainty of the DNN model and prioritize them as fault-revealing test inputs, first, we gather the representations for the training dataset of the model and group them according to their classes. We then obtain the representation of each test data input using the same process. To estimate the uncertainty of a test data input, we calculate its distance to each class in the training dataset. We use the Mahalanobis distance calculation method to measure these distances. For each class, training data representations are fitted to a Gaussian distribution, and the mean and covariance of each distribution are computed. The distance of a data instance to each class is calculated as given in Equation 7 where S is the covariance, μ is the mean of the data belonging to the class c, and x is the representation of the data instance.

$$MD_c(x) = \sqrt{(x - \mu_c)^T S_c^{-1} (x - \mu_c)} \tag{7}$$

Then, we identify the first two minimum distances between the test input and training data classes. The difference between these two minimum distances serves as the uncertainty score, with lower scores indicating higher uncertainty. The rationale behind this approach is that when a test data input exhibits similar distances to two different classes, the model faces greater difficulty in correctly assigning it to the appropriate class. We named this uncertainty score the Mahalanobis Uncertainty Score (MUS), which is given in Equation 8.

$$MUS(x) = MD_k(x) - MD_j(x) \qquad \begin{array}{l}\text{k:class having the minimum Mahalanobis distance} \\ \text{j: class having the second lowest Mahalanobis distance}\end{array} \tag{8}$$

In this study, we used the MUS metric in two different ways to prioritize the test data instances.

- Firstly, we compute the MUS values for the generated test datasets using the model under test (MUT). Then, we sort the test data in ascending order based on their MUS scores, assigning higher prioritization to test data instances with lower score values.
- Secondly, we calculate the MUS using the Deep Ensemble (DE) method. Recall that DE involves several DNN models with the same architecture as the MUT but with different hyperparameter values. For uncertainty estimation, the metrics are calculated separately for each model and then combined to provide a unified value. Therefore, we compute the MUS values for the test data

using DE models, calculate the mean of these MUS values from each individual model, and use this average score to sort and prioritize the test datasets.

5.1.2 Test Data Prioritization with Deep Ensemble Variation Score

Deep Ensembles effectively capture model uncertainty by examining the spread of predictions among different trained models. This approach leverages the diversity in outputs to provide a measure of uncertainty. In the literature, other definitions for the variation in model predictions have been proposed that do not rely on deep ensemble models.

Shi et al. [27] employed prediction variation for test data prioritization by introducing mutated DNN models with mutation operators. In their work, the Variation Ratio measures the proportion of the same number of model prediction results as those of the majority of models relative to the total number of models.

In another study conducted by Ma et al. [23], the uncertainty of the model was determined based on the variation in dropout model predictions for each test input. The dropout model involves integrating dropout layer(s) into the DNN model, which dynamically deactivates neurons during a forward pass. Dropout was initially introduced as a technique to prevent overfitting in neural networks by randomly deactivating neurons during training. However, it was later used during inference time to quantify the model's uncertainty through multiple passes for a given data instance. The variance for an input x in this study is represented by the following equation, where C denotes the number of classes and "var" represents the variance function.

$$Var(x) = \frac{1}{C}\sum_{i} var(p_i(x)) \qquad p_i : \text{predicted probability for class } i \tag{9}$$

To quantify the uncertainty, we employ the variation in predictions of the ensemble models and MUT. For a given input, we define the Deep Ensemble Variation Score as the total number of ensemble models that produce predictions different from the MUT's prediction as given in Equation 10. The high number of distinct predictions suggests that the test input contains features that make it difficult to predict, thereby generating uncertainty for the model. Conversely, when the predictions of the ensemble models closely align with the MUT's prediction, the uncertainty is low.

$$\begin{aligned} \text{DE Variation Score} = \sum_{i} y_{DE_i} \neq y_{MUT} \qquad & y_{MUT} : \text{prediction of the model under test} \\ & y_{DE_i} : \text{prediction of the deep ensemble model } i \end{aligned} \tag{10}$$

We use the predictions of DE models and MUT to compute the DE Variation score values for the test data. Afterward, we rank them in descending order based on their DE Variation Score values, giving priority to those with higher scores.

5.2 Test Selection using Meta-Models with Uncertainty Metrics

In this section, we propose a data prioritization technique that leverages several uncertainty metrics together and integrates their strengths using a meta-model. Some uncertainty metrics are effective at quantifying the uncertainty when the test data closely resembles the training data. However, these metrics often fall short when assessing the uncertainty of adversarially generated data. Our approach combines multiple uncertainty metrics to overcome the limitations of individual metrics, thereby enhancing performance across different scenarios.

In the field of machine learning, a meta-model is a type of model that operates on top of one or more base models. It uses the outputs or decisions of other models, called base models, as its input to learn and make predictions, as shown in Figure 12. A meta-model provides an additional layer of learning on base models. By utilizing the outputs of the base models, meta-models can adapt to different tasks or datasets, detect patterns that might be missed by individual models, and provide more reliable predictions.

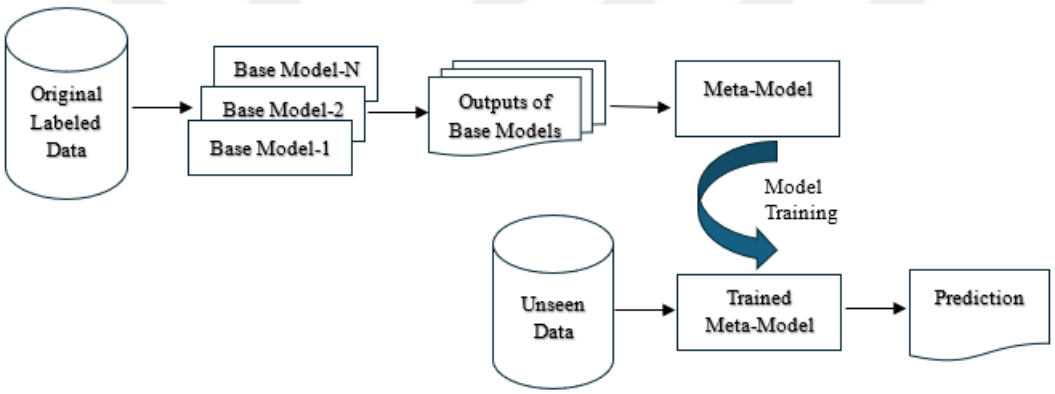


Figure 12: Meta-Model approach.

Meta-models are particularly useful in situations where the base models are not accurate enough on their own or when different models have different strengths and weaknesses. By combining the outputs of multiple base models, a meta-model can achieve higher accuracy and better performance than any single model on its own.

We used a meta-model to forecast whether the tested DNN model would accurately classify a given test input or not. For our base models, we selected the Model Under Test (MUT) and Deep Ensemble (DE) models. We calculated uncertainty metrics based on the outputs of both the MUT and DE models. These metrics served as inputs for the meta-model.

First, we feed the original test data into both the MUT and DE models. From the outputs of these models, we calculate the uncertainty metrics. The original test data refers to the labeled test data that was used during the development of the MUT.

Next, we use these uncertainty metrics as inputs to train the meta-model as illustrated in Figure 13. We train the meta-model as a binary classifier; for this purpose, we assign a label of either 0 or 1 to the uncertainty metrics data. A label of 1 is used to indicate incorrect classification, while 0 is used for

correct classification. To determine if an original test data is misclassified by the MUT, we compare the MUT's predictions for the data with their class labels.

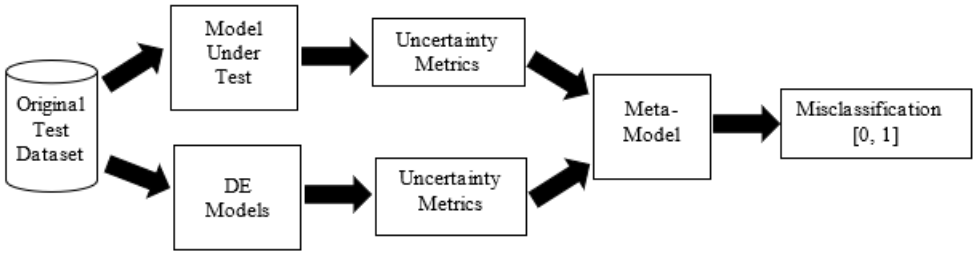


Figure 13: Meta-Model Training Process [120] ©2024 ACM.

We selected the Logistic Regression Model as our meta-model for analyzing the relationship between uncertainty metrics and misclassifications of MUT. Logistic Regression is a type of statistical analysis that is used to examine the relationship between a dependent variable and one or more independent variables. It is well-known for its efficacy in binary classification tasks and is widely used to estimate the probability of an event occurring in situations where the outcome is binary (yes or no, true or false, 1 or 0). The Logistic Regression Model is a good choice for our meta-model as it offers a straightforward and easy to implement method for exploring the relationship between inputs and the output in a binary classification task. In our case, inputs are uncertainty metrics derived from base models for the test input, and output is whether the test input will be misclassified by the tested DNN model. The Logistic Regression formula is given in Equation 11:

$$\begin{aligned} \text{P=f(z) where } f(z) &= \frac{1}{1+e^{-z}} \\ z &= \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k \end{aligned} \tag{11}$$

where P is the dependent variable, β_0 is the constant value (a.k.a "intercept") and β_i are the coefficients (a.k.a "slopes") for the independent variables x_i.

After training, we employed the meta-model on the generated test datasets. We fed these test datasets to base models and calculated the uncertainty metrics from their outputs. Then, we used these uncertainty metrics as input to the meta-model to forecast whether the MUT will make the correct prediction on these data. We used the probability output generated by the meta-model to identify and prioritize the data that is most likely to be misclassified by the MUT.

5.3 OOD Detection with Vision Transformer Models

Furthermore, we explored a different OOD detection method other than stated in Section 4.2.1. The state-of-the-art methods for detecting OOD data have been notably successful with significantly dissimilar or "far" OOD data. However, the detection of OOD data becomes more challenging when distinguishing "near" OOD data - data that is different yet similar to the in-distribution data. Fort et al. [121] introduced a method that effectively identifies such near OOD data using pre-trained transformer models.

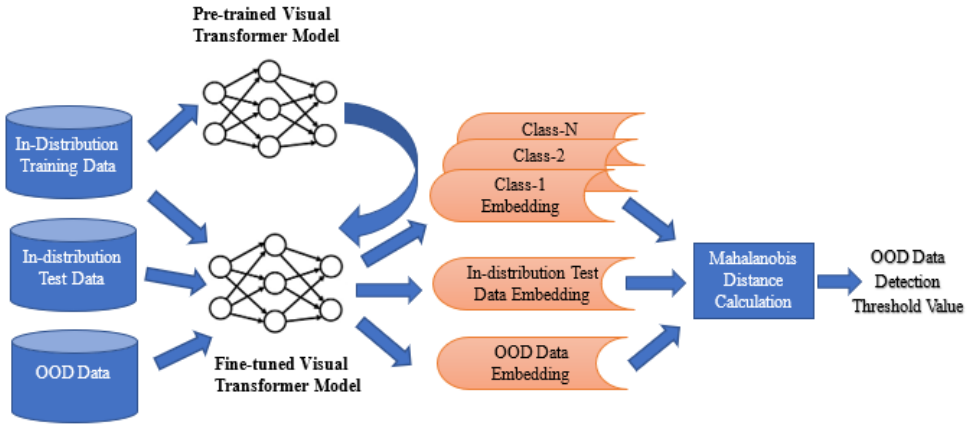


Figure 14: OOD Data Detection Threshold Value Selection Process [120] ©2024 ACM.

This detection method leverages a transformer model pre-trained using a large-scale labeled image dataset. The process begins with fine-tuning this model on the training dataset of the MUT. After fine-tuning, the pre-logit embeddings of the training data, which are the outputs from the transformer model's penultimate layer, are collected. These embeddings are internal representations of the input data that the model has learned. Afterward, they are categorized according to the class labels of the training data.

The method continues by gathering embeddings for the original test dataset and a selected OOD dataset. The next step involves calculating the shortest distances from each OOD data to the embeddings of the training data, grouped by classes. This calculation is also performed between the embeddings of the original test dataset and the training data. The Mahalonobis distance, as defined in Equation 7, serves as the measure for these calculations. The use of Mahalanobis distance for OOD detection using embeddings of data was first proposed by Lee et al. [122]. We employed the resultant distances as OOD scores. These scores determine how similar or different the data is from the training data distribution, aiding in the effective recognition of OOD data. Scores of original test data are used to represent the scores for in-distribution data, while scores belonging to the selected OOD dataset represent the scores for OOD data.

Subsequently, we use this transformer model on unseen test data to identify whether it is OOD data. For this purpose, we determine a threshold value that will be used to discriminate OOD and in-distribution data based on their OOD scores. Our approach involves the construction of two separate line graphs to determine this threshold value. Each graph represents a different dataset, with one graph representing OOD data and the other representing in-distribution test data. The x-axis of each graph represents OOD scores, while the y-axis indicates the frequency of data.

We proceed by selecting a set of OOD scores (distance values) that are close to the intersection of the two graphs, as these values have the potential to effectively discriminate between in-distribution and OOD inputs. To determine the best threshold value from this set of potential thresholds, we make use of the F1-score. The potential threshold values are used to categorize data points that have OOD scores greater than the threshold value as OOD data. The true positive value is the number of data from the OOD dataset that have a greater OOD score than the threshold value, while the false positive value is

the number of data points from the in-distribution test dataset that have a greater OOD score value than the threshold value. We select the OOD score value that results in the highest F1-score as the threshold value. The threshold value is later used to identify whether a new input will be classified as OOD. The visual representation of this process is shown in Figure 14.

The primary purpose of utilizing this OOD detection technique in our research is its remarkable success in differentiating near OOD data from in-distribution data. It is worth noting that test datasets generated through image generation techniques, such as image transformation, adversarial attacks, and generative models, produce data that is semantically similar to the original dataset but exhibits a covariate shift. If the image generation method produces data with a significant shift, making it substantially different from the in-distribution data, it can be referred to as near OOD data.

CHAPTER 6

EXPERIMENTS

We conducted experiments to evaluate the proposed methodologies using image classification datasets and corresponding DNN models. In the following sections, we first detail these datasets and models in Section 6.1. Next, in Section 6.2, we describe the methods used to generate additional test data for the experiments. The datasets, DNN models, and test data generation methods are common across experiments.

In Section 6.3, we detail the experiments conducted to evaluate the DNN testing framework proposed in Chapter 4. The experiments for new test data prioritization metrics described in Section 5.1 are presented in Section 6.4. Additionally, the experiments for the meta-model based test data prioritization approach described in Section 5.2 and the OOD detection method that employs Visual Transformer Models, which is outlined in Section 5.3 are provided in Section 6.5.

We conducted the experiments five times with randomly selected seed data and reported the average results to eliminate the influence of randomness.

In our experiments, we used labeled test data to evaluate the effectiveness of test prioritization methods. Specifically, we aimed to determine how well these methods prioritize the test data that the model misclassifies. By prioritizing the labeled test data and comparing these labels with the model's outputs, we assessed the performance of the prioritization techniques. The results of these experiments demonstrate the ability of these methods to prioritize fault-reveling data within unlabeled datasets in real-world scenarios.

6.1 Test Datasets and DL Models

We used the three most popular datasets for image classification: MNIST [123], CIFAR-10, and CIFAR-100 [124]. The MNIST dataset comprises gray-scale images of handwritten digits, Figure 15. Each image has a resolution of 28x28 pixels and is labeled between 0 to 9. The dataset includes a total of 60000 training images and 10000 testing images. We employed LeNet-5 and LeNet-1 [125], which are part of the LeNet family, as the DNN models for MNIST. We trained both of the models with all MNIST training data for 25 epochs.

LeNet-1 and LeNet-5 are the earliest samples of convolutional neural network (CNN) architectures developed by LeCun et al. LeNet-1 has a simple design featuring two convolutional layers, each followed by a pooling layer and a fully connected layer at the end. LeNet-5, a more advanced and



Figure 15: Sample images from MNIST dataset.

well-known version, comprises seven layers, including two convolutional layers, each followed by a pooling layer, and then three fully connected layers, as shown in Figure 16.

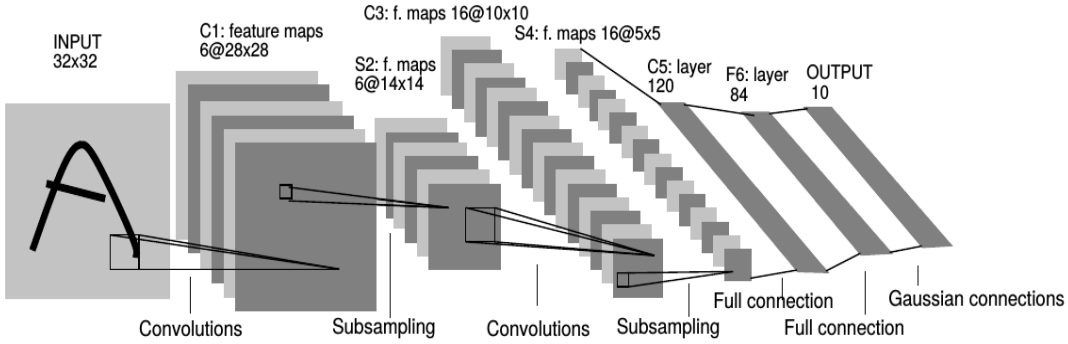
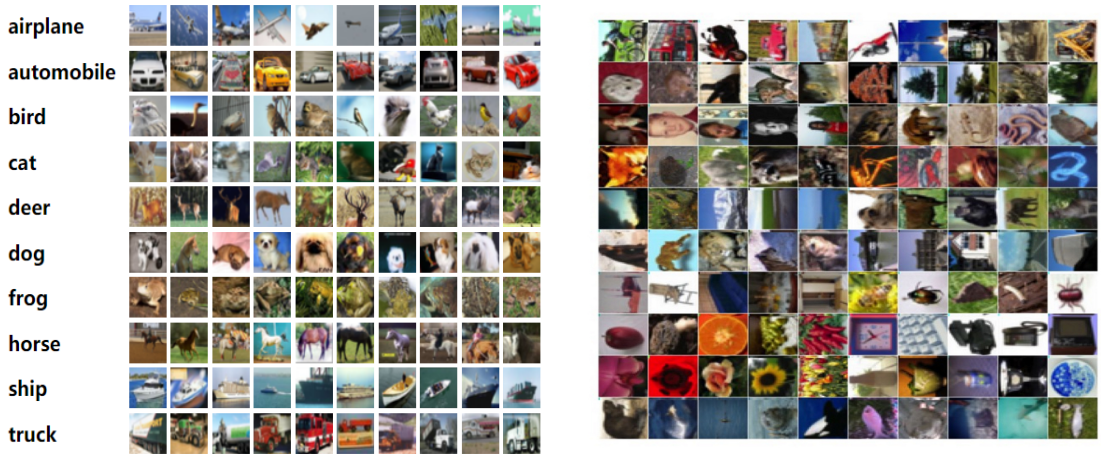


Figure 16: Architecture of LeNet-5 model [125].

The CIFAR-10 and CIFAR-100 datasets are widely used colored image datasets that include images with a resolution of 32x32 pixels, Figure 17. CIFAR-10 comprises a total of 50000 training images and 10000 test images. The images in this dataset are composed of ten different categories, including airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. We conducted experiments on the CIFAR-10 dataset using two popular DNN models, ResNet-32 [126] and VGG-19 [127]. ResNet-32 is a deep residual neural network architecture, which we trained for a total of 200 epochs. VGG-19, on the other hand, is a convolutional neural network that we trained for 150 epochs. During the training of both models, the entire CIFAR-10 training dataset was used in each epoch.

The ResNet DNN model is an influential architecture in DL, which was introduced by He et al. [126]. The model addresses the difficulty of training very deep networks due to the vanishing gradient problem by using residual blocks. A residual block consists of two convolutional layers with batch normalization and ReLU activation. Additionally, it has a shortcut connection that allows the input to be directly added to the output, facilitating the training of deeper networks. The architecture of ResNet-32 used for CIFAR-10 is illustrated in Figure 18. In this DNN, 3X3 convolutions with filter sizes 16, 32, and 64 are employed in residual blocks.



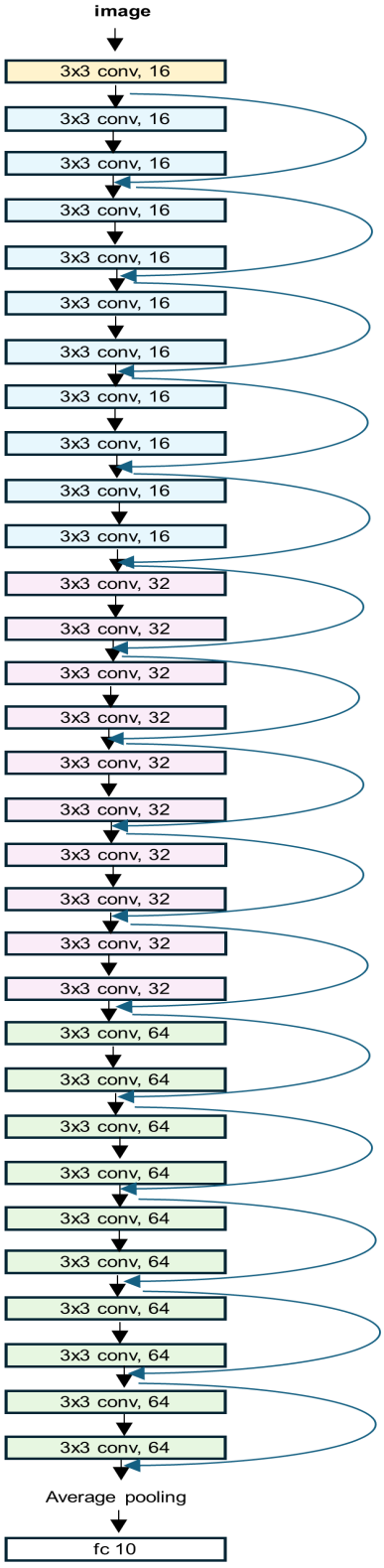
(a) CIFAR-10 (b) CIFAR-100

Figure 17: Sample images from CIFAR-10 and CIFAR-100 datasets.

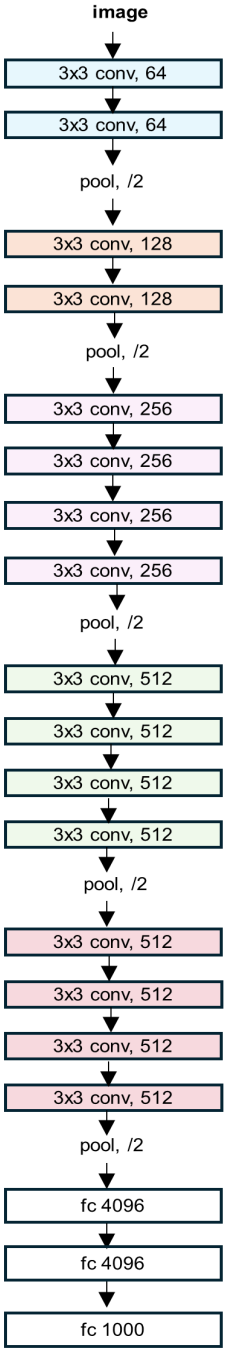
The VGG-19 DNN model comprises a total of 19 layers featuring a sequence of 3x3 convolutional layers with filter sizes ranging from 64 to 512, as shown in Figure 18. These convolutional layers are organized in blocks and separated with max-pooling layers for downsampling. Following the convolutional layers, there are three fully connected layers, and the ultimate layer generates the classification output. The model's depth enables it to capture complex data patterns, resulting in high performance on image recognition tasks, as evidenced by its success in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [128]. We added one 10-unit fully connected layer at the end of this network to get 10 class output probability for the CIFAR-10 dataset.

CIFAR-100 contains a diverse set of 100 classes of objects, including mammals, fish, flowers, fruits, household devices, trees, vehicles, insects, and more. The dataset comprises a total of 50000 training and 10000 test images. To achieve high accuracy on this dataset, we used the ResNet 28-10 model [126], which is deep residual neural network architecture and has proven to be successful in image classification. The wide residual networks are developed based on the original ResNet structure to enhance training effectiveness and results by expanding the width of the residual blocks instead of increasing their depth. We trained the model for 200 epochs with the entire training dataset.

Table 1 provides a summary of each dataset, its corresponding DNN model, and the accuracy achieved by the model on the original test dataset after training. These datasets and models provide a strong basis for comparison with the literature since they are widely accepted and used by the research community. The datasets we selected exhibit diverse characteristics, including variations in color (black-and-white and color images), class sizes, and complexity levels. Furthermore, the models chosen vary in both complexity and network depth. This diverse selection enables us to perform a more thorough and comprehensive evaluation. Additionally, these models have a high level of accuracy, making them well-suited for assessing the effectiveness of various test selection methods in detecting faults. Models with higher accuracy make fewer mispredictions on a given dataset compared to models with lower accuracy. This allows us to more effectively evaluate whether a test prioritization method is successful in identifying data that will be mispredicted by the DNN model.



(a) ResNet-32



(b) VGG-19

Figure 18: Architecture of ResNet-32 and VGG-19 DNN models (adapted from [126]).

Table 1: Datasets and DNN Models used in experiments.

Dataset	DNN Model	Layers*	Parameters	Accuracy
MNIST	LeNet-5	7	61.7 K	99.15
	LeNet-1	5	3246	98.2
CIFAR-10	ResNet-32	33	0.46 M	92.10
	VGG-19	24	20.5 M	90.33
CIFAR-100	ResNet 28-10	30	36.5M	78.76

* Layers include convolutional, pooling, and fully-connected layers.

6.2 Test Data Generation Methods

We generated three types of test data by employing the methods described in Section 4.1 and referred to them as Image Transformation, Adversarial Attacks, and Generative Models test datasets.

We used four image transformation techniques to generate synthetic images: rotation, translation, blurring, and changing brightness. Among these, translation and rotation are affine transformations, while blurring and brightness are pixel transformations. These techniques are used to generate images that represent real-life scenarios, such as changes in lighting conditions or the orientation of the camera. The image transformations applied to seed images do not cause the label of the image to be altered. Image labels are used to describe the class of the object in the image, for example, a cat or a handwritten digit. Image transformations do not alter the class of the objects in the image. This means that if an image of a cat is rotated, it is still a cat, and if an image of a digit is blurred, it is still the same digit. Therefore, we assigned the same labels with the seed images for the transformed images.

For generating adversarial attack test data, we employed three adversarial attack techniques: Fast Gradient Sign Method (FGSM) [129], Carlini & Wagner (C&W) [130], and Projected Gradient Descent (PGD) [131]. These adversarial attack methods are widely used in test data selection studies [23–25, 27]. Modifications made to seed images by adversarial attacks are often so subtle that they can not be perceived by the human eye. Since these alterations do not change the semantic content of the images, in our case, the class of images, they do not alter the labels of the images. As a result, we have assigned the same labels to the perturbed images by adversarial attacks with the seed images.

Lastly, we used generative adversarial networks (GANs) that were either pre-trained or trained by us to generate synthetic test images. For the MNIST dataset, we trained our own GANs. However, for the CIFAR-10 and CIFAR-100 datasets, we opted to use pre-trained GAN models since generating realistic synthetic images is more challenging for these datasets, and the development of a GAN model falls outside our main area of focus. We chose the Diffusion Projected GAN and Diffusion StyleGAN2 models for CIFAR-10, while we selected the Auxiliary Discriminative Classifier GAN for CIFAR-100. With the objective of generating realistic images, we based our selection on the reported Fréchet Inception Distance and Inception Score values of CIFAR-10 and CIFAR-100 GAN models. The Fréchet Inception Distance determines the similarity between the generated images and real images, while the Inception Score evaluates the quality and diversity of the generated images. By selecting the models with high scores in both metrics, we aimed to generate images that are both realistic and diverse. We

used generator networks of GANs to create test input images. We manually labeled the test images generated for MNIST and CIFAR-10. However, for CIFAR-100, we created images by specifying the target labels with the advantage of using conditional GANs.

6.3 Experiments for DNN Testing Framework

In this section, we outline Experiment-1, conducted to address RQ1, and Experiment-2, conducted to address RQ3. The corresponding research questions are restated below:

- RQ1: What is the effectiveness of uncertainty-based test selection methods in identifying and prioritizing fault-revealing data instances within datasets generated through different techniques, and how does this effectiveness vary under different conditions?

  RQ1.1: How effective are the test selection methods when the datasets exclusively consist of in-distribution data?

  RQ1.2: What variations in effectiveness are observed when the datasets contain both OOD and in-distribution data?

- RQ3: Is it possible to use post-hoc explainability methods in DNN testing to understand the root cause of test failures (i.e., incorrect predictions of the tested DNN model)?

In Experiment-1, we assess the effectiveness of uncertainty-based test selection methods in identifying and prioritizing fault-revealing data instances within datasets generated through various techniques. We also examined how this effectiveness changes with/without the inclusion of OOD data in the test datasets and with different dataset sizes.

In Experiment-3, we explore the use of post-hoc explainability methods to understand the root cause of DNN failures in test results.

6.3.1 Test Setup

We used MNIST and CIFAR-10 test datasets in these experiments. We applied image generation methods as discussed in Section 6.2. It should be noted that each method was applied independently, meaning that only one method was used at a time for each seed image, with no combination or sequential use of multiple methods. First, we randomly selected 1000 seed images from the original test dataset, ensuring equal image count across classes. Then, we applied image transformation and adversarial attack methods on these images and created 1000 additional test inputs with each method. Moreover, we generated 1000 new images using each GAN model for CIFAR-10 and MNIST.

To summarize, we generated a total of 10000 additional test images for MNIST and 9000 for CIFAR-10 using the different methods mentioned above. The resultant test datasets are listed in Table 2, where each dataset is named according to the method used to generate it. The row with the "Original" value in the "Applied Method" column represents the seed images randomly chosen from the original test dataset. These images were added as-is to the test datasets composed of generated test data, in addition

to being used as seed images. Note that the "+" symbol in the table indicates that the corresponding method has been applied to the "Original" data.

Table 2: Details of Generated Datasets for Experiment-1 [132] ©2023 IEEE.

ID	Generated Test Dataset	Dataset	Data Count	Applied Method
1	Image Transformation	CIFAR-10	1000	Original
			1000	+Blur
			1000	+Brightness
			1000	+Translate
			1000	+Rotate
2	Adversarial Attacks	CIFAR-10	1000	Original
			1000	+Fast Gradient Sign Method (FGSM)
			1000	+Carlini & Wagner Attacks (C&W)
			1000	+Projected Gradient Descent
3	Generative Models	CIFAR-10	1000	Original
			1000	Diffusion Projected GAN
			1000	Diffusion StyleGAN2
4	Image Transformation	MNIST	1000	Original
			1000	+Blur
			1000	+Brightness
			1000	+Translate
			1000	+Rotate
5	Adversarial Attacks	MNIST	1000	Original
			1000	+Fast Gradient Sign Method (FGSM)
			1000	+Carlini & Wagner Attacks (C&W)
			1000	+Projected Gradient Descent
6	Generative Models	MNIST	1000	Original
			1000	Generative Adversarial Network (Multi-perceptron layers)
			1000	Dep Convolutional Generative Adversarial Network
			1000	Auxiliary Classifier Generative Adversarial Network

We provided the parameters for adversarial attack methods and degrees of image transformations in Tables 3 and 4, respectively. For image transformation, we randomly applied a level of transformation between the lower and upper values specified in Table 4. We used the Cleverhans [133] library to apply adversarial attack methods to the images.

6.3.2 Experiment-1

As the first step after generating test data, we performed out-of-distribution detection on these data. To perform the OOD detection as detailed in Section 4.2.1, we constructed Variational Autoencoder (VAE) networks for both MNIST and CIFAR-10 and trained them using their respective training

Table 3: Parameters of Adversarial Attack methods employed to generate test data [132] ©2023 IEEE.

Method	Parameter Name	CIFAR-10	MNIST
CarliniWagnerL2	binary_search_steps	1	1
	max_iterations	100	1000
	learning_rate	5e-03	5e-03
	initial_const	10	1
FastGradientMethod2	eps	0.1	0.1
	norm	np.inf	np.inf
projected_gradient_descent	eps	0.3	0.1
	eps_iter	0.02	0.02
	nb_iter	40	40
	norm	np.inf	np.inf

Table 4: Degree of image transformations applied to original test data ([132] ©2023 IEEE.

Method	Transformation Value
Blur	%0 - %50
Brightness	%0 - %10
Translate	%0 - %30
Rotate	%0 - %10

datasets. We provided the structure of the VAEs in Table 5. With training the VAEs with the training set of the MNIST and CIFAR-10, our main objective was to enable them to learn the data distribution of these datasets. This would allow them to identify and differentiate data coming from a different data distribution.

Table 5: Architecture of Variational Autoencoder Models used for OOD Data Detection [132] ©2023 IEEE.

Model Name	Model Structure	Training Epochs
MNIST VAE	Input Layer - 3 Dense - Lambda - 3 Dense	50
CIFAR-10 VAE	Input Layer - 4 Conv2D - Flatten - 3 Dense - Lambda - 2 Dense - 3 Conv2DTranspose - Conv2D - Flatten - 2 Dense	200

After training VAEs, the next step is to identify the threshold value for reconstruction probabilities outputted by VAEs. This threshold value is then used to differentiate between in-distribution and OOD data. In order to identify the threshold value, we need two datasets - one for in-distribution and one for OOD data. We used the original test datasets of MNIST and CIFAR-10 as in-distribution data. Additionally, we chose a different dataset for each of them to act as the OOD dataset.

The selection of OOD datasets affects the determination of threshold values. We ensured the OOD datasets matched the in-distribution datasets in terms of color scale, resolution, and size, yet they varied in their semantic content. We selected the datasets from the ones commonly used as OOD datasets for MNIST and CIFAR-10 in the literature. [24, 43, 121].

For our experiments, we opted to use the Fashion MNIST [134] dataset as the OOD dataset for the MNIST dataset. The Fashion MNIST dataset comprises grayscale images that are of the same size as those in the MNIST dataset. However, instead of handwritten digit images, Fashion MNIST contains images of clothing items. This difference in data distribution results in the VAE, trained on the MNIST training dataset, to produce lower reconstruction probabilities for images from Fashion MNIST.

Similarly, we selected the SVHN (Street View House Numbers) [135] dataset as the OOD dataset for the CIFAR-10 dataset. The SVHN dataset comprises images of house numbers captured from the Google Street View dataset. These images have the same resolution as CIFAR-10 images. SVHN's data distribution is distinct from that of CIFAR-10, which contains images of objects, animals, and natural scenes. Consequently, the VAE trained on the CIFAR-10 dataset will produce lower reconstruction probabilities for images from the SVHN dataset.

We fed the trained VAEs with the images in both the original test datasets and selected OOD datasets to obtain the reconstruction probabilities. We drew two line graphs with the reconstruction probabilities of both datasets and selected a set of potential threshold values that were close to the intersection point of these two graphs. We then identified the reconstruction probability threshold value that best distinguishes the OOD dataset from the in-distribution dataset by finding the threshold value that achieved the best F1 score.

Subsequently, we identified the OOD data in generated test datasets for CIFAR-10 and MNIST using the selected threshold values. In order to do this, we input the generated test datasets to VAEs and calculated reconstruction probabilities. If the reconstruction probability of an image is lower than the threshold value, it is identified as OOD data, and if it is higher, it is identified as in-distribution data. After identifying the OOD data in the generated test datasets, we constructed two types of test datasets: one containing only in-distribution data and the other containing both in-distribution and OOD data.

After constructing the test datasets, we developed and trained the Deep Ensemble(DE) and Variational Inference with Flipout(VI-F) models that will be used to prioritize the generated test datasets. For the DE, we trained five DNN models, each sharing the same architecture as the tested model but initialized with different initial values for their hyperparameters.

On the other hand, we developed VI-F models by replacing the standard layers of tested DNN models with their Flipout counterparts. We leveraged the TensorFlow Probability Library within TensorFlow 2 since it offers built-in Flipout DNN layers to perform variational inference. In terms of architecture modifications, we replaced all dense and convolutional layers within the LeNet architectures with Flipout layers. This was straightforward due to the relatively simple structure of LeNet models. However, when it came to more complex models, ResNet-32 and VGG-19, we encountered challenges. Specifically, these complex DNN models exhibited issues with convergence or demonstrated very low accuracy when all their layers were replaced with Flipout layers. Based on the results from the studies in the literature, instead of replacing all layers with Flipout layers, we replaced only certain layers. For ResNet-32, the use of Flipout layers in the final convolutional layer of each residual block yielded the best results. Meanwhile, for VGG-19, we replaced all convolutional layers with Flipout layers.

Previous research in the field of uncertainty quantification [96] has provided insights into the optimal sample size for effective results when utilizing DE and VI methods. Ovadia et al. demonstrated that a sample size of five is sufficient to achieve good results for both DE and VI approaches, and increasing the sample size beyond five tends to yield diminishing improvements. Consequently, in the application of the DE method, we employed an ensemble of five members to have a balance between computational efficiency and the accuracy of uncertainty estimation. Similarly, for the VI method, we chose to collect five samplings from the pass-throughs of the VI model for each test input.

In this experiment, we assessed the effectiveness of test prioritization methods for two categories of test datasets: 1) test datasets that contain only in-distribution data and 2) test datasets that contain both in-distribution and OOD data. To achieve this, we followed the steps outlined in the succeeding paragraphs for each category.

First, we fed the generated test data into all the DE models. The outputs of models for each test input are aggregated by taking the average of these outputs, and uncertainty metrics are calculated from these combined output values. Second, for the VI method, we fed each test input to the VI-F model five times and gathered five outputs for each test input. Similar to the DE method, we aggregated the outputs and calculated the uncertainty metrics for each input. Lastly, we fed each test input to MUT and calculated uncertainty metrics from the output of the MUT model.

Table 6: Uncertainty-based Test Data Prioritization Strategies.

ID	Abbreviation	Uncertainty Estimation Model	Uncertainty Metric
1	MUT-Entropy	Model Under Test	Entropy
2	MUT-Confidence	Model Under Test	Least Confidence
3	MUT-Margin	Model Under Test	Margin
4	DE-Entropy	Deep Ensemble	Entropy
5	DE-Confidence	Deep Ensemble	Least Confidence
6	DE-Margin	Deep Ensemble	Margin
7	VI-Entropy	Variational Inference with Flipout	Entropy
8	VI-Confidence	Variational Inference with Flipout	Least Confidence
9	VI-Margin	Variational Inference with Flipout	Margin

We prioritized the test datasets using values of uncertainty metrics calculated through three different approaches, resulting in nine prioritization strategies that are given in Table 6. Then, we evaluated the effectiveness of these test data prioritization strategies using the evaluation methods defined in Section 4.2.3.

Additionally, we sorted the test datasets randomly. We used this method to serve as the minimum performance baseline for comparison in the effectiveness of test prioritization approaches in terms of identifying fault-revealing inputs.

Furthermore, we conducted a comparison between these nine prioritization strategies and state-of-the-art test data selection methods, including Likelihood-based Surprise Adequacy (LSA), Distance-based Surprise Adequacy (DSA) [29], DeepGini [25], and Multiple Boundary Clustering and Prioritization (MCP) [26]. We selected these methods since they have been widely used in test prioritization studies

for comparison, and in this way, they serve as a benchmarking baseline. For the implementation of these methods, we used the source code provided by the original authors of the studies. We used the activation values of neurons in the layer preceding the final layer of the DNNs to generate the activation traces for both DSA and LSA.

6.3.3 Experiment-2

We conducted an in-depth analysis of the misclassifications that occurred in the test datasets with the goal of understanding the reasons behind the model's incorrect decision-making process.

First, we determined the predicted versus actual classes of the test data in each test dataset and generated a confusion matrix for each dataset. By examining the confusion matrices, we identified the classification error that occurred most frequently between the two classes.

Our next step was to apply visualization methods to images of these two classes from the original test dataset that were correctly classified. This made it possible for us to identify the particular regions of images that had the greatest influence on the model's prediction. Subsequently, we applied visualization techniques to samples from generated test data that belonged to these classes but had been incorrectly classified. By comparing the regions highlighted in the images that were correctly classified and those that were not, we aimed to understand the rationale behind the incorrect prediction of the MUT.

6.4 Experiments for Alternative Test Data Prioritization Approaches

We conducted two experiments to assess the effectiveness of the uncertainty metrics proposed in Chapter 5 for prioritizing test data. In the first experiment, we used the Mahalanobis Uncertainty Score metric, and in the second experiment, we employed the DE Variation Score metric to measure the uncertainty and prioritize the test datasets. Through these two experiments, we continued to evaluate the effectiveness of uncertainty-based test selection methods in identifying and prioritizing fault-revealing data instances in the scope of RQ1.

In both of these experiments, we used the generated test dataset for CIFAR-10 in Section 6.3.1 with the ResNet-32 DNN model. We used the test datasets that do not include OOD data.

6.4.1 Experiment-3

In this experiment, we used the MUS metric to prioritize the generated test datasets. We calculated the MUS metric values by using the MUT and DE models separately. Firstly, we employed the MUT model to generate representations of the CIFAR-10 training dataset. We then obtained the representation of the generated test dataset in the same way. After that, we calculated the MUS scores as described in Section 5.1.1 and prioritized the test data based on their MUS scores. We gave higher priority to test data with higher MUS scores.

Secondly, we used the DE models to generate representations of both the training dataset and generated test datasets. We calculated the average of MUS scores obtained from each DE model and used this

averaged MUS value to prioritize the generated test data. Finally, we compared the prioritization results with the results obtained from test data prioritization methods in Experiment 1.

6.4.2 Experiment-4

In this experiment, we used the DE Variation Score metric to prioritize the generated test datasets. For this purpose, we inputted the test datasets into all DE models along with the MUT and compared the predictions made by the DE models against the MUT. The number of DE models that produced different predictions from the MUT was recorded as the DE Variation Score for each test data. We then assigned priorities to the generated test datasets, favoring those with a high DE Variation Score. Lastly, we compared the prioritization results from this experiment with those obtained from test data prioritization methods in Experiment 1.

6.5 Experiments for Test Selection with Meta-Models

In this section, we outline Experiment-5, conducted to address RQ2, which is restated below:

- RQ2: How effective is the test prioritization approach proposed in this study, which is based on a meta-model that combines different uncertainty metrics, in identifying and prioritizing fault-revealing data instances within datasets, both include and exclude OOD data?

  RQ2.1: How does the effectiveness of the proposed method change according to different labeling budgets (i.e., the number of test data to be selected and prioritized)?

In this experiment, we evaluated the effectiveness of the proposed meta-model based test prioritization method in Section 5.2 for two categories of test datasets: 1) test datasets that are comprised solely of in-distribution data and 2) test datasets that contain both in-distribution and OOD data. To carry out the experiment, we first generated test data as explained in Section 6.5.1. Then, we performed OOD detection and constructed the test datasets. Finally, we prioritized the test data and analyzed the effectiveness of this prioritization in comparison to the SOTA methods. The process that we followed for OOD detection and test data prioritization is detailed in Section 6.5.2.

6.5.1 Test Setup

We used MNIST, CIFAR-10, and CIFAR-100 test datasets in this experiment.

In this experiment, we divided the original test datasets into two sets of 5,000 images each. To ensure balanced representation, we randomly selected an equal number of test data from each class for both sets. We used one of these sets to train the meta-modes, while the other set was used to evaluate the prioritization methods along with generated test datasets.

Table 7: Details of Test Datasets Employed in the Evaluation of Test Selection using Meta-Models [120] ©2024 ACM.

Test Dataset	Data Count	Applied Method	Details of Applied Method
Original Test Dataset	5000	-	-
Generative Models Test Dataset	2000	Original	-
	3000	+GAN	New images generated using generator models of GANs
Image Transformation Test Dataset	2000	Original	-
	2000	+Blur	Random blurring between [%0, %10]
	2000	+Brightness	Random brightness change between [-%10, %10]
	2000	+Translate	Random translation between [-%20, %20] ratios in horizontal and vertical directions
	2000	+Rotate	Random rotation between [-30, 30] degrees
Adversarial Attacks Test Dataset	2000	Original	-
	2000	+FGSM	eps: 0.1 eps_step: 0.1 norm: np.inf targeted: False
	2000	+C&W	binary_search_steps: 10 max_iterations: 10 learning_rate: 0.01 initial_const: 0.01 targeted: False
	2000	+PGD	eps: 0.1 eps_step: 0.02 max_iter: 40 norm: np.inf targeted: False

We randomly selected 2,000 images from the set not used for meta-model training, with an equal distribution among classes. These images served as seeds for adversarial attacks and image transformation methods discussed in Section 6.2. We generated an additional 2,000 test images per method. Furthermore, we created 3000 new synthetic images using the GANs stated in Section 6.2. Consequently, for each MNIST, CIFAR-10, and CIFAR-100, we ended up with 6,000 adversarial, 8,000 transformed, and 3000 synthetic test images. For the adversarial attack methods, we used the Adversarial Robustness Toolbox library (version 1.17.1) [136].

Table 7 lists the test datasets that we used in this experiment. They were referred to as Original, Image Transformation, Adversarial Attacks, and Generative Models. For each of the generated test datasets, we added an additional 2000 original test data. The parameters of adversarial attack techniques and the degree of image transformations are also indicated in Table 7.

6.5.2 Experiment-5

In this experiment, we used the OOD detection method outlined in Section 5.3 to identify test inputs deviating from the training data distribution in generated test datasets. The first step in the OOD detection process is to select a pre-trained Vision Transformer (ViT) model. We opted for the ViT architecture "ViT-B_16" [137] as our transformer model. The ViT-B_16 model was trained on the ImageNet-21k dataset [138], and its parameters were saved as a checkpoint after this training. We employed the ViT model with these saved parameters, and we fine-tuned it for each MNIST, CIFAR-10, and CIFAR-100 by independently performing training using their training datasets.

To evaluate the effectiveness of the method for detecting OOD instances, we employed both near and far OOD datasets for each dataset. Near OOD datasets were selected for their semantic resemblance to the in-distribution training datasets. In contrast, far OOD datasets, although the same in color, resolution, and size as the training datasets, were chosen for their distinct semantics.

We chose to use the same OOD datasets used in Experiment 1 as the far OOD datasets, where these sets were selected from data sets commonly used in the literature. For the MNIST dataset, we opted for FashionMNIST [134] to serve as the far OOD dataset. Similarly, for the CIFAR-10 and CIFAR-100 datasets, our choice was SVHN [135] dataset.

We paired CIFAR-10 and CIFAR-100 with each other as near OOD datasets, considering their semantic similarities and shared characteristics. Furthermore, we designated K-MNIST [139] as the near OOD set for MNIST, a decision grounded in its grayscale nature and visual similarity to MNIST. K-MNIST is a grayscale dataset comprising Kuzushiji characters from ancient Japanese.

While CIFAR-100 is semantically similar to CIFAR-10, SVHN is a colored dataset featuring street-view house numbers in digit form, thus diverging semantically from CIFAR-10. Similarly, Fashion MNIST is a grayscale dataset depicting clothing images and showing semantic differences with the MNIST dataset.

After selecting the OOD datasets, we gathered the embeddings for the original test datasets and chosen OOD datasets. Then, we calculated the OOD scores for original test datasets and OOD datasets as described in Section 5.3.

We determined threshold values for OOD scores to differentiate between OOD data and in-distribution data in each dataset. To evaluate the OOD detection performance, we calculated the F1-Score values for each of the selected threshold values. Furthermore, we also computed the Area Under the Receiver Operating Characteristic curve (AUROC), which serves as a critical metric for evaluating the performance of binary classifiers, as an alternative assessment for the effectiveness of the OOD detection method. In our context, the classification task is to determine whether the input is OOD based on its OOD score compared to the threshold value. If the OOD score of a test input is higher than the threshold value, it is classified as OOD, and vice versa. If the test data classified as OOD is from the OOD dataset, it is a true positive; otherwise, it is a false positive. The Receiver Operating Characteristic curve (ROC) curve depicts the True Positive Rate (TPR) against the False Positive Rate (FPR) at various threshold settings. A higher AUROC score signifies a better differentiation between the two classes, where the highest score is 1.0. An AUROC score of 0.5 indicates that the classifier's performance is not better than random guessing. We have used the AUROC score to evaluate the OOD detection method's ability to differentiate between in-distribution and OOD data with the chosen threshold values.

After determining the threshold value for OOD detection, we applied OOD data detection for all the generated test datasets. Test input that has an OOD score greater than the determined threshold value is identified as OOD data. Then, we created two categories of test datasets - one comprising solely in-distribution data and the other containing both in-distribution and OOD data.

Test Data Prioritization

We applied the test data prioritization method outlined in Section 5.2 to identify the fault-revealing test inputs. For this purpose, we trained three different meta-models using the input features listed in Table 8.

Table 8: Meta-Models and their input features [120] ©2024 ACM.

Meta-Model	Input Features
Meta-Model - 1	Least Confidence DE Variation Score
Meta-Model - 2	Margin DE Variation Score
Meta-Model - 3	Entropy DE Variation Score

To select input features for the meta-models, we conducted an assessment of the performance of uncertainty metrics across different types of test datasets using the results of previous experiments. The experiment results revealed that the DE Variation Score is the most effective metric in identifying fault-revealing data within test datasets containing OOD data and test datasets that include adversarial data. This metric leverages differences in the prediction of ensemble models when compared to the MUT's predictions.

On the other hand, the use of uncertainty metrics such as Least Confidence, Margin, and Entropy with the Deep Ensemble method failed to detect fault-revealing inputs in test datasets with adversarial data. This ineffectiveness is due to the model-specific nature of adversarial data. Since adversarial attacks

produce data that are specific to the model, these data do not introduce uncertainty in ensemble models trained on different parameters.

In contrast to uncertainty estimation that employs outputs of DE models, the uncertainty estimation with metrics that are derived from the output of MUT is simple, has a lower computational cost, and is still effective in identifying uncertainty in both original and synthetic test data.

In light of these results, we designed our meta-models by using the uncertainty metrics obtained from the MUT's output and the DE Variation Score as input features. We aimed to improve the identification of fault-revealing inputs across several data sets by combining their respective strengths.

The initial step in performing test data prioritization with the use of meta-models was to determine the values of selected uncertainty metrics as input. To accomplish this, we fed the test datasets to MUT and computed the margin, entropy, and least confidence metric values from the model's output. Additionally, we used these test datasets with DE models and MUT and calculated the DE Variation Score values. Next, we input these uncertainty metric values into meta-models and collected their outputs. Subsequently, we used the output values of meta-models to order the test datasets and assigned higher priority to those with higher output values.

To evaluate the effectiveness of combining uncertainty metrics using a meta-model, we compared its results with prioritization performed with the individual uncertainty metrics used as input to the meta-model. To perform this comparison, we prioritized the test datasets based on each input of the meta-models separately. We gave higher priority to test data having higher corresponding uncertainty values.

Additionally, we conducted a comparative analysis of our test data prioritization method using a meta-model against other state-of-the-art test selection techniques. These techniques include Likelihood-based Surprise Adequacy (LSA), Distance-based Surprise Adequacy (DSA) [29], DeepGini [25], and Multiple-boundary Clustering and Prioritization (MCP) [26]. This comparison allowed us to assess the strengths and weaknesses of our proposed approach and determine its effectiveness in identifying fault-revealing inputs compared to other existing methods.

CHAPTER 7

EXPERIMENT RESULTS

7.1 Results for DNN Testing Framework

We evaluated the effectiveness of nine test prioritization strategies listed in Table 6 by using the generated test datasets given in Table 2. We performed this assessment with two categories of test datasets: 1) test datasets that contain only in-distribution data and 2) test datasets that contain both in-distribution and OOD data.

7.1.1 Experiment-1 Results

We employed VAEs that were trained for MNIST and CIFAR-10 datasets to detect OOD data in the generated test data. The F1-Scores for the chosen threshold reconstruction probability values that we used to identify OOD data in the generated test data are presented in Table 9.

Table 9: F1-Score Values for OOD detection performed with VAEs using selected threshold values [132] ©2023 IEEE.

Dataset	True Positive	False Positive	F1-Score
MNIST	99.9%	0.29%	0.99
CIFAR-10	83.52%	17.62%	0.83

Comparing the reconstruction probability of each image in the generated test datasets with the selected reconstruction probability threshold values, we classified the images as either OOD or in-distribution data. The percentage of OOD data for each dataset is illustrated in Figure 19. Adversarial attacks tend to create a high amount of OOD data, while Generative Models produce a moderate amount of such data. The properties of the image dataset determine the Image Transformation method that generates the highest ratio of OOD data.

The data produced by FGSM and PGD adversarial attack methods were mostly categorized as OOD data for both MNIST and CIFAR-10 datasets. In contrast, the images subjected to the C&W attack were predominantly classified as in-distribution, with rates of 83.5% for CIFAR-10 and 99% for MNIST.

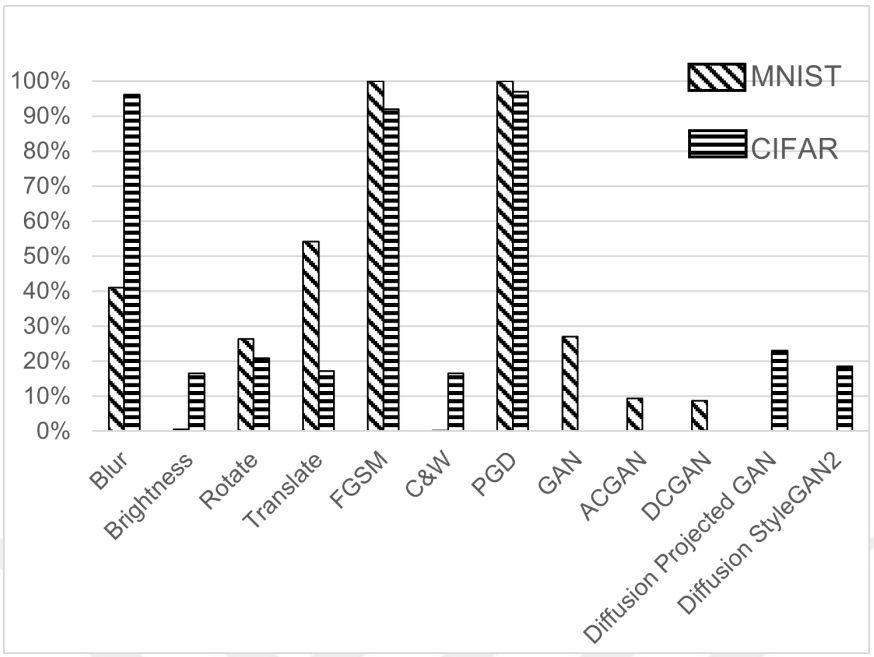


Figure 19: Percentages of OOD Data in Generated Test Datasets using OOD detection with Variational Autoencoders (VAEs) [132] ©2023 IEEE.

Despite using the same transformation settings given in Table 4 for both CIFAR-10 and MNIST datasets, the percentage of OOD data in the generated datasets varies between these datasets. For MNIST, images generated using the Translate method exhibit the highest OOD ratio, 54.2%.

Table 10: Experiment-1 - Accuracy values of DNN models for Generated Test Datasets without OOD data [132] ©2023 IEEE.

	MNIST		
Model	Generative Models	Image Transformation	Adversarial Attacks
LeNet-5	91.8%	92.4%	41.6%
LeNet-1	90.4%	88.0%	7.0%
	CIFAR-10		
Model	Generative Models	Image Transformation	Adversarial Attacks
ResNet-32	79.4%	77.3%	13.5%
VGG-19	79.8%	76.4%	7.2%

We constructed test datasets that only contain in-distribution test data and test datasets with OOD data. The accuracy of the models on test datasets that only contain in-distribution test data is given in Table 10. While the majority of the images produced by the C&W adversarial attack method have been

classified as in-distribution data, these images fooled the DNN models, as evidenced by a very low accuracy value of 7% in LeNet-1 and VGG-19 models.

After creating test datasets, we ordered them based on nine different strategies for test prioritization. We then evaluated these strategies to determine how effectively they identified test data that led to incorrect predictions by the DNN model. We used three evaluation methods, which are described in Section 4.2.3: Ratio of Area Under Curve (RAUC), Average Percentage of Fault Detected (APFD), and Fault Detection Ratio metrics. Additionally, we investigated the correlation between the uncertainty scores produced by each prioritization strategy and the DNN model's incorrect predictions. The results of the experiments conducted using test datasets without OOD data and those with OOD data are presented in the following subsections.

Results for Test Datasets without OOD Data

First, we computed the APFD values for each test data prioritization approach. The APFD, defined in Section 4.2.3, indicates how quickly the failures, which means incorrect predictions of MUT, are identified. APFD values computed using each test dataset are listed in Table 11. To refer to the test data prioritization strategies, we used abbreviations listed in Table 6. These abbreviations were formatted as method-metric tuples. For instance, we used MUT-Confidence to refer to prioritization made by employing the Least Confidence metric calculated using the output of the MUT or DE-Entropy to refer to prioritization performed using the Entropy metric calculated from the outputs of DE models.

Table 11 provides a comparison of these nine test prioritization approaches with selected SOTA methods: LSA, DSA, DeepGini, and MCP. The results reveal that the test prioritization performed using the DE method yields high APFD values ranging from 0.887 to 0.919 for MNIST and 0.792 to 0.830 for CIFAR-10, excluding the adversarial attack test datasets. Conversely, for adversarial attack test datasets, metrics obtained using MUT, DeepGini, LSA, and DSA exhibit higher APFD values compared to other approaches. Notably, the APFD values for MUT-Entropy, MUT-Confidence, and MUT-Margin were very close, differing by less than 0.002.

Next, we performed an evaluation for test prioritization approaches using the Fault Detection Ratio metric described in Section 4.2.3. For this evaluation, firstly, the prioritized test datasets were grouped into ten bins, each containing 10% of the total test data. The first bin represents the highest-priority test data, while the last bin contains the lowest-priority data. For each bin, we identified the number of misclassified test data and calculated the ratio to the total count of misclassified data in the entire test dataset. Then, a line graph of the cumulative ratio of faults detected against the ratio of prioritized test data employed is drawn. The Figures 20 and 21 depicts the Fault Detection Ratio for each generated test dataset. As the test dataset size grows from 10% to 100%, the ratio of fault detected also increases. Approaches that exhibit a higher fault detection ratio for a given percentage of test datasets are more effective in prioritizing the test data that is more likely to be misclassified by the DNN model.

DE methods exhibit the highest fault detection ratios for test datasets generated by Generative Models and Image Transformations. However, for datasets created by adversarial attacks, test prioritization with DSA and DeepGini, as well as metrics obtained by using MUT outputs, outperform other techniques.

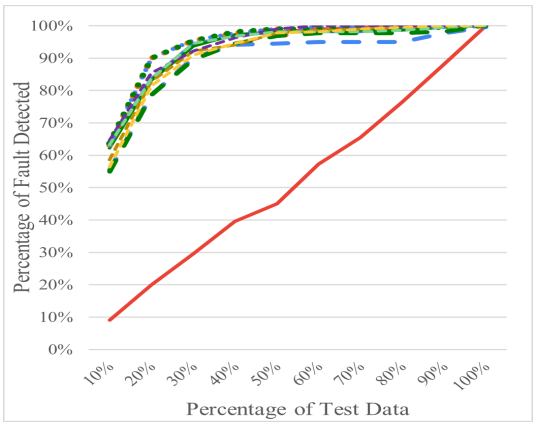
Table 11: Experiment-1 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD [132] ©2023 IEEE.

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	LeNet-5	LeNet-1	LeNet-5	LeNet-1	LeNet-5	LeNet-1
Random	0.486	0.501	0.534	0.518	0.481	0.492
MUT - Entropy	0.893	0.890	0.910	0.857	**0.845**	0.743
MUT - Confidence	0.889	0.889	0.910	0.863	**0.844**	**0.746**
MUT - Margin	0.889	0.889	0.910	0.864	0.843	**0.746**
DE - Entropy	**0.905**	**0.903**	**0.918**	**0.887**	0.806	0.722
DE - Confidence	**0.904**	**0.904**	**0.919**	**0.893**	0.803	0.723
DE - Margin	**0.903**	**0.903**	**0.918**	0.892	0.806	0.723
VI - Entropy	0.868	0.864	0.872	0.800	0.770	0.676
VI - Confidence	0.850	0.864	0.874	0.802	0.768	0.676
VI - Margin	0.861	0.863	0.873	0.801	0.768	0.676
LSA	0.880	0.815	0.903	0.858	0.824	0.710
DSA	0.896	0.850	0.912	**0.893**	0.842	**0.752**
DeepGini	0.889	0.890	0.910	0.861	**0.845**	0.744
MCP	0.872	0.870	0.880	0.842	0.739	0.624

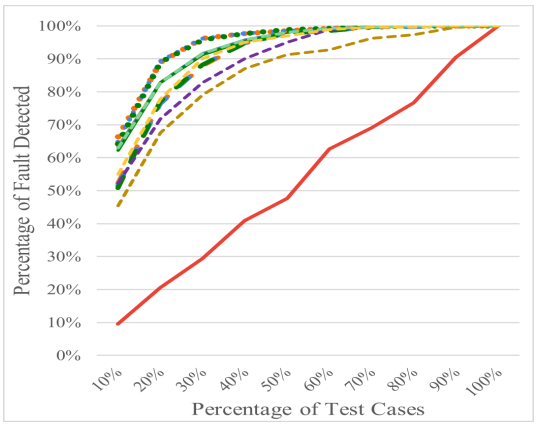
(a) MNIST

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	ResNet-32	VGG-19	ResNet-32	VGG-19	ResNet-32	VGG-19
Random	0.491	0.491	0.502	0.507	0.520	0.523
MUT - Entropy	0.812	0.783	**0.827**	0.799	**0.676**	**0.687**
MUT - Confidence	0.810	0.783	0.825	0.799	0.675	0.686
MUT - Margin	0.807	0.783	0.823	0.799	0.674	0.686
DE - Entropy	**0.821**	**0.794**	**0.830**	**0.802**	0.603	0.606
DE - Confidence	**0.823**	**0.792**	**0.827**	0.800	0.603	0.605
DE - Margin	**0.820**	**0.792**	0.818	0.798	0.602	0.605
VI - Entropy	0.775	0.766	0.776	0.752	0.602	0.577
VI - Confidence	0.763	0.766	0.770	0.751	0.599	0.577
VI - Margin	0.763	0.766	0.755	0.751	0.598	0.576
LSA	0.661	0.787	0.679	**0.804**	0.626	**0.707**
DSA	0.785	0.791	0.772	**0.807**	**0.677**	**0.703**
DeepGini	0.811	0.783	**0.828**	0.801	**0.676**	0.686
MCP	0.742	0.697	0.771	0.730	0.621	0.614

(b) CIFAR-10



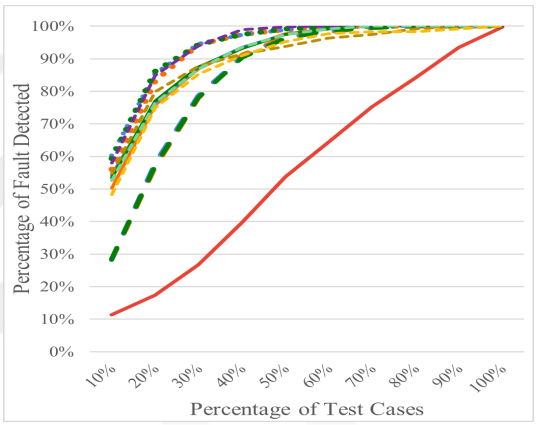
(a) LeNet-5 - Generative Models Test Dataset



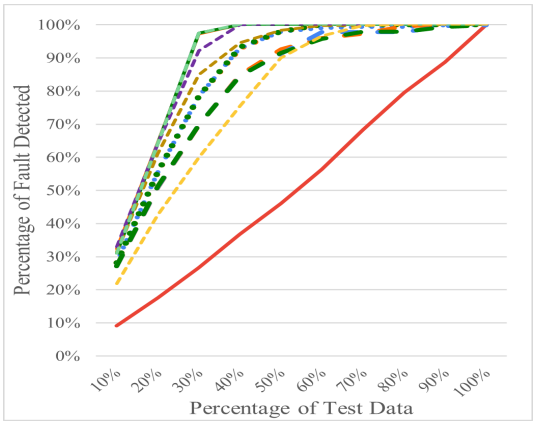
(b) LeNet-1 - Generative Models Test Dataset



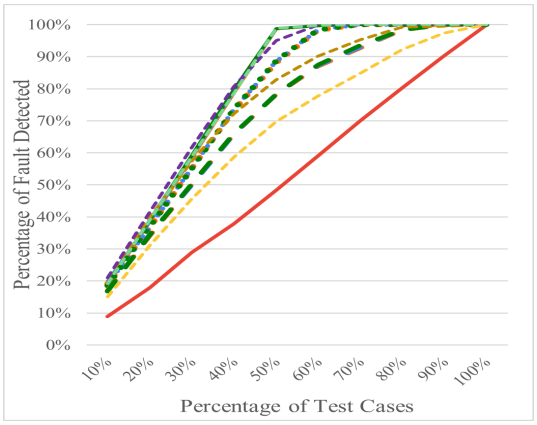
(c) LeNet-5 - Image Transformation Test Dataset



(d) LeNet-1 - Image Transformation Test Dataset



(e) LeNet-5 - Adversarial Attack Test Dataset



(f) LeNet-1 - Adversarial Attack Test Dataset

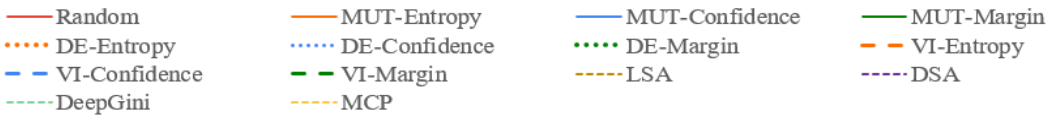
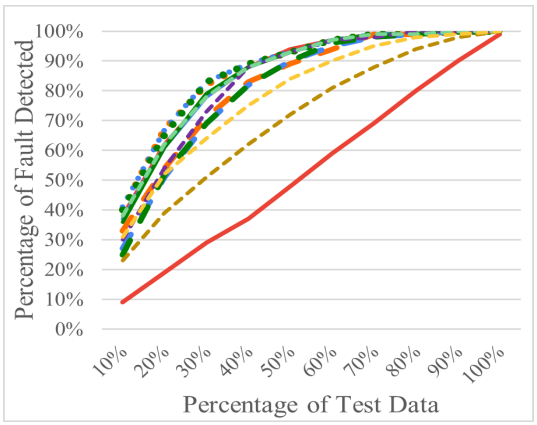


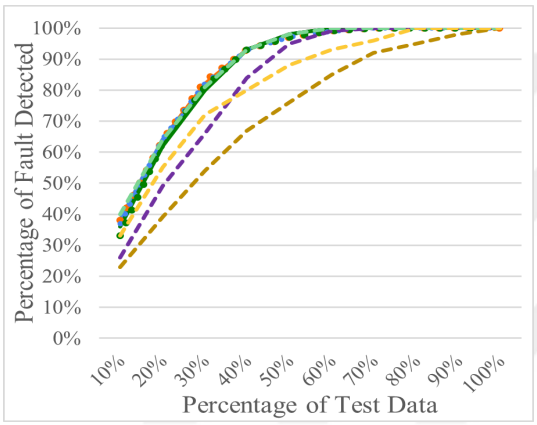
Figure 20: Experiment-1 - Fault Detection Ratios according to the size of Generated Test Datasets (MNIST) [132] ©2023 IEEE.



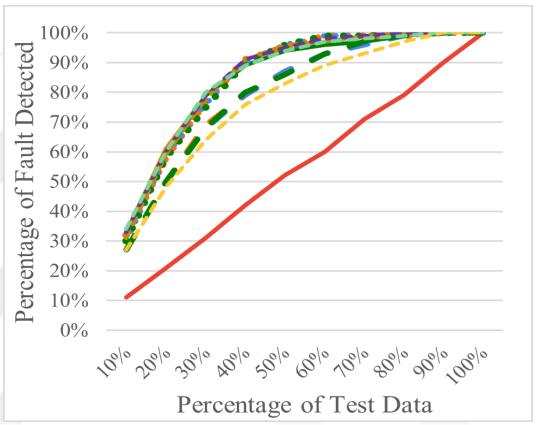
(a) ResNet32 - Generative Models Test Dataset



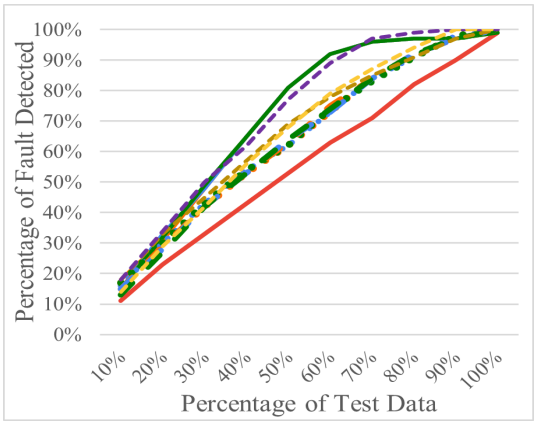
(b) VGG-19 - Generative Models Test Dataset



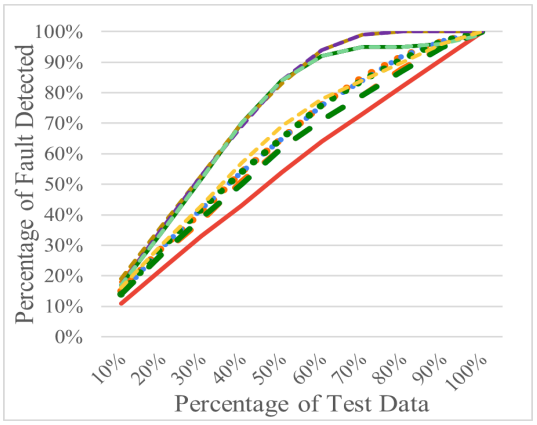
(c) ResNet32 - Image Transformation Test Dataset



(d) VGG-19 - Image Transformation Test Dataset



(e) ResNet32 - Adversarial Attack Test Dataset



(f) VGG-19 - Adversarial Attack Test Dataset



Figure 21: Experiment-1 - Fault Detection Ratios according to the size of Generated Test Datasets (CIFAR-10) [132] ©2023 IEEE.

Uncertainty metrics obtained through the VI-F method and MCP typically underperform relative to other methods. The difference in fault detection rates between prioritization strategies using VI and the nearest competing approaches ranges from 4.03% to 13.06% for MNIST test datasets. For MCP, the difference is between 2.28% and 18.54% on average.

LSA also showed relatively low fault detection rates compared to other methods, except for experiments using the VGG-19 model. In experiments conducted with the ResNet-32 model and datasets generated by the Generative Model and Image Transformation, LSA achieved the lowest fault detection ratios, with more than a 10% gap from the nearest competing approach.

Consequently, we evaluated the effectiveness of test data prioritization approaches using the Ratio of Area Under Curve (RAUC) metric defined in Section 4.2.3. For each test dataset generated, we computed the RAUC value for different sizes of prioritized test datasets: 100, 300, 500, and 1000, as well as for all test data. The RAUC values for generated test datasets of MNIST and CIFAR-10 are provided in Tables 12 and 13, with the top three values for each test dataset size highlighted in bold. Figure 22 displays the RAUC values for MNIST and CIFAR-10, grouped by generated test datasets and presented using boxplots.

Higher RAUC values indicate that the test data is prioritized closer to the ideal prioritization, which means giving more priority to the test data that will be misclassified by the tested model. The prioritization strategies that employ DE models achieve the best RAUC values for the test dataset generated by Generative Models and Image Transformations for MNIST. However, for CIFAR-10, the test prioritization approaches with the DE method achieve the best RAUC values for the Generative Models test dataset, whereas for the Image Transformations test dataset, the MUT-Entropy, MUT-Confidence, and DeepGini provide the best results. On the other hand, for test datasets generated by Adversarial Attacks, the distance-based methods (LSA/DSA) and MUT-Entropy outperform the other prioritization techniques in both MNIST and CIFAR-10.

Finally, we evaluated the effectiveness of test prioritization strategies by performing statistical correlation analyses between misclassifications and test data prioritization strategies. To achieve this, we first assigned a value of '1' to predictions made by the model that resulted in misclassification and a value of '0' to predictions that resulted in correct classification for a given test input. We then calculated the correlation between test data prioritization metric values and the corresponding classification for each test dataset. The correlation results for both the Point-Biserial and Spearman correlation analyses are listed in Tables 14 and 15.

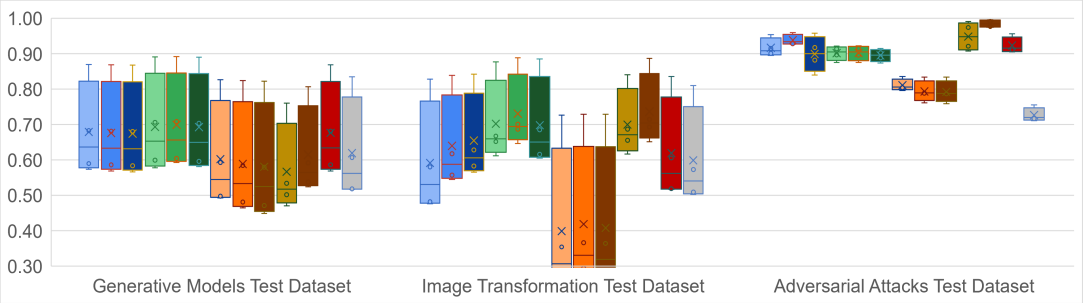
The correlation analysis results showed that there is a moderate correlation between test data prioritization metrics derived from the DE models and misclassifications in the Generative Models and Image Transformation test datasets in both MNIST and CIFAR-10. Additionally, for test datasets generated using Adversarial Attacks, the highest correlation coefficients for MNIST were observed for MUT-Margin, MUT-Confidence, and DeepGini. On the other hand, for CIFAR-10, MUT-Entropy, LSA, DSA, and DeepGini exhibited the highest correlation coefficients for the adversarial test dataset.

Table 12: Experiment-1 - Ratio of Area Under Curve values for Generated Test Datasets (MNIST) [132] ©2023 IEEE.

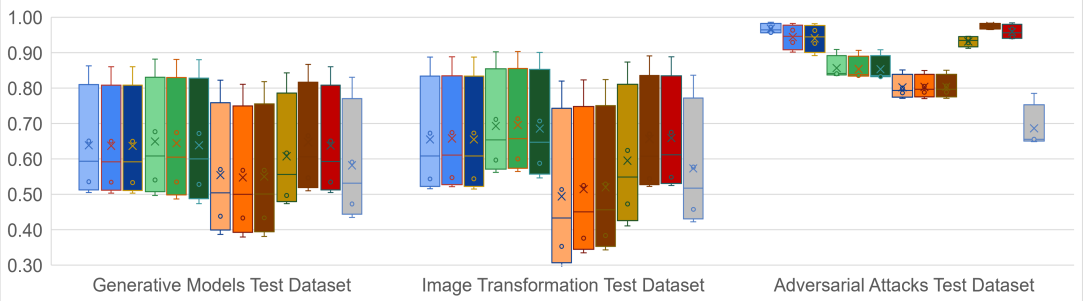
		LeNet-5					LeNet-1				
Test Data Generation Methods	Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
Generative Models	MUT - Entropy	**0.60**	**0.50**	**0.54**	0.65	0.86	0.61	0.57	0.59	0.68	0.87
	MUT - Confidence	**0.58**	**0.50**	0.53	0.65	0.86	0.59	0.57	0.59	0.68	0.87
	MUT - Margin	**0.58**	**0.50**	0.53	0.65	0.86	0.56	0.57	0.58	0.68	0.87
	DE - Entropy	0.53	**0.50**	**0.54**	**0.68**	**0.88**	0.59	**0.58**	**0.60**	**0.71**	**0.89**
	DE - Confidence	0.49	0.49	0.53	**0.67**	**0.88**	0.62	**0.59**	**0.60**	**0.71**	**0.89**
	DE - Margin	0.43	0.47	0.53	**0.67**	**0.88**	0.61	**0.58**	**0.60**	**0.70**	**0.89**
	VI-Entropy	0.39	0.39	0.44	0.57	0.82	0.53	0.49	0.50	0.59	0.83
	VI-Confidence	0.37	0.38	0.43	0.57	0.81	0.47	0.46	0.48	0.58	0.82
	VI-Margin	0.38	0.38	0.43	0.57	0.82	0.44	0.45	0.47	0.58	0.82
	LSA	0.54	0.47	0.50	0.62	0.84	**0.65**	0.50	0.47	0.53	0.76
	DSA	0.56	**0.51**	**0.55**	0.66	0.87	**0.63**	0.54	0.52	0.59	0.81
	DeepGini	**0.59**	**0.50**	0.53	0.65	0.86	0.59	0.57	0.59	0.68	0.87
	MCP	0.47	0.43	0.47	0.59	0.83	**0.63**	0.52	0.52	0.61	0.83
Image Transformations	MUT - Entropy	0.57	0.52	0.54	0.67	0.89	0.50	0.48	0.48	0.58	0.83
	MUT - Confidence	0.57	0.52	0.55	0.67	0.89	0.57	0.56	0.54	0.62	0.84
	MUT - Margin	0.53	0.51	0.54	0.67	0.89	0.58	0.58	0.56	0.63	0.84
	DE - Entropy	**0.59**	**0.56**	**0.60**	**0.71**	**0.90**	0.72	0.65	0.61	0.67	0.88
	DE - Confidence	**0.61**	**0.56**	**0.60**	**0.71**	**0.90**	**0.74**	**0.69**	**0.65**	**0.70**	**0.89**
	DE - Margin	0.57	**0.55**	**0.59**	**0.71**	**0.90**	0.59	0.62	0.60	**0.68**	**0.89**
	VI-Entropy	0.26	0.29	0.35	0.51	0.82	0.32	0.26	0.26	0.35	0.73
	VI-Confidence	0.36	0.33	0.38	0.52	0.82	0.34	0.29	0.28	0.37	0.73
	VI-Margin	0.35	0.34	0.38	0.53	0.82	0.26	0.26	0.27	0.36	0.73
	LSA	0.37	0.41	0.47	0.62	0.87	**0.84**	**0.69**	**0.62**	0.66	0.84
	DSA	**0.62**	0.52	0.54	0.67	0.89	**0.84**	**0.71**	**0.65**	**0.69**	**0.89**
	DeepGini	**0.59**	0.52	0.55	0.67	0.89	0.53	0.52	0.52	0.60	0.84
	MCP	0.43	0.42	0.46	0.58	0.84	0.50	0.51	0.50	0.57	0.81
Adversarial Attacks	MUT - Entropy	**0.94**	**0.96**	**0.96**	**0.97**	**0.99**	0.90	0.89	0.90	0.91	0.95
	MUT - Confidence	0.86	0.90	0.93	0.96	**0.98**	0.93	**0.94**	**0.93**	**0.93**	**0.96**
	MUT - Margin	0.76	0.89	0.93	0.96	**0.98**	0.69	0.84	0.88	0.92	**0.96**
	DE - Entropy	0.88	0.84	0.84	0.84	0.91	0.99	0.92	0.90	0.87	0.91
	DE - Confidence	0.86	0.84	0.83	0.84	0.91	**1.00**	0.92	0.90	0.88	0.91
	DE - Margin	0.85	0.83	0.83	0.84	0.91	0.96	0.90	0.89	0.87	0.91
	VI-Entropy	0.80	0.80	0.79	0.77	0.85	0.83	0.80	0.81	0.80	0.83
	VI-Confidence	0.83	0.80	0.79	0.77	0.85	0.73	0.76	0.79	0.79	0.83
	VI-Margin	0.84	0.81	0.79	0.77	0.85	0.71	0.76	0.79	0.79	0.83
	LSA	**0.95**	**0.94**	0.93	0.91	0.95	**1.00**	**0.99**	**0.97**	**0.92**	0.91
	DSA	**0.98**	**0.98**	**0.97**	**0.97**	**0.98**	**1.00**	**1.00**	**0.99**	**0.97**	**0.98**
	DeepGini	**0.94**	**0.94**	**0.94**	**0.97**	**0.98**	0.91	0.90	0.91	0.92	**0.96**
	MCP	0.63	0.66	0.65	0.65	0.78	0.68	0.71	0.72	0.72	0.76

Table 13: Experiment-1 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-10) [132] ©2023 IEEE.

		ResNet-32					VGG-19				
Test Data Generation Methods	Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
Generative Models	MUT - Entropy	**0.87**	**0.70**	0.62	0.66	0.81	**0.65**	**0.57**	**0.55**	**0.61**	0.77
	MUT- Confidence	0.81	0.65	0.60	0.65	0.80	0.62	**0.57**	0.54	**0.61**	0.77
	MUT- Margin	0.63	0.59	0.57	0.63	0.80	0.58	0.56	0.54	**0.61**	0.77
	DE - Entropy	0.79	**0.70**	**0.64**	**0.69**	**0.82**	0.60	**0.58**	**0.56**	**0.62**	**0.78**
	DE - Confidence	**0.87**	**0.72**	**0.66**	**0.70**	**0.83**	0.62	0.56	0.54	**0.61**	**0.78**
	DE - Margin	0.70	0.67	**0.63**	**0.68**	**0.82**	**0.65**	0.54	0.53	0.60	**0.78**
	VI - Entropy	0.56	0.52	0.49	0.55	0.74	0.63	0.55	0.52	0.57	0.74
	VI - Confidence	0.52	0.49	0.47	0.54	0.73	0.62	0.54	0.51	0.57	0.74
	VI - Margin	0.41	0.40	0.41	0.50	0.72	0.61	0.54	0.51	0.57	0.74
	LSA	0.31	0.34	0.34	0.40	0.59	0.54	0.52	0.51	0.59	0.77
	DSA	0.48	0.48	0.47	0.55	0.76	0.55	0.54	0.53	**0.61**	**0.78**
	DeepGini	**0.85**	0.69	0.62	0.65	0.80	**0.64**	**0.57**	**0.55**	**0.61**	0.77
	MCP	0.66	0.54	0.49	0.53	0.70	0.55	0.46	0.43	0.49	0.65
Image Transformations	MUT - Entropy	**0.80**	**0.78**	**0.75**	**0.70**	**0.86**	**0.74**	**0.73**	**0.70**	**0.65**	**0.82**
	MUT- Confidence	**0.81**	**0.77**	**0.74**	**0.69**	**0.85**	**0.73**	**0.71**	**0.68**	**0.64**	**0.82**
	MUT- Margin	0.66	0.69	0.68	0.66	0.84	0.67	0.69	0.67	**0.64**	**0.82**
	DE - Entropy	0.66	0.72	0.71	0.68	**0.85**	0.76	0.67	0.65	0.62	**0.82**
	DE - Confidence	0.75	0.73	0.70	0.67	**0.85**	0.70	0.66	0.63	0.61	**0.82**
	DE - Margin	0.65	0.64	0.63	0.63	0.83	0.59	0.61	0.60	0.60	0.81
	VI - Entropy	0.62	0.61	0.58	0.56	0.77	0.66	0.62	0.58	0.54	0.75
	VI - Confidence	0.61	0.59	0.55	0.51	0.74	0.65	0.59	0.56	0.54	0.74
	VI - Margin	0.43	0.44	0.44	0.46	0.73	0.59	0.57	0.55	0.53	0.74
	LSA	0.42	0.42	0.42	0.41	0.63	0.71	0.70	0.66	0.62	0.82
	DSA	0.56	0.51	0.50	0.49	0.75	0.73	0.68	0.66	**0.64**	0.83
	DeepGini	**0.80**	**0.78**	**0.75**	**0.70**	**0.85**	**0.73**	**0.73**	**0.69**	**0.65**	**0.82**
	MCP	0.70	0.67	0.64	0.60	0.77	0.58	0.57	0.55	0.51	0.71
Adversarial Attacks	MUT - Entropy	**0.91**	**0.84**	**0.80**	**0.76**	**0.83**	**0.93**	**0.90**	**0.88**	**0.87**	**0.89**
	MUT- Confidence	0.87	0.74	0.74	0.75	**0.82**	0.89	0.85	0.85	0.86	**0.89**
	MUT- Margin	0.49	0.64	0.70	0.74	**0.82**	0.70	0.81	0.83	0.85	**0.89**
	DE - Entropy	0.83	0.79	0.75	0.69	0.71	0.80	0.74	0.73	0.70	0.74
	DE - Confidence	0.90	0.80	0.76	0.69	0.71	0.81	0.73	0.72	0.70	0.74
	DE - Margin	0.78	0.77	0.74	0.68	0.71	0.80	0.72	0.72	0.69	0.74
	VI - Entropy	0.70	0.70	0.69	0.66	0.71	0.76	0.71	0.68	0.66	0.70
	VI - Confidence	0.66	0.69	0.67	0.65	0.70	0.80	0.71	0.68	0.66	0.70
	VI - Margin	0.66	0.64	0.64	0.64	0.70	0.79	0.71	0.68	0.65	0.70
	LSA	**0.89**	**0.84**	**0.80**	0.73	0.76	**0.97**	**0.95**	**0.93**	**0.89**	**0.91**
	DSA	**0.89**	**0.83**	**0.81**	**0.77**	**0.82**	0.84	0.86	**0.87**	**0.87**	**0.90**
	DeepGini	**0.90**	**0.83**	0.78	**0.76**	**0.83**	**0.92**	**0.88**	**0.87**	**0.86**	**0.89**
	MCP	0.74	0.68	0.67	0.66	0.73	0.69	0.76	0.76	0.73	0.76



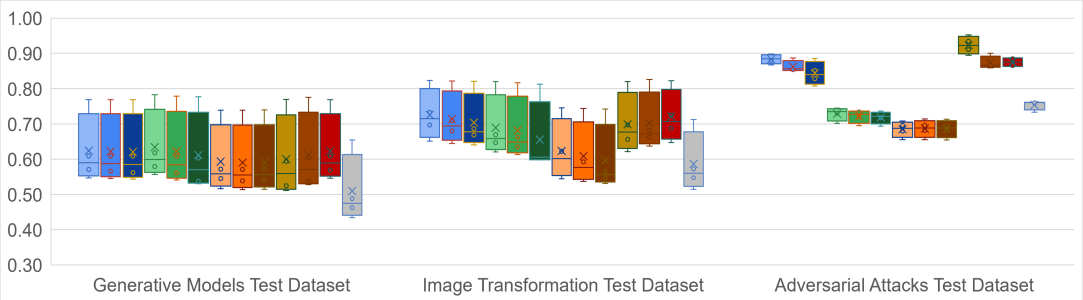
(a) LeNet-1



(b) LeNet-5



(c) ResNet



(d) VGG-19



Figure 22: Experiment-1 - RAUC Values of Test Prioritization Strategies for Generated Test Datasets without OOD Data.

Table 14: Experiment-1 - Correlation Between Test Prioritization Strategies and Misclassifications (MNIST) [132] ©2023 IEEE.

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.43	0.35	0.49	0.35	0.96	0.78
MUT - Confidence	0.40	0.38	0.47	0.37	**0.96**	**0.84**
MUT - Margin	0.40	0.37	0.46	0.37	**0.97**	**0.83**
DE - Entropy	**0.49**	**0.36**	**0.55**	**0.36**	0.41	0.69
DE - Confidence	**0.47**	**0.38**	**0.55**	**0.37**	0.34	0.72
DE - Margin	**0.47**	**0.37**	**0.55**	**0.37**	0.35	0.71
VI - Entropy	0.33	0.33	0.34	0.32	0.29	0.61
VI - Confidence	0.29	0.36	0.33	0.34	0.24	0.62
VI - Margin	0.28	0.34	0.34	0.33	0.24	0.62
LSA	0.41	0.34	0.31	0.35	0.64	0.73
DSA	0.46	0.35	0.46	0.36	0.83	0.77
DeepGini	0.41	0.38	0.49	0.37	**0.97**	**0.84**
MCP	0.37	0.35	0.42	0.35	0.96	0.77

(a) MNIST - LeNet-5

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.55	0.38	0.49	0.40	0.85	0.80
MUT - Confidence	0.53	0.38	0.52	0.40	**0.92**	**0.80**
MUT - Margin	0.53	0.38	0.53	0.40	**0.92**	**0.80**
DE - Entropy	**0.57**	**0.39**	**0.57**	**0.43**	0.62	0.73
DE - Confidence	**0.57**	**0.39**	**0.60**	**0.44**	0.55	0.73
DE - Margin	**0.56**	**0.39**	**0.59**	**0.43**	0.57	0.73
VI - Entropy	0.46	0.36	0.32	0.33	0.36	0.58
VI - Confidence	0.42	0.36	0.30	0.34	0.29	0.58
VI - Margin	0.41	0.35	0.30	0.33	0.28	0.58
LSA	0.36	0.31	0.37	0.40	0.38	0.69
DSA	0.40	0.34	0.54	**0.44**	0.80	0.83
DeepGini	0.55	0.38	0.51	0.40	**0.90**	**0.80**
MCP	0.49	0.38	0.51	0.41	0.89	0.81

(b) MNIST - LeNet-1

Table 15: Experiment-1 - Correlation Between Test Prioritization Strategies and Misclassifications (CIFAR-10) [132] ©2023 IEEE.

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.53	0.46	**0.61**	**0.55**	**0.53**	**0.56**
MUT - Confidence	0.50	0.46	0.59	0.54	0.46	0.56
MUT - Margin	0.48	0.46	0.58	0.53	0.46	0.56
DE - Entropy	**0.56**	**0.48**	**0.60**	**0.54**	0.30	0.33
DE - Confidence	**0.57**	**0.48**	**0.60**	**0.54**	0.29	0.33
DE - Margin	**0.56**	**0.48**	0.58	0.53	0.28	0.33
VI - Entropy	0.43	0.41	0.48	0.46	0.27	0.33
VI - Confidence	0.37	0.39	0.44	0.45	0.23	0.32
VI - Margin	0.36	0.39	0.39	0.42	0.23	0.31
LSA	0.23	0.24	0.28	0.30	0.37	0.40
DSA	0.43	0.42	0.44	0.45	**0.56**	**0.57**
DeepGini	0.51	0.46	**0.60**	**0.54**	**0.52**	**0.57**
MCP	0.43	0.45	0.53	0.53	0.38	0.56

(a) CIFAR-10 - ResNet32

	Generative Models		Image Transformation		Adversarial Attacks	
Test Prioritization Strategy	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.36	0.43	0.46	0.51	**0.51**	**0.64**
MUT - Confidence	0.31	0.43	0.41	0.51	0.44	0.64
MUT - Margin	0.30	0.43	0.40	0.51	0.44	0.64
DE - Entropy	**0.47**	**0.44**	**0.55**	**0.52**	0.26	0.36
DE - Confidence	**0.45**	**0.44**	**0.53**	**0.52**	0.24	0.36
DE - Margin	0.44	0.44	0.53	0.51	0.24	0.36
VI - Entropy	0.30	0.40	0.33	0.43	0.13	0.26
VI - Confidence	0.27	0.40	0.29	0.43	0.13	0.26
VI - Margin	0.28	0.40	0.29	0.43	0.13	0.26
LSA	0.43	0.43	**0.53**	**0.52**	**0.61**	**0.71**
DSA	**0.46**	**0.44**	**0.56**	**0.52**	**0.67**	**0.70**
DeepGini	0.34	0.43	0.44	0.51	0.49	0.64
MCP	0.26	0.43	0.36	0.51	0.36	0.64

(b) CIFAR-10 - VGG-19

Results for Test Datasets with OOD Data

In order to evaluate the effectiveness of test data prioritization strategies with OOD data, we repeated the experiments, this time employing test datasets including OOD data. We merged all the generated test data types into a single dataset for each of MNIST and CIFAR-10 and used these combined test datasets in the experiments.

Since the models were exposed to test data that have a distribution different than the data they were trained on, the accuracy of the MUTs for test datasets containing OOD data decreased, as shown in Table 16. For the ResNet-32 model, the accuracy decreased from 65.5% to 49.3%, while for LeNet-5, it decreased from 84.1% to 78.4%.

Table 16: Experiment-1 - Accuracy Values for Generated Test Dataset including OOD Data [132] ©2023 IEEE.

	LeNet-5	LeNet-1	ResNet-32	VGG-19
Accuracy with OOD Data(%)	78.4	70.2	49.3	49.2
Accuracy without OOD Data(%)	84.1	76.3	65.5	63.9

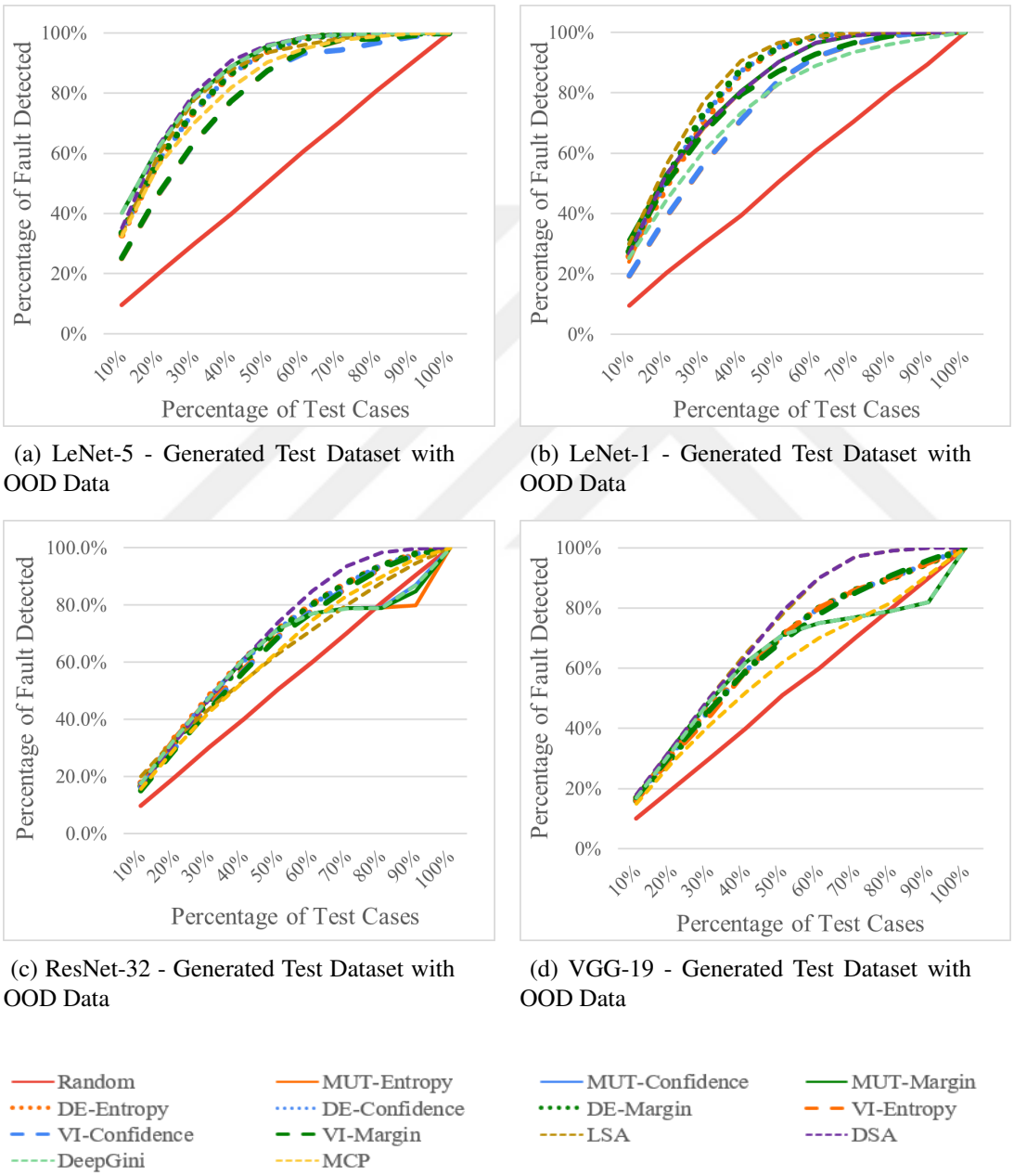
The APFD values from the experiments conducted using test datasets with OOD data are presented in Table 17, where the top three values for each MUT are represented in bold.

Table 17: Experiment-1 - Average Percentage of Fault Detected values for Generated Test Dataset including OOD Data [132] ©2023 IEEE.

Test Prioritization Strategy	LeNet-5	LeNet-1	ResNet-32	VGG-19
Random	0.501	0.500	0.503	0.502
MUT - Entropy	0.815	0.762	0.598	0.596
MUT - Confidence	**0.817**	0.770	0.602	0.595
MUT - Margin	**0.817**	0.772	0.598	0.595
DE - Entropy	0.794	0.777	**0.645**	**0.628**
DE - Confidence	0.794	**0.784**	**0.642**	0.626
DE - Margin	0.796	**0.787**	0.639	0.626
VI - Entropy	0.742	0.706	0.627	0.626
VI - Confidence	0.735	0.708	0.622	0.625
VI - Margin	0.742	0.709	0.619	0.625
LSA	0.798	0.753	0.598	**0.682**
DSA	0.815	**0.802**	**0.659**	**0.682**
DeepGini	**0.816**	0.767	0.603	0.596
MCP	0.774	0.716	0.600	0.568

The DSA method exhibited the highest APFD values in experiments conducted using Lenet-1, ResNet-32, and VGG-19 as the MUT. Moreover, 5/12 top APFD values belong to DE test data prioritization strategies, while the best APFD value (0.817) for LeNet-5 was obtained by MUT-Confidence and MUT-Margin methods.

Additionally, the fault detection ratios with test datasets that include OOD are shown in Figure 23.



(a) LeNet-5 - Generated Test Dataset with OOD Data

(b) LeNet-1 - Generated Test Dataset with OOD Data

(c) ResNet-32 - Generated Test Dataset with OOD Data

(d) VGG-19 - Generated Test Dataset with OOD Data

Figure 23: Experiment-1 - Fault Detection Ratios according to the size of Generated Test Dataset including OOD Data [132] ©2023 IEEE.

The performance of uncertainty metrics is impacted by the presence of OOD data in datasets. As compared to datasets where OOD data are removed, the fault detection rate of uncertainty metrics decreases. For the CIFAR-10 dataset, this decrease is particularly significant, as the performance of uncertainty metrics deteriorates to a level that is close to random sampling. For instance, using 40% of the test data prioritized through DE-Entropy, we were able to identify 89%, 93%, and 53% of the misclassified data in the Image Transformation, Generative Models, and Adversarial Attacks test datasets of CIFAR-10, respectively, Figure 21. For the combined OOD dataset of CIFAR-10, this ratio is 61% with using 40% of prioritized data.

7.1.2 Experiment-2 Results

In this experiment, we explored the application of post-hoc visual explainability methods to unravel the reasons behind the misclassifications of tested models. We focused on the test results from Experiment-1, which employed the ResNet-32 model, using generated test datasets without OOD data for CIFAR-10. We identified the predicted versus actual classes for misclassified test data within each generated test dataset. Confusion matrices illustrating the predicted and correct class labels are presented in Figures 24-26.

Our analysis revealed that the classes most confused by the model in the Generative Model test dataset were the test data where the actual class was Cat, but the model predicted Dog and vice versa, with 27 and 20 misclassifications, respectively. Similarly, in the Image Transformation test dataset, the most frequent misclassifications occurred when data belonging to the Dog class were incorrectly predicted as Cat, with 45 misclassifications. Moreover, test images of the Deer class were frequently misclassified as Bird or Frog, while those of the Airplane class were incorrectly classified as Automobile or Ship within the Image Transformation test dataset.

Adversarial attacks refer to artificial perturbations that are created to deceive machine learning models. As a result, there is a high number of misclassified test images across all classes, ranging from 64 to 108, as demonstrated in Figure 26. The bird class has the least number of misclassifications, which is 64, while the cat class has the highest number of misclassifications, which is 108. The Generative Model test dataset has the least number of misclassifications when compared to the other two test datasets. The highest number of misclassifications in the Generative Models test dataset belongs to the Cat class, with a total of 51 misclassifications.

Predicted Class

Actual Class	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	0	2	4	0	1	0	0	0	5	2
automobile	2	0	1	1	0	0	1	0	1	11
bird	9	0	0	10	11	6	5	2	1	2
cat	0	0	6	0	7	27	8	3	0	0
deer	0	0	6	8	0	6	6	4	0	0
dog	0	0	1	20	7	0	4	5	0	2
frog	1	0	4	0	5	1	0	1	0	0
horse	2	1	0	7	15	4	0	0	0	2
ship	13	1	1	0	0	1	2	1	0	2
truck	7	13	1	2	0	0	3	0	6	0

Figure 24: Experiment-2 - Total Number of Misclassified Test Data According to Classes (Generative Models Test Dataset) [132] ©2023 IEEE.

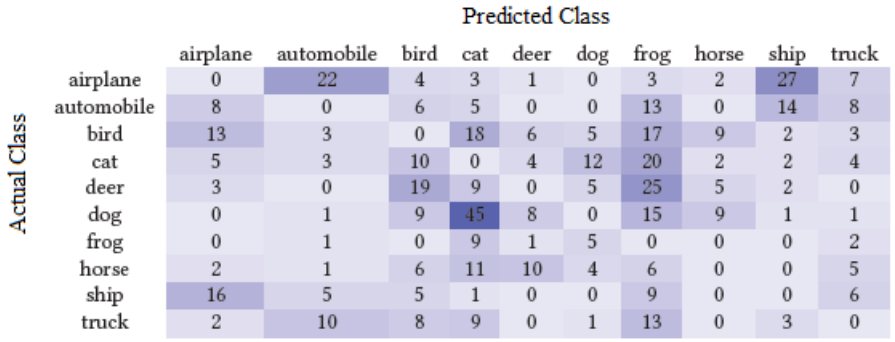


Figure 25: Experiment-2 - Total Number of Misclassified Test Data According to Classes (Image Transformation Test Dataset) [132] ©2023 IEEE.

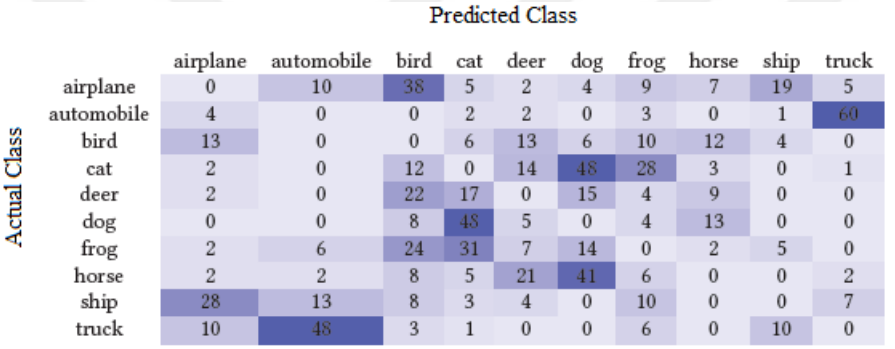


Figure 26: Experiment-2 - Total Number of Misclassified Test Data According to Classes (Adversarial Attacks Test Dataset) [132] ©2023 IEEE.

Subsequently, we closely analyzed the classification errors frequently occurring between the Dog and Cat classes. Our first step was to apply visualization techniques on correctly classified images of Dogs and Cats from the original test dataset. This was done to pinpoint the image regions that significantly influence the model's predictions for correct classification cases. Figures 27 and 28 illustrate the results of these analyses using Grad-CAM, Grad-CAM++, and ScoreCAM techniques.

The analysis uncovered that, for the Dog class, the model's decisions are predominantly affected by the face of the dogs, particularly the nose region. For the Cat class, similarly, the model focuses mainly on the face, though not specifically the nose. Additionally, the body of the cats influenced the model's decision-making for some of the images.

Next, as part of our analysis, we applied visualization techniques to a set of images that were sampled from the incorrectly classified images. These images were part of the Image Transformation test dataset and were generated using the Rotate method. Despite belonging to the Dog class, the ResNet-32 model misclassified these images as Cat. To identify the regions that were influencing the model's decision, we applied visualization methods to highlight the areas of interest.

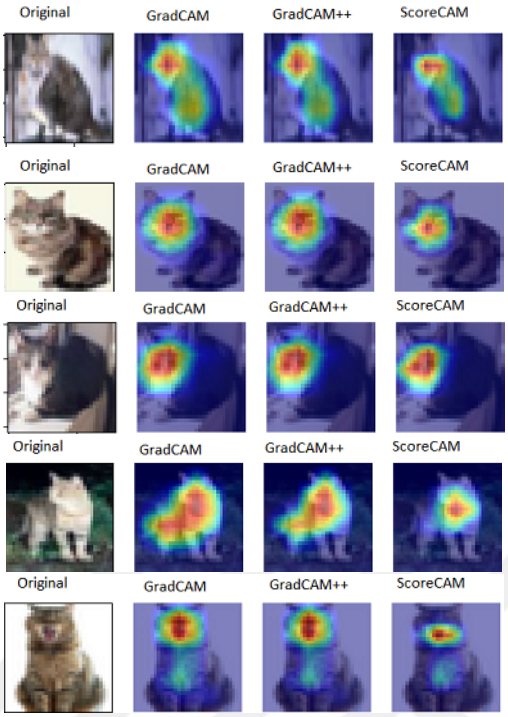


Figure 27: Experiment-2 - Correctly classified Cat images from the original CIFAR-10 test dataset [132] ©2023 IEEE.

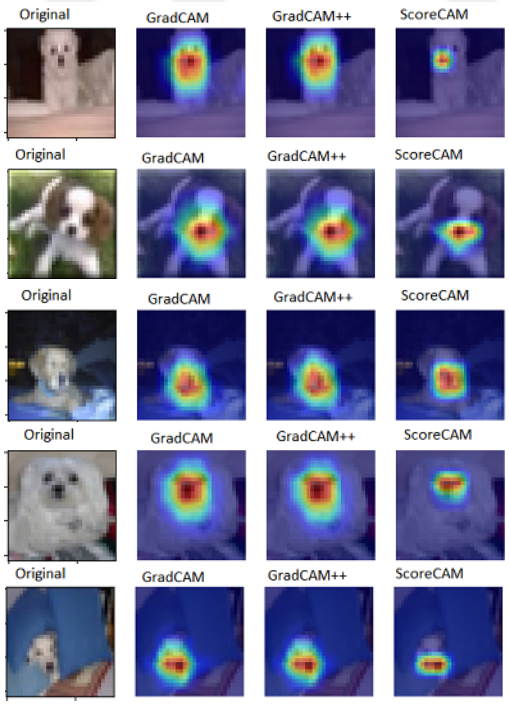


Figure 28: Experiment-2 - Correctly classified Dog images from the original CIFAR-10 test dataset [132] ©2023 IEEE.

In Figure 29, we present the highlighted images, which are organized into four rows. In the first and second rows, the visualization methods highlighted the dog's body as the primary region influencing the model's decision. However, in the case of correctly classified Dog images, the face and especially the nose region of the dog images were the regions that mainly affected the decision of the model. In the third row, we observe that the highlighted region includes the face of the dog, but it does not include the dog's nose. In the fourth row, the highlighted areas include both the dog's ears and face.

In reference to the images shown in Figure 29, we hypothesize that the model struggles to accurately classify a dog image when the nose region is not recognized or when there are additional patterns associated with other classes present. Supporting this idea, we can see in the last row of images in Figure 28 that the model correctly identifies the dog in the image by recognizing its nose, even if the body of the dog is not visible.

In order to address this misclassification issue, it can be recommended to retrain the tested DNN model using a larger dataset that includes images of dogs with non-black noses or images where the nose is not visible.

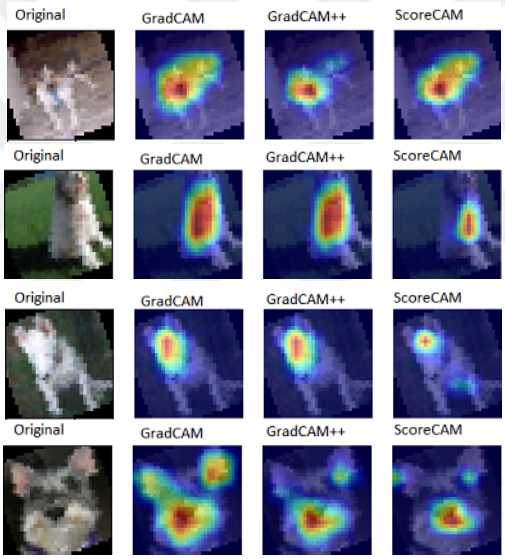


Figure 29: Experiment-2 - Dog images misclassified as Cat in CIFAR-10 Image Transformation Test Dataset [132] ©2023 IEEE.

7.2 Results for Alternative Test Data Prioritization Approaches

In this section, we evaluated the results of test data prioritization performed using Mahalonobis Uncertainty Score and Deep Ensemble Variation Score metrics. These experiments were conducted using the generated CIFAR-10 test datasets that do not include OOD data and employing the ResNet-32 model as the Model Under Test (MUT). We compared the prioritization of test data using MUS and DE Variation Score with test data prioritization approaches that employ MUT and DE models with uncertainty metrics, as used in Experiment 1. The best three values for each evaluation are shown in bold.

7.2.1 Experiment-3 Results

In this subsection, we discuss the results of test data prioritization using the MUS metric by employing MUT and DE models separately. The results obtained using MUT are denoted as MUT-MUS, while those using DE models are referred to as DE-MUS.

The evaluation was conducted using the Average Percentage of Faults Detected (APFD) metric and by calculating the correlation between mispredictions and prioritizations. Table 18 displays the APFD values, while the correlation analysis results are given in Table 19.

Table 18: Experiment-3 - Average Percentage of Fault Detected (APFD) values for test data prioritization with Mahalonobis Uncertainty Scores.

Test Prioritization Strategy	Generative Models Test Dataset	Image Transformation Test Dataset	Adversarial Attacks Test Dataset
MUT - Entropy	0.812	**0.827**	**0.676**
MUT - Confidence	0.810	0.825	**0.675**
MUT - Margin	0.808	0.823	**0.674**
MUT - MUS	0.805	0.825	0.668
DE - Entropy	**0.821**	**0.830**	0.603
DE - Confidence	**0.823**	**0.827**	0.603
DE - Margin	0.820	0.818	0.603
DE - MUS	**0.823**	0.824	0.596

Table 19: Experiment-3 - Correlation Between Mahalonobis Uncertainty Scores and Misclassifications (CIFAR-10).

	Generative Models Test Dataset		Image Transformation Test Dataset		Adversarial Attacks Test Dataset	
Test Prioritization Strategy	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.53	0.46	0.61	0.55	0.53	0.56
MUT - Confidence	0.50	0.46	0.59	0.54	0.46	0.56
MUT - Margin	0.48	0.46	0.58	0.53	0.46	0.56
MUT - MUS	0.39	0.45	0.45	0.54	0.16	0.54
DE - Entropy	0.56	0.48	0.60	0.54	0.30	0.33
DE - Confidence	0.57	0.48	0.60	0.54	0.29	0.33
DE - Margin	0.56	0.48	0.58	0.53	0.28	0.33
DE - MUS	0.44	0.48	0.49	0.53	0.33	0.31

The results indicate that the DE-MUS method achieved the highest APFD value of 0.823 for the Generative Model test dataset. Furthermore, test data prioritization using both DE-MUS and MUT-MUS produced close APFD values to the best APFD result, differing by less than 0.006 in the Image Transformation test dataset. However, for adversarial attack test datasets, prioritizing test data with the DE-MUS method produced a low APFD value of 0.596, as with other metrics used with the DE method. Nevertheless, using the MUT-MUS method resulted in an APFD value of only 0.006, lower than the best APFD value, which belongs to the MUT-Entropy for the adversarial attack test dataset.

The Spearman and Biserial correlation analysis results given in Table 19 follow a similar trend to the APFD values when comparing the MUS metric to previously studied metrics. The Pearson correlation values for the metrics gathered using MUT differ by less than 0.02 for each test dataset. This close similarity is also observed in the test data prioritization approaches employing DE models.

7.2.2 Experiment-4 Results

In this subsection, the results of test data prioritization using the DE Variation Score are presented. Table 20 shows the APFD values, and Table 21 provides the correlation analysis results between the misclassifications and the DE Variation metric.

While the APFD values of the prioritization performed with DE Variation Score in the Generative Models and Image Transformation test data sets are not among the top three, the corresponding APFD value in the Adversarial Attack test dataset has the highest value with 0.742, exceeding the nearest value by 0.066. These findings indicate that the DE Variation Score is the most effective metric for prioritizing test data generated through Adversarial Attacks, outperforming all other metrics. This conclusion is supported by strong correlation values, with a Spearman Correlation of 0.83 and a Biserial Correlation of 0.84.

Table 20: Experiment-4 - Average Percentage of Fault Detected (APFD) values for test data prioritization with DE Variation Scores.

Test Prioritization Strategy	Generative Models Test Dataset	Image Transformation Test Dataset	Adversarial Attacks Test Dataset
MUT - Entropy	0.812	**0.827**	**0.676**
MUT - Confidence	0.810	0.825	**0.675**
MUT - Margin	0.808	0.823	0.674
DE - Entropy	**0.821**	**0.830**	0.603
DE - Confidence	**0.823**	**0.827**	0.603
DE - Margin	**0.820**	0.818	0.603
DE - Variation Score	0.806	0.823	**0.742**

Table 21: Experiment-4 - Correlation Between DE Variation Scores and Misclassifications (CIFAR-10).

Test Prioritization Strategy	Generative Models Test Dataset		Image Transformation Test Dataset		Adversarial Attacks Test Dataset	
	Biserial	Pearson	Biserial	Pearson	Biserial	Pearson
MUT - Entropy	0.53	0.46	0.61	0.55	0.53	0.56
MUT - Confidence	0.50	0.46	0.59	0.54	0.46	0.56
MUT - Margin	0.48	0.46	0.58	0.53	0.46	0.56
DE - Entropy	0.56	0.48	0.60	0.54	0.30	0.33
DE - Confidence	**0.57**	0.48	0.60	0.54	0.29	0.33
DE - Margin	0.56	0.48	0.58	0.53	0.28	0.33
DE - Variation Score	0.56	**0.55**	**0.64**	**0.61**	**0.84**	**0.83**

7.3 Results for Test Selection with Meta-Models

We evaluated the effectiveness of three meta-models given in Table 8 in test data prioritization by using the generated test datasets given in Table 7. We performed this assessment with two categories of test datasets: 1) test datasets that contain only in-distribution data and 2) test datasets that contain both in-distribution and OOD data. For this purpose, we performed OOD data detection on generated test datasets by employing the fine-tuned Vision Transformer(ViT) Models.

First, we fine-tuned ViT models for each of MNIST, CIFAR-10, and CIFAR-100, and then calculated OOD scores using these models on selected near and far OOD datasets. Figures 30-32 illustrate the OOD score values computed for these selected OOD datasets along with MNIST, CIFAR-10, and CIFAR-100 datasets. Using these OOD scores, we determined threshold values as detailed in Section 6.5.2. Table 22 shows the F1-Score values for each chosen threshold value, as well as the AUROC values for OOD detection with the selected OOD datasets. Subsequently, we calculated the OOD scores of each generated test image and categorized them either in-distribution or OOD, comparing their scores with a threshold value. Figure 33 shows the percentage of OOD data in each dataset.

Table 22: Performance results of OOD Detection performed using Visual Transformer (ViT) models with selected OOD datasets [120] ©2024 ACM.

In-distribution Dataset	**OOD Dataset**	**AUROC**	**F1-Score**
CIFAR-10	CIFAR-100 (Near OOD)	98.88	0.95
CIFAR-10	SVHN (Far OOD)	99.87	0.98
MNIST	K-MNIST (Near OOD)	99.73	0.97
MNIST	Fashion MNIST (Far OOD)	99.99	0.98
CIFAR-100	CIFAR-10 (Near OOD)	93.82	0.85
CIFAR-100	SVHN (Far OOD)	93.24	0.86

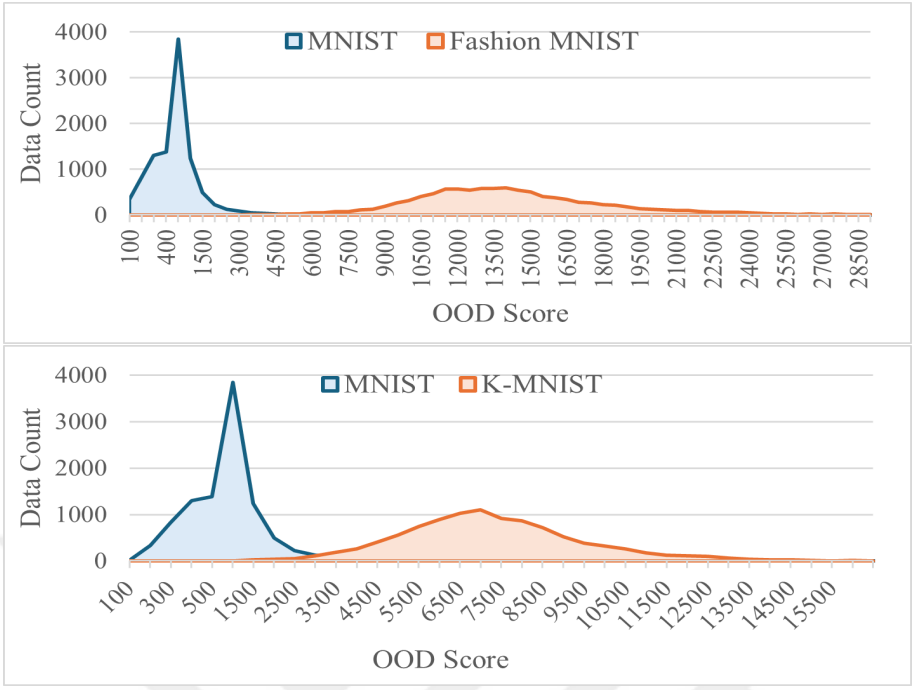


Figure 30: OOD Scores of MNIST vs Selected OOD Datasets (Fashion MNIST/K-MNIST).

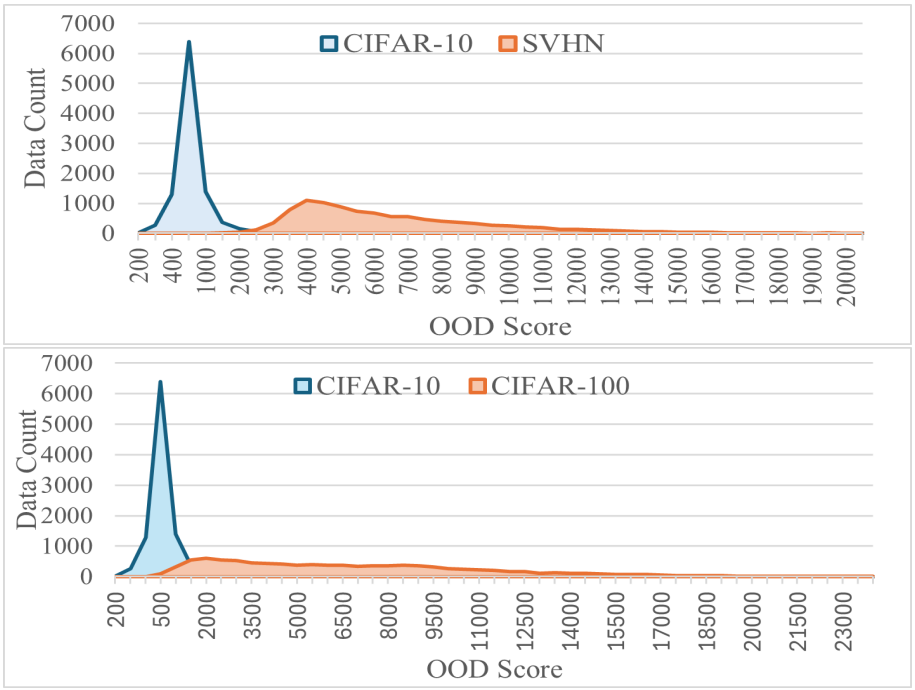


Figure 31: OOD Scores of CIFAR-10 vs Selected OOD Datasets(SVHN/CIFAR-100).

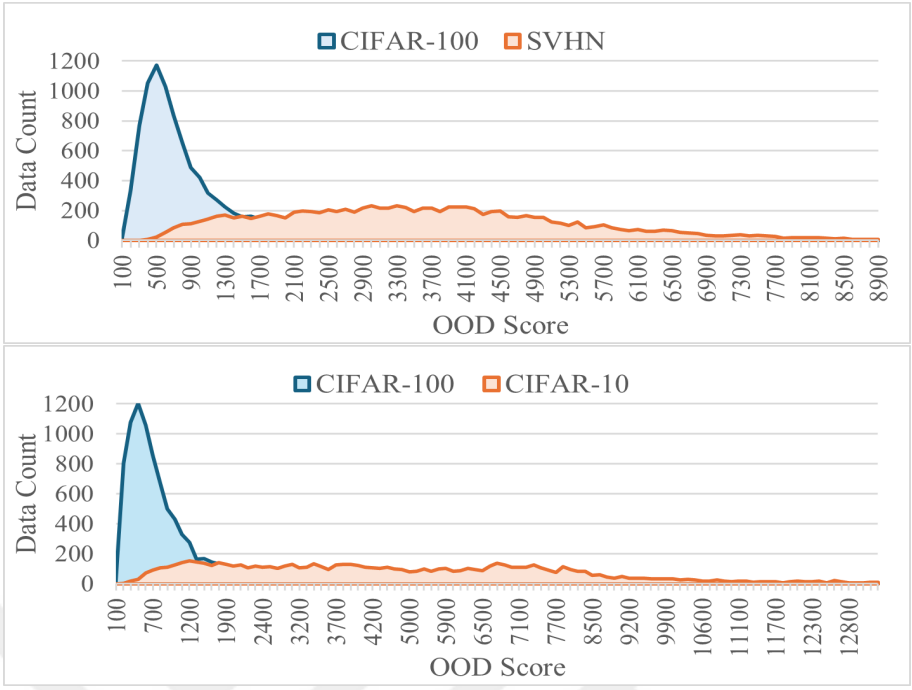


Figure 32: OOD Scores of CIFAR-100 vs Selected OOD Datasets (SVHN/CIFAR-10).

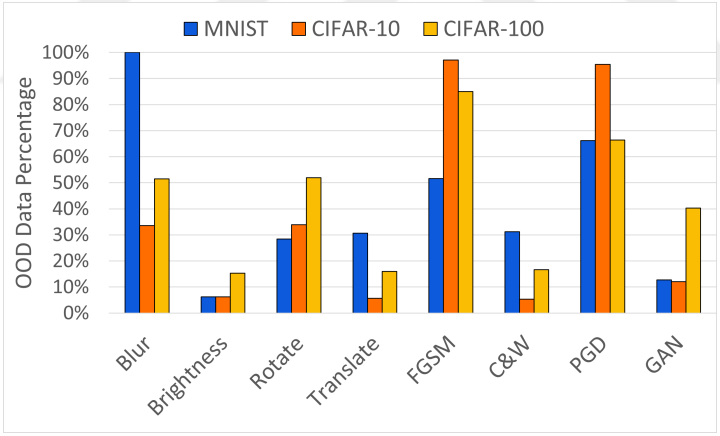


Figure 33: Percentages of OOD Data in Generated Test Datasets using OOD detection with ViT Models [120] ©2024 ACM.

Almost all of the images produced by the FGSM and PGD adversarial attack techniques for CIFAR-10 are identified as OOD data, with percentages of 97% and 95%, respectively. Moreover, the majority of images produced by the C&W attack method are identified as in-distribution data, which is likely attributable to the smaller perturbations caused by this method compared to those produced by the FGSM and PGD methods.

Meanwhile, the data generated with Blur and Rotate image transformations have the highest two percentages of the OOD data in the Image Transformation test dataset.

Synthetic images that were generated by GANs and identified as OOD correspond to a small percentage of data, 12% and 13% for the CIFAR-10 and MNIST datasets, respectively. On the other hand, a higher percentage, 40%, of the CIFAR-100 test data generated by GANs are identified as OOD.

7.3.1 Experiment-5 Results

In this experiment, we prioritized the test data by using three different meta-models separately. We then compared the effectiveness of these meta-models with test data prioritization methods that use uncertainty metrics obtained employing the MUT and DE models, which were also used in Experiment 1. Additionally, we compared this prioritization approach with the same state-of-the-art methods used for comparison in Experiment-1, namely LSA, DSA, DeepGini, and MCP.

LSA calculation relies on the popular SciPy computation package, as mentioned by Kim et al. [140]. The "gaussian_kde" method, used for kernel density estimation, failed with the test datasets generated for CIFAR-100 in our experiments. Consequently, we did not incorporate LSA prioritization in the evaluation of prioritization techniques with CIFAR-100.

Meta-Model Training Results

We trained the meta-models using half of the original test datasets and used the remaining half to generate new test data for evaluating the test data prioritization approaches. This approach mitigates data leakage, which occurs when a model is evaluated using data from its training set.

The Model Under Test (MUT) and Deep Ensemble (DE) models serve as base models for the meta-models, with their outputs used to compute uncertainty metrics. These metrics are then fed as inputs to the meta-models. Consequently, each of the three meta-models is trained separately for each MUT model and its corresponding DE models. During the meta-model training, we standardized the input features. Standardizing values of features ensures that variables with varying scales have an equal impact on the logistic regression model, preventing dominance by those with larger scales. This results in more stable and efficient training and easier interpretation of the coefficients. The resultant coefficient values for input features of the logistic regression models are given in Table 23.

Table 23: Coefficient Values of Input Features in Logistic Regression Models used as Meta-Models.

	LeNet-5	LeNet-1	ResNet-32	VGG-19	ResNet 28-10
Meta-Model-1					
Const Value	-6.15	-5.44	-3.24	-3.04	-1.79
Least Confidence	0.33	0.40	0.58	0.13	0.65
DE Variation Score	0.57	0.89	0.93	1.23	1.06
Meta-Model-2					
Const Value	-6.15	-5.44	-3.25	-3.04	-1.81
Margin	-0.32	-0.40	-0.57	-0.12	-0.68
DE Variation Score	0.57	0.89	0.93	1.23	1.05
Meta-Model-3					
Const Value	-6.17	-5.57	-3.33	-3.04	-1.78
Entropy	0.37	0.52	0.72	0.19	0.59
DE Variation Score	0.52	0.83	0.87	1.19	1.11

The DE Variation Score, Least Confidence, and Entropy have positive coefficients, while Margin has a negative coefficient. The meta-model uses these uncertainty metrics to predict whether the MUT will make a correct prediction. Metrics indicating higher uncertainty with higher values have positive coefficients, whereas those indicating higher uncertainty with lower values, like Margin, have negative coefficients.

The DE Variation Score consistently has a higher coefficient value compared to other input features across all three meta-models. For all MUTs except VGG-19, the ratio of the DE Variation Score to other features ranges from 1.2 to 2.2. However, for VGG-19, this ratio is significantly higher, ranging from 6.2 to 10.3, indicating that the DE Variation Score has a substantially greater impact on the output.

Results for Test Datasets without OOD Data

We evaluated the effectiveness of test prioritization with meta-models by calculating the APFD values for each generated test dataset. Tables 24-26 show the results of this evaluation. We used the Meta-Model-* convention to refer to the test data prioritization performed using the meta-models listed in Table 8. The abbreviations used for the compared prioritization techniques are as follows: MUT-* represents prioritization based on uncertainty metrics gathered using the model under test, and DE-* signifies prioritization using uncertainty metrics derived from the outputs of deep ensemble models. The top three APFD values for each test dataset are highlighted in bold. The APFD values, which are closer to 1.0, indicate better prioritization of test data, which leads to identifying the test data that causes incorrect predictions in the MUT.

The results presented in Tables 24-26 demonstrate that our meta-model based approach outperforms other prioritization techniques across all test datasets and DNN models. For CIFAR-10, our method holds 24 out of the top 25 values, and for CIFAR-100, 10 out of the top 12 values belong to our method. Regarding the MNIST dataset, the meta-model based prioritization is the most effective approach, with 19 out of 32 best values when compared to other approaches.

Furthermore, we observed high APFD outcomes, ranging between 0.889 to 0.922, on Image Transformation and Generative Models test datasets of MNIST with DE-Entropy, DE-Confidence and DE-Margin techniques. While the DE methods employing uncertainty metrics ranked in the top three for these MNIST test datasets, alongside the meta-model-based test data prioritization, the meta-model approach outperformed them in the case of CIFAR-10 and CIFAR-100. For these datasets, the meta-model approach achieved the top three APFD values where the MUTs have lower accuracy compared to LeNet-1 and LeNet-5.

The top three APFD values for test data prioritization in MNIST and CIFAR-10 Adversarial Attacks test datasets belong to Meta-Models and the DE Variation Score. Notably, in the CIFAR-100 Adversarial Attack dataset, the DSA method achieved the highest APFD score at 0.622.

Table 24: Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (MNIST).

Test Prioritization Strategy	Original Test Dataset		Generative Models Test Dataset		Image Transformation Test Dataset		Adversarial Attacks Test Dataset	
	LeNet-5	LeNet-1	LeNet-5	LeNet-1	LeNet-5	LeNet-1	LeNet-5	LeNet-1
MUT - Entropy	0.974	0.963	0.917	0.903	0.901	0.854	0.796	0.725
MUT - Confidence	0.973	0.963	0.915	0.902	0.899	0.852	0.795	0.730
MUT - Margin	0.973	0.963	0.913	0.902	0.900	0.850	0.795	0.731
DE - Entropy	0.979	0.966	**0.922**	**0.919**	**0.916**	**0.891**	0.780	0.753
DE - Confidence	0.979	0.966	**0.921**	**0.920**	**0.914**	**0.889**	0.780	0.757
DE - Margin	0.979	0.965	**0.921**	**0.919**	0.912	0.886	0.780	0.758
DE Variation Score	0.949	0.907	0.875	0.884	0.902	0.882	**0.835**	**0.787**
LSA	0.953	0.906	0.895	0.843	0.903	0.884	0.790	0.737
DSA	0.980	0.949	0.919	0.882	0.910	**0.895**	0.802	0.783
DeepGini	0.973	0.963	0.915	0.902	0.899	0.853	0.795	0.729
MCP	0.953	0.955	0.894	0.881	0.875	0.838	0.711	0.653
Meta-Model-1	**0.982**	**0.978**	0.920	**0.919**	**0.915**	0.887	**0.835**	**0.786**
Meta-Model-2	**0.983**	**0.977**	0.920	**0.919**	**0.914**	0.886	**0.835**	**0.786**
Meta-Model-3	**0.984**	**0.978**	**0.924**	**0.919**	**0.914**	0.887	**0.835**	**0.786**

Table 25: Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (CIFAR-10).

Test Prioritization Strategy	Original Test Dataset		Generative Models Test Dataset		Image Transformation Test Dataset		Adversarial Attacks Test Dataset	
	ResNet-32	VGG-19	ResNet-32	VGG-19	ResNet-32	VGG-19	ResNet-32	VGG-19
MUT - Entropy	0.887	0.866	0.825	0.802	0.807	0.782	0.686	0.651
MUT - Confidence	0.886	0.866	0.824	0.802	0.807	0.782	0.690	0.651
MUT - Margin	0.884	0.866	0.822	0.802	0.805	0.781	0.692	0.651
DE - Entropy	0.871	0.870	0.818	0.808	0.784	0.766	0.578	0.620
DE - Confidence	0.873	0.869	0.820	0.807	0.779	0.767	0.579	0.621
DE - Margin	0.871	0.868	0.819	0.807	0.773	0.765	0.579	0.622
DE Variation Score	0.848	0.846	0.808	0.797	0.742	0.728	0.710	**0.713**
LSA	0.674	0.864	0.639	0.800	0.581	0.753	0.599	0.678
DSA	0.846	0.871	0.798	0.808	0.748	0.777	0.669	0.675
DeepGini	0.887	0.866	0.825	0.802	0.807	0.782	0.690	0.651
MCP	0.814	0.779	0.747	0.707	0.584	0.607	0.610	0.584
Meta-Model-1	**0.903**	**0.890**	**0.839**	**0.819**	**0.817**	**0.790**	**0.716**	**0.713**
Meta-Model-2	**0.902**	**0.890**	**0.838**	**0.819**	**0.817**	**0.790**	**0.716**	**0.713**
Meta-Model-3	**0.904**	**0.890**	**0.840**	**0.819**	**0.820**	**0.791**	**0.715**	**0.713**

Table 26: Experiment-5 - Average Percentage of Fault Detected (APFD) values for Generated Test Datasets without OOD (CIFAR-100).

Test Prioritization Strategy	Original Test Dataset	Generative Models Test Dataset	Image Transformation Test Dataset	Adversarial Attacks Test Dataset
	ResNet 28-10	ResNet 28-10	ResNet 28-10	ResNet 28-10
MUT - Entropy	0.797	0.777	0.739	0.555
MUT - Confidence	0.798	0.779	0.738	0.557
MUT - Margin	0.797	0.776	0.734	0.557
DE - Entropy	0.798	0.782	0.750	0.549
DE - Confidence	0.801	0.788	0.751	0.551
DE - Margin	0.796	0.784	0.743	0.552
DE Variation Score	0.783	0.777	0.726	**0.595**
DSA	0.778	0.751	0.731	**0.622**
DeepGini	0.799	0.779	0.739	0.557
MCP	0.670	0.640	0.597	0.512
Meta-Model-1	**0.820**	**0.806**	**0.759**	0.580
Meta-Model-2	**0.819**	**0.805**	**0.758**	0.580
Meta-Model-3	**0.819**	**0.806**	**0.760**	**0.581**

Next, we calculated the RAUC values with the prioritized test dataset sizes of 100, 300, 500, and 1000, along with the entire test dataset. The RAUC values express the effectiveness of test data prioritization techniques in different sizes of prioritized test datasets. It is possible to easily detect some of the test data that the model will classify incorrectly using prioritization methods, while some may be more difficult to distinguish. Prioritization methods can quickly find and prioritize this easily identifiable data in small sized test datasets. However, when it comes to larger prioritized test datasets, it becomes more difficult to identify challenging data that may cause failures. The ability of a prioritization method to detect data that will lead to misclassification depends on both the process of the prioritization method itself and the characteristics of the data involved. By evaluating the effectiveness of prioritization methods across datasets of varying sizes, we aim to understand how well each method performs under different conditions.

The RAUC values for generated test datasets of MNIST, CIFAR-10, and CIFAR-100 are provided in Tables 28-30, with the top three values for each test dataset size highlighted in bold. Additionally, RAUC values for original test datasets are given in Table 27. Figure 34 presents the RAUC values for original and generated test datasets of MNIST, CIFAR-10, and CIFAR-100 using boxplots. These boxplots illustrate the RAUC values across different test dataset sizes for each prioritization method. A successful prioritization is indicated by higher RAUC values, while narrower boxes suggest closer RAUC values across different dataset sizes. The Meta-Models have the highest RAUC values among all test prioritization methods indicating their effectiveness. Furthermore, these RAUC values show narrower variability, reflecting consistent effectiveness across various test dataset sizes. Specifically, for adversarial attack test datasets of MNIST, the RAUC values are remarkably close to the ideal value of 1.0, ranging from [0.99 to 1.0].

Table 27: Experiment-5 - Ratio of Area Under Curve values for Original Test Datasets.

	LeNet-5					LeNet-1				
Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
MUT - Entropy	0.58	0.77	0.84	0.90	0.98	0.97	0.86	0.73	0.60	0.40
MUT - Confidence	0.58	0.77	0.84	0.90	0.98	0.97	0.86	0.73	0.60	0.39
MUT - Margin	0.57	0.77	0.84	0.90	0.98	0.97	0.86	0.73	0.60	0.38
DE - Entropy	0.55	0.78	0.83	0.92	0.98	0.97	0.88	0.77	0.67	0.48
DE - Confidence	0.56	0.78	0.83	0.92	0.98	0.97	0.88	0.77	0.68	0.45
DE - Margin	0.56	0.77	0.83	0.92	0.98	0.97	0.87	0.76	0.66	0.42
DE Variation Score	0.77	0.86	0.88	0.90	0.95	0.92	0.85	0.83	0.81	**0.77**
LSA	0.29	0.52	0.64	0.79	0.96	0.91	0.73	0.60	0.52	0.47
DSA	0.43	0.72	0.83	0.92	0.98	0.96	0.81	0.71	0.64	0.56
DeepGini	0.58	0.77	0.84	0.90	0.98	0.97	0.86	0.73	0.60	0.40
MCP	0.32	0.58	0.72	0.83	0.96	0.96	0.81	0.68	0.55	0.40
Meta-Model-1	**0.78**	**0.88**	**0.92**	**0.95**	**0.99**	**0.99**	**0.93**	**0.88**	**0.82**	**0.76**
Meta-Model-2	**0.78**	**0.88**	**0.92**	**0.95**	**0.99**	**0.99**	**0.93**	**0.87**	**0.82**	0.75
Meta-Model-3	**0.78**	**0.88**	**0.92**	**0.95**	**0.99**	**0.99**	**0.93**	**0.88**	**0.82**	**0.76**
	ResNet-32					VGG-19				
MUT - Entropy	0.71	0.62	0.57	0.65	0.93	0.62	0.62	0.56	0.53	0.91
MUT - Confidence	0.67	0.60	0.56	0.65	0.93	0.66	0.62	0.55	0.52	0.91
MUT - Margin	0.47	0.54	0.54	0.64	0.93	0.62	0.62	0.54	0.52	0.91
DE - Entropy	0.50	0.52	0.49	0.59	0.91	0.59	0.62	0.56	0.53	0.92
DE - Confidence	0.59	0.56	0.51	0.61	0.91	0.56	0.61	0.56	0.53	0.92
DE - Margin	0.55	0.53	0.49	0.59	0.91	0.49	0.60	0.53	0.52	0.92
DE Variation Score	0.80	0.73	0.64	0.70	0.89	0.81	0.74	0.68	0.71	0.89
LSA	0.24	0.26	0.25	0.30	0.71	0.68	0.59	0.56	0.51	0.91
DSA	0.53	0.47	0.43	0.53	0.89	0.58	0.61	0.54	0.52	0.92
DeepGini	0.72	0.64	0.58	0.66	0.93	0.64	0.62	0.55	0.52	0.91
MCP	0.51	0.44	0.41	0.51	0.85	0.46	0.44	0.43	0.39	0.82
Meta-Model-1	**0.81**	**0.75**	**0.67**	**0.74**	**0.95**	**0.79**	**0.73**	**0.74**	**0.68**	**0.94**
Meta-Model-2	**0.81**	**0.75**	**0.67**	**0.73**	**0.94**	**0.79**	**0.73**	**0.74**	**0.68**	**0.94**
Meta-Model-3	**0.82**	**0.74**	**0.67**	**0.74**	**0.95**	**0.77**	**0.73**	**0.74**	**0.68**	**0.94**
	ResNet 28-10									
MUT - Entropy	0.87	0.78	0.75	0.69	0.89					
MUT - Confidence	0.91	0.81	0.76	0.69	0.89					
MUT - Margin	0.72	0.73	0.71	0.67	0.89					
DE - Entropy	0.79	0.77	0.74	0.67	0.89					
DE - Confidence	0.75	0.76	0.75	0.68	0.90					
DE - Margin	0.65	0.65	0.64	0.63	0.89					
DE Variation Score	0.84	0.84	0.84	0.77	0.88					
LSA	0.20	0.24	0.24	0.24	0.57					
DSA	0.66	0.66	0.65	0.61	0.87					
DeepGini	0.88	0.80	0.77	0.70	0.89					
MCP	0.42	0.44	0.41	0.40	0.75					
Meta-Model-1	**0.96**	**0.91**	**0.87**	**0.79**	**0.92**					
Meta-Model-2	**0.95**	**0.89**	**0.86**	**0.78**	**0.92**					
Meta-Model-3	**0.94**	**0.91**	**0.87**	**0.79**	**0.92**					

Table 28: Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (MNIST).

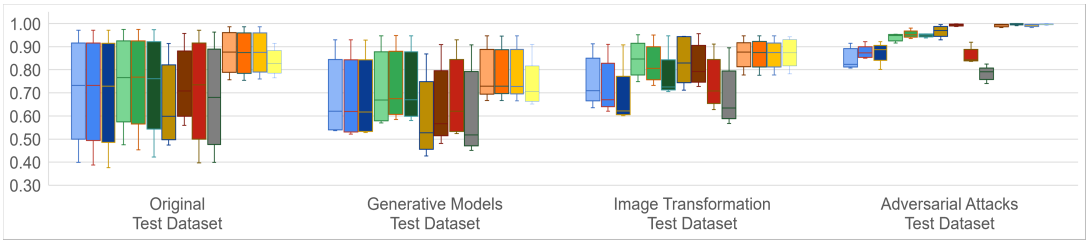
		LeNet-5					LeNet-1				
Test Data Generation Methods	Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
Generative Models	MUT - Entropy	0.58	0.52	0.63	0.78	**0.94**	0.54	0.54	0.62	0.76	0.93
	MUT - Confidence	0.57	0.52	0.63	0.78	**0.94**	0.52	0.54	0.62	0.76	0.93
	MUT - Margin	0.57	0.52	0.63	0.78	**0.94**	0.53	0.54	0.62	0.76	0.93
	DE - Entropy	0.48	0.52	0.64	0.79	**0.94**	0.59	0.57	0.67	0.81	**0.95**
	DE - Confidence	0.55	0.53	0.64	**0.80**	**0.94**	0.63	0.58	0.68	0.81	**0.95**
	DE - Margin	0.54	0.52	0.64	**0.80**	**0.94**	0.62	0.58	0.67	0.81	**0.95**
	DE Variation Score	0.62	0.58	0.60	0.66	0.90	0.71	0.65	0.68	0.72	0.91
	LSA	0.55	0.49	0.57	0.72	0.92	0.53	0.43	0.48	0.63	0.87
	DSA	0.57	0.53	0.62	0.78	**0.94**	0.57	0.48	0.55	0.68	0.91
	DeepGini	0.57	0.52	0.63	0.78	**0.94**	0.52	0.54	0.62	0.76	0.93
	MCP	0.44	0.43	0.54	0.71	0.92	0.49	0.45	0.52	0.68	0.91
	Meta-Model-1	**0.64**	**0.59**	**0.67**	**0.80**	**0.94**	**0.72**	**0.67**	**0.73**	**0.83**	**0.95**
	Meta-Model-2	**0.64**	**0.59**	**0.67**	**0.80**	**0.94**	**0.73**	**0.67**	**0.73**	**0.83**	**0.95**
	Meta-Model-3	**0.64**	**0.59**	**0.67**	**0.80**	**0.95**	**0.72**	**0.67**	**0.73**	**0.83**	**0.95**
Image Transformations	MUT - Entropy	0.81	0.75	0.71	0.70	0.95	0.79	0.71	0.69	0.64	0.91
	MUT - Confidence	0.78	0.73	0.70	0.70	0.95	0.75	0.67	0.66	0.62	0.91
	MUT - Margin	0.73	0.71	0.70	0.69	0.95	0.60	0.62	0.63	0.61	0.91
	DE - Entropy	0.85	0.83	0.79	0.77	**0.97**	0.88	0.85	0.81	0.75	**0.95**
	DE - Confidence	0.81	0.80	0.77	0.76	**0.96**	0.85	0.81	0.78	0.73	**0.95**
	DE - Margin	0.71	0.75	0.74	0.74	**0.96**	0.72	0.74	0.73	0.71	**0.95**
	DE Variation Score	**0.88**	**0.85**	**0.82**	**0.79**	0.95	**0.92**	**0.87**	**0.85**	**0.78**	0.94
	LSA	0.68	0.69	0.67	0.67	0.95	**0.94**	0.83	0.78	0.71	0.94
	DSA	0.79	0.74	0.71	0.71	**0.96**	0.86	0.79	0.77	0.73	**0.96**
	DeepGini	0.81	0.74	0.71	0.70	0.95	0.78	0.71	0.68	0.63	0.91
	MCP	0.64	0.65	0.62	0.60	0.92	0.69	0.63	0.61	0.57	0.90
	Meta-Model-1	**0.88**	**0.86**	**0.82**	**0.79**	**0.96**	0.89	**0.88**	**0.85**	**0.78**	**0.95**
	Meta-Model-2	0.87	**0.85**	**0.82**	**0.79**	**0.96**	**0.90**	**0.87**	**0.85**	**0.78**	**0.95**
	Meta-Model-3	**0.89**	**0.87**	**0.83**	**0.79**	**0.96**	0.88	**0.87**	**0.85**	**0.78**	**0.95**
Adversarial Attacks	MUT - Entropy	0.885	0.89	0.89	0.87	0.95	0.87	0.82	0.81	0.82	0.91
	MUT - Confidence	0.888	0.90	0.89	0.87	0.95	0.85	0.86	0.87	0.88	0.92
	MUT - Margin	0.86	0.89	0.89	0.87	0.95	0.80	0.88	0.89	0.89	0.92
	DE - Entropy	0.90	0.91	0.91	0.87	0.93	0.95	0.95	0.93	0.92	0.95
	DE - Confidence	0.91	0.91	0.91	0.87	0.93	0.98	0.95	0.95	0.94	0.95
	DE - Margin	0.92	0.91	0.91	0.87	0.93	0.96	0.95	0.95	0.94	0.96
	DE Variation Score	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**
	LSA	0.99	0.97	0.96	0.91	0.94	**1.00**	0.98	0.97	0.96	0.93
	DSA	0.94	0.94	0.94	0.90	0.96	**1.00**	**1.00**	**1.00**	0.99	**0.99**
	DeepGini	0.89	0.88	0.89	0.87	0.95	0.84	0.84	0.84	0.86	0.92
	MCP	0.64	0.70	0.70	0.68	0.85	0.79	0.79	0.78	0.74	0.82
	Meta-Model-1	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	0.98	0.99	**1.00**	**1.00**	**0.99**
	Meta-Model-2	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**
	Meta-Model-3	**1.00**	**1.00**	**1.00**	**1.00**	**0.99**	0.98	0.99	**1.00**	**1.00**	**0.99**

Table 29: Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-10).

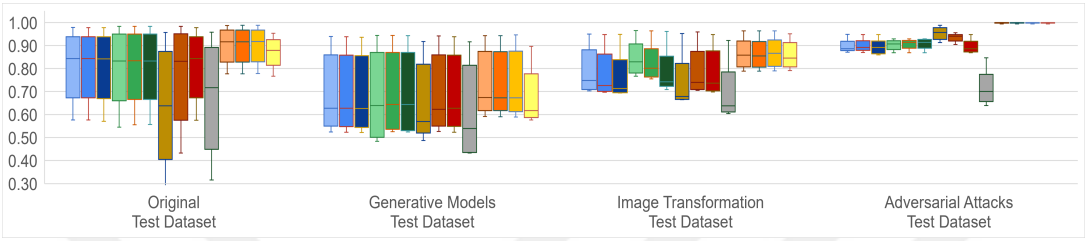
Test Data Generation Methods	Test Prioritization Strategy	ResNet-32					VGG-19				
		$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
Generative Models	MUT - Entropy	0.76	0.64	0.57	0.64	0.89	0.56	0.60	0.53	0.51	0.86
	MUT - Confidence	0.69	0.60	0.56	0.63	0.88	0.58	0.60	0.53	0.51	0.86
	MUT - Margin	0.51	0.55	0.53	0.63	0.88	0.56	0.60	0.53	0.51	0.86
	DE - Entropy	0.64	0.57	0.52	0.61	0.88	0.57	0.60	0.54	0.53	0.87
	DE - Confidence	0.68	0.59	0.54	0.62	0.88	0.51	0.60	0.54	0.53	0.87
	DE - Margin	0.57	0.55	0.52	0.61	0.88	0.48	0.59	0.51	0.52	0.87
	DE Variation Score	0.77	0.69	0.62	0.67	0.87	**0.78**	**0.69**	0.63	**0.65**	0.86
	LSA	0.30	0.32	0.30	0.35	0.69	0.50	0.57	0.47	0.46	0.86
	DSA	0.51	0.48	0.46	0.55	0.86	0.51	0.60	0.50	0.49	0.87
	DeepGini	0.76	0.63	0.57	0.64	0.89	0.58	0.60	0.53	0.51	0.86
	MCP	0.50	0.46	0.42	0.49	0.80	0.41	0.44	0.40	0.37	0.76
	Meta-Model-1	**0.84**	**0.72**	**0.63**	**0.69**	**0.90**	**0.74**	**0.67**	**0.69**	**0.64**	**0.88**
	Meta-Model-2	**0.82**	**0.70**	**0.63**	**0.69**	**0.90**	**0.74**	**0.67**	**0.69**	**0.64**	**0.88**
	Meta-Model-3	**0.87**	**0.72**	**0.64**	**0.70**	**0.90**	**0.75**	**0.67**	**0.69**	**0.64**	**0.88**
Image Transformations	MUT - Entropy	0.75	0.74	0.72	0.69	0.91	0.80	0.67	0.74	0.71	0.88
	MUT - Confidence	0.75	0.71	0.70	0.69	0.91	0.74	0.66	0.71	0.69	0.88
	MUT - Margin	0.71	0.66	0.66	0.66	0.91	0.65	0.65	0.66	0.66	0.88
	DE - Entropy	0.84	0.73	0.71	0.68	0.88	0.78	0.63	0.71	0.68	0.86
	DE - Confidence	0.78	0.72	0.70	0.66	0.88	0.76	0.62	0.66	0.65	0.86
	DE - Margin	0.63	0.64	0.62	0.60	0.87	0.59	0.59	0.61	0.60	0.86
	DE Variation Score	0.86	0.85	0.85	0.80	0.84	**0.87**	**0.82**	0.80	0.75	0.82
	LSA	0.25	0.28	0.29	0.30	0.65	0.51	0.55	0.53	0.54	0.85
	DSA	0.65	0.62	0.61	0.57	0.84	0.76	0.65	0.71	0.69	0.88
	DeepGini	0.77	0.74	0.72	0.69	0.91	0.81	0.66	0.74	0.70	0.88
	MCP	0.65	0.56	0.53	0.48	0.66	0.57	0.49	0.55	0.53	0.69
	Meta-Model-1	**0.93**	**0.91**	**0.89**	**0.82**	**0.92**	**0.85**	**0.76**	**0.85**	**0.82**	**0.89**
	Meta-Model-2	**0.87**	**0.89**	**0.89**	**0.82**	**0.92**	0.79	**0.76**	**0.84**	**0.82**	**0.89**
	Meta-Model-3	**0.96**	**0.90**	**0.88**	**0.83**	**0.92**	**0.89**	**0.77**	**0.86**	**0.83**	**0.89**
Adversarial Attacks	MUT - Entropy	0.88	0.86	0.86	0.86	0.94	0.73	0.79	0.75	0.74	0.89
	MUT - Confidence	0.82	0.84	0.86	0.89	0.95	0.69	0.78	0.67	0.71	0.89
	MUT - Margin	0.70	0.82	0.87	0.90	0.95	0.60	0.78	0.64	0.70	0.89
	DE - Entropy	0.75	0.78	0.77	0.74	0.79	0.81	0.79	0.80	0.80	0.85
	DE - Confidence	0.79	0.79	0.78	0.75	0.79	0.78	0.80	0.80	0.81	0.85
	DE - Margin	0.76	0.78	0.77	0.75	0.79	0.77	0.80	0.81	0.82	0.85
	DE Variation Score	**0.99**	**0.99**	**0.99**	**0.99**	**0.98**	**1.00**	**1.00**	**0.99**	**0.99**	**0.97**
	LSA	0.89	0.88	0.86	0.81	0.82	0.95	0.86	0.89	0.88	0.92
	DSA	0.96	0.92	0.90	0.88	0.92	0.79	0.83	0.81	0.81	0.92
	DeepGini	0.87	0.86	0.87	0.89	0.95	0.72	0.78	0.70	0.72	0.89
	MCP	0.74	0.78	0.80	0.80	0.84	0.72	0.72	0.71	0.71	0.80
	Meta-Model-1	**0.97**	**0.98**	**0.98**	**0.99**	**0.98**	**0.99**	**0.99**	**0.98**	**0.98**	**0.97**
	Meta-Model-2	**0.98**	**0.99**	**1.00**	**0.99**	**0.98**	**0.98**	**0.99**	**0.98**	**0.98**	**0.97**
	Meta-Model-3	0.95	0.96	0.97	0.98	**0.98**	**0.98**	**0.99**	**0.98**	**0.98**	**0.97**

Table 30: Experiment-5 - Ratio of Area Under Curve values for Generated Test Datasets (CIFAR-100).

		ResNet 28-10				
Test Data Generation Methods	Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
Generative Models	MUT - Entropy	0.90	0.77	0.72	0.67	0.89
	MUT - Confidence	0.91	0.78	0.72	0.68	0.89
	MUT - Margin	0.72	0.71	0.68	0.67	0.88
	DE - Entropy	0.86	0.77	0.72	0.68	0.89
	DE - Confidence	0.81	0.78	0.73	0.69	0.90
	DE - Margin	0.70	0.70	0.68	0.67	0.89
	DE Variation Score	0.85	0.84	0.82	0.75	0.89
	LSA	0.25	0.26	0.26	0.26	0.56
	DSA	0.63	0.62	0.61	0.60	0.86
	DeepGini	0.91	0.78	0.73	0.68	0.89
	MCP	0.41	0.37	0.36	0.39	0.73
	Meta-Model-1	**0.97**	**0.86**	**0.83**	**0.77**	**0.92**
	Meta-Model-2	**0.95**	**0.86**	**0.83**	**0.76**	**0.92**
	Meta-Model-3	**0.93**	**0.87**	**0.83**	**0.77**	**0.92**
Image Transformations	MUT - Entropy	0.93	**0.95**	0.89	0.84	0.89
	MUT - Confidence	0.93	0.94	0.90	0.85	0.89
	MUT - Margin	0.81	0.79	0.80	0.79	0.88
	DE - Entropy	0.88	0.89	0.86	0.83	0.90
	DE - Confidence	0.89	0.88	0.88	0.85	0.90
	DE - Margin	0.76	0.73	0.76	0.76	0.89
	DE Variation Score	0.86	0.83	0.87	0.87	0.87
	LSA	0.36	0.34	0.39	0.39	0.61
	DSA	0.83	0.85	0.81	0.78	0.88
	DeepGini	0.93	0.94	0.89	0.85	0.89
	MCP	0.50	0.49	0.49	0.51	0.72
	Meta-Model-1	**0.97**	**0.98**	**0.94**	**0.90**	**0.91**
	Meta-Model-2	**0.94**	**0.95**	**0.92**	**0.89**	**0.91**
	Meta-Model-3	**0.97**	**0.98**	**0.95**	**0.91**	**0.91**
Adversarial Attacks	MUT - Entropy	**0.99**	0.93	0.91	0.89	0.83
	MUT - Confidence	0.98	0.92	0.90	0.88	0.83
	MUT - Margin	0.76	0.84	0.86	0.87	0.83
	DE - Entropy	0.92	0.91	0.90	0.88	0.82
	DE - Confidence	0.90	0.91	0.90	0.88	0.82
	DE - Margin	0.87	0.87	0.88	0.87	0.83
	DE Variation Score	**0.99**	**0.97**	**0.98**	**0.98**	**0.89**
	LSA	0.74	0.73	0.73	0.70	0.75
	DSA	**1.00**	**1.00**	**1.00**	**0.98**	**0.93**
	DeepGini	0.98	0.92	0.91	0.88	0.83
	MCP	0.72	0.74	0.73	0.72	0.77
	Meta-Model-1	0.98	0.95	0.95	**0.96**	**0.87**
	Meta-Model-2	0.98	**0.96**	**0.96**	**0.96**	**0.87**
	Meta-Model-3	0.98	**0.96**	**0.96**	**0.96**	**0.87**



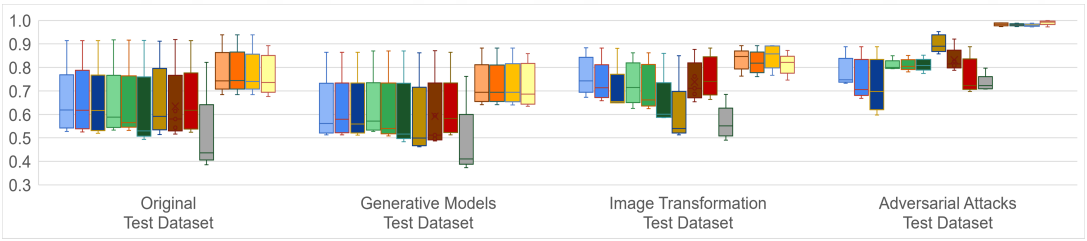
(a) LeNet-1



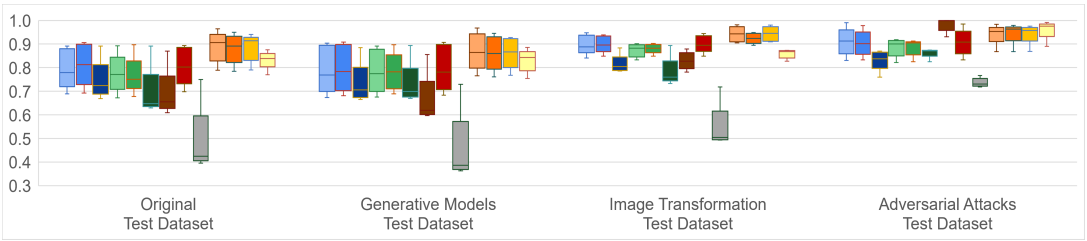
(b) LeNet-5



(c) ResNet



(d) VGG-19



(e) ResNet 28-10



Figure 34: Experiment-5 - RAUC Values of Test Prioritization Strategies for Original and Generated Test Datasets without OOD Data [120] ©2024 ACM.

Finally, we evaluated the effectiveness of test prioritization strategies by analyzing the correlation between metric values and misclassifications using Point-Biserial correlation analysis. Figures 35 and 36 illustrate a strong correlation between misclassifications and the meta-model approach for both the original and generated test datasets. The highest correlation observed with the meta-model approach for the generated MNIST test datasets is 0.81, while the highest correlation values obtained for the meta-model approach for generated test datasets of CIFAR-10 and CIFAR-100 are 0.71 and 0.60 respectively, respectively.

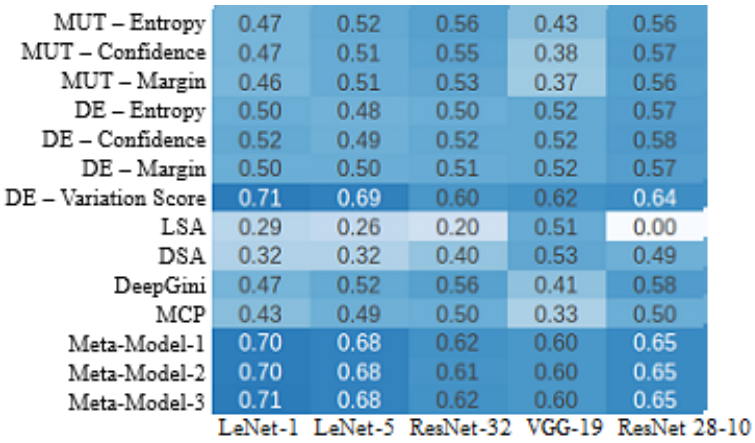


Figure 35: Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Original Test Dataset [120] ©2024 ACM.

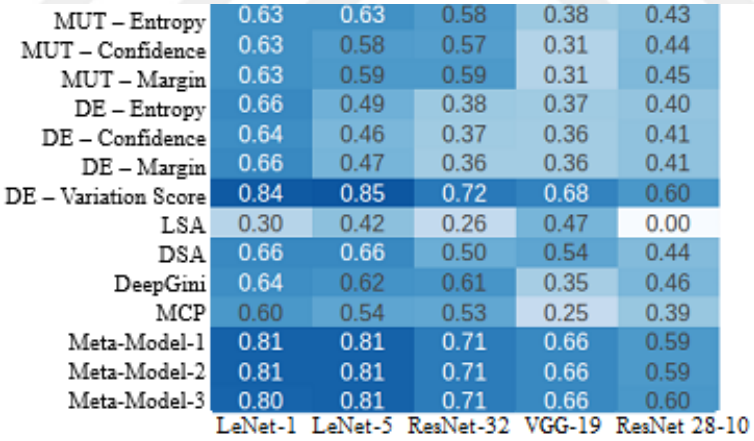


Figure 36: Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Generated Test Datasets without OOD data [120] ©2024 ACM.

Furthermore, we explored the effectiveness of alternative Meta-Models by employing different input feature combinations as shown in Table 31, but they showed poorer performance on identification and prioritization of fault-revealing test inputs compared to Meta-Models given in Table 8.

Table 31: Alternative Meta-Models and their input features.

Meta-Model	Input Features
Meta-Model - 4	Mahalanobis Uncertainty Score Least Confidence DE Variation Score
Meta-Model - 5	Mahalanobis Uncertainty Score Margin DE Variation Score
Meta-Model - 6	Mahalanobis Uncertainty Score DE Variation Score

Results for Test Datasets with OOD Data

We performed experiments without removing OOD data from the generated test datasets to evaluate the effectiveness of the proposed method for prioritizing test datasets containing OOD data. The accuracy results for the MUTs on the test datasets containing OOD are given in Table 32. It is worth noting that the accuracy metrics have decreased for the generated test data in comparison to the original test data provided in Table 1, particularly for the adversarial test inputs.

Table 32: Experiment-5 - Accuracy of models for Generated Test Data.

Model	Generative Models	Image Transformation	Adversarial Attacks
LeNet-5	91.8%	87.6%	45.9%
LeNet-1	90.4%	84.6%	36.4%
ResNet-32	79.4%	70.8%	11.6%
VGG-19	79.8%	70.9%	5.5%
ResNet 28-10	72.5%	58.4%	0.9%

For this experiment, we combined three types of generated test datasets (Image Transformation, Generative Models, and Adversarial Attacks) and the Original test dataset into a single dataset for each of MNIST, CIFAR-10, and CIFAR-100. The APFD values obtained from this experiment, with the best three values highlighted in bold, are listed in Table 33. The table demonstrates that the APFD values decreased when OOD data were included in the test datasets compared to when they were not included. LeNet-1 and LeNet-5 models had higher APFD values compared to CIFAR-10 and CIFAR-100. Similar to the test datasets without OOD data, the prioritization using meta-models resulted in the best APFD outcomes (14 out of 18). Moreover, the OOD data in the test datasets led to successful prioritization using the LSA and DSA methods in the CIFAR-10 and CIFAR-100 test datasets. Additionally, the DE-Variation Score showed the highest APFD values for MNIST following Meta-Models, and it also demonstrated the best value for CIFAR-100, highlighting its effectiveness in prioritizing fault-revealing inputs in test datasets containing OOD data.

Table 33: Experiment-5 - APFD Values of Test Data Prioritization Strategies for Generated Test Datasets with OOD Data [120] ©2024 ACM.

Test Prioritization Strategy	MNIST		CIFAR-10		CIFAR-100
	LeNet-5	LeNet-1	ResNet-32	VGG-19	ResNet 28-10
MUT - Entropy	0.819	0.769	0.677	0.572	0.579
MUT - Confidence	0.819	0.772	0.678	0.572	0.579
MUT - Margin	0.819	0.772	0.677	0.573	0.578
DE - Entropy	0.788	0.797	0.636	0.635	0.584
DE - Confidence	0.787	0.802	0.633	0.633	0.587
DE - Margin	0.788	0.803	0.630	0.632	0.586
DE Variation Score	0.845	0.816	0.669	0.649	**0.619**
LSA	0.804	0.775	0.585	**0.668**	-
DSA	0.822	0.815	0.653	**0.668**	**0.654**
DeepGini	0.819	0.772	0.678	0.573	0.580
MCP	0.772	0.726	0.584	0.563	0.498
Meta-Model-1	**0.861**	**0.833**	**0.700**	**0.656**	**0.601**
Meta-Model-2	**0.861**	**0.833**	**0.700**	**0.656**	0.600
Meta-Model-3	**0.862**	**0.832**	**0.701**	**0.656**	**0.601**

In our evaluation, we computed RAUC values for different test data prioritization strategies across different prioritized dataset sizes—100, 300, 500, 1000, and the entire test dataset. The RAUC values for test datasets with OOD data are provided in Tables 34, with the top three values for each test dataset highlighted in bold. Figure 37 illustrates these RAUC values for each dataset size with boxplots. This figure, unlike the RAUC values presented in Figure 34, uses boxplots to detail the performance based on different test dataset sizes for the combined test datasets of MNIST, CIFAR-10, and CIFAR-100.

The Meta-Model method consistently outperforms other approaches, achieving the highest RAUC-ALL values, which range from 0.86 to 0.98 for the entire test dataset. Although Meta-Models performed well across all dataset sizes, the DE-Variation Score was particularly effective for smaller dataset sizes, ranging from 100 to 1000 data. This was indicated by the highest RAUC values and narrow boxplots, which were represented with yellow-colored boxes in Figure 37.

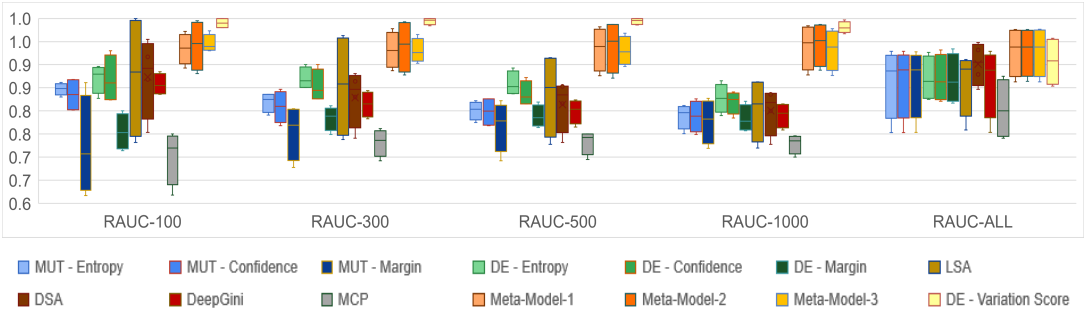


Figure 37: Experiment-5 - RAUC Values of Test Data Prioritization Strategies for Generated Test Datasets with OOD Data [120] ©2024 ACM.

Table 34: Experiment-5 - Ratio of Area Under Curve values for Test Datasets with OOD data.

	LeNet-5					LeNet-1				
Test Prioritization Strategy	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$	$RAUC_{100}$	$RAUC_{300}$	$RAUC_{500}$	$RAUC_{1000}$	$RAUC_{All}$
MUT - Entropy	0.86	0.84	0.81	0.80	0.93	0.83	0.79	0.78	0.75	0.90
MUT - Confidence	0.87	0.83	0.83	0.83	0.93	0.80	0.80	0.77	0.77	0.90
MUT - Margin	0.75	0.81	0.82	0.83	0.93	0.62	0.74	0.78	0.81	0.90
DE - Entropy	0.89	0.85	0.84	0.79	0.89	0.87	0.88	0.87	0.83	0.93
DE - Confidence	0.83	0.83	0.82	0.79	0.89	0.90	0.86	0.85	0.83	0.93
DE - Margin	0.73	0.75	0.76	0.76	0.89	0.80	0.81	0.82	0.82	0.93
DE Variation Score	**1.00**	**1.00**	**1.00**	**1.00**	**0.96**	**0.98**	**0.99**	**0.99**	**0.98**	0.95
LSA	0.73	0.78	0.79	0.77	0.91	0.98	0.94	0.91	0.86	0.90
DSA	0.96	0.86	0.85	0.84	0.93	0.92	0.88	0.85	0.84	0.95
DeepGini	0.87	0.83	0.81	0.82	0.93	0.84	0.78	0.77	0.76	0.90
MCP	0.73	0.74	0.75	0.75	0.88	0.71	0.73	0.74	0.74	0.84
Meta-Model-1	**0.97**	**0.98**	**0.98**	**0.98**	**0.98**	0.93	**0.95**	**0.96**	**0.97**	**0.97**
Meta-Model-2	**0.98**	**0.99**	**0.99**	**0.99**	**0.98**	**1.00**	**0.99**	**0.99**	**0.98**	**0.97**
Meta-Model-3	**0.97**	**0.97**	**0.97**	**0.98**	**0.98**	**0.94**	0.93	0.94	**0.96**	**0.97**
	ResNet-32					VGG-19				
MUT - Entropy	0.85	0.84	0.82	0.81	0.88	0.85	0.82	0.80	0.79	0.75
MUT - Confidence	0.87	0.85	0.83	0.81	0.88	0.80	0.77	0.77	0.75	0.75
MUT - Margin	0.86	0.80	0.77	0.76	0.88	0.66	0.68	0.69	0.72	0.75
DE - Entropy	0.83	0.85	0.84	0.82	0.83	0.90	0.90	0.89	0.87	0.84
DE - Confidence	0.82	0.83	0.82	0.82	0.82	0.93	0.90	0.87	0.84	0.83
DE - Margin	0.78	0.79	0.78	0.76	0.82	0.71	0.79	0.79	0.79	0.83
DE Variation Score	**1.00**	**1.00**	**1.00**	**0.98**	0.87	**0.98**	**0.98**	**0.99**	**0.97**	0.85
LSA	0.79	0.74	0.73	0.72	0.76	**1.00**	**0.96**	**0.92**	0.86	**0.88**
DSA	0.75	0.74	0.73	0.73	0.85	0.87	0.83	0.82	0.80	**0.88**
DeepGini	0.89	0.84	0.82	0.81	0.88	0.84	0.80	0.79	0.78	0.75
MCP	0.75	0.76	0.75	0.73	0.76	0.62	0.69	0.70	0.70	0.74
Meta-Model-1	**0.94**	**0.89**	**0.88**	**0.88**	**0.91**	0.89	**0.92**	**0.92**	**0.92**	**0.86**
Meta-Model-2	**0.91**	**0.88**	**0.87**	**0.89**	**0.91**	0.88	0.90	**0.92**	**0.92**	**0.86**
Meta-Model-3	**0.93**	**0.90**	**0.90**	**0.88**	**0.91**	**0.94**	**0.92**	**0.92**	**0.92**	**0.86**
	ResNet 28-10									
MUT - Entropy	0.97	0.96	0.94	0.89	0.82					
MUT - Confidence	0.95	0.96	0.94	0.90	0.82					
MUT - Margin	0.78	0.81	0.82	0.82	0.82					
DE - Entropy	0.93	0.91	0.89	0.87	0.81					
DE - Confidence	0.89	0.91	0.90	0.89	0.82					
DE - Margin	0.79	0.80	0.81	0.81	0.81					
DE Variation Score	**1.00**	**0.99**	**0.99**	**0.98**	**0.87**					
DSA	**1.00**	**1.00**	**1.00**	**0.98**	**0.89**					
DeepGini	0.96	0.96	0.94	0.90	0.82					
MCP	0.58	0.64	0.63	0.64	0.73					
Meta-Model-1	0.98	**0.98**	**0.97**	**0.94**	0.86					
Meta-Model-2	0.95	0.95	0.95	**0.94**	0.86					
Meta-Model-3	**0.99**	**0.98**	**0.97**	**0.94**	**0.87**					

Lastly, Figure 38 illustrates the correlation between test data prioritization strategies and misclassifications. Notably, the Meta-Model approach demonstrates a correlation with misclassifications, particularly when the combined MNIST test dataset was used with the LeNet-1 and LeNet-5 models. Conversely, this correlation tends to weaken for CIFAR-10 and CIFAR-100 test datasets and respective models. For instance, the highest correlation observed with the ResNet-32 model shows a notable decrease, dropping from 0.71 to 0.59 when comparing scenarios with and without OOD data where the correlation values for test datasets without OOD are given in Table 36.

Furthermore, the DE-Variance Score also exhibited medium to strong correlations, peaking at 0.87 for LeNet-5, indicating its potential in detecting misclassifications for datasets with OOD data. The MCP and MUT-* methods, including MUT-Confidence, MUT-Entropy, and MUT-Margin, showed varying results with generally weak correlations, suggesting their effectiveness may be more limited or context-dependent. Among the compared methods, the lowest correlation values belong to the LSA method.

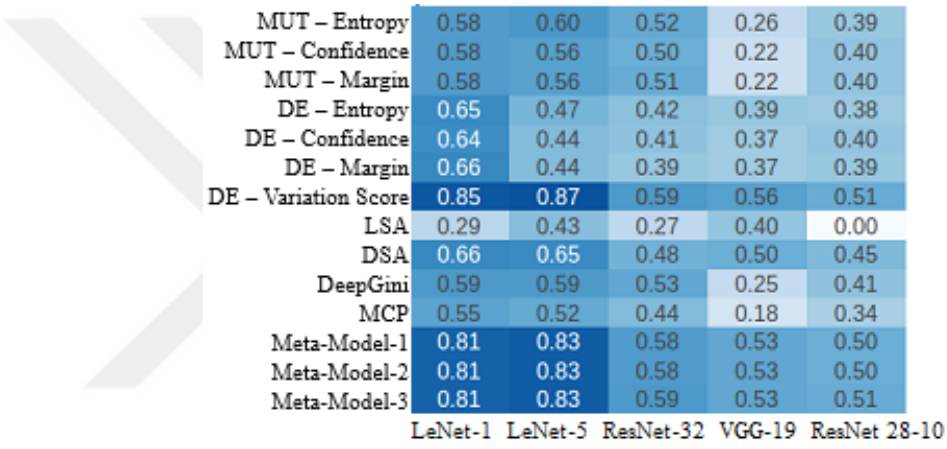


Figure 38: Experiment-5 - Correlation Between Test Prioritization Strategies and Misclassifications for Generated Test Datasets with OOD data [120] ©2024 ACM.

CHAPTER 8

DISCUSSION

8.1 Test Data Selection

In this study, we proposed test data selection methods to identify and prioritize test data that would cause the DNN model to fail in making correct predictions. To achieve this, we leveraged the model's uncertainty against test inputs. Furthermore, we suggested initially focusing on testing with in-distribution data, where the DNN model is expected to make accurate predictions, and then including out-of-distribution (OOD) data in test datasets.

First, we evaluated state-of-the-art uncertainty estimation methods in test data prioritization with three uncertainty metrics: Least Confidence, Entropy, and Margin. Second, we introduced two new uncertainty-based test data prioritization metrics: the Mahalanobis Uncertainty Score and the DE Variation Score. Finally, we proposed a meta-model-based test data prioritization approach.

These test data prioritization approaches were compared against the leading techniques in the literature, including LSA (Likelihood-based Surprise Adequacy) [29], DSA (Distance-based Surprise Adequacy) [29], DeepGini [25], and MCP (Multiple Boundary Clustering and Prioritization) [26], serving as baseline methods.

8.1.1 Uncertainty Based Test Data Prioritization

We evaluated the effectiveness of uncertainty-based test data prioritization performed by using metrics gathered from Deep Ensemble (DE), Variational Inference with Flipout (VI-F), and Model Under Test (MUT) in Experiment-1. The list of test data prioritization strategies evaluated is given in Table 6. We use the abbreviation DE-* to refer to test data prioritization methods that use uncertainty metrics derived from the DE models. Similarly, we use the abbreviation MUT-* for test data prioritization methods based on uncertainty metrics obtained from the outputs of the MUT. Additionally, we use VI-* for the test data prioritization based on the use of uncertainty metrics with VI-F.

In the following two subsections, we discuss the findings from the experiment results evaluating the effectiveness of uncertainty-based test data prioritization, separately for test datasets with and without OOD data.

Prioritization without OOD Data

The experiment results reveal that the test data prioritization performed through the DE method effectively identifies fault-revealing data in test datasets generated by GANs and image transformations, as evidenced by high Average Percentage of Detected (APFD) values in Table 11. The Ratio of Area Under Curve (RAUC) values demonstrate the effectiveness of test data prioritization methods across different dataset sizes. For test datasets generated by image transformation and GANs, the DE method was effective when entire test data was used, Tables 12 and 13. However, for the CIFAR-10 test dataset generated with image transformation, the MUT-Entropy and MUT-Confidence methods outperformed the DE method for test data sizes ranging from 100 to 1000. In this dataset, the DeepGini baseline method was also effective across all dataset sizes. Additionally, supporting the effectiveness of DE method, the correlation analysis results given in Table 14 and 15 showed that there is a moderate correlation between test data prioritization metrics calculated with DE method and misclassifications in the test datasets generated through GANs and image transformations.

Conversely, for test datasets generated by adversarial attacks, test data prioritization using the MUT with uncertainty metrics showed better performance than the DE method, as indicated by higher APFD values. Specifically, MUT-Margin and MUT-Confidence showed the highest correlation coefficients for MNIST, whereas MUT-Entropy has the highest correlation for CIFAR-10.

Since the adversarial attack test datasets are generated using the trained MUT as input to attack methods, the generated images are tailored according to their hyperparameter values. Consequently, these images are unlikely to deceive DNN models used in DE or VI-F methods due to their differing hyperparameter values with MUT, and they do not introduce uncertainty in these models. As a result, the uncertainty metric values acquired from these methods are incapable of identifying fault-revealing data, which leads to lower APFD and RAUC values.

Baseline methods DeepGini, LSA, and DSA also effectively prioritized adversarial test data, achieving APFD and RAUC values in the top three best values. DeepGini uses a formula akin to MUT-Entropy to measure the uncertainty using class probability outputs. This leads to similar APFD values with a minor average difference. On the other hand, the DSA method's effectiveness, particularly in adversarial datasets, is attributed to differences in the activation trace of test data caused by minor perturbations. We conclude this since DSA uses the distance of the activation trace of the test input to the activation trace of the training dataset to determine how different the test input is from the training data set and uses this distance to prioritize the test data.

Additionally, Figures 20 and 21 support the effectiveness of uncertainty-based prioritization in detecting misclassified data. All prioritization approaches consistently outperformed random data selection. For MNIST, prioritizing with the DE method allowed the detection of over 95% of misclassifications with just 30% of the dataset. For CIFAR-10, 50% of the dataset was needed for equivalent detection, demonstrating the DE method's superiority over random selection in identifying fault-revealing test data. Hence, we can conclude that by prioritizing test data using the DE method and labeling these prioritized data for testing, we can identify significantly more test data that will cause DNN failures compared to using random selection in these test datasets.

Moreover, in all scenarios, the VI-F method demonstrated the lowest performance in prioritizing fault-revealing test data compared to the MUT-* and DE-* methods. The implementation of the VI-F method poses significant challenges, particularly in selecting the layers to be implemented with Flipout, select-

ing the prior distribution, and training the model to accurately approximate the posterior distribution. We designed the VI-F models following the prior work [96]. Despite this, the VI-F method still showed poor performance in identifying and prioritizing fault-revealing data. This could be attributed to the complexity of integrating Flipout effectively into DNN layers and the difficulty in achieving a high-quality approximation of the posterior distribution, which is crucial for reliable uncertainty estimation. Consequently, the VI-F method's poorer performance in uncertainty quantification led to less effective prioritization of fault-revealing test data compared to the MUT and DE methods. This suggests that while VI-F has potential, its current implementation requires further refinement to match the effectiveness of other methods in fault detection and uncertainty estimation.

Prioritization with OOD Data

The effectiveness of uncertainty-based test data prioritization approaches decreases when OOD data are included in the test datasets. The results of the experiments show that the top three APFD values for prioritizing test datasets containing OOD data were lower than those for datasets without OOD data, as illustrated in Table 17.

Nevertheless, prioritizing test data using uncertainty metrics derived from the outputs of DE models and the MUT was still effective for test datasets, including OOD data. Notably, the baseline DSA method exhibited the highest APFD values when using Lenet-1, ResNet-32, and VGG-19 models. This suggests that OOD test data have activation traces that differ significantly from those observed in the training data.

As shown in Table 16, the accuracy of models on test datasets with OOD data decreased compared to those without OOD data because the models were exposed to data different from the training data. The effectiveness of uncertainty-based test data prioritization methods, which rely on the class probability outputs of the MUT, is directly affected by the accuracy of the models against the test dataset. For instance, the accuracy of the ResNet-32 and VGG-19 models was 49.3% and 49.2%, respectively, both below 50%. Consequently, for the CIFAR-10 test dataset with OOD data, the highest APFD value for all MUT-* methods was 0.60, where corresponding values were within the range of 0.68 to 0.83 for test datasets without OOD data. The DE method, which uses more than one model, better estimates uncertainty than a single model with a low accuracy. Therefore, the test data prioritization performed with the DE method was more effective for CIFAR-10 than with performing test data prioritization through uncertainty metrics gathered from MUT, as indicated by higher APFD values belonging to the DE method. Conversely, the LeNet-5 model, with an accuracy of 78.4%, demonstrated the effectiveness of the MUT method, achieving the highest APFD values.

Furthermore, the presence of adversarial data within the OOD data negatively impacted the effectiveness of DE methods, leading to reduced APFD values. For instance, the highest APFD value for LeNet-5 on the test dataset with OOD data was 0.80, whereas APFD values for LeNet-5 ranged from 0.80 to 0.92 for test datasets without OOD data. This underscores the challenges posed by adversarial data in accurately prioritizing test data using the DE method.

In summary, while uncertainty-based test data prioritization remains effective in the presence of OOD data, its performance is reduced compared to scenarios without OOD data. The DE method generally outperforms the test data prioritization performed using uncertainty metrics obtained from MUT in scenarios with low accuracy models, while the use of uncertainty metrics from MUT performs better with high accuracy models.

8.1.2 Test Data Prioritization with Mahalanobis Uncertainty Score

Mahalanobis Uncertainty Score (MUS) is a distance-based metric that helps to measure how uncertain a model is about a specific input by comparing it to the training dataset. To calculate MUS, we use the representations of the training data, which are obtained from the layer just before the logits layer and are grouped by class labels. We then measure the distances between the representation of test input and these grouped training data representations. The MUS value is the difference between the two closest distances. We conducted Experiment-3 to evaluate the MUS metric effectiveness in test data prioritization.

MUS delivers results that are on par with existing uncertainty metrics, Least Confidence, Entropy, and Margin. When MUS is used with the DE method, its effectiveness in data prioritization is close to other uncertainty metrics employed with DE, showing a difference in the APFD of less than 0.007 in Table 18.

One of the significant considerations with MUS is its computational complexity. In contrast, simpler metrics like Least Confidence, which relies on the highest class probability, are much easier to compute. Due to its higher complexity and lack of performance superiority over the Least Confidence metric, MUS is less appealing in test data prioritization.

8.1.3 Test Data Prioritization with DE Variation Score

The DE Variation Score measures uncertainty by examining the variance between the predictions of DE models and MUT for a given test input. We performed Experiment-4 to evaluate the DE Variation Score metric effectiveness in test data prioritization. Experimental results highlighted that it is particularly effective in prioritizing adversarial data, significantly increasing the APFD value by 9.7% compared to the closest test prioritization method, as demonstrated in Table 20. This effectiveness is further demonstrated by the very strong correlation value of 0.84 between misclassifications in the adversarial attack test dataset and the DE Variation Score in Table 21. However, while the DE Variation Score is effective on adversarial data, it has a lower performance for test data generated through image transformations and GANs. In these scenarios, other uncertainty metrics (Least Confidence, Entropy, Margin) used with the DE method perform better.

The effectiveness of the DE Variation Score is rooted in its ability to detect uncertainty when test data leads to distinct predictions between DE models and the MUT. This characteristic is most prominently demonstrated with adversarial data. Adversarial data, specifically generated to deceive the MUT, cannot mislead DNN models in DE, which have varying parameter values. Consequently, the predictions of DE models deviate significantly from those of the MUT, revealing high uncertainty.

8.1.4 Meta-model Based Test Data Prioritization

We employed a meta-model to combine the strengths of multiple uncertainty metrics with the goal of improving test data prioritization for identifying fault-revealing data. Using the MUT and DE models as base models, we derived uncertainty metrics from their outputs for test inputs. These metrics served as inputs for the meta-model. The meta-model was then trained to predict whether the tested DNN

model would correctly classify a given test input, thereby improving the prioritization process. We evaluated the effectiveness of meta-model based test data prioritization with test datasets with and without OOD data in Experiment 5. In the following two subsections, we discuss the findings from the results of this experiment.

Prioritization without OOD Data

We evaluated the effectiveness of three Meta-Models with different input features as given in Table 8. These Meta-Models effectively combined the DE Variation Score with the uncertainty metrics obtained from the outputs of the MUT. While the DE Variation score is effective in prioritizing test data that includes adversarial or OOD data, the uncertainty metrics gathered from MUT offer a lower computational cost compared to the DE method and remain effective in identifying uncertainty in both original and synthetic test data.

The test data prioritization using Meta-Models demonstrated that this approach outperforms other prioritization techniques across all test datasets and DNN models as indicated by the APFD values in Tables 24-26. The best APFD values belong to Meta-Models overwhelmingly. The Meta-Models exhibit a very strong correlation with misclassifications of MNIST, with a value of 0.81, and a strong correlation ranging from 0.59 to 0.71 for the misclassifications of CIFAR-10 and CIFAR-100 as demonstrated in Figure 36.

Furthermore, Figure 34 highlights the effectiveness of the meta-model approach, which consistently achieves the best RAUC scores. Prioritization performed using Meta-Models exhibited narrower distributions and higher RAUC values compared to other methods. For instance, with the LeNet-1 and LeNet-5 models, the RAUC values for meta-models on the Original Test Dataset range between [0.76, 0.99] and [0.78, 0.99], respectively. In contrast, the RAUC values for the MUT-* methods range between [0.38, 0.97] and [0.57, 0.98] under the same conditions. The narrow distribution of RAUC values underlines the high level of performance consistency across various dataset sizes tested. The meta-model approach demonstrates superiority on the Adversarial Attack test dataset, with RAUC values approaching the ideal score of 1.0, with the representation of as lines more than boxes in the plots shown in Figure 34. This reflects the method's performance across several DNN models, including LeNet-1, LeNet-5, VGG-19, and ResNet-32. Additionally, the RAUC results indicate the effectiveness of the DE-Variation Score method on Adversarial Attacks test datasets.

DNNs can make incorrect predictions with high confidence for adversarial data. Adversarial Attack methods perform subtle manipulations on the data that mislead the models into making overly confident yet erroneous predictions. This high confidence in incorrect predictions poses risks in the real-world deployment of these systems. Consequently, the effective prioritization of test data to identify fault-revealing instances in adversarial datasets is crucial for uncovering the model's weaknesses.

Notably, the baseline DSA method, which is a distance-based approach relying on the difference between the activation trace of an input and the activation trace of the training data, is successful at identifying data that deviates from the training set and is likely to be misclassified by the DNN model in CIFAR-100 Adversarial Attacks test dataset. The DE Variation Score metric remains effective in this dataset by leveraging the differences between the outputs of MUT and DE models. Consequently, as the DE Variation Score is one of the input features of Meta-Models, they also perform effectively, achieving the third-best APFD value in the CIFAR-100 Adversarial Attacks test dataset.

The results of the experiment show that the meta-model based test data prioritization approach is the most effective method in identifying test data that reveal faults, outperforming all other methods evaluated. All three Meta-Models demonstrate similar performance, with differences in their APFD values being less than 0.004. It is important to note that none of the meta-models consistently achieved the best results across all test datasets. Statistical analysis showed that there were no significant differences in the performances of Meta-Model-1, Meta-Model-2, and Meta-Model-3. However, due to the computational simplicity of the Least Confidence metric used in Meta-Model-1 compared to the Margin and Entropy metrics used in the other meta-models, Meta-Model-1 may be preferred over others.

Prioritization with OOD Data

The inclusion of OOD data in test datasets reduced the effectiveness of all prioritization methods. As shown in Table 33, the APFD values decreased for datasets containing OOD data compared to the test datasets without OOD data. The higher APFD values for LeNet-1 and LeNet-5, compared to those for CIFAR-10 and CIFAR-100, can be attributed to the higher accuracy of these models on OOD datasets, as presented in Table 32.

The number of classes in datasets is a factor affecting the uncertainty of the model, which in turn influences the accuracy of the models. As the number of classes increases, the probability distribution across the classes becomes more spread out. This leads to increased uncertainty since the model is required to differentiate among more potential outcomes. The model needs to learn more complex boundaries between a higher number of classes, which is challenging. This causes higher uncertainty since the model struggle to learn the subtle differences between classes. Additionally, for datasets with the same number of classes, the complexity of the image is an influential factor in the uncertainty of the model. The size of the input images increases the complexity of the input, potentially increasing uncertainty. More complex models better capture and learn patterns in datasets with high complexity and a high number of classes. For this purpose, we employed more advanced models for the CIFAR-10 and CIFAR-100 datasets compared to MNIST. In terms of accuracy and APFD values, the test datasets ranked as follows: MNIST, CIFAR-10, and CIFAR-100.

Even with the inclusion of OOD data, Meta-Models continued to outperform other approaches. Prioritizing with Meta-Models led to the highest APFD results in 14 of 18 instances. This consistent performance shows that this method remains effective across various types of test datasets, including those with OOD data.

The performance of the DE Variation Score was notable. While meta-models showed strong performance across all dataset sizes, the DE Variation Score was particularly effective for smaller datasets, as shown in Figure 37. Moreover, it displayed moderate to strong correlations with misclassifications, emphasizing its potential in identifying misclassifications in datasets with OOD data, as depicted in Figure 36. These findings demonstrate its ability to prioritize fault-revealing inputs in test datasets with OOD data.

Moreover, the inclusion of OOD data led to successful prioritization using the baseline LSA and DSA methods for the CIFAR-10 and CIFAR-100 test datasets. This success can be attributed to the different data distribution of OOD data compared to the training data. The LSA and DSA methods prioritize data that deviates from the training dataset, causing a "surprise" effect on the tested model and enhancing their prioritization effectiveness.

8.2 OOD Data Detection

To detect OOD data in test datasets, we employed existing OOD detection methods from the literature. We employed two different methods, each with its own distinctive processing approach to detect the OOD data. Before applying these OOD data detection techniques to the test datasets we generated, we evaluated their effectiveness in distinguishing two selected datasets. These datasets were selected to have different features but share similar attributes like resolution and color. Both OOD detection methods showed success on these datasets.

8.2.1 OOD Detection with Variational Autoencoders

Variational Autoencoders (VAEs) are commonly used in distribution-aware DNN testing studies for both OOD data detection and generating new test inputs. We employed an OOD detection method based on VAEs to identify OOD data in test data. Then, we created test datasets consisting solely of in-distribution data and those including both in-distribution and OOD data. In Experiment 1, we applied this method to the test data generated by several methods, including GANs, image transformations, and adversarial attacks, to identify OOD data.

The results in Figure 19 indicate that adversarial attacks have a tendency to produce a high percentage of OOD data. The C&W attack generates adversarial images with smaller perturbations, even unnoticeable by humans, making them more similar to the training data distribution, while the FGSM and PGD methods generate a higher degree of perturbations on images. Consequently, nearly all images generated by FGSM and PGD attacks are identified as OOD data, whereas only a small percentage of images produced by the C&W attack are categorized as OOD.

On the other hand, the percentage of data classified as OOD varies based on the image transformation method and the specific characteristics of the data. For example, in the MNIST dataset, the Translate method resulted in the highest percentage of OOD data. This method shifts pixels without altering their values, frequently relocating digits to positions outside the center of the image, where they were typically located in the training images. VAEs are designed to replicate the input image at the output. Therefore, when encoding an image into latent space, VAEs consider not only the individual features present in the image but also their spatial relationships and positions relative to the image's edges. Consequently, even if the object within the image remains unchanged, altering its location can lead to a different encoding in latent space, thereby reducing the reconstruction probability and causing it to be categorized as OOD data. Meanwhile, Blurring produced the highest ratio of OOD data for CIFAR-10. This method alters pixel values in all color channels, causing images to deviate significantly from the training data distribution. The high OOD ratio may be attributed to the possibility of blurred images being mapped to less meaningful or incorrect positions in the latent space of VAEs, reducing the likelihood of successful image reconstruction. This issue is particularly noticeable in CIFAR-10 because it consists of colored images with three channels, whereas MNIST images are grayscale with only one channel. Thus, the change in pixel values affects the CIFAR-10 more than MNIST.

In contrast, GANs did not generate a high number of OOD data for either CIFAR-10 or MNIST. GAN models learn the features of the training data during their training process, resulting in the generation of mostly in-distribution images through the use of their Generator networks.

8.2.2 OOD Detection with Visual Transformer Models

Many studies on OOD detection have primarily focused on distinguishing far-OOD data with semantic differences. However, discriminating between near-OOD data and in-distribution data presents a greater challenge. In this context, we evaluated an OOD detection method that employs a Visual Transformer (ViT) model in Experiment-5. Our assessments, conducted using both far and near-OOD datasets, demonstrated that this method effectively distinguishes between in-distribution data and various types of OOD datasets.

Similar to the VAE-based OOD detection method, this approach identified adversarial images generated by FGSM and PGD as having the highest percentage of OOD data. Image transformation methods accounted for the second-highest percentage of OOD data. Among these transformations, Blur and Rotate resulted in the highest percentages of OOD data. The reason that rotated images are identified as OOD is related to the OOD detection process employed. The ViT model used in the detection process reshapes the input images into patches of smaller sizes and uses these patches as inputs. When the images are rotated, the content of the patches changes significantly, which leads to changes in the input and output of the ViT model. As a result, the rotated data is classified as OOD. The Blurring method applies a Gaussian filter on each pixel of the image and alters its value. The changes in the pixel values of the images affect the embeddings extracted from these images and, in turn, increase their distance to the training data embeddings. As a result, they are identified as OOD data due to high OOD score values.

In line with the VAE-based OOD detection method, the number of test data categorized as OOD from the test data generated through GANs was moderate. GAN models learn the features of the training dataset and use them to generate new images with similar features. However, GANs struggle with the CIFAR-100 dataset due to its higher number of classes compared to CIFAR-10 and MNIST. This complexity results in less realistic synthetic images and a higher OOD data ratio of 40% compared to CIFAR-10 and MNIST. Another reason for the higher ratio of OOD data in CIFAR-100 could be attributed to the different labeling processes applied between CIFAR-100 and other datasets. For the CIFAR-10 and MNIST test datasets, we manually labeled the data and removed any samples that were unrecognizable as digits by humans in the manual labeling process. In contrast, for CIFAR-100, we employed a conditional GAN model to generate labeled data. Consequently, we did not manually label or eliminate invalid data in CIFAR-100, which may have led to a higher presence of OOD data.

These findings underscore the importance of robust OOD detection methods that can effectively handle both far and near-OOD data and highlight the unique challenges posed by different types of OOD data.

8.3 Test Results Interpretation with Post-hoc Explainability Methods

By analyzing the regions of interest highlighted by the visualization methods, we gain insights into the features that the model is using to identify and classify the images. CAM images show which parts of an image the model focuses on when making predictions, revealing potential reasons for misclassifications. In Experiment 2, we examined a specific misclassification between the Cat and Dog classes in the CIFAR-10 test dataset. We employed the GradCAM, GradCAM++, and ScoreCAM methods to highlight the image regions that the DNN model focused on while making its prediction. We manually analyzed the CAM images belonging to these classes to identify the features commonly

highlighted in correctly and incorrectly classified images separately to determine the cause of incorrect predictions. From this analysis, we hypothesized that the model struggles to accurately classify a dog image when the nose region is not recognized or when additional patterns associated with other classes are present in the image. Among the three methods we employed, the ScoreCAM method highlighted smaller regions of the images, enabling a better focus on the areas that significantly influence the decisions of the DNN model.

In order to address the misclassifications, retraining of the tested DNN model using a larger dataset that includes underrepresented images can be performed. By doing so, the model can better learn to identify the unique features of each class and avoid confusing them with other classes. This retraining process has the potential to improve the accuracy of the DNN model.

Improving the DNN model and evaluating the accuracy gain is beyond the scope of our study; therefore, we did not retrain the model. Instead, we explored the potential use of post-hoc explainability methods for root cause analysis of failures in the testing of DNNs. The use of visualization techniques in DL explainability can assist developers and testers in comprehending the reasoning behind the model's decisions, leading to test failures. The insights gained from this analysis can help in planning specific actions for DNN training to address and overcome the model's weaknesses. However, this analysis requires a detailed and manual assessment performed by a domain expert.

8.4 Limitations

In our experiments, we used three widely recognized image classification datasets: MNIST, CIFAR-10, and CIFAR-100. We employed DNN models known for their high performance on these datasets, enabling us to benchmark our approach against existing studies. These datasets vary in aspects such as color (black-and-white vs. color images), class sizes, and complexity. However, the representativeness of these datasets is limited to the image domain. In future studies, other input types and DNN model architectures can be examined to generalize our approach.

Our evaluations of test data prioritization methods are limited by the adequacy of the evaluation methods we use. To evaluate the effectiveness of test data prioritization methods, we employed several distinct evaluation approaches. We used the widely accepted Average Percentage of Faults Detected (APFD) metric from software testing, which is also adopted for DNN testing, as well as the recently introduced Ratio of Area Under the Curve (RAUC) metric from DNN testing studies. Furthermore, we evaluated the Fault Detection Ratio of each approach based on the increasing size of prioritized test data. By incorporating multiple evaluation metrics, we aimed to reduce potential biases associated with relying on a single metric. Additionally, we applied correlation analysis to investigate the relationship between misclassifications and uncertainty metric values.

In OOD detection, there is no labeled data to indicate whether data points are OOD or not. The success of an OOD detection method is judged by its ability to differentiate between two datasets that are semantically different but have similar characteristics, such as resolution and color depth. However, there is no ground truth to determine under which conditions variations within the same test dataset, as opposed to differences between two separate datasets, should be considered OOD. Different OOD detection methods can yield varying results on the same datasets, and no universally accepted metric exists for comparing these approaches. Establishing standardized evaluation protocols and metrics for OOD detection, followed by benchmarking these methods on a variety of datasets, would assist

in selecting the most appropriate OOD detection method for different problem domains. A recent study offers an initial benchmark on OOD detection methods [60], but in this study, only distinct datasets were used as in-distribution and out-of-distribution datasets, and no OOD detection results were presented for data shifts in the same dataset.

Furthermore, in OOD detection, the threshold value used to distinguish OOD data from in-distribution data is crucial. The effectiveness of OOD detection is significantly influenced by the dataset chosen as OOD data when determining this threshold. To determine the threshold values for our experiments, we selected datasets that are commonly used as OOD datasets in the literature [24, 43, 121].

The Deep Ensemble method involves training several DNN models, which only needs to be conducted once, regardless of any changes in the datasets to be prioritized. While this method incurs a higher computational load than single-model methods during training, the increase is not substantial, as only five models are used in the ensemble. During data prioritization, the models are used solely for inference, and no re-training is needed. By favoring a one-time increase in computational time for the training of DE models, early detection of faults during testing could be achieved.

We employed Logistic Regression as the meta-model in our proposed test data prioritization approach. Logistic Regression is a widely used and well-understood technique for modeling the relationship between a dependent variable and one or more independent variables in binary classification tasks. Given that the meta-model in our approach is expected to output a binary prediction on whether the tested model will make the correct classification, Logistic Regression was an appropriate choice. While our approach shows success in test data prioritization, the results are inherently constrained by the capabilities of the Logistic Regression model. Exploring other types of meta-models in the future could provide additional evidence for the benefits of meta-modeling.

CHAPTER 9

CONCLUSION AND FUTURE WORK

In this dissertation, we introduced methods to facilitate the selection of test data that will lead to failures in DNN models and to simplify the identification of the causes of these failures. To achieve this, we developed novel test data prioritization techniques based on the uncertainty of the DNN model. This approach enhances DNN testing by selecting the most relevant data from an extensive unlabeled dataset. Furthermore, we explored the use of DL explainability methods to identify the root causes of incorrect predictions of the DNN model encountered during testing. Our research specifically targeted image classification models within the image domain.

We proposed a DNN test framework and focused particularly on test data selection in our study. Our goal in test data selection is to identify and prioritize test inputs that will lead the DNN model to make incorrect predictions from a large unlabeled data pool. Then, these selected data could be labeled and used in testing. Through this process, weaknesses of the DNN model can be revealed. In this context, we proposed novel test data selection methods and presented them in two methodologies.

We used the uncertainty of the DNN models against an input to prioritize the test data, giving higher priority to test data causing higher uncertainty. We evaluated the effectiveness of test data prioritization methods by assessing their ability to identify and assign higher priority to data that the DNN model predicts incorrectly. To measure the performance of these methods in various scenarios, we used an extended test dataset with synthetic data that simulates real-world conditions and potential adversarial situations.

We employed methods like Generative Adversarial Networks (GANs), adversarial attacks, and image transformations to introduce variety into the test data and represent different data distributions. Through the use of GANs, we generated test data that shared the same features with the training dataset and originated from a similar distribution. Training a GAN with the same data used to train the model being tested allowed the GAN to learn the features of the training data and generate synthetic data that closely mirrors the training distribution. Image transformations were employed to generate images that simulate real-world conditions encountered by a DNN model, such as variations in lighting, camera angles, and defocusing, which are typical in autonomous vehicle applications. Additionally, adversarial attacks were used to create data specifically designed to deceive the DNN models.

In the first methodology, we employed the state-of-the-art uncertainty quantification methods Deep Ensemble (DE) method [64] and Variational Inference with the Flipout (VI-F) [65] method. We computed the uncertainty metrics Margin, Entropy, and Least Confidence through these two methods and used them to prioritize the test data. This is the first study that uses DE and VI-F uncertainty quantification methods for selecting test data in DNN model testing. Furthermore, we utilized the Model

Under Test to obtain the same uncertainty metrics and then used these metrics to prioritize the test data. By evaluating the effectiveness of these uncertainty-based test data prioritization approaches, we answered the RQ1.

- RQ1: What is the effectiveness of uncertainty-based test selection methods in identifying and prioritizing fault-revealing data instances within datasets generated through different techniques, and how does this effectiveness vary under different conditions?

  RQ1.1: How effective are the test selection methods when the datasets exclusively consist of in-distribution data?

  RQ1.2: What variations in effectiveness are observed when the datasets contain both OOD and in-distribution data?

Our evaluation of uncertainty-based test data prioritization methods demonstrates that leveraging uncertainty metrics from Deep Ensemble, Variational Inference with Flipout (VI-F), and Model Under Test (MUT) can effectively identify fault-revealing data in various contexts. The DE method proved particularly effective for prioritizing data in synthetically produced test datasets through the use of image transformations on the original images or generated using Generative Adversarial Networks (GANs). On the other hand, uncertainty metrics obtained using the outputs of MUTs are more effective for prioritizing the test datasets generated through adversarial attacks compared to the DE method. Adversarial test data is generated by employing the tested model and its parameters with adversarial attack methods. Consequently, these data do not introduce uncertainty in DNN models used in DE or VI-F methods and are less likely to deceive these models due to their differing parameters. Therefore, uncertainty metrics derived from the class probability outputs of DE and VI-F models are ineffective at detecting data, leading to incorrect predictions within the adversarial test data. Moreover, in all scenarios, the VI-F method has the lowest performance in prioritizing fault-revealing test data. Despite the potential of VI-F, its current implementation struggled with effective uncertainty quantification, leading to lower performance compared to DE and MUT methods.

When test datasets are prioritized using uncertainty metrics calculated with DE, more than 95% of total misclassifications can be identified by using just 30% of the data for test datasets created by Image Transformations and Generative Models for MNIST. For the CIFAR-10, 45% and 55% of the test dataset are required to detect a similar percentage of misclassifications, respectively. In contrast, without any prioritization technique and with random data selection, the percentage of test data used is directly proportional to the percentage of misclassifications identified. Prioritizing test data allows the identification of misclassifications using a smaller percentage of the test dataset, reducing the amount of data that needs to be labeled to detect the same number of faults.

While uncertainty-based test data prioritization remains effective in the presence of OOD data in test datasets, their performance is reduced compared to scenarios without OOD data. The DE method generally outperforms the test data prioritization performed using uncertainty metrics obtained from MUT in scenarios with low accuracy models, while the use of uncertainty metrics from MUT performs better with high accuracy models.

In the scope of our research performed with the goal of improving test data selection through uncertainty-based prioritization, we introduced two novel uncertainty metrics. The first one is the Mahalanobis Uncertainty Score (MUS), which is a distance-based metric that quantifies a DNN model's uncertainty

for a given input by measuring its similarity to the training dataset. The second one, the Deep Ensemble(DE) Variation Score, assesses uncertainty through the variance between the predictions of DE models and the tested model for a test input.

When the MUS metric is used with the DE method, its performance in data prioritization is comparable to other uncertainty metrics - Least Confidence, Entropy, and Margin- used with DE. Although the MUS metric does not provide a significant improvement, it achieves performance similar to that of commonly used uncertainty metrics. While MUS is a robust metric that performs on par with other uncertainty metrics in test data prioritization, its computational complexity limits its practicality. Simpler metrics like Least Confidence offer better performance with lower complexity, making them more efficient choices for uncertainty-based test data prioritization.

In contrast, the DE Variation Score is highly effective at prioritizing adversarial data by leveraging the prediction variance between DE models and the MUT. However, its effectiveness diminishes with test data generated through image transformations and GANs, where other uncertainty metrics used with DE demonstrate better performance.

As a result, our experiments demonstrated that different uncertainty-based methods effectively measure model uncertainty across different types of datasets, aiding in the identification of data likely to produce incorrect predictions. However, no single metric or method consistently outperformed the others across all data types. To address this, we proposed a novel approach for prioritizing test data using a meta-model that combines the strength of multiple uncertainty metrics.

A meta-model is an advanced model that builds upon one or more base models by using their outputs as input. Meta models are particularly useful when individual base models show insufficient performance or have diverse strengths and weaknesses. By combining the outputs of multiple base models, a meta-model can achieve better performance compared to an individual model. In this study, we employed the Model Under Test (MUT) and Deep Ensemble (DE) models as base models. We derived uncertainty metrics from the outputs of both the MUT and DE models for test inputs and used these metrics as inputs for the meta-model. Then, we trained the meta-model with the goal of predicting whether the tested DNN model would correctly classify a given test input. By evaluating the test data prioritization through the use of a meta-model approach, we answered the RQ2.

- RQ2: How effective is the test prioritization approach proposed in this study, which is based on a meta-model that combines different uncertainty metrics in identifying and prioritizing fault-revealing data instances within datasets, both include and exclude OOD data?

  RQ2.1: How does the effectiveness of the proposed method change according to different labeling budgets (i.e., the number of test data to be selected and prioritized)?

Our evaluation demonstrates that meta-model-based test data prioritization is the most effective method for identifying fault-revealing test data, outperforming all other techniques across various test datasets and DNN models. Meta-Models consistently achieved the highest APFD values and showed strong correlations with misclassifications, underscoring their effectiveness. The RAUC values further highlighted the performance consistency and superiority of the meta-model approach over different test data sizes. By combining the DE Variation Score with uncertainty metrics from MUT, meta-models achieve superior performance over state-of-the-art methods, such as LSA, DSA, DeepGini, and MCP.

By integrating multiple uncertainty metrics, we successfully mitigated the limitations of individual metrics, achieving better performance compared to using a single uncertainty metric across all datasets. The Meta-Models effectively combined the DE Variation Score, which is effective in prioritizing test data that includes adversarial or out-of-distribution (OOD) data, with the MUT-derived uncertainty metrics, which offer a lower computational cost compared to the DE method and remain effective in identifying uncertainty in both original and synthetic test data.

Even with the inclusion of OOD data, Meta-Models continued to outperform other approaches. The prioritization using Meta-Models resulted in the best APFD outcomes in 14 out of 18 cases. This consistent performance indicates that this approach remains effective across different test dataset types, including those containing OOD data. Although Meta-Models performed well across all dataset sizes, the DE-Variation Score was particularly effective for smaller dataset sizes.

The distribution of the test data has a significant impact on the performance of DNNs and is an important factor in evaluating test results. Therefore, we created test datasets with a distribution-aware approach. We employed OOD data detection methods and created test datasets with and without OOD data. We used two different OOD detection methods from the literature: one based on Variational Autoencoders (VAE) and the other using Visual Transformer (ViT) models. We conducted OOD detection to identify test inputs that exhibit covariate distribution shifts—inputs that differ from the training data but retain similar semantic properties and label space. Not all transformed test inputs are categorized as OOD; this classification depends on the specific data instance, the degree of the shift, the OOD detection method used, and its parameters. Adversarial attacks and certain image transformations significantly lead to the generation of OOD data, while GAN-generated data tends to be in-distribution data. Overall, our findings highlight the variability in the results of OOD detection methods depending on the type of perturbation and dataset characteristics.

Most of the OOD detection studies have focused only on distinguishing far-OOD data with semantic differences. However, discriminating between near-OOD data and in-distribution data poses a greater challenge. Our results indicate that OOD detection with ViT models performs effectively even with near-OOD data. Through evaluations with both far and near OOD datasets, we demonstrated that this method successfully discriminates between in-distribution and different types of OOD datasets.

Lastly, we assessed the use of post-explainability methods to identify the causes of misclassifications in DNN models, a process that can be thought of as similar to debugging. We answered the RQ3 with the assessment.

- RQ3: Is it possible to use post-hoc explainability methods in DNN testing to understand the root cause of test failures (i.e., incorrect predictions of the tested DNN model)?

We conclude that visualization techniques used in DL explainability can help developers and testers understand the rationale behind model decisions leading to test failures. Class Activation Mapping (CAM) images provide insights into the model's decision-making process. However, these insights are most valuable when evaluated by experts familiar with the model's intended functionality. This analysis requires a manual and detailed assessment.

Evaluating CAM images can reveal underrepresented or overrepresented data that affect the model's decision-making. Our findings suggest that a thorough analysis of misclassified test data using visualization methods can help uncover the reasons behind failures. This approach allows for more informed

decisions during the retraining process, rather than simply adding all misclassified test data to the training dataset. Based on the insights gained from this analysis, targeted actions can be planned to address the model's weaknesses during subsequent training phases.

The framework proposed in this thesis takes a step toward systematic test data selection for DNN testing by prioritizing fault-revealing test data by leveraging the uncertainty of DNN models. As digital systems continually collect vast amounts of data, DL systems encounter increasingly large datasets. However, selecting the most relevant data from this extensive unlabeled data pool for testing the DNN is a challenging task. Our method enhances test data selection by effectively identifying and prioritizing data that is most likely to be misclassified by the DNN model. Labeling and using this prioritized test data in testing enables the detection of DNN failures at a significantly higher rate compared to randomly selecting the same amount of test data. This prioritized data is most valuable for uncovering the model's weaknesses.

Furthermore, the data distribution-aware approach employed in creating test datasets contributes to the field of distribution-aware DNN testing. Our evaluation of various test data prioritization methods across different data distribution settings revealed that the inclusion of OOD data significantly impacts the effectiveness of these methods. Additionally, the performance of DNN models notably degrades when OOD data is included in test datasets. These findings underscore the importance of considering data distribution in the development of test data selection strategies and evaluation of test results. In this context, the proposed meta-model based test data prioritization approach, which combines the strength of multiple uncertainty metrics, overcomes the limitations of individual metrics and improves the effectiveness in a variety of scenarios, including test datasets with or without OOD data.

Future Work

In this study, we employed a Logistic Regression model as the meta-model to prioritize test data. Future research could benefit from exploring different types of meta-models and incorporating alternative metrics as inputs to these models to enhance the test data prioritization.

We introduced an adaptable test data selection framework through the use of meta-models, which we initially applied to image classification with different datasets and DNN architectures in this study. In future studies, the adaption of the proposed method to a wider range of model architectures, input types, and application domains can be explored. Customizing our method to fit these diverse settings will involve specific modifications.

- Adaptation to Other DNN Model Architectures: The foundational components of our method, such as uncertainty metrics and the meta-model, can be adapted to other types of classification models beyond traditional DNNs. For instance, models like Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) networks could benefit from these adaptations.

- Customization for Different Input Types: Extending our approach to handle different types of input data, such as text or audio, requires substantial changes in the methods used for test data generation and OOD detection. For example, when dealing with text inputs, techniques like tokenization, embeddings, and natural language processing (NLP) frameworks would need to be integrated into the test data generation process. Similarly, OOD detection methods would need to be tailored to identify anomalies in text data, possibly using techniques such as semantic similarity measures or syntactic anomaly detection.

- Expansion to Other Application Areas: Although our approach has been validated for image classification, its underlying principles can be applied to other domains. This expansion would necessitate the development and selection of new uncertainty metrics specific to those applications.

By investigating these potential adaptations in detail, future studies can significantly enhance and expand the applicability of our data selection method. This will improve its utility across different domains, ultimately leading to more efficient and effective DNN testing processes. This comprehensive approach promises early fault detection, contributing to the robust development and deployment of deep learning systems.

Moreover, while showing potential, further investigation is required regarding the application of post-hoc explainability methods with test data that lead to failure in DNN models, as well as the retraining of models based on the insights gained from these analyses. In the future, more comprehensive studies can be conducted to explore how explainability methods in DL can guide developers to the cause of DNN failures instead of relying on manual review.

Concept drift is a phenomenon that impacts the performance of DNN models. Concept drift refers to changes in the underlying relationship between input data and labels of these data [141]. These changes make the learned patterns by the model become outdated as the behavior it aims to predict evolves. On the other hand, data distribution shift, particularly covariate shift, involves changes in the statistical properties of the input data distribution without affecting the input-output relationship. Addressing concept drift requires two main steps: drift detection and model adaptation to learn new patterns [142]. Once the model is updated, it should be tested with test datasets that reflect the new input-output relationship. Our proposed solution for test data selection can be helpful in selecting these data. The proposed approach can be integrated into a pipeline designed to manage concept drift. In this pipeline, continuous monitoring of the data in the deployment environment is essential for detecting concept drift. Following detection, the model is adapted to accommodate the new data patterns. Finally, the updated model is tested using the test data selected through our proposed method, ensuring it accurately reflects the new input-output relationships.

REFERENCES

[1] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[2] W. G. Hatcher and W. Yu, "A survey of deep learning: Platforms, applications and emerging research trends," *IEEE Access*, vol. 6, pp. 24411–24432, 2018.

[3] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019.

[4] J. Chandrasekaran, T. Cody, N. McCarthy, E. Lanus, and L. Freeman, "Test & evaluation best practices for machine learning-enabled systems," *arXiv preprint arXiv:2310.06800*, 2023.

[5] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, pp. 5193–5254, 2020.

[6] "Ieee standard glossary of software engineering terminology," *IEEE Std 610.12-1990*, pp. 1–84, 1990.

[7] M. Felderer and R. Ramler, "Quality assurance for ai-based systems: Overview and challenges (introduction to interactive session)," in *Software Quality: Future Perspectives on Software Engineering Quality: 13th International Conference, SWQD 2021, Vienna, Austria, January 19–21, 2021, Proceedings 13*, pp. 33–42, Springer, 2021.

[8] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda, "A survey of autonomous driving: Common practices and emerging technologies," *IEEE Access*, vol. 8, pp. 58443–58469, 2020.

[9] J. Ker, L. Wang, J. Rao, and T. Lim, "Deep learning applications in medical image analysis," *IEEE Access*, vol. 6, pp. 9375–9389, 2018.

[10] A. Loquercio, A. I. Maqueda, C. R. Del-Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 1088–1095, 2018.

[11] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, pp. 303–314, 2018.

[12] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *Proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

[13] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 739–743, 2018.

[14] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *Proceedings of the International Conference on Machine Learning*, pp. 4901–4911, PMLR, 2019.

[15] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. 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*, pp. 146–157, 2019.

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

[17] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *Proceedings of the 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 132–142, IEEE, 2018.

[18] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "Deepbillboard: Systematic physical-world testing of autonomous driving systems," in *Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 347–358, IEEE, 2020.

[19] S. Kang, R. Feldt, and S. Yoo, "Sinvad: Search-based image space navigation for dnn image classifier test input generation," in *Proceedings of the 42nd IEEE/ACM International Conference on Software Engineering Workshops*, pp. 521–528, 2020.

[20] T. Byun, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Manifold-based test generation for image classifiers," in *Proceedings of the IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 15–22, IEEE, 2020.

[21] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, vol. 48, no. 1, pp. 1–36, 2020.

[22] J. Kim, R. Feldt, and S. Yoo, "Guiding deep learning system testing using surprise adequacy," in *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 1039–1049, IEEE, 2019.

[23] W. Ma, M. Papadakis, A. Tsakmalis, M. Cordy, and Y. L. Traon, "Test selection for deep learning systems," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 2, pp. 1–22, 2021.

[24] Q. Hu, Y. Guo, M. Cordy, X. Xie, L. Ma, M. Papadakis, and Y. Le Traon, "An empirical study on data distribution-aware test selection for deep learning enhancement," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–30, 2022.

[25] Y. Feng, Q. Shi, X. Gao, J. Wan, C. Fang, and Z. 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*, pp. 177–188, 2020.

[26] W. Shen, Y. Li, L. Chen, Y. Han, Y. Zhou, and B. Xu, "Multiple-boundary clustering and prioritization to promote neural network retraining," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 410–422, 2020.

[27] Y. Shi, B. Yin, Z. Zheng, and T. Li, "An empirical study on test case prioritization metrics for deep neural networks," in *Proceedings of the 21st IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pp. 157–166, IEEE, 2021.

[28] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 397–409, IEEE, 2021.

[29] T. Byun, V. Sharma, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Input prioritization for testing neural networks," in *Proceedings of the IEEE International Conference On Artificial Intelligence Testing (AITest)*, pp. 63–70, IEEE, 2019.

[30] Y. Li, M. Li, Q. Lai, Y. Liu, and Q. Xu, "Testrank: Bringing order into unlabeled test instances for deep learning tasks," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20874–20886, 2021.

[31] Z. Aghababaeyan, M. Abdellatif, M. Dadkhah, and L. Briand, "Deepgd: A multi-objective black-box test selection approach for deep neural networks," *ACM Transactions on Software Engineering and Methodology*, 2023.

[32] J. Zhou, F. Li, J. Dong, H. Zhang, and D. Hao, "Cost-effective testing of a deep learning model through input reduction," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 289–300, IEEE, 2020.

[33] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp. 120–131, 2018.

[34] Z. Wang, H. You, J. Chen, Y. Zhang, X. Dong, and W. Zhang, "Prioritizing test inputs for deep neural networks via mutation analysis," in *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 397–409, IEEE, 2021.

[35] M. Weiss and P. 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*, pp. 139–150, 2022.

[36] K. Zhang, Y. Zhang, L. Zhang, H. Gao, R. Yan, and J. Yan, "Neuron activation frequency based test case prioritization," in *Proceedings of the International Symposium on Theoretical Aspects of Software Engineering (TASE)*, pp. 81–88, IEEE, 2020.

[37] R. Yan, Y. Chen, H. Gao, and J. Yan, "Test case prioritization with neuron valuation based pattern," *Science of Computer Programming*, vol. 215, p. 102761, 2022.

[38] Q. Hu, Y. Guo, X. Xie, M. Cordy, L. Ma, M. Papadakis, and Y. Le Traon, "Test optimization in dnn testing: a survey," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 4, pp. 1–42, 2024.

[39] J. Bernhard, T. Schulik, M. Schutera, and E. Sax, "Adaptive test case selection for dnn-based perception functions," in *Proceedings of the IEEE International Symposium on Systems Engineering (ISSE)*, pp. 1–7, IEEE, 2021.

[40] B. Taskazan, J. Navratil, M. Arnold, A. Murthi, G. Venkataraman, and B. Elder, "Not your grandfathers test set: Reducing labeling effort for testing," *arXiv preprint arXiv:2007.05499*, 2020.

[41] P. Zhang, Q. Dai, and P. Pelliccione, "Cagfuzz: Coverage-guided adversarial generative fuzzing testing of deep learning systems," *arXiv preprint arXiv:1911.07931*, 2019.

[42] C. Tao, Y. Tao, H. Guo, Z. Huang, and X. Sun, "Dlregion: coverage-guided fuzz testing of deep neural networks with region-based neuron selection strategies," *Information and Software Technology*, vol. 162, p. 107266, 2023.

[43] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-aware testing of neural networks using generative models," in *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, pp. 226–237, IEEE, 2021.

[44] F. Toledo, D. Shriver, S. Elbaum, and M. B. Dwyer, "Distribution models for falsification and verification of dnns," in *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 317–329, IEEE, 2021.

[45] W. Huang, X. Zhao, A. Banks, V. Cox, and X. Huang, "Hierarchical distribution-aware testing of deep learning," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 2, pp. 1–35, 2023.

[46] D. Berend, X. Xie, L. Ma, L. Zhou, Y. Liu, C. Xu, and J. Zhao, "Cats are not fish: Deep learning testing calls for out-of-distribution awareness," in *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pp. 1041–1052, 2020.

[47] B. Gezici and A. K. Tarhan, "Systematic literature review on software quality for ai-based software," *Empirical Software Engineering*, vol. 27, no. 3, p. 66, 2022.

[48] S. Ghamizi, M. Cordy, Y. Guo, M. Papadakis, *et al.*, "Hazards in deep learning testing: Prevalence, impact and recommendations," *arXiv preprint arXiv:2309.05381*, 2023.

[49] W. Sun, M. Yan, Z. Liu, and D. Lo, "Robust test selection for deep neural networks," *IEEE Transactions on Software Engineering*, vol. 49, no. 12, pp. 5250–5278, 2023.

[50] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," in *International Conference on Learning Representations*, 2018.

[51] E. T. Barr, M. Harman, P. McMinn, M. Shahbaz, and S. Yoo, "The oracle problem in software testing: A survey," *IEEE transactions on software engineering*, vol. 41, no. 5, pp. 507–525, 2014.

[52] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th international symposium on software reliability engineering (ISSRE)*, pp. 100–111, IEEE, 2018.

[53] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, 2019.

[54] A. Hopkins and S. Booth, "Machine learning practices outside big tech: How resource constraints challenge responsible development," in *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, pp. 134–145, 2021.

[55] B. Settles, "Active learning literature survey," *Machine Learning*, vol. 15, no. 2, pp. 201–221, 1994.

[56] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. 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*, pp. 499–509, 2019.

[57] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan, "Practical accuracy estimation for efficient deep neural network testing," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 29, no. 4, pp. 1–35, 2020.

[58] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *arXiv preprint arXiv:2110.11334*, 2021.

[59] D. Hendrycks and K. Gimpel, "A baseline for detecting misclassified and out-of-distribution examples in neural networks," in *International Conference on Learning Representations*, 2016.

[60] J. Yang, P. Wang, D. Zou, Z. Zhou, K. Ding, W. Peng, H. Wang, G. Chen, B. Li, Y. Sun, *et al.*, "Openood: Benchmarking generalized out-of-distribution detection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 32598–32611, 2022.

[61] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 1110–1121, 2020.

[62] A. B. Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. García, S. Gil-López, D. Molina, R. Benjamins, *et al.*, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information fusion*, vol. 58, pp. 82–115, 2020.

[63] F.-L. Fan, J. Xiong, M. Li, and G. Wang, "On interpretability of artificial neural networks: A survey," *IEEE Transactions on Radiation and Plasma Medical Sciences*, vol. 5, no. 6, pp. 741–760, 2021.

[64] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," *Advances in neural information processing systems*, vol. 30, pp. 6402–6413, 2017.

[65] Y. Wen, P. Vicol, J. Ba, D. Tran, and R. Grosse, "Flipout: Efficient pseudo-independent weight perturbations on mini-batches," in *Proceedings of the International Conference on Learning Representations*, pp. 1–16, 2018.

[66] V. Mosin, M. Staron, D. Durisic, F. G. de Oliveira Neto, S. K. Pandey, and A. C. Koppisetty, "Comparing input prioritization techniques for testing deep learning algorithms," in *Proceedings of the 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 76–83, IEEE, 2022.

[67] X. Dang, Y. Li, M. Papadakis, J. Klein, T. F. Bissyandé, and Y. Le Traon, "Graphprior: mutation-based test input prioritization for graph neural networks," *ACM Transactions on Software Engineering and Methodology*, vol. 33, no. 1, pp. 1–40, 2023.

[68] Y. Tao, C. Tao, H. Guo, and B. Li, "Tpfl: Test input prioritization for deep neural networks based on fault localization," in *Proceedings of the International Conference on Advanced Data Mining and Applications*, pp. 368–383, Springer, 2022.

[69] T. Chen, J. Navratil, V. Iyengar, and K. Shanmugam, "Confidence scoring using whitebox meta-models with linear classifier probes," in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics* (K. Chaudhuri and M. Sugiyama, eds.), vol. 89 of *Proceedings of Machine Learning Research*, pp. 1467–1475, PMLR, 16–18 Apr 2019.

[70] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[71] J. Recker, *Scientific research in information systems: a beginner's guide*. Springer Nature, 2021.

[72] P. Offermann, O. Levina, M. Schönherr, and U. Bub, "Outline of a design science research process," in *Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology*, pp. 1–11, 2009.

[73] P. Runeson, E. Engström, and M.-A. Storey, "The design science paradigm as a frame for empirical software engineering," *Contemporary empirical methods in software engineering*, pp. 127–147, 2020.

[74] M.-A. Storey, E. Engstrom, M. Höst, P. Runeson, and E. Bjarnason, "Using a visual abstract as a lens for communicating and promoting design science research in software engineering," in *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp. 181–186, IEEE, 2017.

[75] S. Martínez-Fernández, J. Bogner, X. Franch, M. Oriol, J. Siebert, A. Trendowicz, A. M. Vollmer, and S. Wagner, "Software engineering for ai-based systems: a survey," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 2, pp. 1–59, 2022.

[76] G. Giray, "A software engineering perspective on engineering machine learning systems: State of the art and challenges," *Journal of Systems and Software*, vol. 180, p. 111031, 2021.

[77] S. Amershi, A. Begel, C. Bird, R. DeLine, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, pp. 291–300, IEEE, 2019.

[78] X. Zhang, Y. Yang, Y. Feng, and Z. Chen, "Software engineering practice in the development of deep learning applications," *arXiv preprint arXiv:1910.03156*, 2019.

[79] T. Zhang, C. Gao, L. Ma, M. Lyu, and M. Kim, "An empirical study of common challenges in developing deep learning applications," in *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 104–115, IEEE, 2019.

[80] A. Arpteg, B. Brinne, L. Crnkovic-Friis, and J. Bosch, "Software engineering challenges of deep learning," in *2018 44th euromicro conference on software engineering and advanced applications (SEAA)*, pp. 50–59, IEEE, 2018.

[81] H. B. Braiek and F. Khomh, "On testing machine learning programs," *Journal of Systems and Software*, vol. 164, p. 110542, 2020.

[82] X. Huang, D. Kroening, M. Kwiatkowska, W. Ruan, Y. Sun, E. Thamo, M. Wu, and X. Yi, "Safety and trustworthiness of deep neural networks: A survey (2019)," *arXiv preprint arXiv:1812.08342*, 2019.

[83] D. Marijan, A. Gotlieb, and M. K. Ahuja, "Challenges of testing machine learning based systems," in *2019 IEEE international conference on artificial intelligence testing (AITest)*, pp. 101–102, IEEE, 2019.

[84] D. Marijan and A. Gotlieb, "Software testing for machine learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 13576–13582, 2020.

[85] P. C. Jorgensen, *Software testing: a craftsman's approach*. Auerbach Publications, 2013.

[86] L. A. Johnson *et al.*, "Do-178b, software considerations in airborne systems and equipment certification," *Crosstalk, October*, vol. 199, pp. 11–20, 1998.

[87] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Structural test coverage criteria for deep neural networks," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 5s, pp. 1–23, 2019.

[88] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. 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*, pp. 851–862, 2020.

[89] Z. Li, X. Ma, C. Xu, and C. Cao, "Structural coverage criteria for neural networks could be misleading," in *Proceedings of the 41st IEEE/ACM International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, pp. 89–92, IEEE, 2019.

[90] Y. Dong, P. Zhang, J. Wang, S. Liu, J. Sun, J. Hao, X. Wang, L. Wang, J. Dong, and T. Dai, "An empirical study on correlation between coverage and robustness for deep neural networks," in *Proceedings of the 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pp. 73–82, IEEE, 2020.

[91] Z. Li, X. Ma, C. Xu, and C. 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)*, pp. 89–92, 2019.

[92] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of the International Conference on Machine Learning*, pp. 1050–1059, PMLR, 2016.

[93] J. Navratil, M. Arnold, and B. Elder, "Uncertainty prediction for deep sequential regression using meta models," *arXiv preprint arXiv:2007.01350*, 2020.

[94] M. Shen, Y. Bu, P. Sattigeri, S. Ghosh, S. Das, and G. Wornell, "Post-hoc uncertainty learning using a dirichlet meta-model," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, pp. 9772–9781, 2023.

[95] A. Kendall and Y. Gal, "What uncertainties do we need in bayesian deep learning for computer vision?," *Advances in neural information processing systems*, vol. 30, pp. 5574–5584, 2017.

[96] Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. Dillon, B. Lakshminarayanan, and J. Snoek, "Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift," *Advances in neural information processing systems*, vol. 32, pp. 13991–14002, 2019.

[97] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 427–436, 2015.

[98] M. Abdar, F. Pourpanah, S. Hussain, D. Rezazadegan, L. Liu, M. Ghavamzadeh, P. Fieguth, X. Cao, A. Khosravi, U. R. Acharya, *et al.*, "A review of uncertainty quantification in deep learning: Techniques, applications and challenges," *Information Fusion*, vol. 76, pp. 243–297, 2021.

[99] M. Henne, A. Schwaiger, K. Roscher, and G. Weiss, "Benchmarking uncertainty estimation methods for deep learning with safety-related metrics.," in *Proceedings of the SafeAI@ AAAI*, pp. 83–90, 2020.

[100] J. Caldeira and B. Nord, "Deeply uncertain: comparing methods of uncertainty quantification in deep learning algorithms," *Machine Learning: Science and Technology*, vol. 2, no. 1, 2020.

[101] N. Ståhl, G. Falkman, A. Karlsson, and G. Mathiason, "Evaluation of uncertainty quantification in deep learning," in *Proceedings of the International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pp. 556–568, Springer, 2020.

[102] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," *Advances in neural information processing systems*, vol. 30, 2017.

[103] M. Sundararajan, A. Taly, and Q. Yan, "Axiomatic attribution for deep networks," in *International conference on machine learning*, pp. 3319–3328, PMLR, 2017.

[104] S. Bach, A. Binder, G. Montavon, F. Klauschen, K.-R. Müller, and W. Samek, "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation," *PloS one*, vol. 10, no. 7, p. e0130140, 2015.

[105] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning deep features for discriminative localization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2921–2929, 2016.

[106] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[107] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that?," *arXiv preprint arXiv:1611.07450*, 2016.

[108] A. Chattopadhay, A. Sarkar, P. Howlader, and V. N. Balasubramanian, "Grad-cam++: Generalized gradient-based visual explanations for deep convolutional networks," in *Proceedings of the IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 839–847, IEEE, 2018.

[109] H. Wang, Z. Wang, M. Du, F. Yang, Z. Zhang, S. Ding, P. Mardziel, and X. Hu, "Score-cam: Score-weighted visual explanations for convolutional neural networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 24–25, 2020.

[110] M. T. Ribeiro, S. Singh, and C. Guestrin, "Anchors: High-precision model-agnostic explanations," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[111] A. Dhurandhar, P.-Y. Chen, R. Luss, C.-C. Tu, P. Ting, K. Shanmugam, and P. Das, "Explanations based on the missing: Towards contrastive explanations with pertinent negatives," *Advances in neural information processing systems*, vol. 31, 2018.

[112] O. Li, H. Liu, C. Chen, and C. Rudin, "Deep learning for case-based reasoning through prototypes: A neural network that explains its predictions," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[113] P. Lertvittayakumjorn and F. Toni, "Explanation-based human debugging of nlp models: A survey," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1508–1528, 2021.

[114] J. Adebayo, M. Muelly, I. Liccardi, and B. Kim, "Debugging tests for model explanations," *Advances in Neural Information Processing Systems*, vol. 33, pp. 700–712, 2020.

[115] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial networks," *Communications of the ACM*, vol. 63, pp. 139–144, Oct 2020.

[116] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[117] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[118] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[119] J. Mena, O. Pujol, and J. Vitrià, "A survey on uncertainty estimation in deep learning classification systems from a bayesian perspective," *ACM Computing Surveys (CSUR)*, vol. 54, no. 9, pp. 1–35, 2021.

[120] D. Demir, A. B. Can, and E. Surer, "Test selection for deep neural networks using meta-models with uncertainty metrics," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ACM, 2024.

[121] S. Fort, J. Ren, and B. Lakshminarayanan, "Exploring the limits of out-of-distribution detection," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7068–7081, 2021.

[122] K. Lee, K. Lee, H. Lee, and J. Shin, "A simple unified framework for detecting out-of-distribution samples and adversarial attacks," *Advances in neural information processing systems*, vol. 31, 2018.

[123] Y. LeCun, "The mnist database of handwritten digits," 1998.

[124] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," tech. rep., Univ. Toronto, Toronto, ON, Canada, 2009.

[125] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, IEEE, 1998.

[126] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[127] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proceedings of the International Conference on Learning Representations*, pp. 1–14, 2015.

[128] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[129] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

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

[131] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *Proceedings of the International Conference on Learning Representations*, 2018.

[132] D. Demir, A. B. Can, and E. Surer, "Distribution aware testing framework for deep neural networks," *IEEE Access*, vol. 11, pp. 119481–119505, 2023.

[133] N. Papernot, F. Faghri, N. Carlini, I. Goodfellow, R. Feinman, A. Kurakin, C. Xie, Y. Sharma, T. Brown, A. Roy, *et al.*, "Technical report on the cleverhans v2. 1.0 adversarial examples library," *arXiv preprint arXiv:1610.00768*, 2016.

[134] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[135] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," in *Proceedings of the NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011.

[136] M.-I. Nicolae, M. Sinn, M. N. Tran, B. Buesser, A. Rawat, M. Wistuba, V. Zantedeschi, N. Baracaldo, B. Chen, H. Ludwig, I. Molloy, and B. Edwards, "Adversarial robustness toolbox v1.2.0," *CoRR*, vol. 1807.01069, 2018.

[137] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," in *International Conference on Learning Representations*, 2021.

[138] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, IEEE, 2009.

[139] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha, "Deep learning for classical japanese literature," *arXiv preprint arXiv:1812.01718*, 2018.

[140] J. Kim, R. Feldt, and S. Yoo, "Evaluating surprise adequacy for deep learning system testing," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–29, 2023.

[141] J. Gama, I. Žliobaitė, A. Bifet, M. Pechenizkiy, and A. Bouchachia, "A survey on concept drift adaptation," *ACM computing surveys (CSUR)*, vol. 46, no. 4, pp. 1–37, 2014.

[142] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang, "Learning under concept drift: A review," *IEEE transactions on knowledge and data engineering*, vol. 31, no. 12, pp. 2346–2363, 2018.

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: Demir, Demet

EDUCATION

Degree	Institution	Year of Graduation
Ph.D.	Information Systems, METU	2024
M.Sc.	Computer Engineering, METU	2003
B.Sc	Computer Engineering, METU	2000

PROFESSIONAL EXPERIENCE

Company	Position	Year
PROVEN	Head of Engineering	2017-Present
AYESAŞ	Senior Technical Lead/Project Manager	2006-2017
MILSOFT	Software Engineer	2000-2006

PUBLICATIONS

- D. Demir, A. Betin Can, and E. Surer, “Distribution Aware Testing Framework for Deep Neural Networks,” *IEEE Access*, vol. 11, pp. 119481–119505, 2023.
- D. Demir, A. Betin Can, and E. Surer, “Test Selection for Deep Neural Networks using Meta-Models with Uncertainty Metrics,” in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, ACM, 2024.
- Ö. Özdemir and D. Demir, "TEST-INT: A Testing Platform for Deep Learning Models," in *15th Turkish National Software Engineering Symposium (UYMS)*, pp. 1-3, IEEE, 2021, doi: 10.1109/UYMS54260.2021.9659790.