PhD-FSTM-2023-05
The Faculty of Sciences, Technology and Medicine

DISSERTATION

Defence held on 03/05/2023 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG
EN INFORMATIQUE

by

Hazem MOHAMED FAHMY FARGHALY MOHAMED
Born on 17 November 1992 in Cairo (Egypt)

SUPPORTING SAFETY ANALYSIS OF DEEP NEURAL NETWORKS WITH AUTOMATED DEBUGGING AND REPAIR

Acknowledgement

I would like to dedicate this dissertation to the loving memory of my dear mother, Dr. Hanan Mahmoud Abd El-Azeem. Her unconditional love, and sacrifices have been the guiding light throughout my life and educational pursuits that enabled me to embark on this endeavor. This dissertation is a testament to your profound impact on my life, and I dedicate its completion to you. May your soul find eternal peace.

Lastly, I extend my gratitude to my advisors, professors, and mentors who have shared their expertise, guidance, and wisdom. Your mentorship and insights have shaped my academic growth and enriched my understanding of the subject matter.

Hazem Fahmy
University of Luxembourg
May 14, 2023

Abstract

Deep Neural Networks (DNNs) are increasingly adopted in a variety of safety-critical systems, including Advanced Driver-Assistance Systems (ADAS), Automated Driving Systems (ADS), medical devices, and industrial control systems. DNNs are particularly useful in the perception layer of these systems, where they are used to analyze images and other sensor data to make decisions. However, ensuring the functional safety of DNN-based components remains a challenge because performance metrics for machine learning algorithms, such as accuracy evaluation, do not enable distinguishing the different scenarios leading to DNN failures (i.e., the unsafe execution scenarios of a DNN under test).

To address these challenges, we propose four approaches which leverage the intuition that unsafe scenarios might be cost-effectively identified by automatically clustering images that lead to DNN failures and present commonalities. We call such image clusters root cause clusters (RCCs). The proposed approaches are:

- *Heatmap-based Unsupervised Debugging of DNNs (HUDD)*, a white-box approach that identifies RCCs by applying a clustering algorithm to heatmaps capturing the relevance of every DNN neuron on the DNN outcome. HUDD also composes a dataset for retraining and improving DNNs by selecting images from an improvement set based on their closeness to the generated RCCs.

- *Simulator-based Explanation for DNN failurEs (SEDE)*, a search-based approach that automatically generates explicit descriptions for hazard-triggering events from real-world images. SEDE provides such descriptions in terms of logical expressions constraining the configuration parameters of the simulator used to train the DNN.

- *Safety Analysis using Feature Extraction (SAFE)*, a black-box approach that generates RCCs without relying on DNN internal information. SAFE combines transfer-learning models with dimensionality reduction techniques to extract important features from failure-inducing inputs.

- A pipeline for the generation of RCCs that can integrate different machine learning components. We rely on such a pipeline to assess the effectiveness of composing transfer-learning models, autoencoders, heatmaps of neuron relevance, dimensionality reduction techniques, and various clustering algorithms to derive RCCs.

The proposed approaches have been successfully evaluated on seven case study DNNs from the automotive domain defined in collaboration with our industry partner IEE.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Context and Motivation

RECENTLY in 2022, the World Health Organization (WHO) has stated a critical fact that, annually, 1.3 million people die as a result of road traffic crashes. Additionally, road traffic injuries are the foremost cause of death among children and teenagers between the ages of 5 and 29 [1]. Moreover, the European Road Safety Observatory estimated, in their annual report in 2021 [2], that $18,800$ fatalities are caused by $757,566$ road accidents each year within the European Union only (42 fatalities per million inhabitants), leaving more than 0.5 million people injured.

The high number of annual road traffic fatalities emphasizes the need for safety and reliability in automotive systems. To address these issues, DNN-based systems are increasingly proposed as technical solutions to replace humans in various domains, including autonomous driving [3, 4], computer vision [5], and natural language processing [6, 7]. However, although DNNs are powerful tools for solving complex problems, their complex structure can make it challenging for engineers to understand how they make predictions, especially in the context of automotive systems and medical diagnosis [8] where human lives are at stake. Therefore, the use of DNNs also poses safety problems and ensuring the safety and reliability of DNN-based systems is crucial for their successful integration into automotive systems and many other fields such as finance [9], and industrial control [10].

In safety-critical contexts, DNNs need to be trustworthy and their predictions, especially erroneous ones, should be explainable in order to build trust and credibility among users, especially in contexts that require certification.

To increase trust in DNN models, we must understand why they behave the way they do [11]. Explanation methods aim at making Neural Network (NN) decisions trustworthy [12], and several methods have been proposed in the literature (see Section 2.4). However, there is often a trade-off between the performance of machine learning methods and their explainability. For example, deep learning methods tend to be highly accurate but are often less explainable compared to decision tree methods, which are typically less accurate but more explainable [13].

Standards such as ISO 26262 [14] and ISO/PAS 21448 [15] regulate the functional safety analysis practices that should be put in place for safety-critical automotive systems; similar standards exists for other sectors [16]. They require the identification of situations in which a system may be unsafe (e.g. providing erroneous and unsafe outputs), the identification of countermeasures (e.g. using multiple sensors or removing redundant components), and demonstrating that unsafe conditions are unlikely to occur. Unfortunately, traditional safety analysis methods rely on code inspection or program analysis to demonstrate the absence of unsafe conditions in software [14] and are thus applicable only to traditional software components whose decisions are encoded in their source code, not to DNNs whose decisions derive from hundred mathematical operations performed in multiple DNN layers. Further, performance metrics for machine learning algorithms such as accuracy evaluation are not sufficient to address safety standards requirements. Indeed, according to ISO/PAS 21448, engineers can set a quantitative target for accuracy evaluation to demonstrate that unsafe situations are unlikely; however, ISO/PAS 21448 also points out that engineers remain liable for potentially hazardous scenarios missing from the data set used to assess the DNN (i.e., the test set). Moreover, relying only on accuracy evaluation does not enable identifying the different scenarios leading to DNN failures (i.e., wrong predictions) nor their frequency; underrepresented scenarios in the test set could let serious safety risks go unnoticed.

In practice, in safety-critical systems, engineers should not only assess accuracy but also assess the safety risks related to the DNN failures observed during DNN testing. A straightforward process consists of visually inspecting all the images leading to DNN failures (failure-inducing images) and identifying commonalities across them. Such commonalities should enable an engineer to identify the unsafe scenarios in which the DNN is likely to fail. Identifying the unsafe scenarios for a DNN should enable engineers to determine risks and take actions to improve the system (e.g., retraining the DNN on more inputs belonging to such unsafe scenarios). Unfortunately, when test sets are large, relying on visual inspection is expensive and prone to errors.

Demonstrating the functional safety of DNN-based components remains a critical issue for companies developing safety-critical systems. This is the case for IEE [17], a supplier of sensing solutions active in the automotive market and our partner in this research work. Specifically, the collaboration with IEE helped us identifying the open problems in safety analysis for DNN-based systems, particularly in relation to the safety standards ISO 26262 and ISO/PAS 21448, which IEE has to comply with, like many other organizations in the automotive domain.

To support the safety analysis of DNN-based systems and address issues of trustworthiness, we propose four different techniques. They all leverage the intuition that unsafe scenarios might be cost-effectively identified by automatically clustering images that lead to DNN failures and present commonalities either in their visual content or in the way they trigger responses from DNN neurons. We call such image clusters root cause clusters (RCCs). RCCs can be quickly inspected by an engineer who can then easily determine what the commonalities in the clustered images are and, consequently, determine what the unsafe scenarios are. Based on domain knowledge, engineers can determine the frequency of the identified unsafe scenarios and therefore take appropriate measures to address them (e.g., determine if the risk is acceptable or if the DNN should be improved). Further, RCCs can be utilized to select additional images to be used to retrain the DNN. Finally, simulators can be used to generate additional images belonging to RCCs and rely on the commonalities across the simulators parameters associated to images in a same RCC to generate explicit descriptions of the RCCs.

The proposed approaches have been successfully evaluated on seven case study DNNs from the automotive domain defined in collaboration with our IEE; they include driver's head pose, eye gaze, and drowsiness detection, traffic sign detection, vehicle position estimation, unattended child detection, and steering angle prediction DNNs. Our results demonstrate that the proposed techniques can support safety analysis of DNN-based systems and outperform state-of-the-art alternatives.

1.2 Research Contributions

We developed *Heatmap-based Unsupervised Debugging of DNNs (HUDD)*, a white-box technique that automatically identifies RCCs by applying an unsupervised clustering algorithm to heatmaps that capture the relevance of each DNN neuron on the DNN outcome. HUDD also retrains DNNs with images automatically selected based on their belonging to the identified clusters.

We developed a search-based technique, *Simulator-based Explanation for DNN failurEs (SEDE)*, which generates readable descriptions of RCCs and improves the DNN through retraining based on such descriptions. It requires the availability of real-world scene simulators to generate images to be processed by the DNN under analysis, commonly used to train DNNs for cyber-physical systems. It relies on genetic algorithms to drive such simulators towards generating images belonging to the RCCs generated by HUDD. It then uses rule-learning algorithms to derive expressions that capture commonalities in terms of simulator parameter values. These expressions are later used to generate additional images for debugging the DNN.

Moreover, we developed a black-box approach, *Safety Analysis using Feature Extraction (SAFE)*, to characterize the root causes of DNN failures automatically. It relies on a transfer learning model to extract features from failure-inducing images. Then it applies a clustering algorithm to detect arbitrary-shaped clusters of images that model plausible causes of failure, which are then used for retraining and debugging the DNN using images that belong to such RCCs.

Finally, we propose a generic pipeline for the generation of RCCs that derives from the work on HUDD and SAFE and can integrate different machine learning components. We rely on such a pipeline to assess the effectiveness of RCC generation when different clustering algorithms, dimensionality reduction techniques, and feature extraction methods are used. The evaluation approach for the instantiated pipelines entails controlling the causes of failures by artificially creating scenarios including realistic causes of DNN failures (e.g., poor lighting conditions); it enabled us to objectively assess the performance of these pipelines.

1.3 Dissemination

HUDD was published in IEEE Transactions on Reliability [18]; also, it was presented at the Journal First track of the IEEE 32^{nd} International Symposium on Software Reliability Engineering (ISSRE). Further, the toolset implementing HUDD has been presented in the tool demo track of the 44^{th} ACM/IEEE International Conference on Software Engineering (ICSE 2022) [19]. Our implementation of *HUDD*, the generated root-cause clusters, the DNN models, the datasets and the data generated to address our research questions are available online [20].

SEDE was published in the ACM Transactions on Software Engineering & Methodology [21]. Our implementation of *SEDE*, the IEE simulators, and the data generated to address our research questions are available online [22, 23]. We cannot share the real-world images used for FLD experiments because they were collected by IEE and protected by privacy agreements.

SAFE has been published in ACM Transactions on Software Engineering & Methodology [24]. The first author of the SAFE paper provided the idea of extending the HUDD pipeline with black-box analysis components, the second author provided the expertise concerning their integration in a HUDD-like pipeline and their assessment with the case study subjects used with HUDD. Our implementation of *SAFE*, and the data generated to address our research questions are available online [25].

Our generic pipeline derived from *HUDD* and *SAFE* and the empirical evaluation of its different variations has been submitted to ACM Transactions on Software Engineering & Methodology and is currently under review [26]. All our implementations, the failure-inducing sets, the generated root-cause clusters and the data generated to address our research questions are available online [27].

1.4 Dissertation Outline

In Chapter 2, we explore the background of our work in addition to the related work undertaken in the context of software engineering research, with the aim of identifying the causes of inaccurate predictions made by DNNs and implementing measures for debugging and retraining. We discuss testing and debugging of DNNs in safety-critical contexts. Then, we discuss DNNs and rule-based systems in addition to different supervised and unsupervised machine learning algorithms. Finally, we discuss explainable artificial intelligence methods, including heatmap-based approaches, and present state-of-the-art evolutionary algorithms.

In Chapter 3, we present the *HUDD* technique. Additionally, we present the evaluation results answering two main research questions; (1) *Does HUDD enable engineers to identify the root causes of DNN failures?* and (2) *How does HUDD compare with traditional DNN accuracy improvement practices?*.

In Chapter 4, we present the *SEDE* technique, which includes a dedicated evolutionary algorithm Pairwise Replacement (*PaiR*). We demonstrate our empirical evaluation results which address five main research questions; (1) *How does PaiR fare, compared to state-of-the-art evolutionary algorithms?*, (2) *Does SEDE generate simulator images that are close to RCCs of real-world images?*, (3) *Does SEDE generate, for each RCC, a set of images sharing simmilar characteristics?*, (4) *Do the expressions generated by SEDE delimit an unsafe space?* and (5) *How does SEDE compare to state-of-the-art DNN accuracy improvement practices?*.

In Chapter 5, we present the *SAFE* technique. We demonstrate our evaluation results to answer three research questions; (1) *Does SAFE enable engineers to identify root causes of DNN failures?*, (2) *Does SAFE enable engineers to more effectively and efficiently retrain a DNN when compared with HUDD and baseline approaches?*, and (3) *Does SAFE provide time and memory savings compared to HUDD?*.

In Chapter 6, we present our empirical evaluation of 100 different pipelines to answer three research questions; (1) *Which pipeline generates RCCs with the highest purity?*, (2) *Which pipelines generate RCCs covering more failure scenarios?*, and (3) *How is the quality of RCCs generated affected by infrequent failure scenarios?*.

Finally, in Chapter 7, we summarise our contributions, their limitations, and future prospects.

Chapter 2

Background and Related Work

IN this chapter, we provide a brief and concise overview of machine learning (ML) algorithms, testing and debugging of DNNs in safety-critical contexts, search-based evolutionary algorithms (EA), and explainable artificial intelligence (XAI) approaches used to perform the work presented in this dissertation. Additionally, we provide an overview of the work developed in the context of software engineering research, which aims at determining the reasons for wrong predictions and debug or retrain the DNN.

2.1 Machine Learning (ML)

Machine learning techniques rely on algorithms and statistical models to enable computers to learn from data and improve their performance on a specific task without being explicitly programmed. It involves the use of large datasets to train models, which can then make predictions based on new input data. ML has a wide range of applications, including image recognition, natural language processing, and autonomous driving tasks. ML techniques can be either supervised (i.e., learn from datasets annotated with the information to be learned) or unsupervised (i.e., do not require an annotated dataset). In the following sections, we will delve into the details of two supervised machine learning techniques: neural networks and rule-based systems. Additionally, we will describe unsupervised machine learning techniques referred in the next chapters: clustering algorithms, and dimensionality reduction techniques.

2.1.1 Deep Neural Networks (DNNs)

DNNs are a type of artificial neural network that has revolutionized the field of machine learning. DNNs are composed of multiple layers of interconnected nodes, which are designed to learn and extract increasingly complex features from input data. The depth of these networks allows for the representation of highly nonlinear relationships between inputs and outputs, making them well-suited for a wide range of complex tasks, such as image recognition, natural language processing, and speech recognition. The training of DNNs typically involves the use of large amounts of labeled data, and optimization algorithms that adjust the weights and biases of the network to minimize a loss function. With their ability to learn from large and diverse datasets, DNNs have become a key technology in the development of advanced AI systems.

Among various types of DNNs, Convolutional Neural Networks (CNNs) have emerged as a popular architecture for processing visual data, such as images and videos.The standard CNN architecture [28, 29, 30] comprises three types of layers: convolutional layers, pooling layers, and fully connected layers. The convolutional layer is considered the primary building block of a CNN. This layer extracts relevant features from input images during training. Convolutional and pooling layers are stacked to form a hierarchical feature extraction module. The model captures the resulting feature map from the last pooling layer as a 3D matrix of size (N, N, N_c), where N is its width and height, and N_c is its depth. For feature extraction, this feature map is flattened to form a vector of size (1, $N \times N$). In summary, the CNN model receives an input image of size $(224, 224, 3)$. This image is then passed through the network's layers to generate a vector of features. The feature extraction process from all images generates raw data represented by a 2D matrix (denoted as X) formalized below:

$$X = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1m} & l_2 \\ x_{21} & x_{22} & ... & x_{2m} & l_c \\ ... & ... & ... & ... & ... \\ x_{k1} & x_{k2} & ... & x_{km} & l_1 \end{bmatrix}, l_i \in \{l_1, l_2, ..., l_c\} \tag{2.1}$$

where l_i represent the class labels, c is the number of categories, $m = N \times N$ is the number of features, and k is the size of the dataset. The class categories are useful if the user is working on a supervised problem or if fine-tuning is required.

A well-known example of NNs is VGGNet [31], which is a CNN characterized by a high number of layers (11 to 19 layers). The purpose of this architecture is to minimize the number of trainable parameters. Controlling the number of parameters helps to reduce overfitting issues. To this end, VGGNet proposes to increase the network's depth and to decrease the size of filters from 7×7 and 5×5 to 3×3. The comparative study between the number of parameters in 3 stacked convolutional layers associated with 3×3 filters and a single convolutional layer associated with 7×7 filters demonstrated that small filters reduce the total number of parameters. Also, it enhances non-linearity through ReLu activation functions in intermediate layers. VGGNet proposes six different configurations: A, $A - LRN$, B, C, D, and E, where the depth varies from 11 to 19 layers.

2.1.2 Rule-based Systems

Rule-based algorithms are a type of machine learning model that focuses on the creation of decision rules that can be easily understood and interpreted by humans. These models are particularly useful in scenarios where the ability to explain the decision-making process is critical. The main goal of rule-based algorithms is to provide a transparent and interpretable method for making predictions or classifications, which can help build trust and confidence in the decision-making process.

A *decision rule* [32] is an IF-THEN statement consisting of a condition and a prediction. For example, a decision rule may indicate that if the horizontal angle used to generate an image is above 50 degrees (i.e., the head of the person is so turned that her left eye is barely visible), then the image is failure-inducing. Decision rules are machine learning models that can be easily interpreted by humans [32].

The support of a rule indicates the percentage of instances to which the condition of a rule applies, while its accuracy measures the proportion of classes correctly predicted for the instances to which the condition of the rule applies.

To combine multiple rules, rule learning algorithms derive either decision lists or decision sets. A decision list is ordered, and we should rely on the prediction of the first rule whose condition evaluates to true. In a decision set, predictions are based on some strategy (e.g., majority voting).

Rule-based Approaches

Kim et al. [33] process images generated with simulators and rely on rule extraction algorithms to characterize correctly and incorrectly classified images in terms of simulator parameters. However, the approach of Kim et al. cannot be applied to real-world images, which are instead necessary to ensure the applicability of the DNN in the field.

Further, recent work studies the effectiveness of decision trees in characterizing the input space of the simulator-based testing process [34, 35].

In particular, the PART algorithm [36] which combines the approach of RIPPER [37], a rule learning algorithm, with decision trees, have been successfully applied in related work to characterize DNN inputs [34].

PART, similarly to RIPPER, relies on a *separate-and-conquer* approach. It relies on a partial C4.5 decision tree to derive each rule. It starts by deriving a rule that provides accurate predictions for some of the data points. Such result is achieved by building a pruned C4.5 decision tree from all the data points, and then selecting the leaf with the largest support to transform it into a rule. The decision tree is then discarded and all the data points that are covered by the rule are excluded from the training set. This rule learning procedure is repeated until the whole data set is processed.

2.1.3 Unsupervised Learning Algorithms

Unsupervised learning is a type of machine learning that involves finding patterns and relationships in a dataset without any prior knowledge of its structure or labels. There are various categories of unsupervised learning algorithms, including clustering, dimensionality reduction, density estimation, anomaly detection, and association rule learning, among others. While there are many types of unsupervised learning algorithms, clustering and dimensionality reduction are two of the most commonly used.

On one hand, a clustering algorithm tries to group data points that are similar to each other; on the other hand, a dimensionality reduction algorithm tries to reduce the number of features in the data set by eliminating useless information and only retaining important ones.

Clustering Algorithms

Clustering is a data analysis method that mines essential information from a dataset by grouping data into several groups called *clusters*. In clustering, similar data points are grouped into the same cluster, while non-similar data points are put into different clusters. There are two main objectives in data clustering; the first objective is to minimize the dissimilarity within the cluster, and the second objective is to maximize the inter-cluster dissimilarity. In the following paragraphs, we will describe some of the well-known, state-of-the-art, clustering algorithms, including Hierarchical Agglomerative Clustering

(HAC) [38], K-Means [39], Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [40], and Hierarchial Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [41]

HAC is a bottom up approach in which each observation starts in its own cluster and pairs of clusters are iteratively merged into a sequence of nested partitions. The input of HAC is a matrix capturing the distance between every observation pair. The grouping that occurs at each step aims to minimize an objective function.

In HAC, widely adopted objective functions are (1) the error sum of squares within clusters (i.e., Ward's linkage method [42]), to help minimize within-cluster variance, (2) the average of distances between all pairs of elements belonging to distinct clusters (i.e., average linkage [43]), to help maximize diversity among clusters, and (3) the shortest distance between a pair of elements in two clusters (i.e., single linkage [44]), to merge clusters that are closer for at least one element.

K-Means is a well-known clustering algorithm. It receives a number K as input and divides the data into K clusters based on the distance calculated from the data points to the center of the cluster. This algorithm's main function is to minimize the distance between the data points and their cluster center as much as possible. In the original K-means algorithm, the number of clusters (K) is set manually, which can affect the quality of the clusters since we do not have any prior knowledge of the data.

DBSCAN is an algorithm that defines clusters using local density estimation. It can be divided into four steps:

1. The ϵ-neighborhood of a data point is determined as the set of data points that are at most ϵ distant from it.
2. If a data point has a number of neighbors, including itself, above a configurable threshold (called *MinPts*), it is then considered a core point, and a high-density area has been detected.
3. Since core points can be in each other's neighborhoods, a cluster consists of the set of corepoints that can be reached through their ϵ-neighborhoods and all the data points in these ϵ-neighborhoods.
4. Any data point that is not a core point and does not have a core point in its neighborhood is considered noise.

To obtain clusters using DBSCAN, the selection of two configuration parameters is needed: (1) The distance threshold, ϵ, to determine the ϵ-neighborhood of each data point, and (2) the minimum number of neighbors, *MinPts*, needed for a data point to be considered a core point. In Section 6.2.3, we introduce a method to automatically derive and select these two configuration parameters, ϵ and *MinPts*.

HDBSCAN is an extension of DBSCAN to solve its main limitation: selecting a global ϵ. DBSCAN uses a single global ϵ value to determine the clusters. When the clusters have varying densities, using one global value can lead to a suboptimal partitioning of the data. Instead, HDBSCAN overcomes such a limitation by relying on different ϵ values for each cluster, thus finding clusters of varying densities.

HDBSCAN first builds a hierarchy using varying ϵ to figure out which clusters end up merging together and in which order. Based on the hierarchy of the clusters, HDBSCAN selects the most persisting clusters as final clusters. Cluster persistence represents how long a cluster stays without splitting when decreasing the value of ϵ. After selecting a cluster, all its descendants are omitted.

Figure 2.1 shows an example of the clusters' hierarchy found by HDBSCAN. The y-axis represents the values of ϵ. Vertical bars represent clusters; the color and width of each vertical bar captures the size of the cluster. We can notice that certain clusters split after the value of ϵ is increased, while some other persist. HDBSCAN decides which subclusters to select based on their persistence, which is represented by different color regions in the plot. Precisely, HDBSCAN selects the clusters having the highest persistence. The unselected data points are considered noise.

In our example, only six clusters are selected (circled bars); they are the most persisting groups in the hierarchy.

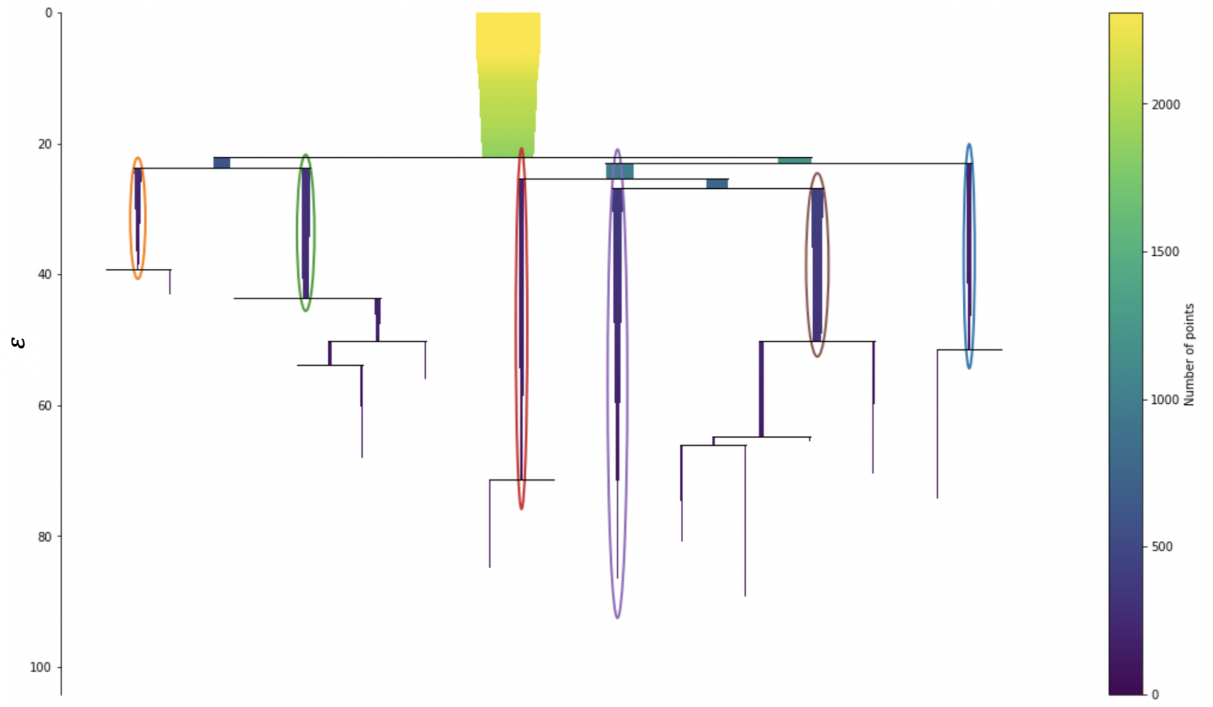


Figure 2.1: An example of the clusters selected by HDBSCAN.

Dimensionality Reduction Techniques

Several dimensionality reduction techniques exist in the literature. In this section, we focus on two state-of-the-art techniques: Principal Component Analysis (PCA) [45], and Uniform Manifold Approximation and Projection (UMAP) [46]. PCA is used for its simplicity of implementation and because it does not require much time and memory resources. UMAP is used for its effectiveness when applied before clustering. UMAP groups data points based on relative proximities, which optimizes the clustering results. PCA and UMAP are described below.

PCA To reduce dimensionality, PCA creates a 2-dimensional matrix of variables and observations. Then, for this matrix, it constructs a variable space with a dimension corresponding to the number of variables available. Finally, it projects each data point onto the first few maximum variance directions in the variable space. This procedure allows PCA to obtain a lower-dimensional data representation while maximizing data variation. The first principal component can equivalently be defined as the direction that maximizes the variance of the projected data [47].

UMAP is a dimension reduction technique that can be used for visualization but also for general non-linear dimension reduction. UMAP is fast, and scaling well in terms of both dataset size and dimensionality. The main limitation of UMAP is that it doesn't preserve the density of the data, which is, instead, better preserved by PCA.

First, UMAP forms a weighted graph representation between each pair of data points, where the edge weights are the probability of two data points being connected to each other. This graph is obtained by extending a radius outward each data point such that two data points are connected if their radii overlap. However, since an underestimation of such a radius can lead to the generation of small, isolated clusters, and its overestimation can lead to connecting all data points together, UMAP selects such a radius locally. The radius selection is performed based on the distance from each data point to its '$n - th$' neighbor. Finally, UMAP decreases the likelihood of two data points getting connected past the first neighbor (as the radius grows larger), which preserves the balance between the high-dimensional and low-dimensional representations. Once the high-dimensional graph is constructed, UMAP optimizes the layout of a low-dimensional representation to be as similar as possible. The general idea is to initialize the low-dimensional data points and then move them around until they form clusters that have the same structure as the high-dimensional data, preserving the connectedness of the data points. UMAP calculates Similarity Scores (distances) in the high dimensional graph to help identify the clustered points and tries to preserve that clustering in the low dimensional graph.

Autoencoders (AE)

AEs are unsupervised artificial neural networks that learn how to compress and encode the data before reconstructing it from the compressed encoded version to a representation that resembles the original input as much as possible. AEs can extract features that can be used to improve downstream tasks, such as clustering or supervised learning, that benefit from dimensionality reduction and higher-level features. In other words, AEs try to learn an approximation to the identity function and, by placing various constraints on the network's architecture and activation functions, they extract useful representations [48].

Figure 2.2 illustrates the neural network architecture of a simple AE. It consist of four components:

- **Encoder:** compresses the input data and reduces its dimensions into an encoded representation.
- **Bottleneck:** contains the encoded representation of the input data (i.e., extracted features vector).
- **Decoder:** reconstructs the input data from the encoded version (retrieved from the *Bottleneck*) such that it resembles the original input data as much as possible.
- **Reconstruction Loss:** the difference between the *Encoder*'s input and the reconstructed version (the *Decoder*'s output). The objective is to minimize such loss during training.

The objective of an AE's training process is to minimize its reconstruction loss, measured as either the mean-squared error or the cross-entropy loss between original inputs and its constructed inputs.

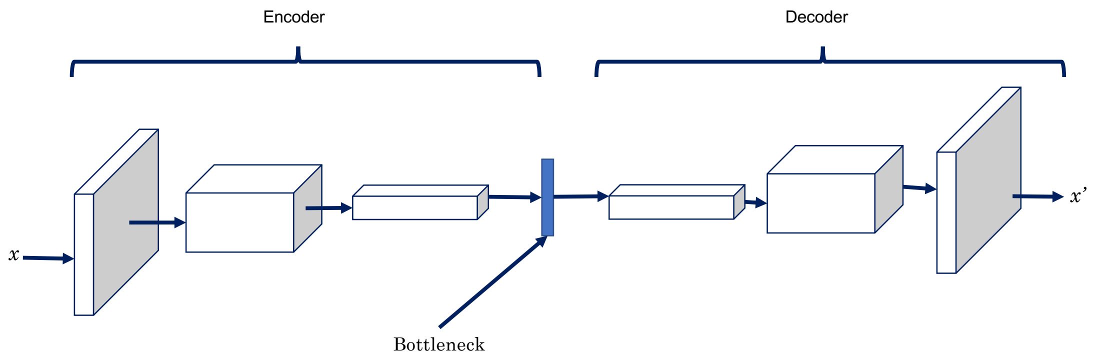


Figure 2.2: Autoencoder Architecture

2.2 Testing and Debugging of DNNs in Safety-critical Contexts

In this section, we introduce the practical context of our research, which in short is the safety analysis and debugging of DNN-based automotive systems. We explain why automated root cause analysis is necessary to enable functional safety analysis. Also, we show how DNN accuracy improvement can be facilitated by the automated characterization and identification of *failure-inducing inputs* (i.e., inputs that make the DNN generate erroneous results). Though the issues raised below and many of our insights are not specific to automotive systems, but also relevant to many cyber-physical systems in general, this is the practical domain and context in which this work took place.

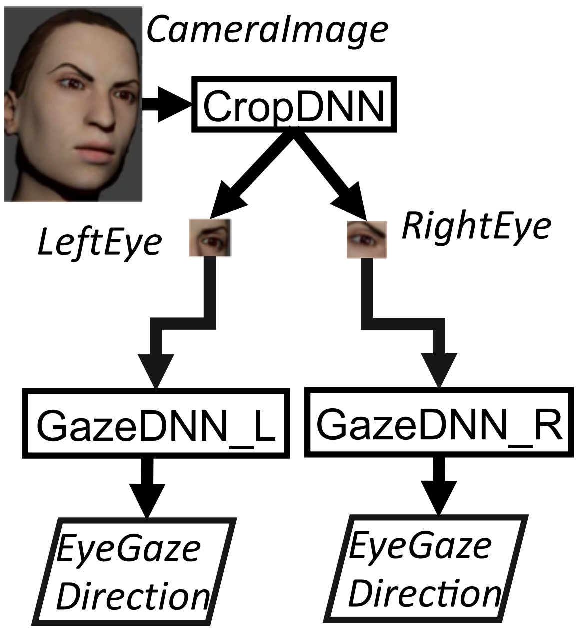


Figure 2.3: DNN-based system for gaze detection.

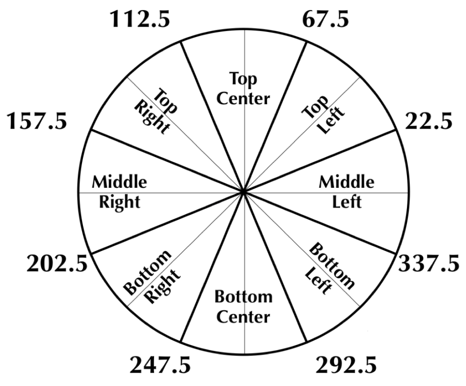


Figure 2.4: Gaze directions.

2.2.1 DNN-based Automotive Systems

Our work is motivated by the challenges encountered in industry sectors developing safety-critical, cyber-physical systems, such as the automotive sector. In particular, IEE has investigated the development of a Gaze Detection System (GDS) which uses DNNs to determine the gaze direction of the driver, from images captured by a camera on the instrument panel of the car.

Figure 2.3 shows a possible architecture for the GDS architecture consisting of three DNNs (i.e., CropDNN, GazeDNN_L, GazeDNN_R). CropDNN identifies face landmarks that enable the cropping of images containing the eyes only. GazeDNN_L and GazeDNN_R classify the gaze direction into eight classes (i.e., TopLeft, TopCenter, TopRight, MiddleLeft, MiddleRight, BottomLeft, BottomCenter, and BottomRight).

To reduce training costs, IEE relies on training sets containing images that are collected from driving scenes and images generated by simulation software. Simulators are used to reduce the costs related to data collection and data labeling. Indeed, models of the dynamics of real-world elements (e.g., eyeballs) are used to generate, in a controlled way through the selection of parameter values, hundreds of images in a few hours [49]. Further, and this is important in terms of cost saving, simulation enables the automated labeling of images by analyzing model parameters. However, while simulator images alleviate the costs of training, testing ultimately requires real-world images as well, since simulators do not exhibit perfect fidelity.

Additional DNN-based systems under development at IEE, which we used as cases studies, are presented in Sections 3.3, 5.3, 4.4, and 6.3.

2.2.2 Debugging of DNN-based Systems

IEE engineers train the DNNs that compose their systems by following the standard machine learning process depicted in Figure 2.5-a. They first train the DNN using a training set with labeled images (Step A) and then execute the DNN against a labeled test set (Step B). This process enables engineers to evaluate the DNN accuracy (e.g., the percentage of images leading to correct results).

When the accuracy of the system is not adequate, engineers can improve the DNN by augmenting the training set with failure-inducing images. This process is depicted in Figure 2.5-b. First, engineers generate a set of new images to be used to retrain the DNN (Step C). We call this set of images *improvement set*. The improvement set generally consists of images collected from the field since these tend to be failure-inducing when DNNs have been trained using simulator images. Real-world images must be manually labelled (Step D). The DNN model is tested with the improvement set and images that lead to DNN failures are identified (Step E). This set of failure-inducing (unsafe) images is considered to retrain the DNN (Step G), using as initial configuration for DNN weights the ones in the previously trained model.

To improve the DNN, it is necessary to process a sufficiently large number of unsafe images. For this reason, the number of unsafe images can be augmented by applying *bootstrap resampling* (i.e., by replicating samples in the unsafe set [50]) till a target is achieved (Step F). Finally, the improved DNN can be assessed on the test set (Step H).

In general, generating a sufficiently diverse set of failure-inducing inputs that include all possible causes of DNN failures is very difficult. Further, when the labeling of such images is manual, the costs of labeling becomes prohibitive and DNN improvement is hampered. For this reason, **automatically**

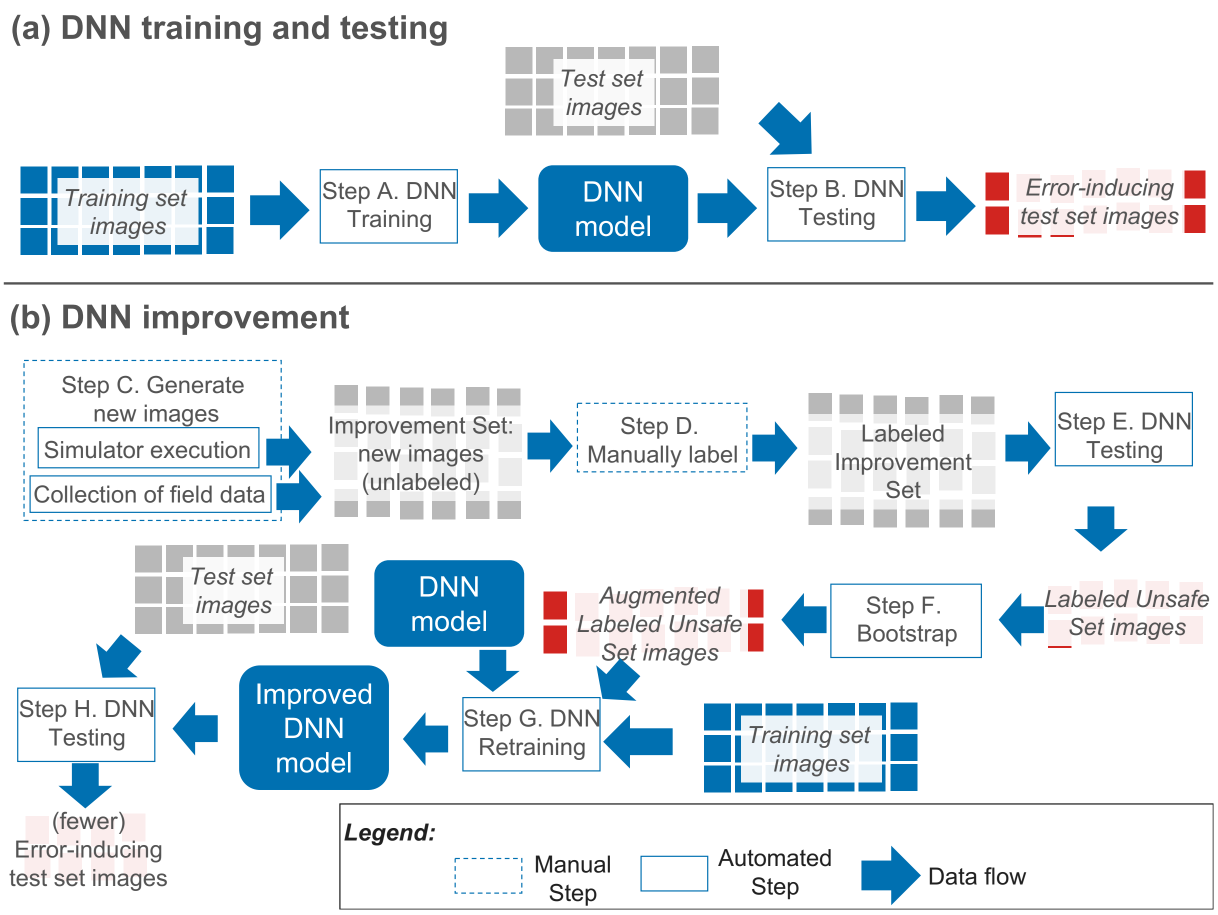


Figure 2.5: Training and debugging DNNs.

characterizing images that are likely to lead to DNN failures would allow the image generation or selection process to target specific types of images and increase the efficiency of the DNN retraining process.

2.2.3 Functional Safety Analysis

Like many other organizations in the automotive domain, IEE products must comply with the functional safety standards ISO 26262 and ISO/PAS 21448. Functional safety is addressed by identifying, for each component of the product (e.g., the DNNs of the GDS), the *unsafe conditions* that could lead to hazards, by identifying countermeasures (e.g., redundant components), and by demonstrating that these unsafe conditions are unlikely to occur.

ISO/PAS 21448, specifically targeting autonomous systems, recommends to determine unsafe conditions by following the traditional DNN testing process depicted in Figure 2.5-a and by manually inspecting the failure-inducing images to look for root causes of DNN failures. In a DNN context, *unsafe conditions thus correspond to root causes of DNN failures*.

In addition, the manual identification of unsafe conditions is error-prone. For example, engineers may overlook unsafe conditions that are underrepresented in the test set. Also, such conditions may lead to a biased estimate of the accuracy of the DNN. For example, UnityEyes generates eye images where the horizontal angle of the head is determined based on a uniform distribution, between 160 (head turned right) and 220 degrees (head turned left). As a result, very few images with an angle of 160 or 220 degrees are generated and, though it may be an unsafe condition (i.e., one eye is barely visible and the gaze direction prediction may be inaccurate), experiments based on test sets generated with UnityEyes may suggest that the DNN is on average very accurate. It is, however, important for engineers to know that such a DNN, in some rarely occurring cases in the test set, is unsafe when the driver turns his head while driving.

In summary, accuracy estimation results depend on the test set, which may not include all unsafe conditions in a representative or balanced manner. Automated root cause analysis helps making sure, through clustering, that even rare, unsafe conditions are made visible to the analyst, especially when safety analysis time is limited. In other words, clustering based on heatmaps makes safety analysis robust, to some extent, to imperfect test sets.

2.3 Evolutionary Algorithms (EA)

Evolutionary algorithms are state-of-the-art solutions to find a set of nondominated solutions in multi-objective optimization problems. In software engineering, they are widely adopted to identify test inputs that maximize some coverage criterion [51]. Also, evolutionary algorithms have been successfully used to test autonomous driving systems [52, 53, 54, 55].

2.3.1 Multi-objective Algorithms

Multi-objective algorithms are optimization methods that are designed to find a set of solutions that simultaneously optimize multiple conflicting objectives. These algorithms are used in a wide range of applications, including engineering design, resource allocation, and machine learning.

One popular multi-objective optimization algorithm is the Non-dominated Sorting Genetic Algorithm II (NSGA-II) [56, 54, 53, 55]. NSGA-II is a state-of-the-art solution for optimization problems with up to three objectives. It is shown in Fig. 2.6. NSGA-II works by first generating a random population (Line 2), which is evolved in a number of iterations (Lines 3 to 16). A new offspring (Q_t) is generated by applying mutation and crossover operators (Line 4). The algorithm preserves elitism since the new population (P_{t+1}) is generated by selecting the best individuals in the union of the current population (P_t) and the offspring (Line 5). The best individuals are selected by first ranking individuals based on their belonging to the l-th nondominated fronts (Line 6), which is efficiently performed by the fast-nondominated-sorting procedure [56]. Then, the algorithm adds individuals to the population according to their rank (Line 9), till it finds a ranked subset that cannot be fully added (Line 8). Finally, to preserve diversity in the population, NSGA-II selects the items belonging to the last ranked subset according to a crowding distance function that prioritizes individuals being more distant from others along with individuals at the boundaries of the objective space (Lines 11 to 15).

2.3.2 Many-objective Algorithms

Many-objective algorithms are optimization methods that are designed to find a set of solutions that simultaneously optimize a large number of objectives. These algorithms are used in cases where the number of objectives is so high that traditional multi-objective optimization algorithms are not effective.

The term many-objective is used for algorithms that tackle problems with more than three objectives [57]. A popular many-objective algorithm used in software testing is Many-Objective Simulated

```
Require: O = o1, o2, ... the objectives
Require: M, population size
Require: T, search budget (i.e., number of iterations to perform)
Ensure: a population whose Pareto front includes the best found solutions
 1: $t \leftarrow 0$ //current generation
 2: $P_t \leftarrow$ CREATE RANDOM POPULATION WITH SIZE M
 3: while $t < T$ do
 4:     $Q_t \leftarrow$ GENERATE OFFSPRING FROM $P_t$
 5:     $R_t \leftarrow P_t \cup Q_t$
 6:     $(F_1, F_2, ..., F_l) \leftarrow$ APPLY fast-nondominated-sorting TO $R_t$
 7:     $r \leftarrow 0$ //counter for the Pareto front rank
 8:     while $|P_{t+1}| + |F_r| \leq M$ do //iterate over all the Pareto fronts till they fit into the population
 9:         $P_{t+1} \leftarrow P_{t+1} \cup F_r$
10:         $r \leftarrow r + 1$
11:     ASSIGN CROWDING DISTANCE TO INDIVIDUALS IN $F_r$ //see Fig. 4.4; $F_r$ is the front that does not fully fit into $P_{t+1}$
12:     SORT $F_r$ BASED ON CROWDING DISTANCE
13:     while $|P_{t+1}| \leq M$ do //till $P_{t+1}$ is not full, add the individuals in the sorted $F_r$
14:         $ind \leftarrow$ EXTRACT FIRST INDIVIDUAL IN $F_r$
15:         $P_{t+1} \leftarrow P_{t+1} \cup ind$
16:     $t \leftarrow t + 1$
17: Return $P_t$
```

Figure 2.6: NSGA-II

Annealing (MOSA) [51]. Given test cases (i.e., sequences of inputs for program APIs) as individuals, MOSA models branch coverage as a search objective but does not account for the length of individuals. Shorter test cases are, however, easier to read than longer ones and should be prioritized. Therefore, MOSA extends NSGA-II by relying on an archive that is used to keep track of the shortest test cases accidentally identified during the search. In addition, since MOSA aims to fulfil all the objectives and not only a subset of them, instead of relying only on the nondominance relation to select elitist individuals (i.e., Line 6 in Fig. 2.6), it makes use of the *preference criterion*, a strategy that ensures preserving individuals that minimize the objective score for uncovered objectives (i.e., objectives without an individual already in the archive). To generate inputs for DNN-based systems with MOSA [58], it is necessary to specify a fitness threshold to determine objective coverage (e.g., a safety violation occurs if the ego vehicle gets too close to the vehicle in front). At every iteration, the archive is updated to keep the best individual found during the search; however, the probability of replacing an individual already in the archive is low. Indeed, thanks to the preference criterion, the population includes only the best individuals for the objectives not covered yet. When it is not possible to specify a threshold to determine when an objective is covered, the main benefit of MOSA is to ensure that the best individual for each objective is preserved; indeed, the archive becomes useless because the best individuals are preserved directly in the population. However, a simple modification of NSGA-II can achieve the same purpose (see Section 4.3.1).

2.4 eXplainable Artificial Intelligence (XAI)

Approaches that aim to explain DNN results have been developed in recent years [59]. Most of these concern the generation of heatmaps that capture the importance of pixels in image predictions. They include black-box (Model-agnostic) [60, 61] and white-box (heatmap-based) approaches [62, 63, 64, 65, 66]. In this section, we explore approaches developed by the ML and computer vision community that can be used to explain either correct or wrong predictions and are partially reused in the rest of this dissertation.

2.4.1 Heatmap-based DNN Explanations

Heatmap-based approaches [60, 61, 62, 63, 64, 65, 66] explain the DNN's prediction of an image by highlighting which region of an image influenced the DNN output most. For example, Grad-CAM generates a heatmap from the gradient flowing into the last layer. The heatmap is then superposed on the original image to highlight the regions of the image that activated the DNN and influenced the decision [63]. The main limitation of these approaches is that they require the inspection of all the heatmaps generated for each image processed by the DNN (e.g., failure-inducing images) and do not provide engineers with guidance for their inspection (i.e., one cluster for each failure root cause).

Heatmap-based Approaches

Under-approximation boxes [67] consist of the minimal set of neurons, belonging to a specific layer, that ensure a postcondition (e.g., the generation of a specific DNN output). When applied to explain misclassifications, they lead to heatmap-like images showing the minimal set of input pixels leading to the same DNN result. Similarly, Ribeiro et al., identify the image chunks that are sufficient to generate a certain DNN result [68]. Like heatmap generation techniques, these two approaches cannot automatically identify the root cause for a group of error-inducing images but require manual inspection for every error-inducing image.

MODE automatically identifies the images to be used to retrain a DNN [69]. However, it cannot identify the root causes of DNN errors, which is a major limitation in our context. The selection of images to be used for retraining, in the case of MODE, is not based on heatmaps but on training additional DNN layers that capture commonalities among neuron activations leading to DNN errors. MODE, therefore, entails repeated modification and retraining of the DNN under test, just to select the improvement set, which is a very expensive endeavor.

Surprise adequacy measures the degree of variation in neuron activations between a new image and the training images belonging to the same class [70]. Empirical results show that a retraining set with a varying degree of surprise adequacy improves DNN robustness against adversarial examples. However, it has never been adopted to improve accuracy for non-adversarial inputs. Also, like previous techniques, it cannot be used to identify root causes of DNN errors.

DeepFault [71] identifies a set of suspicious neurons to synthesize new, adversarial images and improve DNN adversarial robustness. Since it relies on synthesized adversarial inputs, it cannot improve accuracy for unsafe, non-adversarial inputs. Once again, it does not distinguish different root causes.

INNvestigate [72] and TorchRay [73] are well known tools supporting DNN explanation. However, for explanations concerning DNNs that process images, they generate one heatmap for every failure-inducing input image; each heatmap must be visually inspected by engineers, which makes the investigation of many DNN failures highly expensive.

AI-Lancet [74] optimizes deep learning models by locating the error-inducing (EI) neurons and fixing them using either neuron-flip or neuron-fine-tuning methods. It starts by revealing the EI regions in the input sample and then extracts the EI features activated by the EI regions of the input. Finally, the EI neurons can be located with the guidance of the EI features. AI-Lancet requires the modification and the retraining of the DNN to find the erroneous neurons. Another limitation is that it does not explain the root causes of errors. Instead, it attempts to fix them by fine-tuning or flipping neurons.

SHAP (SHapley Additive exPlanation) [75] generates explanations by calculating the contribution of each feature to the prediction, thus explaining what features are the most important for a prediction. In the case of an image CNN, SHAP considers a group of pixels as a feature and calculates their contribution to the decision made by the DNN. Like heatmap-based approaches, SHAP does not provide guidance for the investigation of multiple failure-inducing images.

Finally, solutions [66] backpropagating only the difference in activations between the different classes may compromise clustering since they do not account for information about all available neurons but only the ones related to the predicted output class. Deconvolutional networks [64] and guided backpropagation [65] lead to sparse heatmaps that do not fully explain the DNN result [76]. Grad-CAM [63] does not work with convolutional DNN layers. In contrast, Layer-Wise Relevance Propagation (LRP) [62] generates precise, non-sparse heatmaps for all the DNN layers because it takes into account all the different factors affecting the relevance of a neuron, which include the DNN structure and the neuron activations. In the following, we detail LRP because selected as building block for HUDD (see Chapter 3).

Layer-Wise Relevance Propagation (LRP) LRP redistributes the relevance scores of neurons in a higher layer to those of the lower layer. Assuming j and k to be two consecutive layers of the DNN, LRP propagates the relevance scores computed for a given layer k into a neuron of the lower layer j. It has been theoretically justified as a form of Taylor decomposition [77].

Figure 2.7 illustrates the execution of LRP on a fully connected network used to classify inputs. LRP analyzes the data processed by a DNN and can be applied to any DNN architecture. In the forward pass, the DNN receives an input and generates an output (e.g., classifies the gaze direction as TopLeft) while keeping trace of the activations of each neuron. The heatmap is generated in a backward pass. The heatmap in Figure 2.7 shows that the pupil and part of the eyelid, which are the non-white parts in the heatmap, had a significant effect on the DNN output. Furthermore, the heatmap in Figure 2.8 shows that the mouth and part of the nose are the input pixels that mostly impacted on the DNN output.

In Figure 2.7, blue lines show that the DNN score of the selected class is backpropagated to lower layers. Plain lines show the connections concerned by the propagation formula used to compute the relevance (R_{ji}) of neuron i at layer j from all the connected neurons in layer k. $R_{ji} = \sum_l (\frac{z_{ji\_kl}}{\sum_i z_{ji\_kl}} * R_{kl})$, where z_{ji_kl} captures the extent to which neuron ji has contributed to make neuron kl relevant, and R_{kl} captures the relevance of neuron l at layer k. For example, for linear layers, $z_{ji\_kl} = a_{ji} * w^{+}_{ji\_kl}$, where a_{ji} is the activation of neuron i at layer j and $w^{+}_{ji\_kl}$ is the value of the weight on the connection between neuron ji and neuron ki, considering positive weights only. The denominator is used to redistribute the relevance received by a neuron to the lower layer proportionally to relative contributions.

The heatmap in Figure 2.7 shows that the result computed by the DNN was mostly influenced by the pupil and part of the eyelid, which are the non-white parts in the heatmap.

An additional key benefit of LRP is that it enables the computation of *internal heatmaps*, i.e., heatmaps for the internal layers of the DNN, based on the relevance score computed for every neuron in every layer. An internal heatmap for a layer k consists of a matrix with the relevance scores computed for all the neurons of layer k.

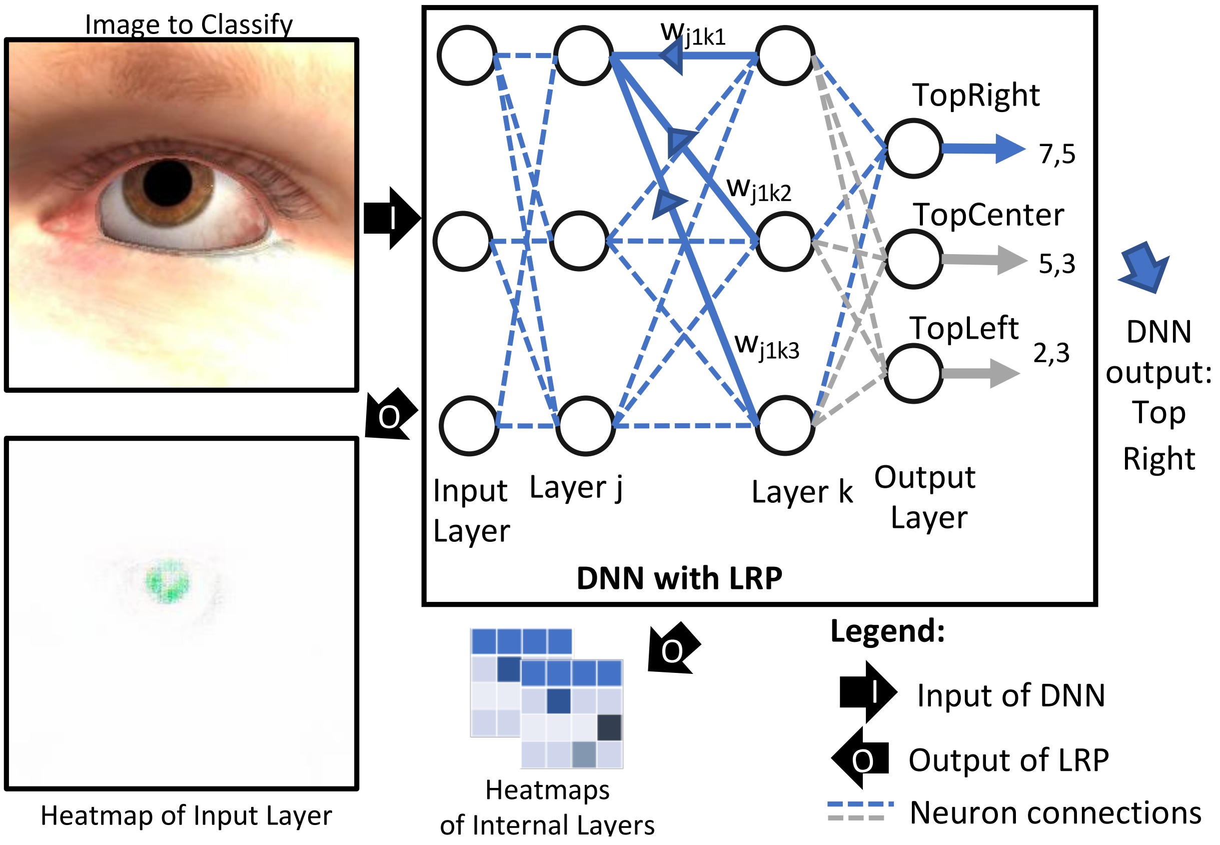


Figure 2.7: Layer-Wise Relevance Propagation.

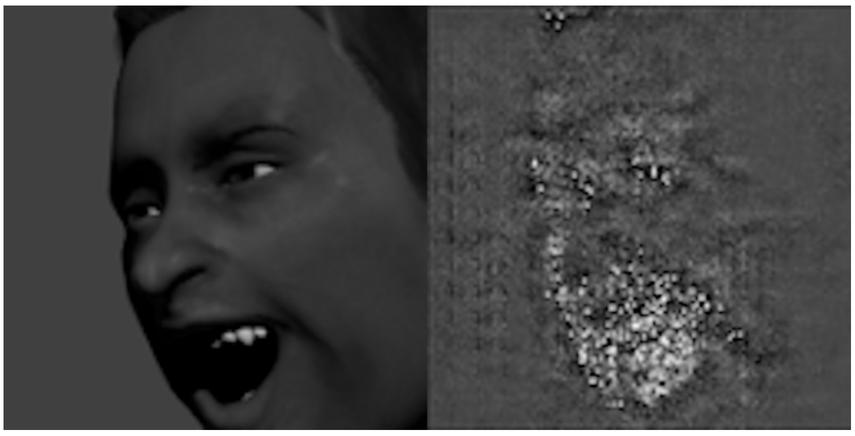


Figure 2.8: An example image of head-pose detection subject (on the left) and applied LRP (on the right) showing that the mouth had a large influence on the DNN behavior.

2.4.2 Model-agnostic DNN Explanations

Model-agnostic explanation approaches for DNNs are methods that aim to provide explanations for the predictions made by DNNs, without requiring access to the internal structure or training details of the model. These approaches are designed to be independent of the specific architecture or training procedure used to create the DNN, and can be applied to a wide range of different models.

One popular model-agnostic explanation method is called perturbation-based feature importance, which measures the contribution of each input feature to the model's output. This can be done through techniques such as permutation importance, which involves randomly shuffling the values of a single feature and measuring the resulting change in the model's predictions. One example of such approaches is *Anchors*. *Anchors* are if-then rules that constrain a subset of the input features so that changes to the unconstrained features do not influence the output of the model to be explained [68]. The Anchors algorithm constructs an explanation rule iteratively, by interacting with the model. At each iteration it alters the values associated to one input feature, till it identifies a range within which the accuracy is above a given threshold. The Anchors algorithm works with the test dataset, not simulators; also, when applied to DNN processing images, it does not generate expressions but identifies the image chunks influencing the DNN output, similarly to heatmap-based approaches.

Though designed for textual datasets, the Anchors algorithm may provide decision rules that can be easily generalized to multiple inputs (e.g., the ones that match the same rule and lead to the same result); for image inputs, Anchor simply emphasizes the image chunk that is sufficient for the classifier to make the prediction. Therefore, Anchors do not help engineers in analyzing large input datasets because it requires all the chunks belonging to each input image to be visualized.

Finally, Randomized Input Sampling for Explanation (RISE) [78] allows researchers to gain insights into the decision-making process of a NN without accessing its internal structure. It achieves this by sub-sampling input images using random masks and recording the network's response to each masked image. The paper also proposes causal metrics as an evaluation method for the generated explanations, aiming to assess the true cause of the model's decision rather than solely relying on human-annotated importance regions. However, such approach does not support the automated debugging and retraining of DNNs.

2.4.3 Explanations for DNN Testing Approaches

Some DNN testing approaches can provide explanations for portions of the input space in which DNN failures are observed [79, 34, 35, 80].

For instance, Abdessalem et al. [79] rely on evolutionary algorithms to search for test inputs using simulators and, to maximize test effectiveness, decision trees are used during the search process to learn the regions of the input space that are likely unsafe and, hence, should be targeted by testing. Finally, engineers are presented with decision tree leaves that characterize such portions.

Furthermore, DeepHyperion [80] configures a generative model using a metaheuristic search algorithm directed towards generating test inputs in a specific dimension of the inputs space and provides a set of feature maps which visualize the degree of accuracy obtained for different values of dimensions pairs.

Finally, DeepJanus characterizes the frontier of DNN misbehaviours by identifying pairs of inputs that are close to each other, with one input leading to a correct DNN output and the other to a DNN failure [81]. It relies on the popular Non-dominated Sorting Genetic Algorithm II NSGA-II algorithm extended with an archive (to keep the best individuals found in the search) and with repopulation (to escape from stagnation by replacing the most dominated individuals with random ones). DeepJanus relies on a fitness function that includes a measure of sparseness of the solutions; sparseness is measured as the distance from the closest input in the archive, which is populated with inputs having a distance above a given threshold. Finally, DeepJanus cannot relate failures observed with images generated by a simulator to failures observed with real-world data.

Chapter 3

Heatmap-based Unsupervised Debugging of DNNs

We propose Heatmap-based Unsupervised Debugging of DNNs (HUDD), an approach that automatically supports the identification of root causes of DNN failures. HUDD identifies root causes by applying a clustering algorithm to heatmaps capturing the relevance of every DNN neuron on the DNN outcome. Automated root cause analysis ensures that even uncommon and hazardous scenarios are brought to the attention of analysts, particularly when time is limited for safety analysis. Through clustering, this technique utilizes heatmaps to make safety analysis more robust and less dependent on perfect test sets. In essence, the clustering-based approach enhances the visibility of rare and unsafe conditions, thereby improving the overall reliability of safety analysis. Also, HUDD retrains DNNs with images that are automatically selected based on their relatedness to the identified image clusters.

3.1 Introduction

A root cause is *a source of a defect such that if it is removed, the defect is decreased or removed* [82]. With DNN-based systems, root cause analysis consists in characterizing system inputs that lead to failing DNN results. For example, in image classification tasks, a root cause of DNN failures could be severe gender imbalance in the training set leading the DNN to label most female doctors as nurses; it might be detected after noticing that failure-inducing inputs are characterized by doctors with long hair [63]. The DNN can be efficiently retrained after including in the training set additional images featuring these failure-inducing characteristics.

When DNN inputs are images, which is our focus here, existing solutions for root cause analysis generate heatmaps that use colors to capture the importance of pixels in their contribution to a DNN result [63, 62]. By inspecting the heatmaps generated for a set of erroneous results, a human operator can determine that these heatmaps highlight the same objects, which may suggest the root cause of the problem (e.g., long hair [63]). Based on the identified root cause, engineers can then retrain the

DNN using additional images with similar characteristics. Unfortunately, this process is expensive and error-prone because it relies on the visual inspection of many generated heatmaps. MODE goes beyond visual inspection and supports the automated debugging of DNNs through the identification of likely failure-inducing images to be used for retraining [69]. However, MODE cannot support safety analysis since it does not provide support to identify plausible and distinct root causes leading to DNN failures.

To address these problems in the context of DNNs analyzing images, we propose HUDD. HUDD relies on hierarchical agglomerative clustering [38] combined with a specific heatmap-based distance function to identify clusters of failure-inducing images with similar heatmaps for internal layers. Since heatmaps capture the importance of neurons regarding their contribution to the DNN result, failure-inducing images with similar heatmaps should share characteristics that drive the generation of erroneous DNN results. Each cluster should thus characterize a distinct root cause for the observed DNN failures, even in cases where such causes are infrequent. Images in such clusters should then help identify clear and distinct root causes and can serve as a basis for efficient and effective retraining. We focus on internal DNN layers because they act as an abstraction over the inputs (e.g., ignore image background). More precisely, HUDD relies on the computed clusters to identify new images to be used to retrain the DNN.

Given a potentially large set of collected or generated unlabeled images, HUDD selects the subset of images that are closer to the identified clusters according to a heatmap-based distance. These images are then labeled by engineers and used to retrain the network. Labeling only a subset of images reduces retraining cost.

3.2 The HUDD Approach

Figure 3.1 provides an overview of our approach, HUDD, which provides two main contributions: (1) it automatically identifies the root causes of DNN failures, (2) it automatically identifies unsafe images for retraining the DNN. HUDD consists of six steps, described below.

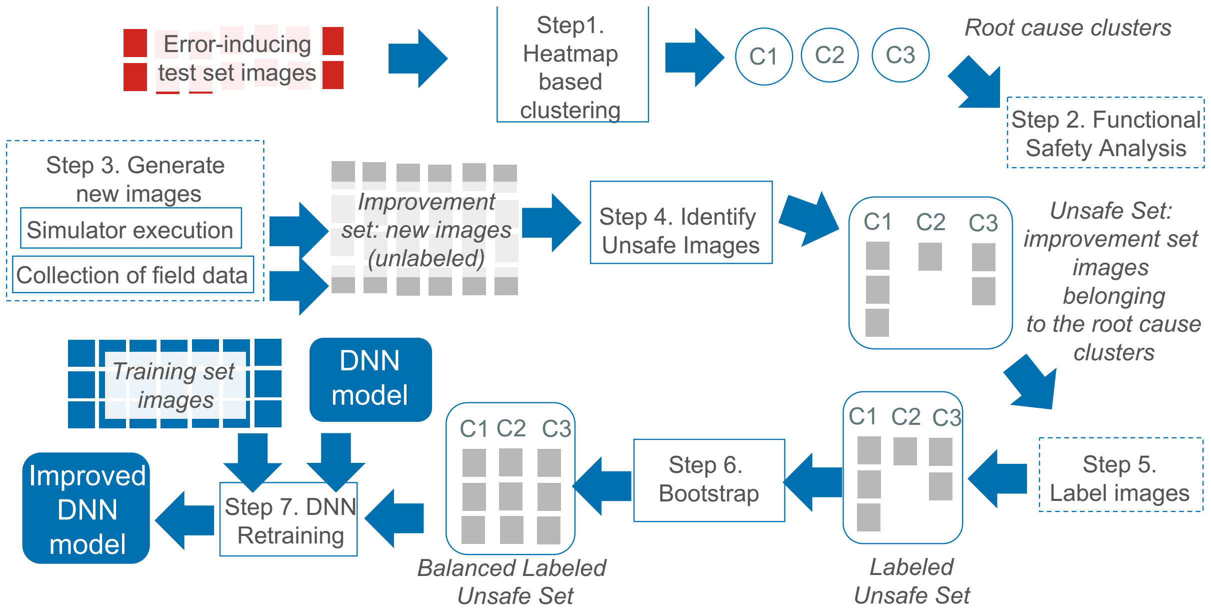

Error-inducing test set images
Step1. Heatmap based clustering
C1
C2
C3
Root cause clusters
Step 2. Functional Safety Analysis
Step 3. Generate new images
Simulator execution
Collection of field data
Improvement set: new images (unlabeled)
Step 4. Identify Unsafe Images
C1 C2 C3
Unsafe Set: improvement set images belonging to the root cause clusters
Training set images
DNN model
Improved DNN model
Step 7. DNN Retraining
C1 C2 C3
Balanced Labeled Unsafe Set
Step 6. Bootstrap
C1 C2 C3
Labeled Unsafe Set
Step 5. Label images

Figure 3.1: Overview of HUDD.

In Step 1, HUDD performs heatmap-based clustering. This is a core contribution of this approach and consists of three activities: (1) generate heatmaps for the failure-inducing test set images, (2) compute distances between every pair of images using a distance function based on their heatmaps, and (3) execute hierarchical agglomerative clustering to group images based on the computed distances. Step 1 leads to the identification of root cause clusters, i.e., clusters of images with a common root cause for the observed DNN failures.

In Step 2, engineers inspect the root cause clusters (typically a small number of representative images) to identify unsafe conditions, as required by functional safety analysis. The inspection of root cause clusters is an activity performed to gain a better understanding of the limitations of the DNN and thus introduce countermeasures for safety purposes (see Section 2.2.3), if needed. However, the inspection of root cause clusters has no bearing on the later steps of our approach, including retraining.

In Step 3, engineers rely on real-world data or simulation software to generate a new set of images to retrain the DNN, referred to as the *improvement set*.

In Step 4, HUDD *automatically* identifies the subset of images belonging to the improvement set that are likely to lead to DNN failures, referred to as the *unsafe set*. It is obtained by assigning the images of the improvement set to the root cause clusters according to their heatmap-based distance.

In Step 5, engineers manually label the images belonging to the unsafe set, if needed (e.g., in the case of real images). Different from traditional practice (see Figure 2.5-b), HUDD requires that engineers label only a small subset of the improvement set.

In Step 6, to improve the accuracy of the DNN for every root cause observed, regardless of their frequency of occurrence in the training set, HUDD balances the labeled unsafe set using a bootstrap resampling approach.

In Step 7, the DNN model is retrained by relying on a training set that consists of the union of the original training set and the balanced labeled unsafe set.

In general, generating a sufficiently diverse set of failure-inducing inputs that include all possible causes of DNN failures is very difficult. Further, when the labeling of such images is manual, the costs of labeling becomes prohibitive and DNN improvement is hampered. To enhance the efficiency of the DNN retraining process, it would be beneficial to automatically identify images that are prone to causing DNN failures. This would enable the image generation or selection process to focus on specific image characteristics, allowing for a more targeted approach to retraining the DNN.

Three out of the seven steps are manual (Steps 2, 3, and 5). However, these steps are also part of state-of-the-art solutions (see above). But in the case of HUDD the manual effort required in such steps is much more limited than in existing approaches. With HUDD, in Step 2, engineers inspect a few images per root cause clusters rather than the whole set of images, thus resulting in (a) significant cost savings (see Section 3.3.2) and (b) effective guidance towards the identification of root causes. In Step 5, with HUDD, engineers label only a subset of the improvement set that contains likely unsafe images identified by HUDD. Such unsafe images can be effectively used for retraining. Without HUDD, engineers would label a randomly selected subset of the improvements set, which would likely contain less unsafe images and thus be less effective during retraining (see Figure 3.11).

Finally, Step 3 is common practice and entails limited effort (e.g., buying field images or configuring a simulator).

The quality of HUDD results does not depend on the personal ability of engineers involved in manual steps; indeed, manual steps either concern the inspection of HUDD results or involve simple activities.

The first contribution of HUDD (i.e., identify root causes of DNN failures) is provided by Step 1, which is fully automated. Step 2, which is manual, concerns the visual inspection of the generated clusters, does not require particular skills, and is part of state-of-the-art approaches. However, with HUDD, this step is facilitated by the quality of the generated clusters; for example, in Section 3.3.2 we demonstrate that HUDD generates root cause clusters presenting a common set of characteristics that are plausible causes of DNN failures, thus facilitating the identification of these root causes. The other manual steps (i.e., Step 3 and Step 5) are simple. In Step 3, engineers simply generate additional images (their selection is automated by HUDD in Step 4). In Step 5, engineers provide additional labels, which is an activity that, despite being time-consuming, can be assumed to be correct most of the time and is unavoidable when supervised learning (e.g., DNNs) is involved. The other steps leading to the second contribution of HUDD (i.e., identification of unsafe images and DNN retraining), are fully automated.

The following sections describe in detail all the steps of the approach, except Steps 3 and 5, which were introduced above.

3.2.1 Step 1. Heatmap-based Clustering

HUDD is based on the intuition that, since heatmaps capture the relevance of each neuron on DNN results, failure-inducing inputs sharing the same root cause should show similar heatmaps. For this reason, to identify the root causes of DNN failures, we rely on clustering based on heatmaps. Figure 3.2 provides an overview of our clustering approach.

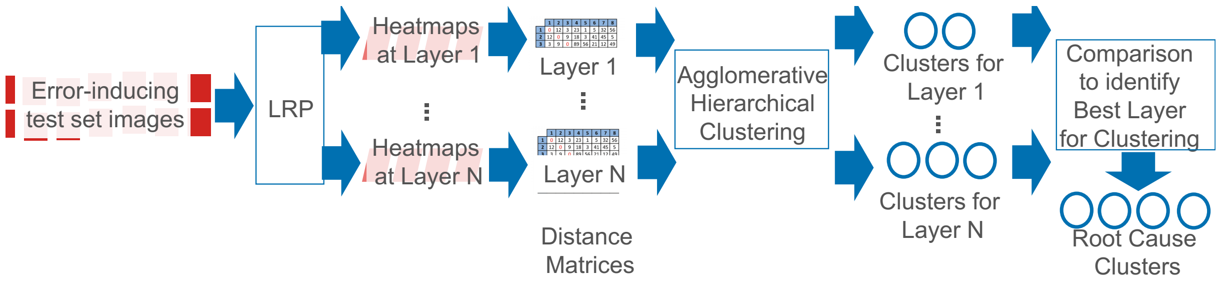


Figure 3.2: Heatmap-based Clustering

In this chapter, we rely on LRP because of the limitations of other approaches (see Section 2.4).

In our experiments, we have applied LRP to Convolutional Neural Networks (CNNs [83]), for classification tasks, and Hourglass Neural Networks [84], for regression tasks. We rely on the LRP implementation provided by LRP authors [85].

For each failure-inducing image in the test set, HUDD relies on LRP to generate heatmaps of internal DNN layers. Each heatmap captures the relevance score of each neuron in that layer.

A heatmap is a matrix with entries in $\mathbb{R}$, i.e., it is a triple (N, M, f) where $N, M \in \mathbb{N}$ and f is a map $[N] \times [M] \to \mathbb{R}$. We use the syntax $H[i,j]^L_x$ to refer to an entry in row i (i.e., $i < N$) and column j (i.e., $j < M$) of a heatmap H computed on layer L from an image x. The size of the heatmap matrix (i.e., the number of entries) is $N \cdot M$, with N and M depending on the dimensions of the DNN layer L. For convolution layers, N captures the number of neurons in the feature map, while M captures the number of feature maps. For example, the heatmap for the eighth layer of AlexNet has size 169×256 (convolution layer), while the the heatmap for the tenth layer has size 4096×1 (linear layer).

Since distinct DNN layers lead to entries defined on different value ranges [77], to enable the comparison of clustering results across different layers, we generate normalized heatmaps by relying on min-max normalization [86]. For a layer L, we identify min_L as the minimum value observed for all the heatmaps generated for a layer L, i.e.,

$$min_L \leq H[i,j]^L_x \; \forall \; i < N, j < M \tag{3.1}$$

with x being an image belonging to the unsafe test set. The maximum value max_L is derived accordingly. An entry $\tilde{H}[i,j]^L_x$ belonging to a normalized heatmap $\tilde{H}^T_x$ is derived as

$$\tilde{H}[i,j]^L_x = \frac{H[i,j]^L_x - min_L}{max_L - min_L} \tag{3.2}$$

The generated normalized heatmaps are used to build, for each DNN layer, a distance matrix that captures the distance between every pair of failure-inducing image in the test set. The distance between a pair of images $\langle a, b \rangle$, at layer L, is computed as follows:

$$heatmapDistance_L(a,b) = EuclideanDistance(\tilde{H}^L_a, \tilde{H}^L_b) \tag{3.3}$$

where $\tilde{H}^L_x$ is the normalized heatmap computed for image x at layer L.

$EuclideanDistance$ is a function that computes the euclidean distance between two $N \times M$ matrices according to the formula

$$EuclideanDistance(A,B) = \sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M}(A_{i,j} - B_{i,j})^2} \tag{3.4}$$

where $A_{i,j}$ and $B_{i,j}$ are the values in the cell at row i and column j of the matrix.

In this chapter, we rely on hierarchical agglomerative clustering (HAC) [38] to identify groups of failure-inducing images with similar characteristics.

Since we aim to generate clusters including images with similar characteristics, for each layer, we identify clusters of images by relying on the HAC algorithm with Ward linkage, which minimises within cluster variance.

HAC leads to a hierarchy of clusters that can be represented as a dendrogram (see Section 2.1.3). To identify the *optimal number of clusters*, we rely on the knee-point method [87], a recent approach that has been applied in different contexts, including fault localization [88] and performance optimization [89]. It automates the *elbow method* heuristics [90], which is commonly used in cluster analysis and consists of plotting the variance within a cluster as a function of the number of clusters, and picking the curve's elbow as the number of clusters to use. The knee-point method automates it by fitting a spline to raw data through univariate interpolation and normalizing min/max values of the fitted data. The knee-points are the points at which the curve differs most from the straight line segment connecting the first and last data point.

More specifically, we select the optimal number of clusters for a layer using the *knee-point method* applied to the weighted average intra-cluster distance.

We chose HAC over K-means [91] since the latter requires the number of clusters to be known or predicted. In our case, K-means would thus need to be repeatedly executed in order to determine the optimal number of clusters; in the case of HAC, instead, the generated dendrogram provides all the required information in a single run. Further, HAC does not require the computation of cluster centroids [92], which is particularly expensive when differences between observations are computed from large matrices [93]. To more formally motivate our choice, we compare the worst case time complexity of

HAC and K-means. HAC's running time is in $\mathcal{O}\left(\frac{d \cdot n \cdot (n-1)}{2} + n^2\right) \approx \mathcal{O}\left(d \cdot n^2\right)$, with n being the number of instances to cluster, and d being number of features to consider during clustering (i.e., the entries of the heatmap matrix). In other words, this complexity depends on the cost of computing the distance matrix, which is equal to the number of features multiplied by the number of image pairs (first addend), and the time complexity of HAC with Ward linkage (second addend [92]). The worst case time complexity of a single iteration for K-means is $\mathcal{O}\left(n^{k \cdot d}\right)$, with k being the number of clusters to consider [94, 95]. In our experiments, n lies in the range [506-5371], the *optimal number of clusters* is between 11 and 20 (see Section 3.3.2), and d is very large for DNN convolutional layers (e.g., 169×256). These numbers show that the worst case complexity of K-means is much larger than that of HAC, which further motivates our choice. We leave to future work the empirical evaluation of K-means and other clustering solutions.

In our context, clustering results are informative if they group together images that are misclassified for a same reason (i.e., if clusters are cohese) and if similar images belong to a same cluster (i.e., clusters are not fragmented). We determine cluster cohesion based on the weighted average intra-cluster distance ($WICD$), which we define according to the following formula:

$$WICD(L_l) = \frac{\sum_{j=1}^{|L_l|} \left(ICD(L_l, C_j) * \frac{|C_j|}{|C|} \right)}{|L_l|} \tag{3.5}$$

where L_l is a specific layer of the DNN, $|L_l|$ is the number of clusters in the layer L_l, ICD is the intra-cluster distance for cluster C_i belonging to layer L_l, $|C_j|$ is the number of elements in cluster C_j, while $|C|$ is the number of images in all the clusters.

In Formula 3.5, $ICD(L_l, C_j)$ is computed as follows:

$$ICD(L_l, C_j) = \frac{\sum_{i=0}^{N_j} heatmapDistance_{L_l}(p_i^a, p_i^b)}{N_j} \tag{3.6}$$

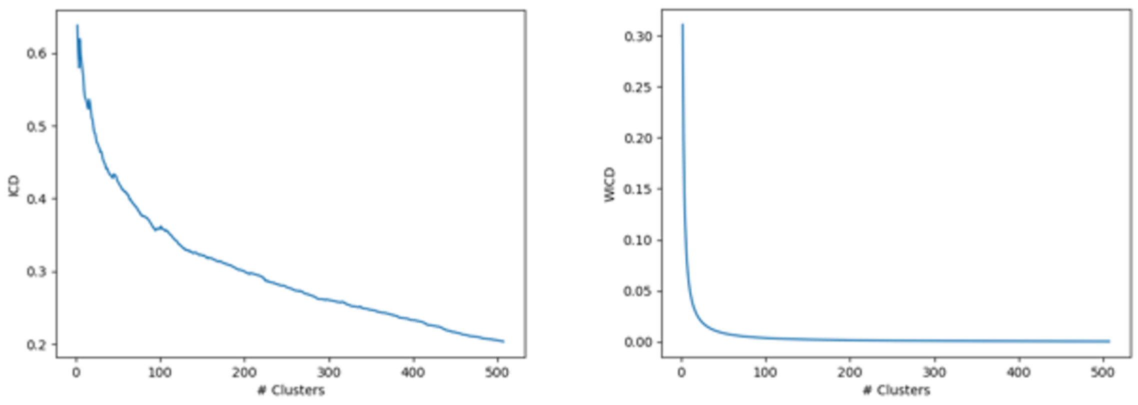
where p_i is a unique pair of images in cluster C_j, and N_j is the total number of pairs it contains. The superscripts a and b refer to the two images of the pair to which the distance formula is applied.

In Formula 3.5, the factor $\frac{|C_j|}{|C|}$ normalizes the average ICD with respect to the relative size of the cluster. It helps determine the optimal number of clusters within a layer and enables the identification of the best clustering result across layers, as explained in the following paragraphs.

Since Ward linkage groups together elements that minimize within-cluster variance, an increase in the number of clusters (i.e., less elements per cluster) leads to a proportional decrease in the average ICD. In other words, the ICD slope is mild and smooth, which complicates the identification of the optimal number of clusters through the elbow method. By taking into account the relative size of the cluster, WICD helps determine when a larger number of clusters leads to suboptimal results, which happens when an increased number of cluster does not break down large clusters but rather divide small clusters into tiny ones (i.e., they are fragmented).

Figure 3.3 shows the slope obtained for both ICD and WICD for a growing number of clusters; the plot for WICD clearly helps identify the sub-range on the X-axis leading to a drastic change in the slope, thus enabling the identification of an optimal number of clusters beyond which WICD barely decreases.

To determine when WICD stops decreasing significantly, we rely on its derivative that we approximate by relying on the fourth order central difference method [96]. We then rely on the knee-point method to identify the point with the maximum curvature in the derivative [87]. Figure 3.4 shows an example knee-point automatically identified with our method.



(a) ICD (b) WICD

Figure 3.3: Slope of ICD and WICD for GazeDNN

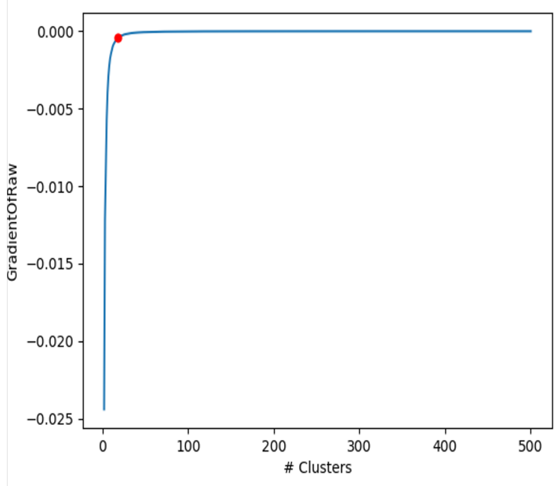


Figure 3.4: Gradient and knee-point (in red) for Figure 3.3-b

HUDD selects the layer L_m with the minimal $WICD$. By definition, the clusters generated for layer L_m are the ones that maximize cohesion and we therefore expect them to group together images that present similar characteristics, suggesting root causes for DNN failures.

When comparing clusters for distinct layers, the normalization based on the relative size of the cluster (i.e., the factor $\frac{|C_j|}{|C|}$ in Equation 3.5) enables HUDD to penalize layers including large clusters with high ICD. These clusters group together images with heatmaps that are different from each other and thus may be associated with different root causes for DNN failures.

3.2.2 Step 2. Root Causes Inspection

Root cause clusters are then inspected by engineers to determine unsafe conditions. For example, Figure 3.5 shows the clusters generated for the GD in Figure 3.7 on a test set with eye images generated by UnityEyes. To simplify the understanding of root causes, we printed the gaze angle on each image. Clusters $C1$ and $C2$ group together images that lead to DNN failures because the pupil is barely visible. In contrast, clusters $C3$ and $C4$ group images that are misclassified because the gaze angle is close to the classification threshold. Cluster $C5$, however, shows images that are misclassified because the training set labels are incomplete and do not capture the case of an eye looking middle center.

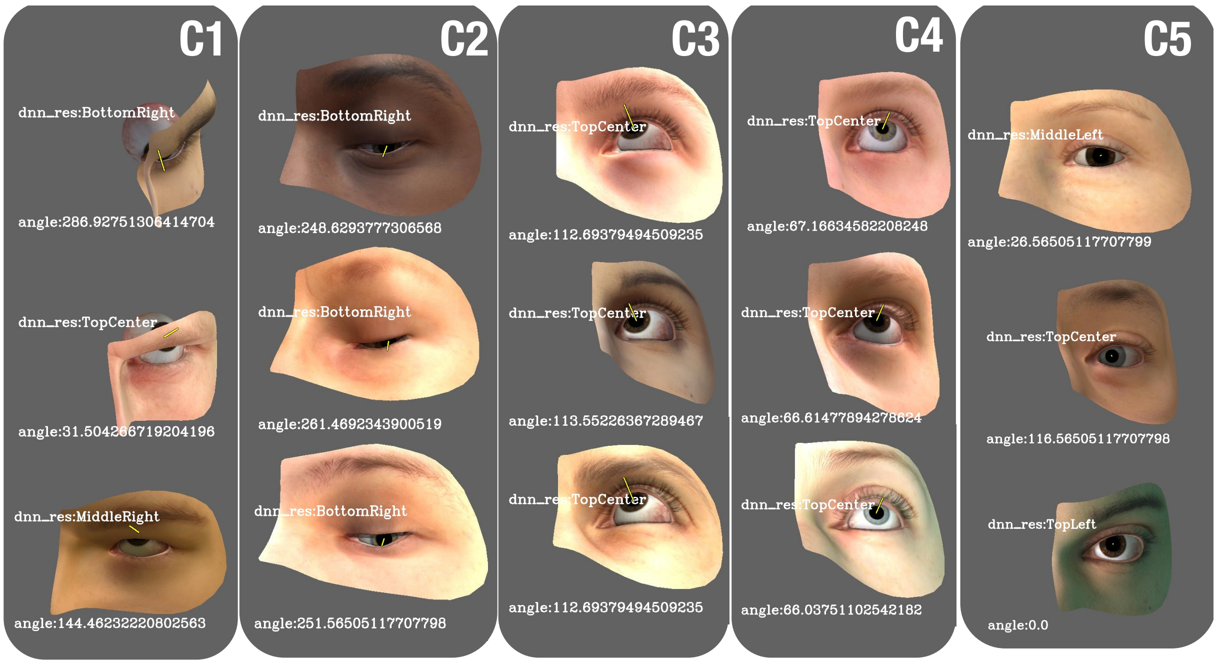


Figure 3.5: Clustering results for GD.

HUDD correctly handles both (1) the case in which erroneous DNN results with different output labels share the same root cause and (2) the case in which erroneous DNN results with the same output label are caused by distinct root causes. The first case is exemplified by cluster $C5$, which includes images that lead to different erroneous results (e.g., TopCenter or TopLeft) due to the same root cause (i.e., the eye is looking middle center but the DNN was not trained to detect it). The second case is exemplified by clusters $C5$ and $C3$, both including images erroneously classified as TopCenter. In cluster $C3$, this is due to the gaze angle close to the threshold with class TopRight whereas Cluster $C5$ includes images erroneously classified as TopCenter that are actually middle center.

In addition, the clusters in Figure 3.5 show that **HUDD identifies root causes that are associated with an incomplete training set** (e.g., borderline cases for gaze angle detected by $C3$ and $C4$) but also with **an incomplete definition of the predicted classes** (i.e., the middle center gaze detected by cluster $C5$ and the closed eyes detected by cluster $C2$) and **limitations in our capacity to control the simulator** (i.e., unlikely face positions detected by cluster $C1$).

The first case is addressed by HUDD retraining procedures (i.e., Steps 4-7) whereas the other causes require that engineers modify the DNN (e.g., to add an output class) or improve the simulator.

To further simplify the inspection of root cause clusters, our toolset also generates a set of animated *GIF* images, one for each cluster [97]. Each generated GIF image shows all the images belonging to a cluster one after the other. Animated GIFs enable engineers to inspect a large number of images in a few seconds (e.g., we configure our tool to visualize 100 images in a minute) thus facilitating the detection of the common characteristics among them.

3.2.3 Step 4. Identification of Unsafe Images

HUDD processes the improvement set to automatically identify potentially unsafe images. This is done by assigning improvement set images to root cause clusters while limiting the number of assigned images.

To assign images to clusters, HUDD relies on the *single linkage method* (see Section 2.1.3). According to the single linkage method, an image y belonging to the improvement set IS, is assigned to the cluster containing the closest failure-inducing image from the test set. We rely on single linkage since it has a desirable property: If applied to the failure-inducing test set images used to generate the clusters, it ensures that every image is assigned to the cluster it belongs to. This is important since, in a realistic scenario where test set and improvement set images are collected from the field, it ensures the selection of unsafe images that are highly similar to failure-inducing ones.

Unfortunately, single linkage alone is not sufficient to identify unsafe images. Since the root cause clusters capture only the unsafe portion of the input space, every image in the improvement set, including the safe ones, will be assigned to a root cause cluster. In other words, a safe image might be assigned to a cluster simply because it happens to be closer to an image belonging to this particular cluster.

To address this problem, we heuristically select, for every cluster, the images that are likely failure-inducing by estimating the number of failure-inducing images for each failure-cause cluster in the improvement set. Since the improvement set is generally derived from the same population as the test set (e.g., real world images collected according to the same strategy as for the test set), we can assume that (1) the improvement set is characterized by the same accuracy as the test set and (2) the causes of DNN failures in the improvement set should follow the same distribution as the one observed in the test set (i.e., for every root cause cluster, we should observe a number of failure-inducing images proportional to what observed in the test set).

We compute the number U_{C_i} of images to be selected for a root cause cluster C_i, as follows:

$$U_{C_i} = (|\mathit{TestSet}| * \mathit{sf}) * (1 - \mathit{TestSetAcc}) * \frac{|C_i|}{|C|} \tag{3.7}$$

The term $(|\mathit{TestSet}| * \mathit{sf}) * (1 - \mathit{TestSetAcc})$ estimates the number of failure-inducing images that should be selected from the improvement set. The term $\frac{|C_i|}{|C|}$ indicates how to distribute these images across root cause clusters in order to preserve the proportion of the test set across clusters in the improvement set. The term $(|\mathit{TestSet}| * \mathit{sf})$ provides an upper bound for the unsafe set size as a proportion of the test set size, which is determined by the available budget for labelling. Indeed, $|\mathit{TestSet}|$ is the size of the test set, while sf is a selection factor in the range [0-1] (we use 0.3 in our experiments). The term $(1 - \mathit{TestSetAcc})$ indicates the proportion of failure-inducing images that should be observed in the improvement set, based on assumption (1) above. Multiplied by the unsafe set upper bound, it ensures that we select a fraction of it. Finally, the term $\frac{|C_i|}{|C|}$ indicates the fraction of unsafe set images that should be assigned to the root cause cluster C_i. The term $|C_i|$ is the size of the cluster C_i. The term $|C|$ is the number of failure-inducing images in the test set. Based on assumption (2) above, the fraction $\frac{|C_i|}{|C|}$ corresponds to the proportion of failure-inducing images from the test set belonging to the root cause cluster C_i.

Figure 3.6 shows the pseudocode of our algorithm for identifying unsafe images. It requires the root cause clusters R, the identifiers of the images belonging to the improvement set (IS), the selection factor sf, the test set accuracy (acc_{TS}), the size of the test set ($size_{TS}$), and a distance matrix DM_{IS} with the distances between the images in IS and the images in the failure-inducing test set, for the layer selected

Require: (1) R, root cause clusters. (2) IS, set with the identifiers of the images belonging to the improvement set. (3) sf, the selection factor used in Equation 3.7. (4) $size_{TS}$ the size of the test set. (5) acc_{TS} the accuracy for the test set. (6) DM_{IS}, distance matrix capturing the distance between images in the improvement set and images in *TS*.
Ensure: an associative array with the unsafe images associated to each root cause cluster

```
    //Initialize the array that will contain all the images to be processed
 1: $rankedClustersPerImage \leftarrow$ new associative array that will contain,
                    for every image, the IDs of clusters,
                    ranked based on their distance from the image
    //Set the max number of images to be assigned to each cluster
 2: for clusterID in R do
 3:     $Uc[clusterID] \leftarrow (size_{TS} * sf) * (1 - acc_{TS}) * \frac{sizeOf(R[clusterID])}{sizeOf(R)}$
    //For each image, rank clusters, based on HeatmapDistance
 4: for $img$ in $IS$ do
      //Generate an associative array capturing, for every cluster,
      the distance of $img$ from the closest image of the cluster
 5:   for $clusterID$ in R do
 6:       $clusterDists \leftarrow$ new associative array to store the distance of $img$
                    from every cluster
 7:       $closest \leftarrow$ use $DM_{IS}$ to identify the test set image that is closer to $img$
 8:                 among the ones belonging to $clusterID$
 9:       $clusterDists[clusterID] \leftarrow$ distance between $closest$ and $img$
      //Put clusters in the correct rank for $img$
10:   for rank in 1 .. $|R|$ do
11:       $clusterID \leftarrow$ position in $clusterDists$ containing the lowest value
12:       $rankedClustersPerImage[rank][img] \leftarrow$
13:                 $< clusterID, clusterDists[clusterID] >$
14:       set $clusterDists[clusterID]$ to undefined
    //Assign images to clusters, trying to assign every image to the closer cluster first
15: for rank in 1 .. $|R|$ do
16:   $img \leftarrow$ the index $img$, in $rankedClustersPerImage[rank][img]$
              containing the lowest value, i.e., the image that is closer to any of
              the clusters
      //Save the ID of the cluster that is closer to $img$
17:   $clusterId \leftarrow rankedClustersPerImage[rank][img]$
18:   delete $rankedClustersPerImage[rank][img]$
      //Add the image to the cluster, if this is not already full
19:   if sizeOf ( $unsafeSet[clusterId]$ ) $< Uc[clusterId]$ then
20:       add $img$ to $unsafeSet[clusterId]$
          //Remove the image from the array with the images to process
21:       delete $rankedClustersPerImage[rank][img]$ for all the ranks
22: Return $unsafeSet$
```

Figure 3.6: Algorithm for the identification of unsafe images

for the identification of root cause clusters. To speed up the identification of unsafe images, since we focus on one specific layer, we compute the distance matrix based on the heatmaps returned by LRP, not the normalized ones.

The algorithm works by assigning U_{C_i} improvement set images to each root cause clusters C_i. It ensures that every image is assigned to the closest cluster C_i that has not been already filled with U_{C_i} images. The algorithm also handles the presence of spurious clusters, that is, clusters that group together diverse images for which a common root cause cannot be clearly identified. Spurious clusters may be assigned with safe images or failure-inducing images that should belong to other clusters. By relying on a ranking strategy for the assignment of images to clusters, we alleviate the effect of spurious clusters. When spurious clusters are fully assigned, the algorithm correctly assigns to their respective clusters all the remaining images, even if they are accidentally closer to the spurious cluster.

In Figure 3.6, Lines 2 to 3 compute, for every cluster, the value of U_{C_i} according to Equation 3.7. Lines 4 to 14, for every image, rank clusters, based on single linkage distance. This is performed by computing the distance of the image from each cluster (Lines 5 to 9) and then sorting clusters accordingly (Lines 9 to 14), such that the cluster in rank 1 is the closest one.

In Lines 15 to 21, the algorithm assigns every image to the closer cluster that is not already full. It iterates over the ranks (Line 15), and for each rank r, it loops over the improvement set images starting from the ones that are closer to a cluster (Line 16). If the cluster (i.e., $clusterId$, Line 17) is not already full (Line 19), it assigns the image to the cluster (Line 20). If the cluster is full (e.g., if it is a spurious cluster), the image is processed in the next iteration, i.e., when the algorithm tries to assign images to clusters ranked as $r + 1$. Note that, for each rank, every image is processed only once (Lines 18 and 21 delete processed images).

The algorithm returns an unsafe set with U_{C_i} images selected for each cluster C_i. The selected images are labeled by engineers when required (Step 6 in Figure 3.1) and then used for retraining.

3.2.4 Steps 6-7. DNN Retraining

IEE engineers train the DNNs that compose their systems by following the standard machine learning process depicted in Figure 2.5-a. They first train the DNN using a training set with labeled images (Step A) and then execute the DNN against a labeled test set (Step B). This process enables engineers to evaluate the DNN accuracy (e.g., the percentage of images leading to correct results).

When the accuracy of the system is not adequate, engineers typically improve the DNN by augmenting the training set with failure-inducing images. This process is depicted in Figure 2.5-b. First, engineers generate a set of new images to be used to retrain the DNN (Step C). We call this set of images *improvement set*. The improvement set generally consists of images collected from the field since these tend to be failure-inducing when DNNs have been trained using simulator images. Real-world images must be manually labelled (Step D). The DNN model is tested with the improvement set and images that lead to DNN failures are identified (Step E). This set of failure-inducing (unsafe) images is considered to retrain the DNN (Step G), using as initial configuration for DNN weights the ones in the previously trained model.

To improve the DNN, it is necessary to process a sufficiently large number of unsafe images. For this reason, the number of unsafe images can be augmented by applying *bootstrap resampling* (i.e., by replicating samples in the unsafe set [50]) till a target is achieved (Step F). Finally, the improved DNN can be assessed on the test set (Step H).

Similarly, HUDD retrains the DNNs by executing the DNN training process against a data set that is the union of the original training set and the labeled unsafe set. HUDD uses the available model to set the initial configuration for the DNN weights. The original training set is retained to avoid reducing the accuracy of the DNN for parts of the input space that are safe (i.e., showing no failure in the test set).

Furthermore, HUDD balances the unsafe set with bootstrap resampling [50], i.e., it randomly duplicates the images belonging to the cluster until every cluster has the same size. This is done to maximize the chances of eliminating every root cause of failure, even the ones that are rare (i.e., the ones for which we identify less unsafe set images). More formally, assuming $Max(|U_{C_i}|)$ being the size of the largest root cause cluster, bootstrap resampling ensures that every cluster contains $Max(|U_{C_i}|)$ members.

The retraining process is expected to lead to an improved DNN model compared to that based on the original training set.

3.3 Evaluation

Our empirical evaluation aims to address the following research questions:

RQ1 *Does HUDD enable engineers to identify the root causes of DNN failures?* We aim to investigate whether images belonging to a same cluster, as generated by HUDD, present a common set of characteristics that are plausible causes of DNN failures.

RQ2 *How does HUDD compare with traditional DNN accuracy improvement practices?* We aim to investigate whether HUDD enables engineers to efficiently drive the retraining of a DNN compared with state-of-the-art approaches.

To perform our empirical evaluation, we have implemented HUDD as a toolset that relies on the PyTorch [98] and SciPy [99] libraries. Our toolset, case studies, and results are available for download [20].

3.3.1 Subject Systems

To address RQ1, we need to objectively and systematically identify commonalities among images belonging to the same cluster. To do so, we rely on images generated using simulators as it allows us to associate each generated image to values of the configuration parameters of the simulator. These parameters capture information about the characteristics of the elements in the image and can thus be used to objectively identify the likely root causes of DNN failures.

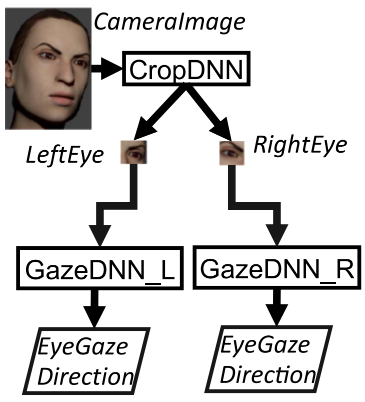


Figure 3.7: DNN-based system for gaze detection.

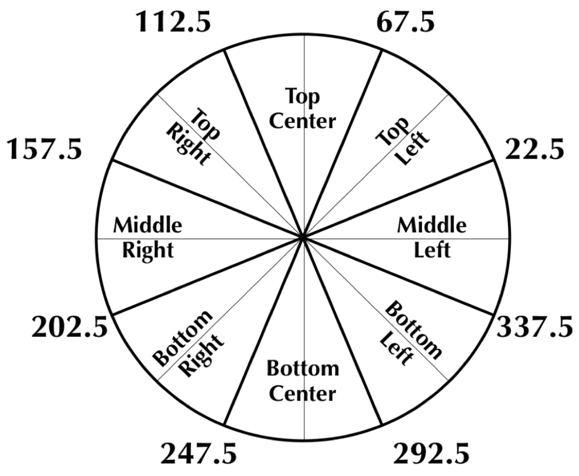


Figure 3.8: Gaze directions.

We consider DNNs that implement the key features of gaze detection, drowsiness detection, headpose detection, and face landmarks detection systems under development at IEE. In our experiments, for the gaze detection system (hereafter referred as GD), we rely on the UnityEyes simulator to generate eye images [49]. UnityEyes combines a generative 3D model of the human eye region with a real-time rendering framework based on Blender [100]. We determine the gaze direction label from the gaze angle parameter provided by UnityEyes, based on predefined gaze ranges depicted in Figure 3.8. For example, we assign the label *TopCenter* when the gaze angle is between 67.5 and 112.5 degrees.

The drowsiness detection system (OC) features the same architecture as the gaze detection system, except that the DNN predicts whether eyes are closed. The headpose detection system (HPD) receives as input the cropped image of the head of a person and determines its pose according to nine classes (straight, turned bottom-left, turned left, turned top-left, turned bottom-right, turned right, turned top-right, reclined, looking up). The face landmark detection system (FLD) receives as input the cropped image of the head of a person and determines the location of the pixels corresponding to 27 face landmarks delimiting seven face elements: nose ridge, left eye, right eye, left brow, right brow, nose, mouth. Each face element is delimited by several face landmarks.

GD, OC, and HPD follow the AlexNet architecture [101] which is commonly used for image classification. FLD, which addresses a regression problem, relies on an Hourglass-like architecture [84]. It includes 27 output neurons, each one predicting the position (i.e., pixel) of a distinct face landmark. Since a small degree of error in the detected landmarks is considered acceptable, the output of FLD is considered erroneous if the average distance of the identified landmarks from the ground truth is above four pixels. To apply HUDD to FLD, we generate heatmaps by backpropagating the relevance of the worst output neuron, i.e., the output neuron with the highest distance from the ground truth. Since face elements present very different characteristics, we apply the HUDD clustering algorithm seven times, once for each face element, by selecting images whose worst output neuron corresponds to the considered face element.

The first four rows of Table 3.1 provide details about the four DNNs described above. Column *Data Source* reports the name of the simulator generating the images used to train and test the network. GD and OC have been trained and tested with images generated by UnityEyes. Since classes need to be balanced in order to properly train the DNN, for OC, we selected a subset of images consisting of all the closed eyes and the same number of open eyes. For GD, this is not needed since UnityEyes selects the gaze angle according to a uniform distribution. HPD and FLD have been trained and tested with images generated using a simulator developed in-house by IEE. The IEE simulator relies on 3D face models built with the MakeHuman [102] plug-in for Blender [100]. To emulate car cameras, it generates grey images. HPD and FLD share the same training and test sets. To generate images, we used six face models for the training set, one for the test set. The use of different face models for training and testing emulates realistic scenarios in which the images processed in the field belong to persons different than the ones considered for training the DNN. Figure 3.9 shows examples of nine head poses generated with the same face model.

In Table 3.1, column *Epochs* reports the number of epochs considered to train the network. All the DNNs have been trained for a number of epochs that was sufficient to achieve a training set accuracy above 80%. Columns *Training Set Size* and *Test Set Size* report the size of the training and test sets. Columns *Accuracy Training* and *Accuracy Test* indicate the accuracy obtained by the DNN when executed against images in the training and test sets. Though training set accuracy is above 87% for all the four DNNs, in the case of HPD and FLD, we observe a lower test set accuracy. This is due to the prediction task being more complex for HPD and FLD than for GD and OC; indeed, the DNNs for HPD and FLD are tested with images belonging to face models that are different than the ones used for training. This was not the case for GD and OC since UnityEyes does not provide the means to control face features and automatically selects them during simulation. In addition, in contrast to UnityEyes, the images generated with the IEE simulator using different face models are likely to be more diverse. Indeed, the six face models integrated in UnityEyes capture only the face area surrounding the eye (i.e., eyelid and a portion of the nose) while the face models of the IEE simulator capture the entire face. Consequently, when testing is based on new face models, it is more likely to lead to DNN failures in the case of HPD and FLD. The number of face models considered for training HPD and FLD is limited to six since the definition of a

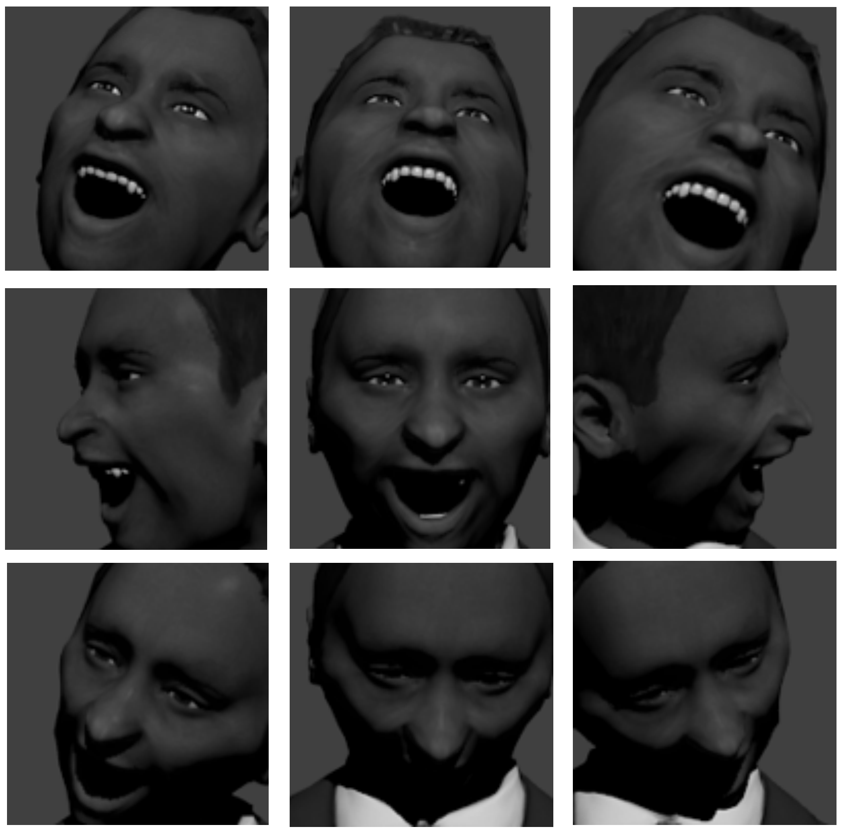


Figure 3.9: Example of distinct head poses of the same person generated with the simulator based on MakeHuman/Blender.

Table 3.1: Case Study Systems

DNN	Data Source	Training Set Size	Test Set Size	Epochs	Accuracy	
					Training	Test
GD	UnityEyes	61,063	132,630	10	96.84%	95.95%
OC	UnityEyes	1,704	4,232	10	87.38%	88.03%
HPD	Blender	16,013	2,825	10	94.45%	44.07%
FLD	Blender	16,013	2,825	10	88.97%	44.99%
OD	CelebA [103]	7916	5276	13	83.67%	84.11%
TS	TrafficSigns [104]	29,416	12,631	12	92.64%	81.65%

face model is an expensive manual task.

Since HUDD can be applied to DNNs trained using simulator or real images, to address RQ2, which concerns the improvement achieved after retraining the DNN, we also considered additional DNNs trained using real-world images. We selected DNNs implementing traffic sign recognition (TS), and object detection (OD), which are typical features of automotive, DNN-based systems. They are reported in the last two rows of Table 3.1. TS recognizes traffic signs in pictures. OD determines if a person wears eyeglasses. OD has been selected to compare results with MODE, a state-of-the-art retraining approach whose implementation is not available (see Section 3.3.2), but which is close in objective to HUDD. OD has been trained on the same dataset used for evaluating MODE but we selected a subset of the available images to balance classes (common practice). Though the original trained model is not available, we achieved the same accuracy as the one reported. The other two case studies considered in the MODE evaluation were discarded because they are either not representative (i.e., low accuracy) or lack information for enabling replication (i.e., description of inputs and outputs). TS and OD follow the AlexNet architecture [101].

3.3.2 Measurements and Results

RQ1. *Does HUDD enable engineers to identify the root causes of DNN failures?*

We refine RQ1 into three complementary subquestions (i.e., RQ1.1, RQ1.2, and RQ1.3), which are described in the following, along with the results obtained.

RQ1.1: Is the visual inspection of root cause clusters practically feasible?

Design and measurements.

We discuss whether the number of clusters generated by HUDD is small enough to make visual inspection feasible.

Since this research question does not concern the quality of the generated clusters, we considered all the case studies, including the ones trained and tested with real-world images. For each case study system, we thus report the number of root cause clusters generated by HUDD. Also, under the assumption that engineers visually inspect five images for each root cause cluster, we discuss the ratio of failure-inducing images that should be visually inspected when relying on HUDD. This ratio provides an indication of the time saved with respect to current practice (i.e., manual inspection of all the failure-inducing images). A user study concerning the time savings introduced by HUDD is part of our future work.

Experiment Results.

Table 3.2 shows, for each case study, the total number of failure-inducing images belonging to the test set, the number of root cause clusters generated by HUDD, and the ratio of failure-inducing images that should be visually inspected when using HUDD.

For the respective DNNs, HUDD identifies 16 (GD), 14 (OC), 17 (HPD), 71 (FLD), 14 (OD), and 20 (TS) root cause clusters. For all the case studies except FLD, the number of root cause clusters generated is below or equal to 20. Assuming that engineers inspect few images (e.g., five) for each cluster in order to determine plausible root causes, manual inspection based on HUDD appear to be practically feasible. In the case of FLD, the larger number of clusters is due to the identification of distinct root cause clusters for each face element; on average, we derive ten root cause clusters per element (within a [6 - 11] range). IEE engineers agreed that the manual inspection of a larger number of clusters is justified given the complexity of the case study.

Table 3.2: Root cause clusters generated by HUDD.

Case study	# Error-inducing images	# Root cause clusters	Ratio of inspected images
GD	5371	16	1.49%
OC	506	14	13.82%
HPD	1580	17	5.38%
FLD	1554	71	22.84%
OD	838	14	8.35%
TS	2317	20	4.31%

In general, the ratio of failure-inducing images that is inspected with HUDD is low, ranging from 1.49% (GD) to 22.84% (FLD), which shows that the analysis supported by HUDD saves a great deal of effort with respect to current practice (i.e., manual inspection of all the failure-inducing images).

RQ1.2: Do the clusters generated by HUDD show a significant reduction in the variance of simulator parameters?

Design and measurements.

This research question assesses if images belonging to the same cluster present similar characteristics. To address this research question, we rely on case studies trained and tested with simulator images. When images are generated with a simulator, images belonging to the same cluster should present similar values for a subset of the simulator parameters. In turn, this should result in a reduction of variance for these parameters in comparison to the entire failure-inducing test set. For a cluster C_i, the rate of reduction in variance for a parameter p can be computed as follows:

$$RR^{p}_{C_i} = 1 - \frac{variance\ of p\ for\ the\ images\ in\ C_i}{variance\ of\ p\ for\ the\ entire\ failure - inducing\ set} \tag{3.8}$$

Positive values for $RR^{p}_{C_i}$ indicate reduced variance.

Table 3.3 provides the list of parameters considered in our evaluation. In the case of GD and OC, we selected all the parameters provided by the simulator except the ones that capture coordinates of single points used to draw the pictures (e.g., eye landmarks) since these coordinates alone are not informative about the elements in the picture. However, we considered these coordinates to compute metrics that capture information about the scene in the image. We refer to such metrics as derived parameters. For example, we compute the distance between the bottom of the pupil and the bottom eyelid margin (*PupilToBottom* in Table 3.3). It determines if the eye is in an unusual position, e.g., if the eye is at the bottom of the orbit.

In the case of HPD, similarly to GD and OC, we considered the parameters provided by the simulator, excluding once again landmark coordinates. For parameters expressed with X-Y-Z coordinates, we considered the coordinate on each axis as a separate parameter. In the case of FLD, since a DNN failure may depend on the specific shape and expression of the face being processed (i.e., on the specific position of a landmark), we considered the coordinates of the 27 landmarks on the horizontal and vertical axes as distinct parameters (54 parameters in total).

We compute the percentage of clusters showing reduction in variance for at least one of the parameters. Since we do not know a priori the number of parameters that capture common failure causes, we consider variance reduction in one parameter to be sufficient. More precisely, we compute the percentage of clusters with a reduction in variance between 0.0 and 0.9, with incremental steps of 0.1. To answer positively our research question, a high percentage of the clusters should show a reduction in variance for at least one of the parameters.

Experiment Results.

Figure 3.10 shows the percentage of clusters with variance reduction for at least one of the simulator parameters, at different reduction rates.

We can positively answer RQ1.2 since all the clusters present at least one parameter with a positive reduction rate (> 0 in Figure 3.10). Also, a very high percentage of the clusters (i.e., 57% for OC, 96% for FLD, and 100% for both GD and HPD) include at least one parameter with a reduction rate above or equal to 0.5, i.e., 50% reduction in variance. Expectedly, as the threshold considered for variance

Table 3.3: Image parameters considered to address RQ1.1

DNN	**Parameter**	**Description**
GD/OC	Gaze Angle	Gaze angle in degrees.
	Openness	Distance between top and bottom eyelid in pixels.
	H_Headpose	Horizontal position of the head (degrees)
	V_Headpose	Vertical position of the head (degrees)
	Iris Size	Size of the iris.
	Pupil Size	Size of the pupil.
	PupilToBottom	Distance between the pupil bottom and the bottom eyelid margin.
	PupilToTop	Distance between the pupil top and the top eyelid margin.
	DistToCenter	Distance between the pupil center of the iris center. When the eye is looking middle center, this distance is below 11.5 pixels.
	Sky Exposure	Captures the degree of exposure of the panoramic photographs reflected in the eye cornea.
	Sky Rotation	Captures the degree of rotation of the panoramic photographs reflected in the eye cornea.
	Light	Captures the degree of intensity of the main source of illumination.
	Ambient	Captures the degree of intensity of the ambient illumination.
HPD	Camera Location	Location of the camera, in X-Y-Z coordinate system.
	Camera Direction	Direction of the camera (X-Y-Z coordinates).
	Lamp Color	RGB color of the light used to illuminate the scene.
	Lamp Direction	Direction of the illuminating light (X-Y-Z coordinates).
	Lamp Location	Location of the source of light (X-Y-Z coordinates).
	Headpose	Position of the head of the person (X-Y-Z coordinates). It is used to derive the ground truth.
FLD	X coordinate of landmark	Value of the horizontal axis coordinate for the pixel corresponding to the i^{th} landmark.
	Y coordinate of landmark	Value of the vertical axis coordinate for the pixel corresponding to the i^{th} landmark.

reduction increases, the percentage of clusters tends to decrease. However, a 0.9 threshold is still matched by 29% (OC) to 93.33% (GD) of the clusters (69.48% on average), a very high proportion, thus showing that most of the clusters should present a noticeable common characteristic.

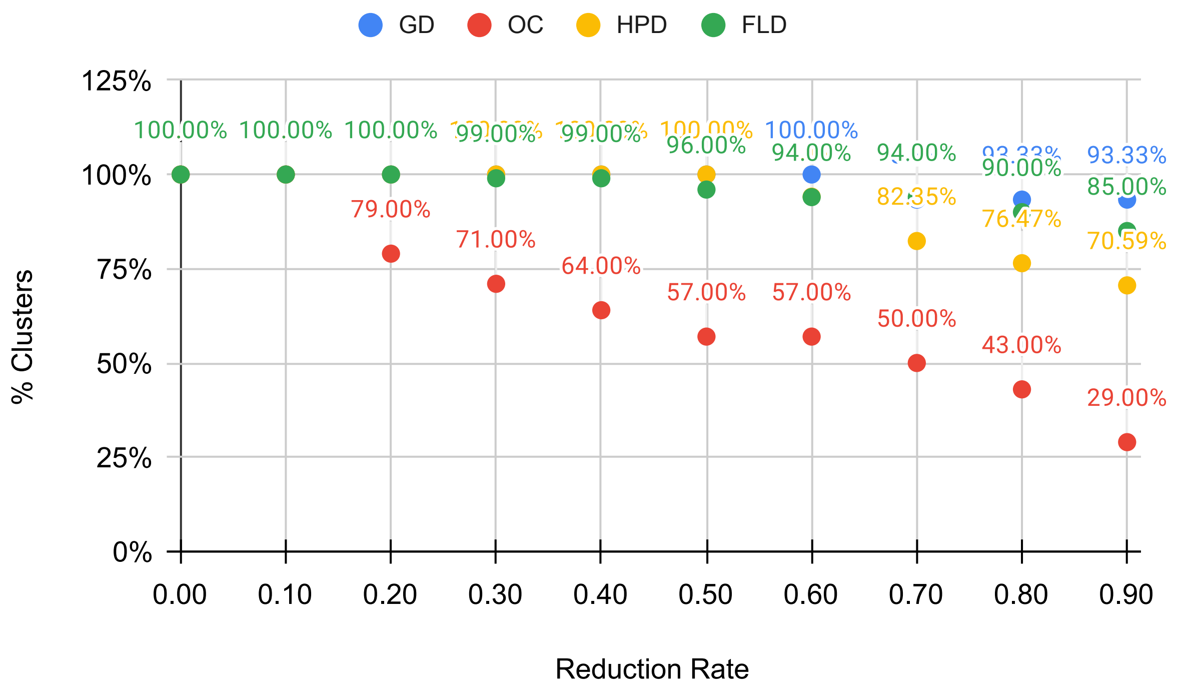


Figure 3.10: RQ1.1: Clusters with at least one parameter showing a reduction rate above thresholds in the range (0.0 - 1.0).

RQ1.3: Do parameters with high reduction in variance identify the plausible cause for DNN failures?

Design and measurements.

With RQ1.3, we ask whether the commonalities of the images belonging to the root cause clusters can help engineers determine the root causes of the DNN failures.

We expect DNN failures to be triggered in specific parts of the input space, each one capturing characteristics of the input images. To identify the input sub-spaces that are unsafe for our case studies, based on domain knowledge, we have identified a set of parameters (hereafter, *unsafe parameters*) for which it is possible to identify values (hereafter, *unsafe values*) around which, or below which, we are likely to observe a DNN failure. However, for FLD, it was not possible to determine, a priori, a set of unsafe parameters that might affect the results and we had to leave out that case study. Indeed, the position of a landmark may depend on many factors including the shape of the face element (e.g., thick lips), the element position (e.g., mouth being open), the headpose, and the camera position. Since the IEE simulator does not export information about the shape and position of face elements, it was not possible to define a set of metrics capturing plausible failure causes.

Table 3.4 provides the list of unsafe parameters, along with the unsafe values identified. For example, for the Gaze Angle parameter, unsafe values consist of the boundary values used to label images with the gaze direction.

Root cause clusters that are explanatory should present at least one characteristic that is noticeable by the engineer, i.e., they should have at least one parameter with high (i.e., 50%) reduction in variance. In addition, at least one of the parameters with high variance reduction should be an unsafe parameter. Finally, the cluster average should be close to one unsafe value. For $GazeAngle$, $Openness$, $H\_Headpose$,

Table 3.4: Safety parameters considered to address RQ1.3

DNN	Parameter	Unsafe values
GD, OC	Gaze Angle	Values used to label the gaze angle in eight classes (i.e., 22.5°, 67.5°, 112.5°, 157.5°, 202.5°, 247.5°, 292.5°, 337.5°).
	Openness	Value used to label the gaze openness in two classes (i.e., 20 pixels) or an eye abnormally open (i.e., 64 pixels).
	H_Headpose	Values indicating a head turned completely left or right (i.e., 160°, 220°)
	V_Headpose	Values indicating a head looking at the very top/bottom (i.e., 20°, 340°)
	DistToCenter	Value below which the eye is looking middle center (i.e., 11.5 pixels).
	PupilToBottom	Value below which the pupil is mostly under the eyelid (i.e., -16 pixels).
	PupilToTop	Value below which the pupil is mostly above the eyelid (i.e., -16 pixels).
HPD	Headpose-X	Boundary cases (i.e.,-28.88°,21.35°), values used to label the headpose in nine classes (-10°,10°), and middle position (i.e., 0°).
	Headpose-Y	Boundary cases (i.e.,-88.10°,74.17°), values used to label the headpose in nine classes (-10°,10°), and middle position (i.e., 0°).

$V\_Headpose$, $Headpose-X$, and $Headpose-Y$, since unsafe values split the parameter domains into subranges, we determine that the cluster average is close to one unsafe value if the difference between them is below 25% of the subrange including the average value. For $DistToCenter$, $PupilToBottom$, and $PupilToTop$, we simply check if the average is below or equal to the unsafe value. Finally, we compute the percentage of clusters for which the conditions above hold. To answer positively to RQ1.3, this percentage should be high.

Experiment Results.

In the case of GD, according to the conditions defined above, the percentage of clusters that identify the likely root cause of DNN failures is very high: 86.66% (13 out of 15). The identified unsafe parameters are *Angle*, *Openness*, and *DistToCenter*. For one cluster not meeting the conditions, the unsafe parameters (i.e., *DistToCenter*) have a reduction in variance of 44%, below the 50% threshold. This threshold is, however, arbitrary and a manual inspection of the cluster clearly shows that the commonality is the eye being abnormally open. The other non-compliant cluster shows pupils being partially masked by the eyelid; however, we could not define a measure to systematically capture this situation based on simulator parameters.

For OC, we obtain 57.14% (8 out of 14), with *Openness* and *X_Headpose* being the unsafe parameters. The remaining clusters are characterized either by thin almond eyes, an aspect of the simulation that is not controllable with parameters, and pupils being partially masked by the eyelid.

In the case of HPD, we obtain 88.24% (15 out of 17). For the two remaining clusters, the common characteristic is the presence of visible white teeth, which are not visible in training set images and may confuse the DNN since they stand out in grey-scale images. Based on the above observations, we respond positively to RQ1.3 since, in all cases, clusters are clearly associated with image characteristics that are plausible causes of failures.

RQ2. *How does HUDD compare to traditional DNN accuracy improvement practices?*

Design and measurements.

This research question aims to compare the accuracy improvements achieved by HUDD with the improvements achieved by baseline approaches, which do not rely on the automated selection of predicted unsafe images.

We consider two baseline approaches, namely $B1$ and $B2$. $B1$ has been introduced in Section 3.2.4 and consists of selecting for retraining the misclassified images belonging to the labeled improvement set. $B2$ is depicted in Figure 3.11. It follows the HUDD process except that it selects unsafe images randomly (i.e., the *Reduced improvement set*) instead of relying on root cause clusters. $B2$ enables the evaluation of the benefits of selection based on root cause clusters over random selection.

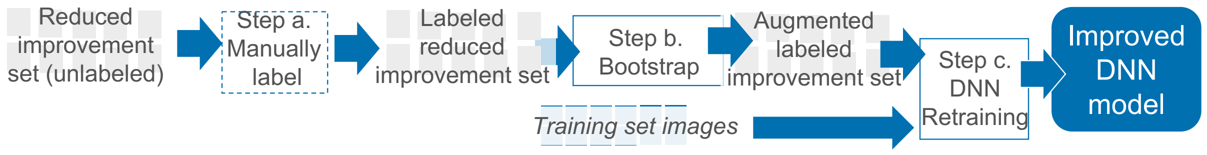

Reduced improvement set (unlabeled)
Step a. Manually label
Labeled reduced improvement set
Step b. Bootstrap
Augmented labeled improvement set
Step c. DNN Retraining
Improved DNN model
Training set images

Figure 3.11: Baseline 2 (B2).

To not introduce bias in the results, we rely on the same experiment setting for all the approaches (i.e., same configuration of the DNN training algorithm and same number of images to be labeled). In the case of HUDD, only the images in the unsafe set need to labeled. In the other cases, all the images in the improvement set must be labeled. For this reason, for the two baselines, we select an improvement set that is a random subset of the improvement set used by HUDD (referred to as *reduced improvement set*) and has the same size as the unsafe set generated by HUDD. To account for randomness, we repeat the experiment 10 times.

With HUDD, retraining the DNN was done by applying the approach described in Section 3.2.4. For $B1$ and $B2$, we configure bootstrap resampling to generate an *augmented labeled unsafe set* and an *augmented labeled improvement set* with the same size as the *balanced labeled unsafe set* for HUDD.

To answer the research question, we compute the accuracy of the retrained models on the test set and compare the accuracy improvement obtained by HUDD with that obtained by the baselines. We considered all the case studies listed in Table 3.1. The improvement set for GD and OC has been generated through additional executions of UnityEyes. To simulate a realistic scenario in which engineers collect additional data from the field or construct additional simulator models to improve DNN accuracy, the improvement sets for HPD and FLD have been generated with additional executions of the IEE simulator configured to use two new face models, which were not used for generating the training and test sets. For the other cases, we selected images of the original datasets which had not been used for the training and test sets.

Experiment Results.

The first eight columns of Table 3.5 provide the number of images used to retrain the DNNs. The remaining columns of Table 3.5 show the accuracy of the retrained models, the delta with respect to the best baseline, and $\hat{A}_{12}$ effect size [105]. For the accuracy, negative values indicate that the accuracy of the retrained model is worse than that of the original model. HUDD always fares better than the baseline approaches. Vargha and Delaney's $\hat{A}_{12}$ effect size is always equal or above 0.60, which indicates that, in all cases, HUDD has higher chances of generating accurate DNNs than baselines [105, 106]. In the paper by Vargha and Delaney [105], the authors specify the an effect size above 0.56 suggests a significant difference, with higher thresholds for medium (0.64) and large (0.71) effects.

HUDD accuracy improvements range from 0.28% to 30.24%. In contrast, $B1$ and $B2$ improvements range from -0.18% to 27.03% and -0.15% to 28.84%, respectively. For DNNs with an accuracy above

Table 3.5: RQ2. Size of Images Set used for Retaining and Accuracy Improvement

	Size of Images Sets for Retraining								**Accuracy**	**Accuracy**			**Delta**	$\hat{A}_{12}$	
	HUDD			**B1**			**B2**		**Original**	**(Accuracy Improvement)**			**wrt best**	**HUDD vs**	
DNN	**IS**	**US**	**BLUS**	**IS**	**LUS**	**ALUS**	**IS**	**ALIS**	**Model**	**HUDD**	**B1**	**B2**	**Baseline**	**B1**	**B2**
GD	72500	1615	4192	1615	156.4	4192	1615	4192	95.95%	96.23% **(+0.28)**	95.77% **(-0.18)**	95.80% **(-0.15)**	+0.43%	0.72	0.71
OC	4103	160	336	160	43.4	336	160	336	88.03%	94.41% **(+6.38)**	91.65% **(+3.62)**	92.33% **(+4.30)**	+2.08%	1.00	1.00
HPD	4700	481	697	481	12.7	697	481	697	44.07%	68.13% **(+24.06)**	66.73% **(+22.66)**	66.30% **(+22.23)**	+1.40%	0.70	0.72
FLD	6864	502	970	502	34.9	970	502	970	44.99%	75.23% **(+30.24)**	72.02% **(+27.03)**	73.83% **(+28.84)**	+1.40%	0.70	0.60
TS	9775	704	1860	704	53.2	1860	704	1860	81.65%	93.03% **(+11.38)**	92.63% **(+10.98)**	92.73% **(+11.08)**	+0.30%	0.83	0.67
OD	13194	258	770	258	69.1	770	258	770	84.12%	97.04% **(+12.92)**	96.63% **(+12.51)**	96.67% **(+12.55)**	+0.37%	0.79	0.60

IS: Improvement Set, US: Unsafe Set, BLUS: Balanced Labeled Unsafe Set; LUS: Labeled Unsafe Set (average over all the runs); ALUS/ALIS: Augmented Labeled Unsafe/Improvement Set.

80%, we have the following ranges: from 0.28% to 12.92% (HUDD), from -0.18% to 12.51% ($B1$), and -0.15% to 12.55% ($B2$). For HPD and FLD, which have lower initial accuracy, we have the following ranges: from 12.92% to 24.06% (HUDD), from -0.18% to 12.51% ($B1$), and -0.15% to 12.55% ($B2$). We can therefore conclude that HUDD is most useful when DNNs have lower accuracy and there is more room for improvement. Those are also the cases where retraining is most particularly important. The negative results obtained by the baselines for GD suggest that retraining the DNN without targeting the DNN-failure root causes may lead to worse accuracy. The choice of an inadequate strategy for retraining DNNs is therefore particularly detrimental since one could invest significant time and effort in labeling improvement set images without getting any benefit.

The difference in accuracy improvement between HUDD and the best baseline ranges between 0.30 (TS) and 2.08 (OC). Given that all techniques cost the same according to our experiment design, it is therefore recommended to use HUDD. In addition, there is a larger average difference (≥ 1.40) between HUDD and baselines in the cases of OC, HPD, and FLD. There are three plausible reasons to explain these differences. One is that, for HPD and FLD, there is significant room for accuracy improvement in the original models. Second, to increase the accuracy of HPD and FLD, we require improvement set images that are very different from the training set ones (i.e., pictures generated with new face models). We deem this to be a realistic situation that generalizes beyond our case studies; for example, it might be observed also in autonomous driving systems where certain types of vehicles (e.g., e-scooters) are missing from the training set. Third, for the three DNNs above, the training set is missing unsafe situations where the predictions are expected to be challenging (e.g., very dim light in the image). The type of retraining described above is particularly important in our context since the reduction of the unsafe input space is a key objective of safety engineering practices for the automotive industry [15].

Though the above accuracy differences may appear small, they may nevertheless be important in the context of critical applications where every percentage point in improvement matters. Furthermore, one should recall that, when we are dealing with highly accurate DNNs, room for improvement is limited. Finally, results with OD show that HUDD achieves better accuracy than MODE (97.04% vs 89% [69] after DNN retraining). Despite differences in the DNN architecture used by HUDD and MODE, both models are trained on the same images. While MODE improves the model's overall accuracy from 83% to 89% ($+6\%$), HUDD improves the model's overall accuracy from 84% to 97% ($+13\%$). These results show the potential of HUDD which, in addition to a higher accuracy than MODE, also provides root cause clusters.

Execution Time

Table 3.6 provides details about the time required to perform our experiments. It reports on the total execution time and how it is distributed across the different steps of HUDD and the execution of the baseline approaches.

Table 3.6: Experiments Execution Time

DNN	Training	Testing	HUDD Step 1	HUDD Steps 4-7	BL1	BL2	Testing Improved DNNs	Total (hours)
GD	2.66%	2.00%	3.82%	26.91%	31.97%	26.64%	5.99%	375.3
OC	1.34%	8.93%	18.30%	15.63%	15.63%	13.39%	26.79%	3.7
HPD	2.01%	2.41%	21.16%	22.93%	24.14%	20.11%	7.24%	20.7
FLD	2.81%	1.41%	2.81%	28.43%	35.60%	28.10%	0.84%	177.9
OD	2.80%	0.73%	6.83%	29.13%	30.81%	28.01%	1.68%	29.8
TS	1.89%	1.94%	29.36%	20.47%	22.63%	18.86%	4.85%	30.9

In Table 3.6, columns 5 to 7 refer to the cumulative time over 10 repetitions, column 8 refers to the cumulative time required for testing the retrained models for HUDD, $B1$, and $B2$. Our experiments took between 3.7 (OC) and 375.3 hours (GD) across DNNs, which highlights the large endeavor entailed by repeating experiments ten times in order to be able to draw statistical conclusions. Execution time is driven by the size of the data sets (i.e., executions with DNNs trained and tested with larger data sets took more time). The time required by HUDD to improve the DNN (i.e., *Steps* 4-7)— which includes also the identification of unsafe images—is similar with the time required by baseline approaches (columns 6 and 7), thus showing that HUDD does not introduce delays in the retraining process. The time required to perform HUDD Step 1 is significant (between 40 minutes and 14 hours); however, Step 1 can be executed overnight and help reduce human effort to identify root cause clusters.

To be able to execute experiments for 638.3 hours, we parallelized the executions of experiments using the HPC cluster of the University of Luxembourg [107]. We relied on Intel Xeon Gold 6132 nodes (2.6 GHz with four Tesla $V100$ $16GB$ SXM2).

3.3.3 Threats to Validity

We target DNNs performing image analysis in the perception layer of safety-critical systems. To address threats to external validity, for RQ2, we have considered DNNs performing classification of body parts and road objects, which are typical features in automotive systems. Further, one regression DNN was also analyzed, thus showing the applicability of the approach beyond classification; to this end, we considered a DNN representative of the typical task of landmark detection. Though four subject DNNs out of six implement tasks motivated by IEE business needs, they address problems that are quite common in the automotive industry (i.e., angle determination and landmarks detection). To strengthen generalizability, for two of these four case studies (i.e., GD and OC), we relied on a third-party simulator used in computer vision research (i.e., UnityEyes). Further, we considered two case studies from related work (i.e., TS and OD) that rely on real images. Our benchmark DNNs are therefore both diverse and representative.

For RQ1, we could only consider a subset of the case studies having high-resolution simulators available. Simulation is based on Blender [100], a readily available and widely used technology, thus making our results more representative. In our experiments, to objectively and systematically evaluate the quality of the generated clusters, we relied on the analysis of simulator parameters rather than a user study, which, however, should be undertaken in the future. Though the fidelity of simulator images is always a question, our experimental results do not show different trends for real and simulated images with respect to the number of clusters and accuracy improvements. Indeed, the number of root cause clusters identified for the two DNNs working with real-world images (OD and TS), which are 14 and 20, are similar to the ones observed in DNNs with simulator images (ranging from 11 to 17). Also, the accuracy improvement obtained for OD and TS, i.e., +11.38 and +11.92 (see Table 3.5), is within the range observed with simulator images (i.e., +0.28 and +30.24).

In our work, we do not rely on the popular K-means algorithm because of its higher computational cost in our context (see Section 3.2). However, more computation might be justified by better performance results, which may include a higher variance reduction in root cause clusters, a larger number of explanatory root cause clusters, and higher accuracy improvement. Since our experiments have shown that (1) HUDD generates cohese and explanatory root cause clusters (RQ1), (2) the room for accuracy improvement is small (RQ2 results show that, for four out of six DNNs, HUDD accuracy is above 93%), and (3) our experiments already took 638.3 hours to complete, the empirical comparison of HAC with K-means and other clustering algorithms has not been considered a priority in this work.

Though HUDD background technology (i.e., LRP and HAC) is context-independent, future work will investigate the evaluation of the approach in different contexts (e.g., space industry).

3.4 Conclusion

In this chapter we introduced HUDD, an approach that automatically identifies the different situations in which an image processing DNN is likely to produce erroneous results. HUDD generates clusters (i.e., root cause clusters) containing misclassified input images sharing a common set of characteristics that are plausible causes for failures. This is achieved through an hierarchical agglomerative clustering algorithm applied to heatmaps capturing the relevance of neurons across different DNN layers on the result.

In addition, HUDD minimizes the effort required to select and label additional images to be used to augment the training set and improve the DNN.

This is done by automatically selecting images that are close to members of root cause clusters and thus unsafe. Only these selected images then need to be labeled by engineers. Since DNN failures are often due to an incomplete training set (e.g., lack of images with a gaze angle close to borderline), HUDD alleviates the problem by augmenting the training set with unsafe images.

With respect to a recent taxonomy of DNN faults [108], HUDD can identify different types of *training* and *input* faults, while automatically addressing problems due to *training data quality* (see Section 3.2.2). A more extensive evaluation of HUDD based on this taxonomy is part of our future work.

To summarize, HUDD is the first approach that facilitates the scalable identification of distinct failure root causes in DNNs, by applying clustering algorithms to heatmaps generated by DNN explanation techniques. For the latter, we rely on Layer-Wise Relevance Propagation (LRP), which is based on theoretical foundations that are generalizable to other DNN architectures. Further, HUDD relies on standard retraining procedures based on back propagation and gradient analysis, that have been widely applied and validated, and does not entail the direct modification of the learned DNN model.

As demonstrated in our empirical evaluation, HUDD can successfully support the debugging of DNNs that implement either classification or regression tasks.

Empirical evaluation with simulator images show that HUDD generates clusters of images sharing similar values for some of the simulation parameters driving the generation of images. We can conclude that such clusters can then serve as a useful instrument for the identification of root causes of DNN failures, as exemplified in our case studies. In turn, this information is important to safety analysis as it helps clearly characterize unsafe inputs, a requirement in safety standards. Our results, on both simulated and real images, also show how these clusters can be effectively used to select new images for retraining in a way that is more efficient than existing practices and leading to better DNN accuracy.

Chapter 4

Simulator-based Explanations for DNN failurEs

THE problem of automatically generating explicit descriptions for unsafe scenarios identifies by clustering real-world images is addressed in this chapter. Such descriptions are provided in terms of logical expressions constraining the configuration parameters of the simulator used to train the DNN (e.g., rotation angle of the driver's head and illumination angle). For example, we may report that the hazard-triggering event that prevents the gaze angle from being correctly estimated is the driver's head turned by more than 60 degrees with an illumination angle above 45 degrees (i.e., both eyes are barely visible and covered by a shadow). We name our approach Simulator-based Explanations for DNN failurEs (SEDE).

4.1 Introduction

Most of DNNs used in autonomous systems are trained and tested using both images generated by simulators and real-world images. Because of the labelling costs, real-world images are limited and mostly used for testing. Training is mostly based on simulator images, while testing should include a larger proportion of real-world images to ensure that the system is ready to be delivered.

The reference DNN training process is depicted in Fig. 4.1. It leverages simulators to reduce the cost related to collecting and labelling a large number of images. In general, engineers tend to initially train the DNN using images automatically generated with a high-fidelity simulator (e.g., a human body simulator). When the DNN is deemed accurate, to enable the correct processing of real-world images, engineers fine-tune the trained DNN using real-world images (e.g., images of real people seated in a car). Fine-tuning consists of relying on the DNN training algorithm to update the weights of the DNN trained with simulator images; retraining may concern the whole set or a subset of the DNN weights (e.g., the fully connected layers in a CNN). The fine-tuned DNN is then tested with real-world images.

In the presence of DNN failures, the failure-inducing images in the test set can be inspected by engineers to improve the DNN. The visual inspection of such images may enable engineers to identify scenarios that are unsafe (i.e., the DNN generates erroneous results), and collect additional related real-world images or configure the simulator to generate them. Given that this is an expensive activity, in this chapter, we propose a solution to automate the identification and characterization of unsafe scenarios observed in real-world images to enable generating simulator images that present similar triggering events and minimize retraining costs.

To overcome the cost and error-proneness of manual inspection of failure inducing images, the HUDD technique, presented in Chapter 3, automatically groups images showing a same root cause by analyzing heatmaps derived from DNN outputs and neuron activations — such groups of images are named root cause clusters (RCCs). The rationale behind HUDD is that images sharing the same root causes (e.g., a face turned left) should present similar neuron activations and, consequently, similar heatmaps. However, with HUDD, root cause analysis still remains error-prone because it relies on the capability of the engineer to interpret the generated outputs (e.g., she may not notice that DNN failures depend not only on a face being turned left but also on a specific illumination angle). Further, it does not fully leverage the availability of simulators to retrain DNNs; precisely, it does not guide the generation of images but simply selects a subset of images available fro retraining.

In this chapter, we propose *Simulator-based Explanations for DNN failurEs (SEDE)*, an approach to characterize the root causes (events) leading to DNN failures; we call such events hazard-triggering events. It targets contexts in which DNNs are partly trained using simulators, which is common practice in safety-critical contexts with complex inputs [109, 110, 111]; for example, DNNs implementing vision-based driving tasks or interpreting human postures. This is the case for our industry partner, which relies on a simulator capable of generating images of human bodies, seated in a car environment, to train DNNs that interpret human postures (e.g., determine gaze or drowsiness).

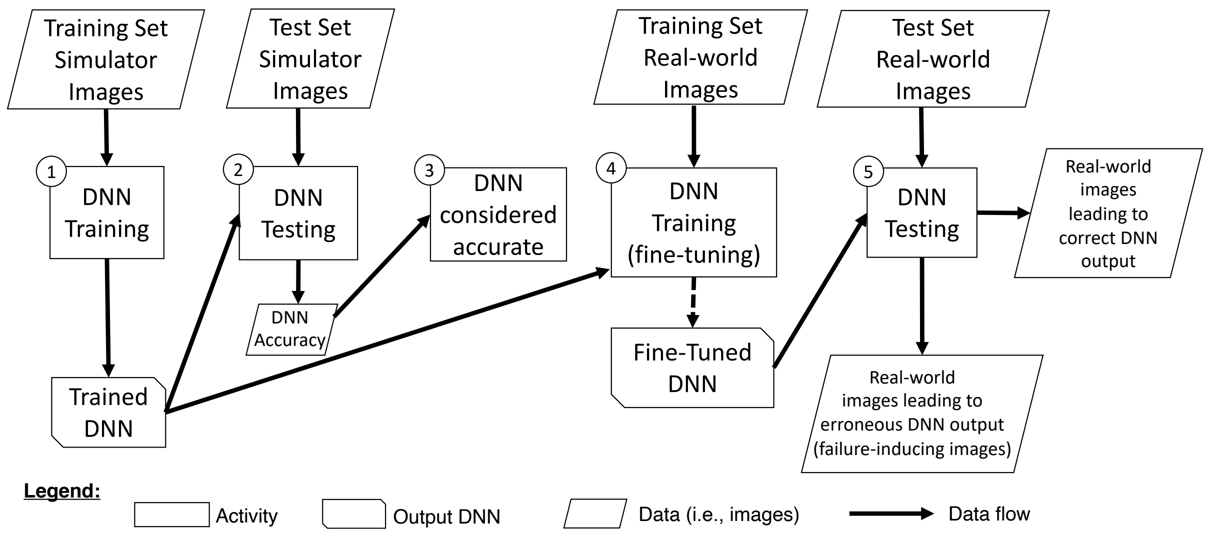

Training Set Simulator Images
Test Set Simulator Images
Training Set Real-world Images
Test Set Real-world Images
1 DNN Training
2 DNN Testing
3 DNN considered accurate
4 DNN Training (fine-tuning)
5 DNN Testing
Real-world images leading to correct DNN output
DNN Accuracy
Fine-Tuned DNN
Trained DNN
Real-world images leading to erroneous DNN output (failure-inducing images)
Legend:
Activity
Output DNN
Data (i.e., images)
Data flow

Figure 4.1: DNN training and testing

Given a set of failure-inducing real-world images (e.g., the failure-inducing images in the test set), SEDE relies on HUDD to generate RCCs. We assume that all the images belonging to a RCC present the same hazard-triggering events, as suggested by HUDD's empirical results. For each RCC identified by HUDD, SEDE relies on the simulator used for DNN training to generate more images that belong to the RCC. Since a RCC characterizes a small portion of the input space, SEDE relies on evolutionary algorithms to efficiently generate RCC images; indeed, evolutionary algorithms explore the input space guided by fitness functions measuring how close an input is to satisfying a given objective. To better identify the commonalities among RCC images and avoid generating images that present characteristics that are accidentally shared [1], we propose PaiR (Pairwise Replacement), a genetic algorithm that not only generates images belonging to the RCC but also maximizes their diversity.

4.2 Background on Evolutionary Algorithms for Diversity Optimization

In our work, we aim to rely on evolutionary algorithms to generate a set of diverse images belonging to a RCC. As described in the related literature [52, 34, 35, 81], an image can be modelled as an individual (or chromosome) of the population, represented as a vector whose components capture the values assigned to simulator parameters used to generate the image.

Although, in principle, genetic algorithms can be used to evolve a population of individuals and achieve our objectives (i.e., generate individuals belonging to the RCC and maximize the diversity between them), the use of well-known algorithms for this purpose (e.g., NSGA-II) is not feasible because it is not possible to define a fitness function for diversity, as explained next.

To maximize diversity, since it is the property of a set of images, it is not possible to define a fitness function that assesses how much an offspring individual contributes to diversity prior to knowing what the final set will be. Also, note that selection based on crowding distance, which is part of NSGA-II, does not help because such distance is defined over the objective space, as opposed to the simulator parameter space. Consequently, the NSGA-II algorithm cannot simply be adopted to address our problem.

We can represent a solution for one of our objectives (i.e., a set of n diverse images belonging to a RCC) as an individual where each image is modelled with a subsequence of the chromosomes (e.g., the first subsequence of m chromosomes are for the first images, the second for the second image, and so on).

This would enable *whole test suite generation* (i.e., the generation, within a same search run, of all the solutions for our objectives) [112]. However, different from traditional software testing, individuals achieving different objectives are unlikely to present dependencies. For example, in traditional software testing, nested conditions lead to dependencies between search objectives, whereas two RCCs should include images that are diverse (e.g., one picture of a person looking left and another picture with a person looking right); therefore, in our context, generating an image that fulfills one objective should not help with achieving another objective. Further, modelling multiple images as a single individual would lead to many simulator runs to generate a single individual, which leads to an inefficient exploration of the search space and, therefore, entails a large budget to generate the desired solutions. For these reasons we chose to model a single image as an individual.

An alternative solution consists of relying on an archive where, at each iteration of the search algorithm, we add the individuals in the first non-dominated front that contribute to increasing diversity. Related work suggests adding to the archive the individuals with a *sparseness* value (i.e., the distance from the nearest neighbour) that is above a set threshold (hereafter, *sparseness threshold*) [81, 113]. DeepJanus [81] and DeepMetis [113] are two state-of-the-art solutions that rely on a *sparseness threshold*. They extend

[1]An evolutionary algorithm may get stuck in a local optimum and, for example, generate images of persons with blue eyes whose head is turned 45 degrees, though the eye color does not affect the DNN outputs.

the popular NSGA-II algorithm with an archive and with repopulation (to escape from stagnation by replacing the most dominated individuals with random ones); we refer to such algorithm as *DeepNSGA-II*. Unfortunately, in our context, the sparseness threshold may directly affect the quality of the results; indeed, if the threshold is too low, most of the selected individuals will be similar to each other, thus making the learning algorithm derive rules that do not characterize the whole RCC. Similarly, if the threshold is too high, we end up selecting only a small number of individuals, which prevents the generation of accurate rules. Unfortunately, identifying an appropriate threshold requires multiple executions of the algorithm, which, in our context, shall be repeated for every RCC; indeed, since it may be easier for the simulator to generate certain images than others, some portions of the input space may be denser than others and, therefore, different thresholds might be needed for different RCCs. In practice, this would lead to an expensive process, which is practically infeasible. One simple solution is to rely on a sparseness threshold of zero (i.e., we add to the archive any image that differs from the ones already in it) [113]. However, such choice may render the algorithm less efficient as it leads to larger archives and the larger the archive the more costly the distance is to compute. Further, in our context (Section 4.3), the best individuals identified through the search (in the archive) are used as input for additional search iterations; therefore, the *DeepNSGA-II* algorithm should be combined with a strategy to select the best individuals in the archive (Section 4.4.2 provides details about the strategy adopted in our empirical evaluation).

In addition to the limitations indicated above, please note that DeepJanus and DeepMetis address problems that are different than ours. They both rely on two fitness functions, namely F_1 (to be maximized) and F_2 (to be minimized). In both DeepJanus and DeepMetis, at every search iteration, every individual of the population is added to the archive if F_1 is above the sparseness threshold and if F_2 is below a user-defined threshold capturing if the individual addresses the other objective of the search further described below. DeepJanus characterizes the frontier of DNN misbehaviours by identifying pairs of inputs that are close to each other, with one input leading to a correct DNN output and the other to a DNN failure [81]. Function F_1 focuses on sparseness and closeness of input pairs and is computed as follows:

$$F_1 = \{sparseness(i, A) - k * distance(i.m1, i.m2)\}$$

with $x.m1$ and $x.m2$ being a pair of inputs, $sparseness$ the function that computes the distance from the nearest individual in the archive A, and $distance$ the distance between two individuals. Function F_2 focuses on the closeness to the frontier and is computed as

$$F_2(i) = \begin{cases} eval(i.m1) \cdot eval(i.m2) & \text{if} > 0 \\ -1 & \text{otherwise} \end{cases}$$

with $eval$ *being the difference between (a) the confidence level associated with the expected label and (b) the maximum confidence level associated to any other class.* Based on the above, DeepJanus cannot be directly applied in our context because we do not aim to generate pairs of individuals. DeepMetis aims to identify inputs leading to DNN failures in mutated DNNs (i.e., DNN models trained or modified to be less accurate than the DNN under test) [113]. Different from DeepJanus, in DeepMetis, an individual is a single image. Function F_1 focuses only on sparseness

$$F_1 = \{sparseness(i, A)\}$$

Function F_2 measures how well an individual triggers inaccurate outputs in mutated DNNs and is computed as

$$F_2 = \sum_{mutant \in Mutant} eval_{mutant}(i)$$

with $eval_{mutant}$ returning the *difference between the confidence associated with the expected class and the maximum confidence associated with any other class when the prediction is correct (-1 if the prediction is wrong)*. Although F_1 might also be used in our context to drive the generation of sparse solutions, F_2 needs to be modified to capture our goals (i.e., the input belongs to a *RCC* and leads to a failure).

To address our problem, we thus defined an algorithm (i.e., the PaiR algorithm, described in Section 4.3.1) that takes advantage of every image generated by the simulator in an iteration and does not require a sparseness threshold. Further, we have separated the identification of images belonging to a RCC, which is achieved by PaiR, from the identification of failing images, which is achieved with additional search iterations performed through an extension of the NSGA-II algorithm. In Section 4.4.2, we compare PaiR with a DeepNSGA-II solution with a sparseness threshold of zero and F_2 matching the same fitness function used by PaiR to identify individuals belonging to a *RCC*.

PaiR does not support whole test suite generation but it is executed once for each RCC.

Once PaiR has generated images belonging to the RCC, to ensure that these images include hazard-triggering events, SEDE produces additional images that are mispredicted, in addition to belonging to the RCC. This is achieved by relying on a modified version of the multi-objective algorithm NSGA-II (hereafter, *NSGA-II'*). Different from recent extensions of NSGA-II [113, 81] that aim to maximize diversity (hereafter, DeepNSGA-II), PaiR does not rely on an archive and does not require the definition of thresholds for selecting the final set of images. Finally, to precisely characterize hazard-triggering events (i.e., to distinguish between commonalities that cause DNN failures and accidental similarities), SEDE relies on one additional execution of *NSGA-II'* to generate additional images that are similar to failure-inducing images but do not cause a DNN failure. The availability of failing and passing images enables SEDE to derive, leveraging the PART decision rule learning algorithm [36], a set of expressions for the simulator parameters that capture commonalities among failure-inducing images. These expressions characterize hazard-triggering events and also delimit part of the input space that shall be considered unsafe. The expressions generated by SEDE can be used to either estimate the probability of exposure to the hazard-triggering events (based on domain knowledge) or improve the DNN. Improvements in DNN accuracy can be achieved by retraining the model using additional images generated using a simulator, configured with parameters that match the expressions generated by SEDE.

4.3 The SEDE Approach

SEDE works in four steps depicted in Fig. 4.2. In the *first step*, SEDE relies on a state-of-the-art solution (HUDD, see Chapter 3), to automatically identify clusters of images (RCCs) leading to a DNN failure because of common hazard-triggering events.

In the *second step*, SEDE relies on evolutionary search driven by the analysis of heatmaps and simulator parameters to generate images that are associated to RCCs, enabling accurate learning to characterize unsafe scenarios. To precisely derive constraints on simulator parameter values (configurations) for distinct hazard-triggering events, while avoiding overfitting, for each RCC, we are interested in generating many diverse images leading to correct and erroneous DNN outputs.

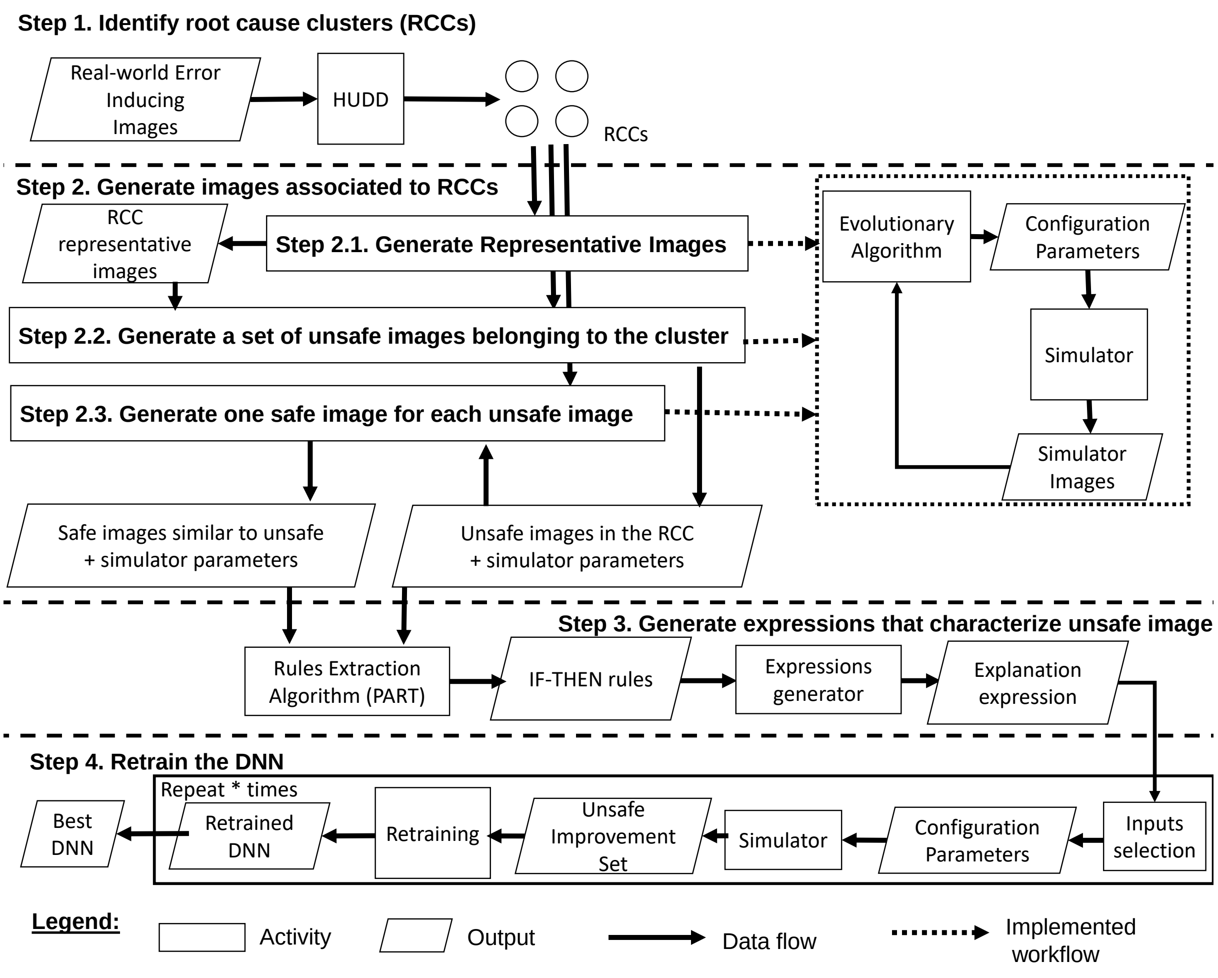

Step 1. Identify root cause clusters (RCCs)
Real-world Error Inducing Images
HUDD
RCCs
Step 2. Generate images associated to RCCs
RCC representative images
Step 2.1. Generate Representative Images
Step 2.2. Generate a set of unsafe images belonging to the cluster
Step 2.3. Generate one safe image for each unsafe image
Evolutionary Algorithm
Configuration Parameters
Simulator
Simulator Images
Safe images similar to unsafe + simulator parameters
Unsafe images in the RCC + simulator parameters
Step 3. Generate expressions that characterize unsafe image
Rules Extraction Algorithm (PART)
IF-THEN rules
Expressions generator
Explanation expression
Step 4. Retrain the DNN
Repeat * times
Best DNN
Retrained DNN
Retraining
Unsafe Improvement Set
Simulator
Configuration Parameters
Inputs selection
Legend:
Activity
Output
Data flow
Implemented workflow

Figure 4.2: The SEDE workflow

Addressing multiple hazard-triggering events entails multiple search runs, as justified in Section 4.3.1. More precisely, for each RCC, our algorithm executes one search to generate a set of diverse images belonging to that RCC, one search to generate a set of unsafe (i.e., failure-inducing) images that are close to each of the diverse images and belonging to the RCC, and another search to identify a set of safe images (i.e., not failure-inducing) that are close to the unsafe ones. For classifier DNNs, a failure is triggered when the output class differs from the ground truth, where the latter is automatically derived from the parameters of the simulator. In the case of regression DNNs, a DNN failure is triggered when the difference between the DNN output and the reference value (ground truth) is above a set threshold, which is defined by engineers based on domain knowledge.

In the *third step*, SEDE relies on the availability of a sufficiently large number of diverse safe and unsafe images, generated in the previous step, to identify those parameter ranges that characterize unsafe images. More precisely, SEDE relies on a machine-learning algorithm (i.e., PART) to extract such ranges, under the form of logical expressions that accurately predict and therefore represent unsafe images.

In the *fourth step*, SEDE relies on the derived expressions to generate an additional set of images that are used to retrain and improve the DNN.

Please note that SEDE can also be applied to test sets generated using simulators; however, compared to test sets with real-world images, simulator-based failure-inducing test sets are less challenging to characterize. Indeed, when test set images are generated using simulators, decision trees or any other rule learning algorithm can be used to identify the commonalities among the failure-inducing images [79, 35]. For example, it is sufficient to generate RCCs with HUDD and then extract PART rules from them. However, if the number of available failure-inducing images is limited, learning may not lead to accurate results. In such cases, SEDE may be useful to generate additional failure-inducing images, thus enabling the generation of more accurate explanatory rules.

Below, we describe Steps 2 to 4 since Step 1 simply involves the execution of HUDD to generate RCCs.

4.3.1 Step 2. Automated Generation of Unsafe and Safe Images

In the second step of SEDE, for each RCC, we aim to automatically generate images that (goal 1) exhibit the same hazard-triggering events observed in the RCC and (goal 2) enable a learning algorithm to extract accurate rules predicting unsafe images. These rules shall ideally characterize parameter values that are observed only with unsafe images belonging to a specific RCC.

To achieve goal 1, it is sufficient to explore the input space by generating diverse images that are similar to the ones belonging to the RCC.

To achieve goal 2, instead, we need to generate both safe images and unsafe images. To be accurate, the rules derived by SEDE shall provide ranges for parameter values that are observed only or mostly with unsafe images and are as wide as possible.

To this end, the unsafe images used to generate the rules must be as diverse as possible and for each unsafe image, we should identify a very similar safe image so that the algorithm learns to precisely distinguish unsafe images from safe ones, i.e., precisely learn the boundary between them.

To achieve the above, we rely on a divide-and-conquer approach that consists in executing three genetic algorithms for each RCC in a sequence. We make use of genetic algorithms because they have been successfully used by related work to explore the input space of DNNs using simulators [81].

First (Step 2.1), we generate diverse images belonging to the RCC, safe or unsafe. The goal is to cover the cluster with representative images.

Second (Step 2.2), we rely on the generated representative images to generate an additional set of unsafe and diverse images belonging to the cluster; this is achieved by generating, for each representative image, one unsafe image that is close to it and belongs to the RCC.

Third (Step 2.3), for each unsafe image derived by the second evolutionary search, we generate one safe image that is as close to it as possible.

In summary, we aim to generate a sufficient number of diverse safe and unsafe images belonging to each RCC to enable effective learning.

Step 2.1 - Generate Representative Images

We have two objectives: (1) generate a set of images that belong to a cluster and (2) minimize their similarity (i.e., maximize their diversity); however, since RCCs tend to contain images that are similar, the two objectives are unlikely to compete. Indeed, two dissimilar images are unlikely to belong to the same cluster. Further, for our purpose, it is useless to generate a set of diverse images if they do not belong to the RCC. Consequently, it is not desirable to rely on a multi-objective search algorithm to address them. To drive the search process, we thus need a fitness function that enables an evolutionary search algorithm to first generate images that belong to a cluster and then decrease their similarity.

In Section 4.1, we have clarified why state-of-the-art approaches are inapplicable or ineffective when applied to achieve our objectives; in this section, we propose a dedicated fitness function and algorithm.

In the presence of a function that measures the distance of an individual i from the center of the RCC C (hereafter, $RCC_{distance}(C, i)$) and a function that measures how an individual contributes to minimizing the similarity across cluster members (hereafter, $F_{similarity(i)}$), to obtain a mathematically adequate behavior, our fitness function $F_1^C(i)$ can be defined as follows:

$$F_1^C(i) = \begin{cases} F_{similarity}(i) & \text{if } RCC_{distance}(C, i) \leq 1 \\ RCC_{distance}(C, i) & \text{otherwise} \end{cases} \tag{4.1}$$

Our goal is to minimize $F_1^C(i)$ and given that (1) $F_{similarity(i)} \leq 1$ (see Equation 4.7 below) and (2) $RCC_{distance} \leq 1$ is only true when the image belongs to the RCC, we obtain the targeted property: fitness is always lower in cases where an individual i belongs to the RCC. Below, we describe the functions $F_{similarity}$ and $RCC_{distance}(C, i)$.

To determine if an individual belongs to a RCC, we can measure its distance from the RCC centroid and verify if it is shorter than the RCC radius (i.e., the max distance between the centroid and the farthest image). For the distance metric, as in HUDD, we can use the Euclidean distance between two heatmaps (hereafter, $HeatmapDistance$); however, to generate a heatmap we need a concrete image and, therefore, instead of relying on the cluster centroid, we identify its medoid. The formula for the $HeatmapDistance$ function is:

$$HeatmapDistance(A, B) = \sqrt{\sum_{m=1}^{M} \sum_{n=1}^{N} (A_{m,n}^L - B_{m,n}^L)^2} \tag{4.2}$$

where A and B are two heatmap matrices generated with LRP, and m, n indicate the row and column of a cell in a matrix (M and N indicate the total number of rows and columns, respectively). Since LRP generates one heatmap matrix for every neuron layer of the DNN, we rely on the heatmap generated for layer L as selected by HUDD to generate the RCC.

Similar to related work [34, 58], within our evolutionary algorithms, we represent an individual using a vector of simulator parameter values. For this reason, to compute the heatmap distance, our search algorithm, for every offspring individual, first generates one image using the simulator, then processes the generated image using the DNN under test, and finally generates its heatmap using the LRP algorithm.

The medoid of a RCC C is the image that minimizes the average pairwise distance from the other images of the RCC; it is computed as follows:

$$Medoid(C) = \underset{y \in C}{\arg\min} \left\{ \frac{\sum_{x \in C, x \neq y} HeatmapDistance(x, y)}{|C| \cdot (|C| - 1)} \right\} \tag{4.3}$$

The radius of a RCC C is the maximum distance between its medoid and any other image in C:

$$Radius(C) = \underset{y \in C}{\arg\max} \left\{ HeatmapDistance(y, Medoid(C)) \right\} \tag{4.4}$$

Since clusters may have different radiuses, we compute $RCC_{distance}(C, i)$ for a RCC C as the heatmap distance between individual i and C's medoid $Medoid(C)$, normalized by C's radius $Radius(C)$:

$$RCC_{distance}(C, i) = \frac{HeatmapDistance(i, Medoid(C))}{Radius(C)} \tag{4.5}$$

```
Require: $RCC_C$, the RCC under analysis
Require: s, population size
Require: r, number of random populations of images to generate initially
Require: b, search budget (i.e., number of iterations to perform)
Ensure: $P_1^C$, an optimal set of diverse images belonging to cluster $RCC_C$
 1: repeat
 2:     $R \leftarrow R \cup \{RANDOM\ POPULATION\ with\ size\ s\}$
 3: until $r$ random populations have been generated
 4: $t \leftarrow 0$
 5: $P \leftarrow$ selects the population in $R$ with the best fitness
 6: while $t < b$ do
 7:     EVALUATE the FITNESS of $P$ based on $RCC_i$
 8:     $O \leftarrow$ GENERATE the OFFSPRING population from $P$
 9:     EVALUATE the FITNESS of $O$ based on $P$ and $RCC_C$
10:     while $SIZE(O) > 0$ do
11:         SORT $O$ based on decreasing FITNESS
12:         $i_o \leftarrow$ extract and remove first individual in $O$
13:         if there is at least one individual in $P$ that is outside the RCC then
14:             $i_p \leftarrow$ IDENTIFY the individual in $P$ with the highest fitness
15:         else
16:             $i_p \leftarrow$ IDENTIFY the individual in $P$ that is closer to $i_o$
17:         if fitness of $i_o <$ fitness of $i_p$ then
18:             Within $P$, replace $i_p$ with $i_o$
19:             EVALUATE the FITNESS of $P$ based on $RCC_C$
20:             EVALUATE the FITNESS of $O$ based on $P$ and $RCC_C$
21:     $t \leftarrow t + 1$
22: $P_1^j \leftarrow$ all the individuals in $P$ that belong to $RCC_C$
23: return $P_1^C$
24: exit
```

Figure 4.3: PaiR, the Genetic Algorithm integrated into SEDE

To compute $F_{similarity}(i)$, we follow related work [81] and measure how much an individual contributes to the diversity of the population by measuring its distance from the closest individual in the population. We can measure the distance between two individuals i and j based on their chromosome vectors v_i and v_j (containing simulator parameter values), as follows:

$$\begin{aligned} ChromosomeDistance(i,j) &= \cos(i,j) \\ &= \frac{v_i \cdot v_j}{\|v_i\|\|v_j\|} \\ &= \frac{\sum_{p=1}^{|i,j|}(v_i[p] \cdot v_j[p])}{\sqrt{\sum_{p=1}^{|v_i|} v_i[p]^2}\sqrt{\sum_{p=1}^{|v_j|} v_j[p]^2}} \end{aligned} \tag{4.6}$$

where $\cos(i,j)$ is the cosine similarity between v_i and v_j, whose range is [0, 1]; $v_i[p]$ and $v_i[p]$ indicate the value of the p-th component in the vectors v_i and v_j, respectively.

F_1^C can be used to drive the search algorithm, described below.

The SEDE search algorithm: PaiR *Pairwise Replacement* (PaiR) is our algorithm to generate representative RCC images; it is presented in Fig. 4.3 and described below.

PaiR aims to evolve a whole population of individuals to generate a population of individuals that both belong to the RCC under analysis and are diverse. The size of the population is a parameter of the algorithm. PaiR leverages all the images generated by the simulator at every search iteration by replacing one or more images in the parent population with offspring individuals having a better fitness.

PaiR works by looking for individuals that minimize F_1^C. At every iteration, PaiR creates a new population P' obtained by replacing an individual i_p with an individual i_o having a better fitness (i.e., $F_1^C(i_o) < F_1^C(i_p)$), as described in the following.

The PaiR algorithm receives as input the RCC under analysis (hereafter, RCC_C), the configuration parameters s (population size) and r (number of random populations to generate initially), and the search budget b indicating the number of iterations to perform.

To maximize the chances of finding an optimal solution, as in related work [114], the PaiR algorithm starts by generating several random populations (Line 1 to 3) and selecting the population including the individual with the lowest fitness value (P, Line 5).

In its main loop (Lines 6 to 21), PaiR computes the fitness values of individuals in the parent population P and, from P, generates an offspring population O by relying on the same strategy commonly adopted with NSGA-II in similar contexts (i.e., tournament selection, simulated binary crossover, crossover probability p_c, and mutation probability p_m).

The offspring individuals are sorted to process first the ones with better fitness (Line 11). PaiR considers each individual in O (i_o) as a replacement of an individual in P (i_p), if the former has a better fitness. PaiR distinguishes between the cases in which P only contains individuals belonging to the RCC (Line 15) and when at least one individual does not (Line 13). Indeed, if we do not have enough individuals belonging to the RCC, it is not important to maximize their diversity.

If P contains at least one individual not belonging to the RCC, PaiR simply looks for the individual in P with the highest fitness (i.e., the one that is furthest from the cluster medoid). Such individual will be replaced by i_o if the latter has a lower fitness (Line 17), which is always the case if i_o belongs to the RCC; otherwise, it depends on the value of $RCC_{distance}$: i_o replaces i_p if i_o is closer to the RCC medoid.

If all the individuals in P belong to the RCC, PaiR selects the individual in P that is closer to i_o (i.e., i_p, Line 16). The offspring individual i_o replaces i_p if i_o has a lower fitness than i_p (Line 17). Since i_p belongs to the RCC, according to Equation 4.1, i_o has a lower fitness than i_p when i_o belongs to the RCC and has a lower $F_{similarity}$.

$F_{similarity}(i)$ is computed differently for individuals in the parent and in the offspring population; the reason is that we must decide if an individual i_o in the offspring population O, would lead to a population P' with a larger diversity than P. P' is obtained by replacing an individual i_p in the parent population P with i_o; therefore, we obtain a different P' from a same P, depending on the selected individuals i_p and i_o. To help identify the most diverse P' and avoid computing the similarity for all possible P' obtained from every individual in O, inspired by related work [81], we compare (a) the distance between i_p and its closest neighbour in P (i.e., the closest individual in P excluding i_p itself) with (b) the distance between i_o and the closest individual in P'. Note that P' includes i_o and $P \setminus \{i_p\}$. Therefore, for an individual i_p in the parent population P, we compute one minus its distance from $closest_{i_p}$, the closest neighbour to i_p

$$F_{similarity(i_p)} = 1 - ChromosomeDistance(i_p, closest_{i_p}) \tag{4.7}$$

For an individual i_o in the offspring population O, we compute one minus its distance from $closest_{i_o}$, the individual in $P \setminus \{i_p\}$ that is closest to i_o:

$$F_{similarity(i_o)} = 1 - ChromosomeDistance(i_o, closest_{i_o}) \tag{4.8}$$

Based on the above, given two individuals i_o and i_p in the offspring and parent populations, respectively, the individual i_o has a better fitness than i_p if they both belong to the RCC and the similarity between i_o and the closest individual in $P \setminus \{i_p\}$ is lower than the similarity between i_p and its closest individual in P. Since the distance from the closest individual should increase (lower similarity) when replacing i_p with i_o in the population, we can assume that a population P' will have higher diversity than P, which we demonstrate in Section 4.4.2.

The process is repeated until O is empty (Lines 10 to 20); since F_1^C depends on the distance between individuals being close to each other, given that both O and P vary over iterations, the fitness of the two populations is recomputed after every replacement (Lines 19 and 20).

After b iterations, the algorithm has generated a population P_1^C of individuals that are diverse and very likely to belong to RCC_C.

Step 2.2 - Generate a set of unsafe images belonging to the cluster

For each RCC_C, we aim to generate a set of diverse, unsafe images, belonging to the cluster. To preserve the diversity of the images generated by PaiR in Step 2.1 (P_1^C), we can identify, for each image in P_1^C, an image that is close to it and makes the DNN fail. We can thus model our problem as a many-objective optimization problem with q objectives, where q is the number of images in P_1^C.

We define q fitness functions, one for each individual in P_1^C. An individual is an image generated with the simulator, modelled as described in Section 4.3.1. All fitness functions share the same parameterized formula $F_2^{C,t}$, where $t \in \{1, ..., q\}$:

$$F_2^{C,t}(i) = \begin{cases} ChromosomeDistance(i, P_1^C[t]) & if\ (RCC_{distance}(RCC_C, i) \leq 1)\ \wedge \\ & \quad (CLOSEST(i, P_1^C) = P_1^C[t]) \wedge (DNN_{failure}(i)) \\ 1 + F_{uncertainty}(i) & if\ (RCC_{distance}(RCC_C, i) \leq 1)\ \wedge \\ & \quad (CLOSEST(i, P_1^C) = P_1^C[t]) \wedge (DNN_{correct}(i)) \\ 2 + ChromosomeDistance(i, P_1^C[t]) & if\ (RCC_{distance}(RCC_C, i) \leq 1)\ \wedge \\ & \qquad (CLOSEST(i, P_1^C) \neq P_1^C[t]) \\ 3 + RCC_{distance}(RCC_C, i) & if\ RCC_{distance}(RCC_C, i) > 1 \end{cases} \tag{4.9}$$

$F_2^{C,t}$ is a piecewise-defined function implemented through four sub-functions that return a value in the range [0, 1] incremented by a constant (from 0 for the first sub-function to 3 for the fourth sub-function).

The fourth sub-function of the equation drives the search towards generating an individual belonging to RCC_C; indeed, when this is not the case, $F_2^{C,t}$ measures how far the individual is from the RCC medoid with $RCC_{distance}(RCC_C, i)$. Since not belonging to RCC_C is the worst case for an individual, $F_2^{C,t}$ must be higher than in other cases. This is achieved by returning $3 + RCC_{distance}(RCC_C, i)$, given that the codomain of the first three sub-functions referred to in $F_2^{C,t}$ lay in the range [0, 3].

The third sub-function of the equation ensures that we generate one image for each individual in P_1^C; indeed, if the individual i belongs to RCC_C but the individual in P_1^C that is closest to i is not the t-th individual (i.e., $CLOSEST(i, P_1^C) \neq P_1^C[t]$), $F_2^{C,t}$ returns the cosine distance between i and the t-th individual in P_1^C, incremented by 2.

The second sub-function helps generating an individual that makes the DNN fail. Indeed, if the individual i belongs to RCC_C, is the closest to the t-th individual targeted by $F_2^{C,t}$, but does not make the DNN fail, then the fitness function returns a measure of DNN uncertainty increased by one.

```
Require: I, vector whose components represent an individual
1: $l = |I|$
2: for $i = 0; i < |I|$ do
3:     $I[i] = 0$
4: for each objective $o$ do
5:     $I \leftarrow$ sort $I$ according to objective $o$
6:     $I[1]_{distance} \leftarrow \infty$
7:     $I[l]_{distance} \leftarrow \infty$
8:     for $i = 2$ to $(l - 1)$ do
9:         $I[i]_{distance} \leftarrow I[i]_{distance} + \frac{(I[i+1]_m - I[i-1]_m)}{(max\ fitness\ for\ o) - (min\ fitness\ for\ o)}$
```

Figure 4.4: Crowding-distance-assignment function for NSGA-II. Red part indicates the instruction commented out in *NSGA-II′* to ensure that, for each objective, the best individual is preserved.

To compute the uncertainty component of the fitness, we return one minus the cross-entropy loss function, which is commonly used to measure uncertainty:

$$F_{uncertainty}(i) = 1 - entropy(i) \tag{4.10}$$

Cross-entropy measures how uncertain is the DNN about the provided output where lower cross-entropy values indicate higher certainty; therefore, by using one minus cross-entropy loss, we direct the search (minimization) towards the identification of images for which the DNN is less certain about outputs and, therefore, likely to produce an erroneous output.

The first sub-function of $F_2^{C,t}$, instead, for individuals that make the DNN fail, helps select the individual that is closest to the individual targeted by $F_2^{C,t}$; indeed, it returns the chromosome distance between the individual i and the individual $P_1^C[t]$.

Search Algorithm. To address our problem, we rely on a modified version of NSGA-II. NSGA-II is known to underperform in the presence of a large number of objectives (>3), mainly because of the exponential growth in the number of non-dominated solutions required for approximating the Pareto front [115]. However, we do not aim to find, as a solution to our problem, one single individual (i.e., image) that finds the best balance among different objectives but a set of images, each optimizing one independent objective (indeed, each image shall be close to a reference image in P_1^C); therefore, we are unlikely to find a set of nondominated solutions larger than $|P_1^C|$ (i.e., we will find one solution for each objective). Also, by definition, $F_2^{C,t}$ is always different than zero, which renders ineffective algorithms that look for an individual that *covers* an objective like MOSA. A solution to enable the adoption of MOSA would be to set a threshold to determine when the objective has been *covered*; however, since RCCs differ with respect to their radius, we choose not to introduce a threshold that may lead to varying performance results across RCCs. Since we aim to optimize all the objectives, we rely on an extended version of NSGA-II with a modified crowding-distance function that ensures we select the minimal value for each objective, thus resembling the preference criterion of MOSA. Finally, we do not require an archive because the population will always include the best individual found for each objective and we do not need to evaluate an individual according to objectives not explicitly modeled (e.g., inputs' length in MOSA).

We propose a modified version of NSGA-II (hereafter, *NSGA-II′*) that includes a modified crowding-distance-assignment which ensures preserving, for each objective, the individual with the lowest fitness value.

Our modified crowding-distance-assignment is shown in Fig. 4.4; different from the original crowding-distance-assignment used by NSGA-II, for each objective, we assign infinite crowding distance only to the individual that minimizes the fitness for the objective. NSGA-II, instead, assigns infinite distance also to the individual that maximizes the fitness for the objective (Line 7 in Fig. 4.4). Fig. 4.4 provides the pseudocode for the function that computes the crowding-distance for a set of individuals. Lines 6 and 7 prioritize the individuals with the best and worst fitness, respectively, by assigning them infinite distance. As in NSGA-II, if more than one individual has the same minimal fitness, *NSGA-II′* assigns infinite distance only to one, randomly selected individual. Our choice ensures that, when the Pareto front includes a number of individuals larger than the population size s, for each objective, we preserve the individual that minimizes the fitness value. When the Pareto front has a number of individuals lower than the population size s, our crowding-distance assignment ensures the selection of individuals that minimize the fitness and are likely unsafe, which is what we intend to preserve in the final population.

However, this situation is unlikely. Indeed, it may occur only when one individual has the same fitness value for several (i.e., z, with $z \leq q$) objectives, in one of the following unlikely scenarios:

- one image has the lowest $RCC_{distance}$ and no other individual falls in the cases covered by the first three sub-functions in $F_2^{C,t}$, for the same z objectives;
- one image has the lowest $\mathit{ChromosomeDistance}(i, P_1^C[t])$ for z reference images and no other individual falls in the cases covered by the first two sub-functions in $F_2^{C,t}$, for the same z objectives;
- one image has the lowest $F_{uncertainty}(i)$ and no other individual falls in the cases covered by the first sub-function in $F_2^{C,t}$, for the same z objectives;
- one image is failure-inducing and has the lowest $\mathit{ChromosomeDistance}(i, P_1^C[t])$, for the same z objectives.

We apply *NSGA-II′* for k iterations. Please note that when P_1^C already includes unsafe images belonging to the RCC C, *NSGA-II′* simply retains such images at every iteration; indeed, their fitness is already optimal according to $F_2^{C,t}$. After the k-th iteration of *NSGA-II′*, SEDE generates a population of individuals that likely lead to a DNN failure, which we refer to as P_2^C; at the end of the search, images not leading to DNN failures are removed from P_2^C.

Step 2.3 - Generate one safe image for each unsafe image

We aim to generate a set of images that are similar to the unsafe images in P_2^C but do not lead to a DNN failure. As for the previous case, we model our problem as a many-objective optimization problem with q objectives, where q is the number of images in P_2^C; for each image in P_2^C, we aim to generate an image that is close to it and makes the DNN pass. We rely on the same *NSGA-II′* configuration used for Step 2.2 except for the fitness function, which in this step needs to be adapted to drive the generation of safe images; also, it is not necessary for these generated images to belong to RCC_C (indeed, we may not have any safe image within RCC_C). In Step 2.3, *NSGA-II′* evolves a population P_3^C that initially matches P_2^C.

As for Step 2.2, we define q fitness functions, one for each image in P_2^C; these functions share the same parameterized formula, for $t \in \{1, ..., q\}$:

$$F_3^{C,t}(i) = \begin{cases} \mathit{ChromosomeDistance}(i, P_2^C[t]) & \text{if } (\mathit{CLOSEST}(i, P_2^C) = P_2^C[t]) \wedge \\ & \quad (DNN_{correct}(i)) \\ 1 + \mathit{entropy}(i) + |1 - RCC_{distance}(RCC_C, i)| & \text{if } (\mathit{CLOSEST}(i, P_2^C) = P_2^C[t]) \wedge \\ & \quad (DNN_{failure}(i)) \\ 2 + \mathit{ChromosomeDistance}(i, P_2^C[t]) & \text{if } \mathit{CLOSEST}(i, P_2^j) \neq P_2^C[t] \end{cases} \tag{4.11}$$

The third sub-formula in $F_3^{C,t}$ matches the third formula in $F_2^{C,t}$ and aims to generate one image close to each each unsafe image in P_2.

The definition of the second sub-formula in $F_3^{C,t}$ is motivated by the fact that, since the initial population is P_2^C, all the images in the population initially cause a DNN failure and, therefore, the fitness should drive the search algorithm to find a close image leading to a correct output. To do so, we should look for images that either increase the confidence of the DNN output (i.e., leading to a lower entropy) or are close to the border of the RCC. Concerning the latter, since a RCC characterizes unsafe images, we expect that images next to its border are similar to the ones in the RCC but are less likely to be failure-inducing. In addition, moving further away from the border is expected to lead to images that are increasingly different from the images in the RCC. Such observations are captured by the absolute difference between 1 and $RCC_{distance}$ (i.e., how close the image is to the RCC border), which is added to $\mathit{entropy}(i)$ to help generate safe images.

The codomains of the second and the third sub-formula overlap as the former may return a fitness value above 3; this choice reflects the fact that an image being far away from the RCC border (when $|1 - RCC_{distance}(C, i)| > 1$) is unlikely to provide useful information as it is not helpful to derive rules that characterize safe images that are not similar to the ones in the RCC. Therefore, such image would be as useless as an image that is not close to the target image (i.e., $P_2^C[t]$). Also, if the DNN is highly uncertain about an image i (i.e., $\mathit{entropy}(i)$ is close to 1), it is unlikely for that image to help driving the algorithm towards a correct output. For the reasons above, the output of the second sub-formula falls into the codomain of the third sub-formula when the sum $\mathit{entropy}(i) + |1 - RCC_{distance}(RCC_C, i)| > 2$.

The first sub-formula in $F_3^{C,t}$, in the presence of images leading to correct DNN outputs, aims to minimize the distance from the t-th image; therefore, it returns the chromosome distance between the individual i and the target image $P_2^C[t]$.

The algorithm *NSGA-II′* terminates after k iterations with a population P_3^C including images that are likely safe and close to the images in P_2^C; though unlikely, images leading to DNN failures are removed from P_3^C before SEDE's Step 3.

4.3.2 Step 3. Characterize Unsafe Images

In Step 3, we aim to generate an expression that characterizes unsafe images by using PART, to learn decision rules. For example, we could learn that images likely lead to a DNN failure when the parameters $HeadPose_X > 10 \; and \; HeadPose_Y > 50$.

PART generates as output a list of mutually exclusive rules (see Section 2.1.2) that predict the class of a data point; in our context, these rules predict the DNN output generated for an image (i.e., wrong or correct output) based on the values of the simulator parameters used to generate the image.

```
1: $HeadPose_Y > 50.34$ : $class = DNN - error$
2: $HeadPose_Y < 13.34$ : $class = DNN - correct$
3: $HeadPose_Z > 60$ & $HeadPose_Y > 30$ : $class = DNN - error$
4: $HeadPose_Z \leq 60$ : $class = DNN - correct$
5: (default) : $class = DNN - error$
```

Figure 4.5: Example PART output

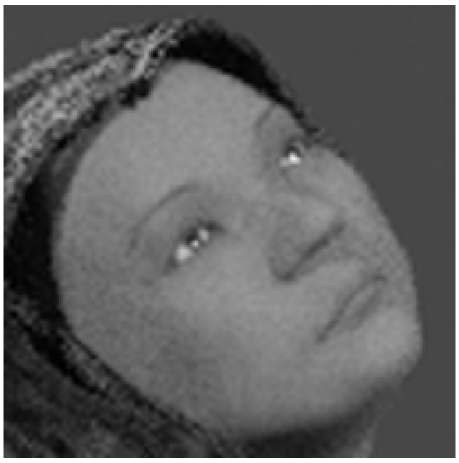


Figure 4.6: Example image generated with IEE-Humans, one of the simulators used in our empirical evaluation. It is labeled as looking top-right (i.e., towards the top-right corner of the picture).

Since, in our context, for each image under analysis we know (1) the DNN outcome (i.e., a DNN failure or a correct output) and (2) the set of simulator parameter values used to generate it, we can configure PART to consider the DNN outcome as the class to predict and the simulator parameter values as the features to derive rules from.

Fig. 4.5 provides an example output generated by PART when processing images from our case study subjects. Each image depicts a person's head and is represented by a number of simulator parameters configured to generate the image (an example is shown in Fig. 4.6). The considered parameters include, among others, the orientation of the head (pitch, yaw, and roll captured by the parameters $HeadPose_X$, $HeadPose_Y$, and $HeadPose_Z$).

In Fig. 4.5, Line 1 indicates that we observe a DNN failure if the value of the parameter $HeadPose_Y$ is above 50.34 (i.e., the person is looking to the right of the viewer, with the head turned so much that her left eye is barely visible). Line 2 indicates that, when the rule in Line 1 does not apply and the person is looking center or to her right (i.e., if $HeadPose_Y$ is below 13.34), then the DNN produces a correct output. Line 3 indicates that, when the two rules above do not hold, we observe a DNN failure if the value of parameter $HeadPose_Z$ is above 60 and the value of parameter $HeadPose_Y$ is above 30 (i.e., when the head is tilted and the person is looking to her left then the DNN makes a mistake even if her left eye is visible). Line 4 indicates that, when the three rules above do not hold, the DNN output is correct if the value of parameter $HeadPose_Z$ is below 60 (i.e., the head is not tilted). Finally, Line 5 is the default rule, which indicates that, when all the rules above do not hold, the DNN output is incorrect.

Since in Step 2, for each RCC, SEDE generated a set of unsafe (P_2^C) and safe (P_3^C) images, associated with simulator parameters, to learn decision rules with PART, SEDE selects from P_2^C all the images that belong to the RCC and are failing, and all the images from P_3^C that are not failing.

To generate an expression from the PART output (hereafter, referred to as *PART expression*), we implemented a procedure that, for all the rules leading to a DNN failure, generates a subexpression that joins the negation of all the preceding rules with the current rule. The generated subexpressions are mutually exclusive. For the example in Fig. 4.5, it leads to the following expression:

$$\begin{aligned}
&(HeadPose_Y > 50.34) \parallel \\
&(\neg(HeadPose_Y > 50.34)\& \neg(HeadPose_Y < 13.34) \\
&\quad \&(HeadPose_Z > 60 \;\&\; HeadPose_Y > 30)) \parallel \\
&(\neg(HeadPose_Y > 50.34) \;\& \neg(HeadPose_Y < 13.34) \\
&\quad \& \neg(HeadPose_Z > 60 \;\&\; HeadPose_Y > 30) \\
&\quad \& \neg(HeadPose_Z <= 60))
\end{aligned} \tag{4.12}$$

Indeed, for Line 1 in Fig. 4.5, SEDE simply reports the expression generated by PART (there are no preceding expressions), which leads to "$HeadPose_Y > 50.34$". For Line 2, SEDE does not generate any subexpression because it concerns cases in which the DNN generates a correct output; the same happens for Line 4. For Line 3, SEDE negates the preceding expressions (i.e., "$HeadPose_Y > 50.34$" and "$HeadPose_Y < 13.34$") and joins them with the PART expression of Line 3 (i.e., "$HeadPose_Z > 60 \;\&\; HeadPose_Y > 30$"). For Line 5, SEDE simply generates a subexpression joining the negation of all the preceding expressions.

Unfortunately, PART does not generate expressions for parameters whose range is shared by the two classes under consideration (i.e., DNN failure and DNN correct). This is an issue as SEDE generates safe images that are similar to the unsafe ones. For example, the PART output in Fig. 4.5 does not include an expression with "$HeadPose_X > 10$", which indicates that the head is looking top because this characteristic is observed in both safe and unsafe images. Such commonalities, however, might be important to characterize unsafe inputs; indeed, in our example, if the RCC includes images with a person looking top, safe images close to the generated unsafe images will likely be looking top. To address this issue, SEDE joins the PART expression with an expression that constrains the parameters not appearing in the PART expression. The additional expression is generated by identifying the min and max values assigned to the parameters for all the unsafe images. For the example above, it joins the expression "$HeadPose_X > 10$" with the PART expression in Equation 4.12, thus leading to:

$$\begin{aligned}
&HeadPose_X > 10 \;\& \\
&\quad ((HeadPose_Y > 50.34) \parallel \\
&\qquad (\neg(HeadPose_Y > 50.34)\&\neg(HeadPose_Y < 13.34) \\
&\qquad\quad \&(HeadPose_Z > 60 \;\&\; HeadPose_Y > 30)) \parallel \\
&\qquad (\neg(HeadPose_Y > 50.34)\&\neg(HeadPose_Y < 13.34) \\
&\qquad\quad \&\neg(HeadPose_Z > 60 \;\&\; HeadPose_Y > 30) \\
&\qquad\quad \&\neg(HeadPose_Z \leq 60)))
\end{aligned} \tag{4.13}$$

4.3.3 Step 4. Generate Unsafe Images for Retraining

In Step 4, for each RCC, SEDE automatically generates a set U of unsafe images by relying on the simulator. Thanks to the simulator, these images can indeed be automatically labeled and, consequently, can be used to retrain the DNN. We refer to this set of images as the *unsafe improvement set*. More precisely, we aim to train a new DNN model by relying on an extended training set that consists of part of the training set used to train the DNN under analysis and the unsafe improvement set. The retraining process is configured as a fine-tuning process where the weights of the DNN to be trained are initially set to be equal to the weights of the DNN under analysis. Our rationale is that by retraining the DNN with an additional set of images belonging to the unsafe portion of the input space, we enable the DNN to better learn how to provide correct outputs for such inputs.

To avoid losing information derived from the original training process, the extended training set consists of $S\%$ of the simulator images and $R\%$ of the real-world images belonging to the original training set. We suggest configuring S and R such that the number of simulator images is not overwhelmingly larger than the number of real-world images (e.g., twice the number of real-world images, at most). More precisely, we should limit the number of simulator images because they may overly influence the DNN with characteristics that are not present in real-world images (e.g., faces with regular shapes); nevertheless, we suggest keeping a representative set of simulator-based images (e.g., 5%) because they may include a number of scenarios (in our case studies, head positions) not covered by real-world images. For real-world images, since they are generally low in number, we suggest keeping most or all of them to prevent the risk of losing any image that captures an under-represented scenario. For our experiments, we set S=5% and R=100% (see Section 4.4.2). We leave to future work the identification of a solution for the automated selection of the best configuration for S and R (e.g., by selecting the DNN with the best accuracy among the ones generated by repeating the retraining process with different values for S and R). For example, recent results suggest that the proportion of simulator ($S\%$) and real-world ($R\%$) images may also depend on the degree of fidelity of the simulator: the more realistic simulator images are, the less real-world images are required [116].

The accuracy of the DNN training process may depend on a wide range of factors, including the specific simulator images selected from the original training set, the images belonging to the unsafe improvement set, and other random factors (e.g., the order of images in the training batches). For this reason, SEDE repeats the retraining process (i.e., generate unsafe images in U for each RCC and retrain the DNN) multiple times and keeps the DNN that provides the best output. Though repeating the training process obviously increases the training cost (e.g., buying additional GPU time in Cloud systems), for safety-critical systems, such additional cost may be justified by improvements in accuracy.

4.4 Evaluation

Our empirical evaluation addresses the following research questions:

RQ1 *How does PaiR fare, compared to NSGA-II and DeepNSGA-II, for the generation of diverse images belonging to RCCs?*

The generation of a sufficiently large and diverse set of images belonging to all the RCCs under analysis is essential to enable the characterization of DNN failures. To that end, SEDE integrates a dedicated genetic algorithm (PaiR) whose performance is evaluated by this RQ and compared to NSGA-II and DeepNSGA-II in terms of percentage of RCCs for which an individual has been generated, percentage of individuals belonging to each cluster, and image diversity within clusters.

RQ2 *Does SEDE generate simulator images that are close to the center of each RCC?*

The generation of images belonging to a RCC is a necessary step to generate expressions that characterize the RCC. Since there is no guarantee of generating, using simulators, images that are similar to real-world images, this research question evaluates if the images generated by SEDE are closer to the center (medoid) of a cluster than random simulated, unsafe images.

RQ3 *Does SEDE generate, for each RCC, a set of images sharing similar characteristics?*

To successfully characterize RCCs with PART, it is necessary, for each RCC, to rely on a set of images presenting similar characteristics while preserving diversity regarding other aspects. In other words, the generated images shall present similar values for a subset of the simulator parameters while being diverse regarding the other parameters.

RQ4 *Do the RCC expressions identified by SEDE delimit an unsafe space?*

This research question aims to determine if the analyzed DNN underperforms when processing images matching RCC expressions. In other words, is the DNN accuracy significantly lower for such images, as expected, than for random images from the whole input space?

RQ5 *How does SEDE compare to state-of-the-art DNN accuracy improvement practices?*

This research question evaluates the effectiveness of SEDE (Step 4), in terms of accuracy improvements, compared to HUDD and a baseline consisting of randomly generated images for each RCC (namely, Random BaseLine (RBL)).

4.4.1 Subject Systems

We consider DNNs that implement head pose detection and face landmarks detection. They are building blocks for in-car monitoring systems (e.g., driver's drowsiness detection) under study at IEE Sensing, our industry partner.

The head pose detection DNN (HPD) receives as input the cropped image of the head of a person and determines its pose according to nine classes (straight, turned bottom-left, turned left, turned top-left, turned bottom-right, turned right, turned top-right, reclined, looking up).

The facial landmarks detection DNN (FLD) determines the location of the pixels corresponding to 27 face landmarks delimiting seven face elements: nose ridge, left eye, right eye, left brow, right brow, nose, and mouth. Several face landmarks match each face element (e.g., there are four landmarks to outline a mouth). The accuracy of this regression DNN is computed as the percentage of images with landmarks being accurately predicted. For each landmark, we compute the prediction error as the Euclidean distance between the predicted and correct landmark pixel on the input image; an image is considered accurately predicted if the average error is below 4 pixels, as suggested by IEE engineers.

Our DNNs have been trained with simulator images, fine-tuned with real-world images, and tested with real-world images, following the process in Fig. 4.1. For our experiments we relied on two different simulators developed by IEE and based on the 3D simulation of MakeHuman [102] and MBLab [117] using the rendering engine Blender [100]. The first simulator is called IEE-Faces, it relies on seven preset MakeHuman models of human faces developed by IEE; it provides 13 parameters that enable controlling different characteristics of the image such as illumination angle and head orientation. Table 4.1 provides a description of all the parameters of IEE-Faces, where the first three rows capture nine parameters, three for each row. IEE-Faces generates one image in 20 seconds, on the hardware used for our experiments. The second simulator is called IEE-Humans and it generates images that are more realistic (i.e., have more details) than the ones generated by IEE-Faces. IEE-Humans provides 23 different configuration parameters, which are described in Table 4.2 where the first two rows capture six parameters, three for each row. It takes 100 seconds to generate an image.

Table 4.1: IEE-Faces simulator parameters

Parameter	**Description**
Camera Direction	Direction of scenario camera (X, Y, Z)
Camera Location	Location of scenario camera (X, Y, Z)
Lamp Location	Location of scenario lamp source (X, Y, Z)
$HeadPose_X$	Vertical position of the head (degrees)
$HeadPose_Y$	Horizontal position of the head (degrees)
$HeadPose_Z$	Tilting position of the head (degrees)
Makehuman Model	Preset makehuman face model (9 face models)

Table 4.2: IEE-Humans simulator parameters

Parameter	**Description**
Lamp Location	Location of scenario lamp source (X, Y, Z)
Lamp Direction	Direction of scenario lamp source (X, Y, Z)
Lamp Color	Color of scenario lamp source (R, G, B)
Camera Height	Height of scenario camera (pixels)
$HeadPose_X$	Vertical position of the head (degrees)
$HeadPose_Y$	Horizontal position of the head (degrees)
$HeadPose_Z$	Tilting position of the head (degrees)
Age	Age of the generated humanoid
Gender	Gender of the generated humanoid
Iris Size	Size of the humanoid iris
Pupil Size	Size of the humanoid pupil
Eye Saturation	Level of the humanoid eye saturation
Eye Color	Color of the humanoid eye
Eye Value	Controls the lightness level of iris
Skin Freckles	Amount of procedural freckles added to the skin
Skin Oil	Brightness of subtle oil effect on the skin
Skin Veins	Amount of procedural veins added to the skin

We trained three DNNs in total, two HPD DNNs and one FLD DNN. One HPD DNN (hereafter, HPD-F) has been trained using IEE-Faces and another one (hereafter, HPD-H) using IEE-Humans. The FLD DNN has been trained using IEE-Faces; we could not train FLD with IEE-Humans because it does not provide landmarks for the generated images.

To fine-tune and test the HPD DNNs we relied on the BIWI real-world dataset [118], which contains over $15,000$ pictures of 20 people faces (6 females and 14 males) annotated with head pose angle. People were recorded sitting in front of a Kinect [119], which is a motion sensor add-on for the Xbox 360 gaming console, and were asked to turn their head around trying to span all possible yaw/pitch angles they could perform. For each image, the head pose angle is computed using FaceShift [120].

For our experiments, we automatically labeled each image with a head pose class derived from the provided angles; we considered 1,000 close-up pictures— similar to what can be obtained with in-car, DNN-based sensors—belonging to two persons where one is used for training and the other one for testing. To generate close-up pictures we performed face detection and image cropping with the Dlib framework [121]. We could select only two persons from the BIWI dataset because the pictures belonging to other subjects were taken far from the camera and thus not useful to mimic a realistic situation with an in-car camera shooting a driver. Further, to simulate a real-world scenario where the DNN processes images of people never observed during training, we do not test the DNN using images belonging to the person used for training. To fine-tune and test the FLD DNN, since we require accurately annotated landmarks, we relied on a dataset provided by IEE.

Table 4.3 provides the number of images used to fine-tune the DNNs along with the obtained accuracy (i.e., the percentage of images for which the DNN provides correct outputs).

Table 4.3: Case Study Systems

	Data Source		**Simulator-based Training**			**Fine-tuning**			
DNN	Simulator	Real-world	Training Set Size **(Accuracy)**	Test Set Size **(Accuracy)**	Epochs	Training Set Size **(Accuracy)**	Simulator-based Test Set Size **(Accuracy)**	Real-world Test Set Size **(Accuracy)**	Epochs
FLD	IEE-Faces	IEE	16,000 **(99.92%)**	2,750 **(44.41%)**	10	9,000 **(95.44%)**	2,825 **(43.22%)**	1,000 **(80.06%)**	50
HPD-F	IEE-Faces	BIWI	21,500 **(91.20%)**	2,750 **(85.43%)**	18	21,976 **(95.35%)**	2,200 **(87.10%)**	500 **(51.65%)**	9
HPD-H	IEE-Humans	BIWI	15,400 **(85.38%)**	3,000 **(80.21%)**	25	18,476 **(90.23%)**	2,750 **(85.23%)**	500 **(51.03%)**	28

Column *Data Source* provides the names of both the simulator and the real-world dataset used to train and fine-tune the DNNs. The columns under *Simulator-based Training* and *Fine-tuning* provide details about the size (i.e., number of images) of the training and test sets used in those phases along with the accuracy of the DNN. The data set used for training the fine-tuned DNNs (FLD, HPD-F, HPD-H) consists of simulator images ($3,000$ for FLD, $21,500$ for HPD-F, $18,000$ for HPD-H) and real-world images ($6,000$ for FLD, 476 for HPD-F, 476 for HPD-H); in Table 4.3, we report the size of the whole fine-tuning training set, including both simulator and real-world images.

Columns *Simulator-based Training – Epochs* and *Fine-tuning – Epochs* report the number of epochs considered to train and fine-tune the DNNs, respectively. All the DNNs have been fine-tuned for a number of epochs sufficient to achieve an accuracy above 90% with a training set of simulator and real-world images.

All the DNNs were implemented with PyTorch [98]. HPD follows the AlexNet architecture [101] which is commonly used for image classification tasks, while FLD follows the Hourglass architecture [84], which is optimized for landmarks detection (regression tasks).

In SEDE Step 1, for each case study DNN, we process all the failure-inducing images of the case study with HUDD to generate RCCs. HUDD's execution led to 10 RCCs for HPD-F DNN, 11 RCCs for HPD-H DNN, and 10 RCCs for FLD DNN.

Table 4.4 provides further information about the configuration of SEDE. Since in Steps 2.2 and 2.3 each individual of the initial population matches the reference one (i.e., the individual that should be similar to the one identified in the final population), we use low crossover and mutation probability to mutate only a few chromosomes at a time (i.e., we depart from the initial individual slowly).

Table 4.4: SEDE Configuration

Step	Population Size	# of iterations	Crossover probability	Mutation probability	Avg. Execution Time (hrs.)		
					FLD	HPD-F	HPD-H
2.1	25	100	0.7	0.3	40	40	200
2.2	25	100	0.3	0.3	20	20	100
2.3	25	100	0.3	0.3	20	20	100

4.4.2 Measurements and Results

RQ1. *How does PaiR fare, compared to NSGA-II and DeepNSGA-II, for the generation of diverse images belonging to RCCs?*

Design and measurements.

We aim to demonstrate that PaiR, our dedicated algorithm to derive a diverse image population belonging to a RCC, performs better than NSGA-II and DeepNSGA-II. To this end, we measure the diversity in the populations generated by the three algorithms over time, for all the RCCs under analysis.

We choose NSGA-II as a baseline for comparison since its crowding distance function is designed to preserve and optimize the diversity between individuals by prioritizing individuals being more distant from others, along with individuals at the boundaries of the objective space.

We configured NSGA-II with an objective function (Equation 4.14) that drives the generation of individuals belonging to the RCC through their normalized distance from the cluster's medoid, as for PaiR (Equation 4.5).

$$F^C(i) = RCC_{distance}(C, i) \tag{4.14}$$

Additionally, we compare PaiR with DeepNSGA-II because the latter is a state-of-the-art solution to generate a population of individuals that are diverse. To this end, we extended the implementation provided for DeepJanus [81]. For our experiments, we rely on the following fitness functions:

$$F_1(i) = ChromosomeDistance(i, closest_{i_A}) \tag{4.15}$$

$$F_2^C(i) = RCC_{distance}(C, i) \tag{4.16}$$

F_1, consistent with DeepJanus and DeepMetis, is computed as the distance between individual i and the closest individual in the archive ($closest_{i_A}$). F_2^C measures the distance between an individual i and the medoid of the RCC C under analysis, thus enabling the algorithm to drive the search towards its objective: generating individuals that belong to C. We execute four independent runs of DeepNSGA-II for each RCC under analysis. At every search iteration, individuals are added to the archive if they belong to the RCC under analysis (i.e., $F_2^C(i) \leq 1$); we consider a sparseness threshold of zero (i.e., we add to the archive any individual that differs from the ones already in the archive, which happens when $F_1(i) > 0$). Since, at the end of the search, the archive may contain a number of individuals larger than the one required for the next steps of the algorithm (i.e., 25, in our experiments), consistent with PaiR, we select the 25 individuals with the highest F_1.

For all three compared algorithms (NSGA-II, DeepNSGA-II, and PaiR), to measure the diversity of the population, since every individual is represented by a vector with the simulator parameter values used to generate the image (chromosome), we compute the average of the chromosome distance across pairs of individuals in the population. Individuals that do not belong to any cluster are ignored from the computation of diversity; indeed, such an individual might increase diversity but it would not be useful for SEDE.

We configured PaiR, NSGA-II, and DeepNSGA-II with the same time budget; precisely, we let NSGA-II and DeepNSGA-II execute for the duration required by our algorithm to perform 100 iterations. It amounts to 40 hours for HPD-F and FLD, 200 hours for HPD-H; differences are due to the time taken by the simulators to generate a single image. We executed PaiR, NSGA-II and DeepNSGA-II for all the 31 RCCs identified for our case study DNNs. To account for randomness, we ran the experiment four different times for each RCC. With a total of 31 RCCs, the four experiment runs led to the collection of 124 data points, thus enabling statistical analysis within practical execution time.

To compare the increase in diversity achieved over time by the three algorithms, for each RCC, we recorded the diversity achieved by the three algorithms every hour. Also, we tracked the percentage of individuals belonging to any RCC—a larger number of such individuals is expected to lead to better results in later stages of SEDE— and the percentage of covered clusters (i.e., RCCs with at least one individual belonging to them).

PaiR performs better than NSGA-II and DeepNSGA-II if, over time, the following conditions hold: (1) the diversity achieved by PaiR is significantly higher than that achieved by NSGA-II and DeepNSGA-II, which would indicate that PaiR fits better our purpose (i.e., generating a diverse population of individuals); (2) PaiR generates a larger proportion of individuals belonging to any RCC, which is useful to characterize RCCs since PART rules can be expected to be more accurate if a larger set of data points is considered; (3) PaiR covers a larger number of RCCs (i.e., PaiR generates at least one image belonging to each cluster), thus enabling the characterization of a larger number of RCCs in later SEDE steps.

Experiment Results.

Fig. 4.7 shows the evolution of diversity obtained with PaiR, DeepNSGA-II, and NSGA-II for our case study DNNs. To simplify visual comparisons, we plot the average, minimum, and maximum diversity observed at each timestamp (every hour); data points are diversity values observed across the four executed runs for all RCCs.

In Fig. 4.7, we can observe that, after 13 hours of execution, both PaiR and DeepNSGA-II outperform NSGA-II; also, with the largest test budget (i.e., 200 hours for HPD-H and 40 hours for HPD-F and FLD) PaiR performs either similar to DeepNSGA-II (for HPD-F and HPD-H) or outperforms it (FLD). However, the performance of PaiR and DeepNSGA-II differ, depending on the case study: PaiR has better initial performance (i.e., within 10 hours) than DeepNSGA-II for the case study subjects relying on a low-fidelity simulator (i.e., HPD-F and FLD), while DeepNSGA-II has better initial performance than PaiR for the case study with a high-fidelity simulator (i.e., HPD-H). We believe that such result can be explained by the fact that, since the low-fidelity simulator provides less configuration parameters, it is more difficult to generate diverse images than with a high-fidelity simulator. Therefore, with a low-fidelity simulator it might be easier to achieve higher diversity by continuously evolving the same population (as done by PaiR) rather than by relying on repopulation (what is integrated into DeepNSGA-II). Indeed, repopulation forces the algorithm to use the test budget to re-generate images belonging to the RCC rather than diversifying images already belonging to a RCC.

Further, we can observe that, during the first 10 hours of execution, both PaiR and NSGA-II present

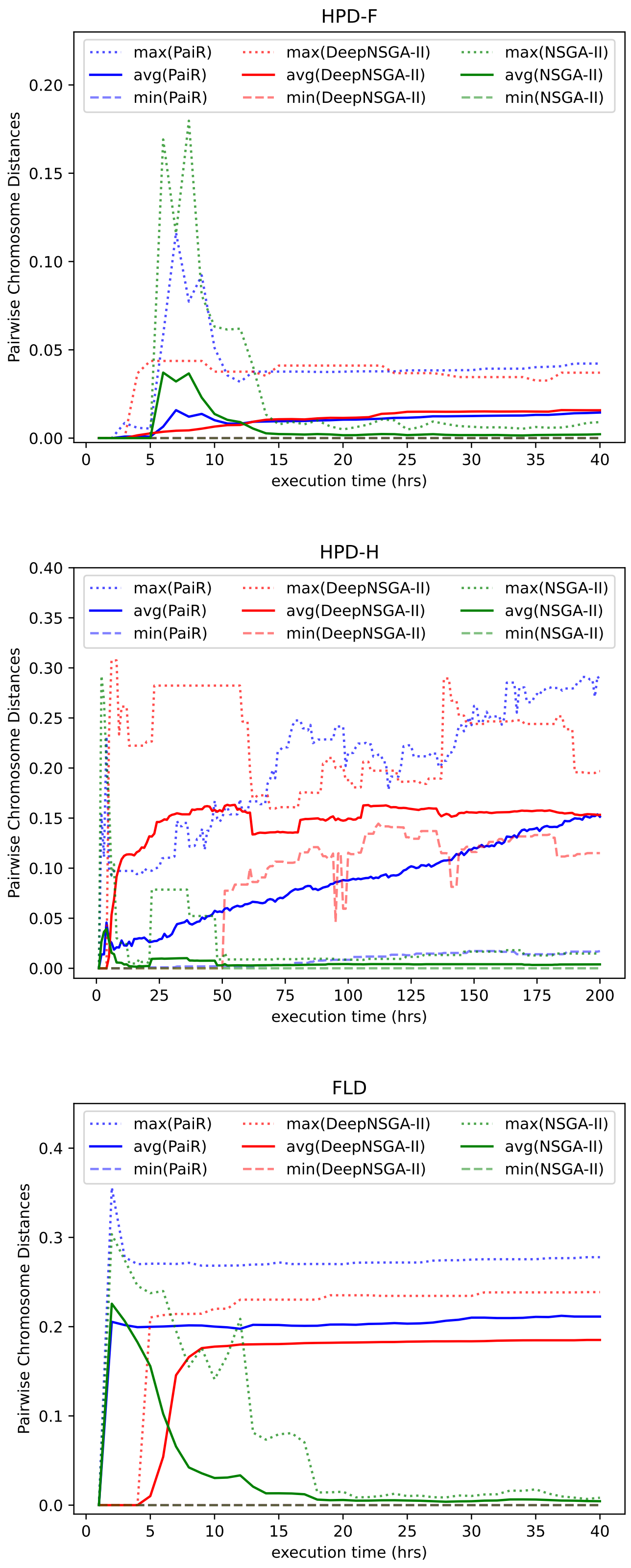


Figure 4.7: Evolution of diversity achieved by PaiR compared to NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs.

Table 4.5: RQ1: Average diversity across RCCs for HPD-F

Time (hrs.)	**Avg. diversity (standard deviation)**			**Statistical test**			
				NSGA-II		**DeepNSGA-II**	
	PaiR	**NSGA-II**	**DeepNSGA-II**	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.001 (0.002)	0.000 (0.000)	0.003 (0.010)	3.18e-03	0.60	0.159	0.555
10	0.010 (0.017)	0.014 (0.020)	0.007 (0.011)	0.870	0.51	0.190	0.576
15	0.010 (0.013)	0.002 (0.003)	0.011 (0.012)	8.53e-03	0.66	0.843	0.512
20	0.010 (0.014)	0.002 (0.002)	0.011 (0.013)	5.53e-04	0.71	0.843	0.512
25	0.012 (0.013)	0.002 (0.002)	0.015 (0.014)	6.18e-06	0.78	0.711	0.476
30	0.013 (0.014)	0.002 (0.002)	0.015 (0.014)	5.45e-05	0.75	0.771	0.481
35	0.013 (0.014)	0.002 (0.002)	0.015 (0.014)	6.18e-06	0.78	0.741	0.479
40	0.014 (0.015)	0.002 (0.003)	0.016 (0.014)	1.31e-05	0.77	0.640	0.470

Table 4.6: RQ1: Average diversity across RCCs for HPD-H

Time (hrs.)	**Avg. diversity (standard deviation)**			**Statistical test**			
				NSGA-II		**DeepNSGA-II**	
	PaiR	**NSGA-II**	**DeepNSGA-II**	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.028 (0.039)	0.028 (0.056)	0.011 (0.045)	0.062	0.612	2.58e-08	0.805
10	0.028 (0.039)	0.005 (0.008)	0.109 (0.083)	5.43e-03	0.669	2.45e-03	0.314
15	0.029 (0.037)	0.002 (0.002)	0.113 (0.077)	5.45e-10	0.880	4.12e-04	0.282
20	0.028 (0.035)	0.002 (0.002)	0.125 (0.067)	5.66e-09	0.860	1.80e-06	0.205
25	0.029 (0.036)	0.010 (0.022)	0.146 (0.061)	4.06e-06	0.785	4.32e-10	0.114
50	0.056 (0.049)	0.003 (0.002)	0.156 (0.047)	2.30e-12	0.934	3.09e-11	0.089
75	0.072 (0.061)	0.003 (0.003)	0.135 (0.014)	1.20e-13	0.959	5.37e-10	0.115
100	0.088 (0.063)	0.004 (0.002)	0.149 (0.015)	1.89e-15	0.992	1.54e-06	0.202
125	0.102 (0.071)	0.004 (0.003)	0.159 (0.012)	6.37e-16	1.0	8.51e-04	0.293
150	0.120 (0.083)	0.004 (0.004)	0.155 (0.020)	6.37e-16	1.0	0.021	0.357
175	0.140 (0.095)	0.003 (0.003)	0.157 (0.017)	6.37e-16	1.0	0.049	0.378
200	0.151 (0.093)	0.004 (0.004)	0.153 (0.016)	6.37e-16	1.0	0.369	0.444

Table 4.7: RQ1: Average diversity across RCCs for FLD

Time (hrs.)	**Avg. diversity (standard deviation)**			**Statistical test**			
				NSGA-II		**DeepNSGA-II**	
	PaiR	**NSGA-II**	**DeepNSGA-II**	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	0.200 (0.077)	0.156 (0.073)	0.010 (0.045)	4.01e-03	0.685	6.06e-13	0.932
10	0.200 (0.077)	0.030 (0.044)	0.178 (0.061)	4.36e-10	0.905	0.015	0.657
15	0.202 (0.077)	0.013 (0.023)	0.181 (0.063)	4.36e-10	0.905	0.028	0.643
20	0.202 (0.077)	0.006 (0.004)	0.182 (0.063)	4.36e-10	0.905	0.013	0.660
25	0.203 (0.077)	0.005 (0.003)	0.183 (0.064)	4.36e-10	0.905	0.028	0.643
30	0.210 (0.080)	0.004 (0.003)	0.184 (0.064)	4.36e-10	0.905	2.08e-03	0.700
35	0.211 (0.080)	0.006 (0.005)	0.185 (0.065)	4.36e-10	0.905	1.60e-03	0.705
40	0.211 (0.080)	0.004 (0.002)	0.185 (0.065)	4.36e-10	0.905	1.60e-03	0.705

a sharp peak for both average and maximum diversity. This is mainly due to the fact that in the first hours of execution the two algorithms generate a limited number of images belonging to each RCC (see Fig. 4.8), thus leading to high diversity regardless of the generation strategy. After 10 hours of execution and the initial peak, for PaiR, we can observe an up-trend in average diversity reaching 0.014, 0.151, and 0.211 for HPD-F, HPD-H, and FLD, respectively. In contrast, NSGA-II presents an average diversity that keeps decreasing till 15 hours (HPD-F), 50 hours (HPD-H), and 20 hours (FLD), and then stabilizes around 0.002, 0.004, and 0.004, at much lower levels than PaiR. Such initial peak is not observed with DeepNSGA-II, likely because of repopulation. Indeed, repopulation slows down the identification of images belonging to a cluster: DeepNSGA-II requires between 10 and 75 hours to generate the same number of RCC images generated by PaiR in five hours, as shown in Tables 4.5 to 4.7, discussed below. However, repopulation leads to the identification of more diverse images since those generated by DeepNSGA-II in later iterations are not similar to the ones generated in previous iterations, as shown by the up-trending DeepNSGA-II curve. For DeepNSGA-II, we observe an average diversity reaching 0.016 (HPD-F), 0.153 (HPD-H), and 0.185 (FLD), respectively.

Tables 4.5, 4.6, and 4.7 report statistics about the diversity obtained for all RCCs, for PaiR, NSGA-II, and DeepNSGA-II along with the standard deviation. To compare the three techniques, we also provide the Vargha and Delaney's $\hat{A}_{12}$ effect size and p-values resulting from a non-parametric Mann-Whitney U-test, where each individual observation is the diversity for one RCC in one of the four runs. PaiR achieves a significantly higher diversity (*p-value* < 0.05) compared to NSGA-II in all subjects and for all but two timestamps. For HPD-F, PaiR performs similarly to NSGA-II up to 10 hours of execution but then outperforms it with a large effect size (i.e., > 0.714)[2]. Given the safety-critical contexts in which the DNN under analysis should be adopted, we believe a budget of 20 hours to be acceptable for DNN analysis. PaiR performs similarly to DeepNSGA-II (i.e., *p-value* ≥ 0.05), with a higher average diversity for PaiR up to 10 hours of execution.

For HPD-H, PaiR significantly outperforms NSGA-II at all timestamps except one (i.e., *p-value* ≥ 0.05 for 5-hour execution). However, DeepNSGA-II performs significantly better than PaiR for all the timestamps except two. Indeed, at 5 hours of execution PaiR performs significantly better than DeepNSGA-II with a large effect size and converges to a similar diversity after 200 hours.

For FLD, PaiR performs significantly better than both NSGA-II and DeepNSGA-II, with large and medium effect sizes, respectively. Compared to HPD-F, which relies on the same simulator used by FLD, the performance of PaiR could be attributed to the ease of generating images belonging to FLD clusters; indeed, the radius of FLD clusters range between $[27.33, 74.65]$ while this range for HPD-F is $[0.039, 0.145]$. In such situation, PaiR can spend most of the test budget to maximize the diversity of the generated images.

Fig. 4.8 shows the average, minimum, and maximum percentage of individuals belonging to one of the RCCs under analysis, observed at each timestamp over the four executed runs. For PaiR, NSGA-II and DeepNSGA-II, the number of individuals belonging to any RCC increases over time, though this trend is much steeper for PaiR. Indeed, on average, a larger proportion of the individuals generated by PaiR belongs to a RCC for a given execution time. This should result in better supporting the generation of PART rules at later stages of SEDE.

Tables 4.8, 4.9, and 4.10 report statistics about the percentage of individuals belonging to any RCC, for PaiR, NSGA-II and DeepNSGA-II along with the standard deviation. We report the $\hat{A}_{12}$ statistics and p-values resulting from a non-parametric Mann-Whitney U-test, where each observation is the percentage of individuals belonging to one cluster in one of the four runs. PaiR achieves a significantly higher number of individuals belonging to RCCs (*p-value* < 0.05) compared to NSGA-II, for all subjects and timestamps except three. Compared to DeepNSGA-II, PaiR identifies a significantly higher number of individuals belonging to RCCs for all the cases except FLD, where both techniques fare similarly.

For HPD-F, around 10 hours of execution, PaiR performs similarly to NSGA-II; both PaiR and NSGA-II perform significantly better than DeepNSGA-II. However, with a time budget of 25 hours PaiR

[2]Please note that effect size is considered medium when $0.638 < \hat{A}_{12} < 0.714$, large when $\hat{A}_{12} \geq 0.714$ [122].

Table 4.8: RQ1: Percentage of individuals belonging to RCCs for HPD-F

Time (hrs.)	Avg. (standard deviation) % of individuals belonging to RCCs			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	6.7% (14.162)	0.0% (0.0)	0.60% (2.134)	2.06e-04	0.65	8.72e-03	0.616
10	42.0% (47.27)	32.0% (44.36)	10.2% (22.72)	0.309	0.56	7.01e-03	0.656
15	60.0% (49.61)	40.0% (49.61)	25.7% (33.77)	0.076	0.60	7.48e-04	0.705
20	60.0% (49.61)	40.0% (49.61)	31.7% (37.10)	0.076	0.60	7.48e-04	0.705
25	60.7% (48.82)	40.0% (49.61)	38.9% (38.84)	0.024	0.63	9.77e-04	0.706
30	70.0% (46.41)	40.0% (49.61)	42.1% (40.36)	7.48e-03	0.65	2.10e-05	0.764
35	70.0% (46.41)	40.0% (49.61)	44.5% (40.98)	7.48e-03	0.65	2.09e-05	0.764
40	70.0% (46.41)	40.0% (49.61)	45.2% (41.17)	7.48e-03	0.65	2.88e-05	0.760

Table 4.9: RQ1: Percentage of individuals belonging to RCCs for HPD-H

Time (hrs.)	Avg. (standard deviation) % of individuals belonging to RCCs			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	50.91% (39.87)	27.27% (35.60)	0.91% (2.84)	1.24e-03	0.694	2.26e-12	0.909
10	70.91% (40.80)	41.09% (42.93)	8.55% (4.69)	3.75e-06	0.776	8.78e-08	0.824
15	85.45% (32.38)	45.45% (43.00)	9.45% (4.32)	1.24e-10	0.880	6.01e-12	0.909
20	90.91% (23.41)	49.09% (40.58)	10.27% (4.26)	6.38e-12	0.909	4.36e-17	1.0
25	92.73% (23.26)	59.27% (36.63)	10.91% (3.89)	2.71e-13	0.929	1.48e-17	1.0
50	98.18% (5.82)	66.18% (35.71)	20.64% (19.92)	1.87e-14	0.950	3.41e-17	1.0
75	100.0% (0.0)	66.18% (35.71)	61.09% (34.37)	5.02e-18	1.0	5.39e-18	1.0
100	100.0% (0.0)	68.00% (32.60)	71.36% (26.34)	5.14e-18	1.0	5.48e-18	1.0
125	100.0% (0.0)	74.55% (29.29)	73.36% (24.22)	4.44e-18	1.0	5.38e-18	1.0
150	100.0% (0.0)	74.55% (29.29)	77.91% (18.79)	4.44e-18	1.0	5.24e-18	1.0
175	100.0% (0.0)	74.55% (29.29)	82.55% (13.89)	4.44e-18	1.0	4.92e-18	1.0
200	100.0% (0.0)	74.55% (29.29)	84.36% (11.60)	4.44e-18	1.0	4.76e-18	1.0

Table 4.10: RQ1: Percentage of individuals belonging to RCCs for FLD

Time (hrs.)	Avg. (standard deviation) % of individuals belonging to RCCs			Statistical test			
				NSGA-II		DeepNSGA-II	
	PaiR	NSGA-II	DeepNSGA-II	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)	*p-value* (U-test)	Effect Size ($\hat{A}_{12}$)
5	90.0% (30.38)	78.80% (28.13)	5.0% (22.07)	5.42e-11	0.905	4.06e-14	0.925
10	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
15	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
20	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
25	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
30	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
35	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5
40	90.0% (30.38)	78.80% (28.13)	90.0% (30.38)	5.42e-11	0.905	1.0	0.5

performs significantly better than both NSGA-II and DeepNSGA-II, with an effect size above 0.60. For HPD-H, PaiR significantly outperforms NSGA-II and DeepNSGA-II (*p-value* < 0.05); effect size is always large (above 0.78 in our cases) except for one case (medium, for 5-hour execution with NSGA-II).

For HPD-F and HPD-H, we conjecture that DeepNSGA-II performs worse than PaiR because the images in a RCC are very similar to each other and several mutations of the best images in a population are needed to generate additional images belonging to a same RCC; such observation is supported by the plot for HPD-F, where PaiR lays in an intermediate plateau before reaching its best average result. The repopulation operator adopted by DeepNSGA-II may prevent DeepNSGA-II from generating individuals that belong to the RCC and are different from the already generated ones (e.g., after repopulation, DeepNSGA-II end-ups with images that match the ones already generated because they are simpler to generate).

For FLD, PaiR quickly reaches (in less than 5 hours) a plateau for the average (90%) and max (100%) percentages of individuals per RCC. Though NSGA-II reaches its plateau in less than 5 hours, its plateau is lower than PaiR's (78.8% vs. 90.0%). Further, DeepNSGA-II reaches the same plateau values as PaiR but requires more time (i.e., 8 hours for the peak of the average curve). These results are mainly due to the ease of generating images belonging to most FLD clusters; indeed, all three algorithms quickly reach their peaks. However, both PaiR and DeepNSGA-II do not improve over 90% because they do not cover one RCC. In other words, for FLD, when PaiR and DeepNSGA-II analyse a cluster that they can cover, both generate a population of images that fully belongs to the cluster. The same observation cannot be made for NSGA-II; indeed, although all three algorithms cover the same FLD clusters, PaiR and DeepNSGA-II achieve a higher percentage of individuals per cluster. NSGA-II probably gets stuck in local optima.

Fig. 4.9 shows the percentage of clusters covered by PaiR, NSGA-II and DeepNSGA-II, for all subjects. PaiR is capable of covering (generating representative images of) a larger number of RCCs (27, in total): 7 out of 10 RCCs (70.0%) for HPD-F, 11 out of 11 RCCs (100.0%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. DeepNSGA-II covers a lower number of clusters (26, in total): 6 out of 10 RCCs (60.0%) for HPD-F, 11 out of 11 RCCs (100.0%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. NSGA-II, instead, covers only 23 RCCs: 4 out of 10 RCCs (40.0%) for HPD-F, 10 out of 11 RCCs (90.9%) for HPD-H, and 9 out of 10 RCCs (90.0%) for FLD. Since the cases in which PaiR does not cover all the clusters are the ones involving a simulator with a lower number of configuration parameters (i.e., HPD-F and FLD), we believe that our imperfect results are due to our limited control of the simulator in use. However, PaiR still performs better than NSGA-II and DeepNSGA-II, thus showing that it can better leverage the capabilities of the simulator. We conclude that PaiR is a better choice than both DeepNSGA-II and NSGA-II since it helps explain a larger number of RCCs, 27 (87.1%) compared to 26 (83.8%) with DeepNSGA-II and 23 (74.1%) with NSGA-II.

Tables 4.11, 4.12, and 4.13 report the percentage of covered RCCs across the four runs, for PaiR, NSGA-II and DeepNSGA-II. Further, we report the p-value computed with a non-parametric Fisher's exact test where we compare the number of clusters covered by PaiR and the competing approaches. Unsurprisingly, there is no significant difference for FLD, where the three algorithms cover almost all the clusters because of the easiness of the task (see discussion above). Differences with DeepNSGA-II tend to be not significant, which indicates that covering a cluster is a simple task for both algorithms. However, differences are always significant for small time budget; indeed, up to 5 hours, PaiR covers more clusters. A higher number of significant differences is instead observed when comparing PaiR to NSGA-II.

To summarize, our results suggest that the adoption of PaiR is the best choice in our context; indeed, PaiR can (1) generate images for a larger number of RCCs than NSGA-II and DeepNSGA-II, (2) generate significantly more images belonging to each RCC, compared to both DeepNSGA-II and NSGA-II, and, (3) achieve significantly higher image diversity than NSGA-II and similar diversity to that of DeepNSGA-II, except for one case study, where PaiR outperforms DeepNSGA-II.

In the rest of our evaluation, we ignore the clusters for which PaiR was not able to identify representa-

Table 4.11: RQ1: Percentage of covered RCCs for HPD-F

Time (hrs.)	**Avg. % of covered RCCs**			***p-value*** (Fisher's Exact)	
	PaiR	**NSGA-II**	**DeepNSGA-II**	**NSGA-II**	**DeepNSGA-II**
5	30.0%	0.0%	7.5%	1.85e-04	0.019
10	50.0%	40.0%	32.5%	0.5	0.172
15	60.0%	40.0%	47.5%	0.117	0.369
20	60.0%	40.0%	47.5%	0.117	0.369
25	70.0%	40.0%	57.5%	0.012	0.352
30	70.0%	40.0%	57.5%	0.012	0.352
35	70.0%	40.0%	57.5%	0.012	0.352
40	70.0%	40.0%	60.0%	0.012	0.482

Table 4.12: RQ1: Percentage of covered RCCs for HPD-H

Time (hrs.)	**Avg. % of covered RCCs**			***p-value*** (Fisher's Exact)	
	PaiR	**NSGA-II**	**DeepNSGA-II**	**NSGA-II**	**DeepNSGA-II**
5	81.82%	45.45%	11.36%	7.54e-04	1.84e-11
10	81.82%	54.55%	93.18%	0.011	0.195
15	90.91%	54.55%	100.0%	2.27e-04	0.116
20	100.0%	63.64%	100.0%	5.75e-06	1.0
25	100.0%	81.82%	100.0%	5.51e-03	1.0
50	100.0%	81.82%	100.0%	5.51e-03	1.0
75	100.0%	81.82%	100.0%	5.51e-03	1.0
100	100.0%	90.91%	100.0%	0.116	1.0
125	100.0%	90.91%	100.0%	0.116	1.0
150	100.0%	90.91%	100.0%	0.116	1.0
175	100.0%	90.91%	100.0%	0.116	1.0
200	100.0%	90.91%	100.0%	0.116	1.0

Table 4.13: RQ1: Percentage of covered RCCs for FLD

Time (hrs.)	**Avg. % of covered RCCs**			***p-value*** (Fisher's Exact)	
	PaiR	**NSGA-II**	**DeepNSGA-II**	**NSGA-II**	**DeepNSGA-II**
5	90.0%	90.0%	5.0%	1.0	1.47e-15
10	90.0%	90.0%	90.0%	1.0	1.0
15	90.0%	90.0%	90.0%	1.0	1.0
20	90.0%	90.0%	90.0%	1.0	1.0
25	90.0%	90.0%	90.0%	1.0	1.0
30	90.0%	90.0%	90.0%	1.0	1.0
35	90.0%	90.0%	90.0%	1.0	1.0
40	90.0%	90.0%	90.0%	1.0	1.0

tive images (Step 2.1), as described above. Also, we ignore one cluster for which SEDE did not identify unsafe images (Step 2.2), possibly because the hazard-triggering event is the presence of jewelry, which is not generated by our simulator. In total, we ignore three clusters in the case of HPD-F, and one cluster for HPD-H and FLD.

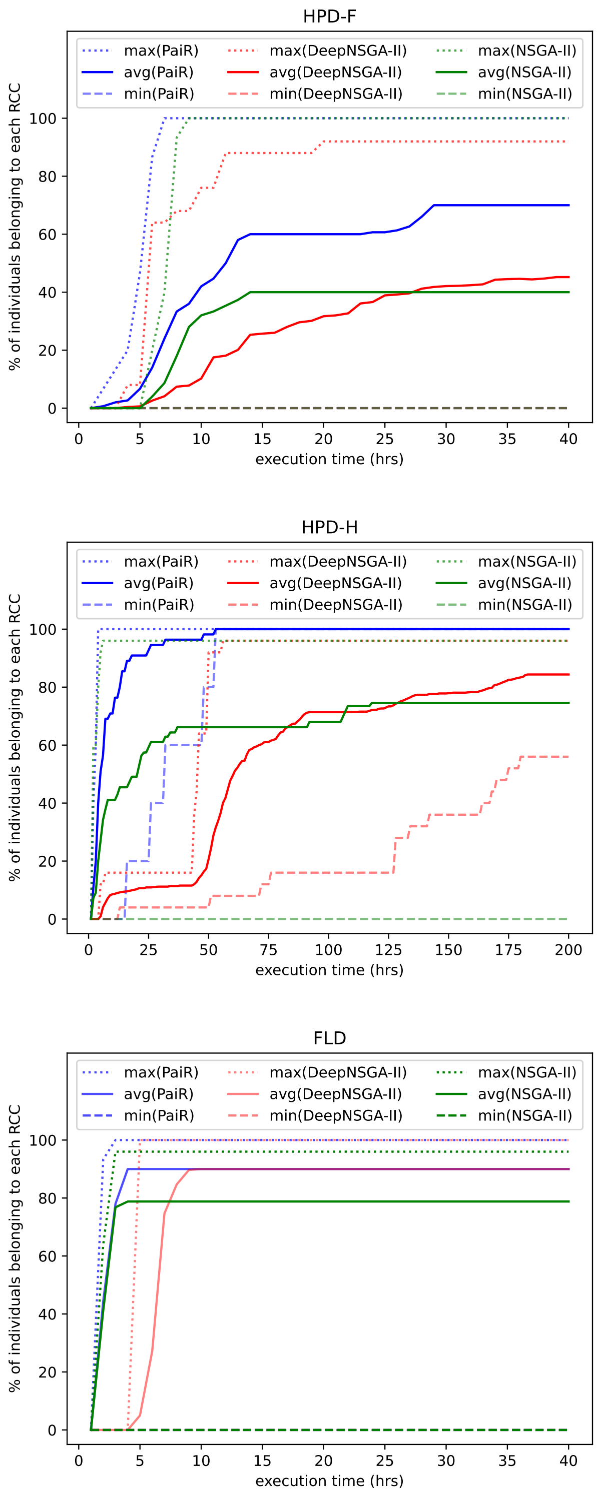
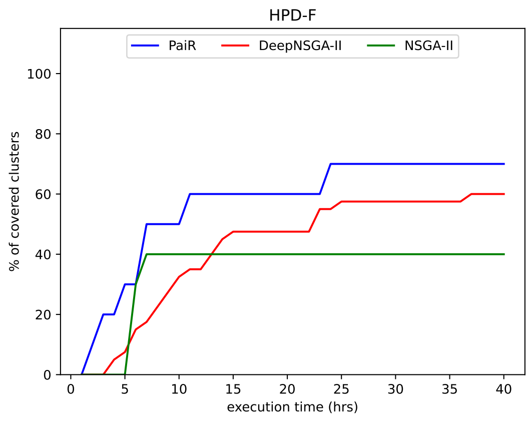
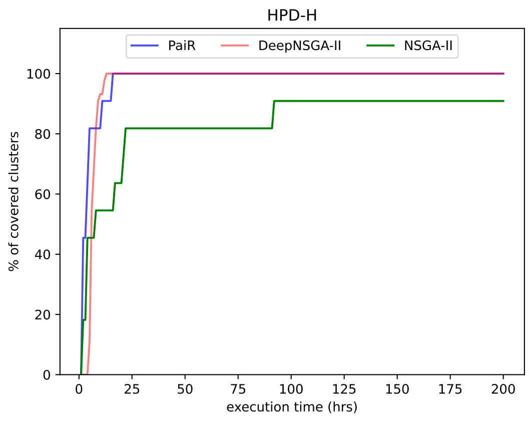


Figure 4.8: Percentage of individuals belonging to a cluster generated by PaiR compared to NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs.





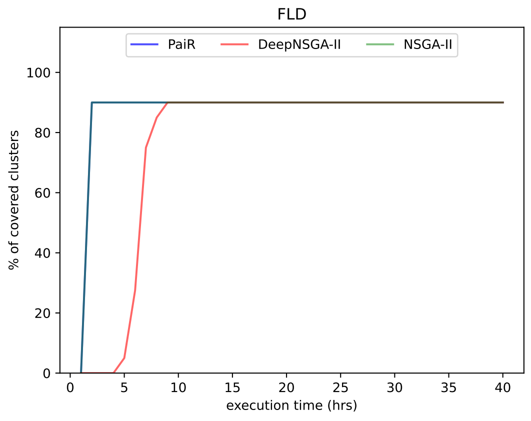


Figure 4.9: Percentage of clusters covered by PaiR, NSGA-II and DeepNSGA-II for HPD-F, HPD-H, and FLD DNNs.

RQ2. *Does SEDE generate simulator images that are close to the center of each RCC?*

Design and measurements.

This research question evaluates whether the images generated by SEDE for each RCC are closer to the medoid of the RCC than random failing images. Otherwise, we cannot claim that SEDE contributes to generating images that help characterize the RCC.

To measure the distance of an image from the RCC medoid we once again rely on the heatmap distance between the RCC medoid and the image (Equation 4.2).

For each RCC, we compute the distance between the RCC medoid and every image generated by SEDE to characterize the RCC in Step 2.1. Also, we compute the distance of the unsafe images in the simulator-based test set (i.e., random failing images) from the RCC medoid. To positively answer our research question, for each RCC, the average distance from the medoid to the images generated by SEDE should be significantly smaller than the average distance obtained with random unsafe simulated images, based on a non-parametric Mann-Whitney U-test.

Table 4.14: RQ2: Average distance from RCC's medoid for HPD-F (RCCs 1 to 7), HPD-H (RCCs 8 to 17) and FLD (RCCs 18 to 26).

Case Study Subject	**RCC**	**Avg. distance from medoid**		**p-value** (U-test)	**Effect Size** ($\hat{A}_{12}$)
		SEDE Images	Unsafe Images		
HPD-F	1	0.075	0.469	5.11e-17	1.0
	2	0.118	0.480	4.52e-16	1.0
	3	0.058	0.476	7.17e-17	1.0
	4	0.072	0.459	2.12e-16	1.0
	5	0.085	0.457	6.45e-16	1.0
	6	0.128	0.470	1.48e-16	1.0
	7	0.096	0.439	5.43e-17	1.0
HPD-H	8	0.057	0.135	1.52e-14	1.0
	9	0.047	0.120	3.47e-14	1.0
	10	0.067	0.134	1.43e-08	0.98
	11	0.050	0.110	5.61e-12	1.0
	12	0.098	0.183	4.81e-17	1.0
	13	0.068	0.208	3.33e-17	1.0
	14	0.101	0.214	1.13e-13	1.0
	15	0.060	0.156	2.28e-17	1.0
	16	0.067	0.159	7.11e-17	1.0
	17	0.107	0.126	5.43e-03	0.91
FLD	18	25.66	46.28	1.79e-08	0.985
	19	20.03	41.90	2.87e-10	1.0
	20	45.17	56.84	4.12e-03	0.910
	21	32.95	49.99	1.22e-05	0.965
	22	49.99	57.64	0.037	0.845
	23	35.63	48.03	3.84e-03	0.905
	24	51.10	58.63	0.048	0.645
	25	64.45	72.75	5.12e-04	0.940
	26	42.11	53.30	4.32e-04	0.945

Experiment Results.

Fig. 4.10 shows boxplots reporting, for all RCCs across DNNs, the heatmap distances from RCC medoids, for images in the unsafe test set and those generated by SEDE for every RCC in Step 2.1. Fig. 4.10 shows that the images generated by SEDE (i.e., boxplots with IDs 1 to 26) are much closer to the RCC medoid than random unsafe images (i.e., boxplots with IDs UI-1 to UI-26). Indeed, with SEDE, the median lays in the ranges $[0.056, 0.125]$ for HPD-F, $[0.048, 0.113]$ for HPD-H, and $[18.6, 64.3]$ for FLD. For random unsafe images, the median tends to be much higher and lays in the ranges $[0.38, 0.48]$ for HPD-F, $[0.08, 0.24]$ for HPD-H, and $[42.0, 73.6]$ for FLD.

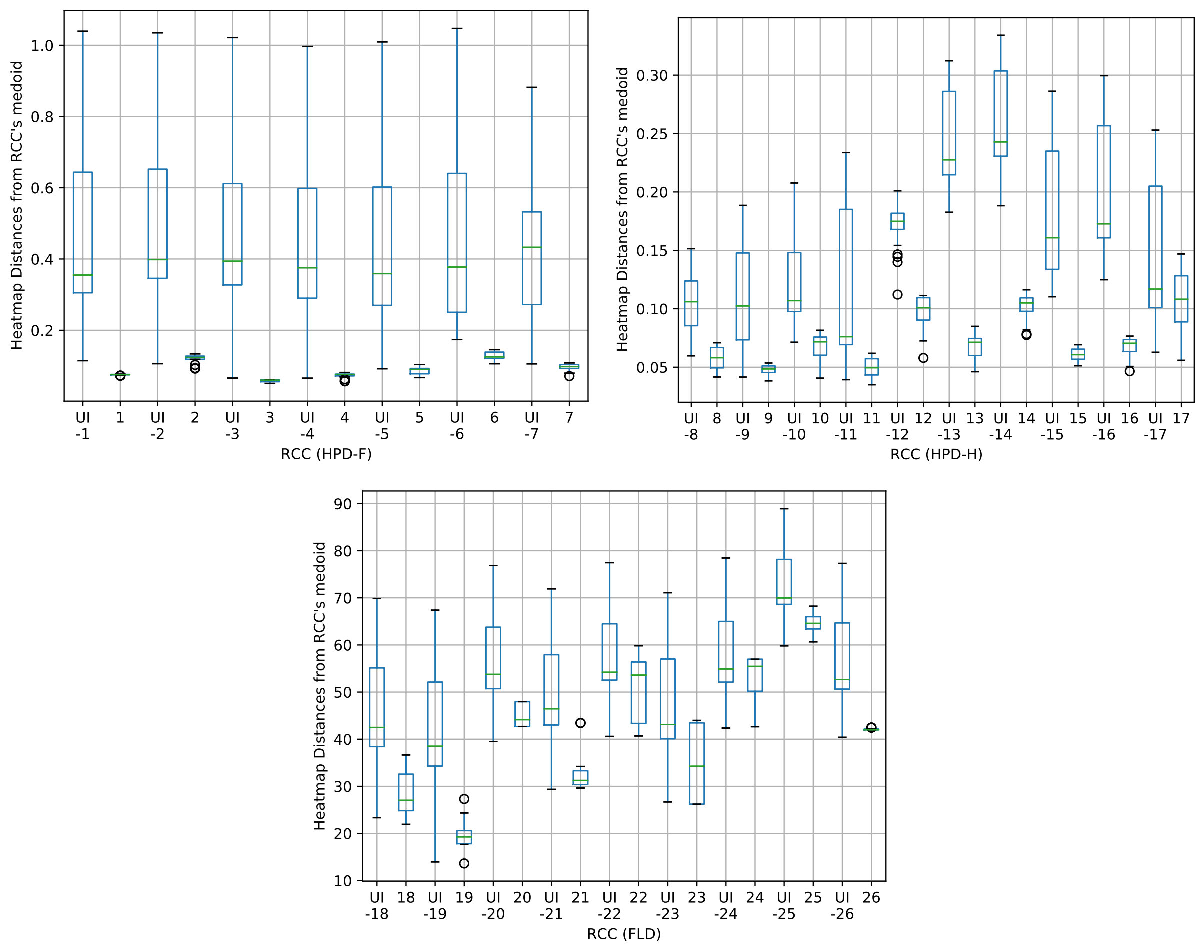


Figure 4.10: Heatmap distances of SEDE images from the medoid of HPD-F, HPD-H, and FLD RCCs compared to unsafe test set images (UI).

Table 4.14 provide, for each RCC across DNNs, the average distance of SEDE images and random unsafe images, along with p-values and effect size.

On average, SEDE images are closer to the medoid of the clusters by 80.7% for HPD-F, 54.6% for HPD-H and 24.9% for FLD. These differences with unsafe test set images are always statistically significant ($p\text{-}value < 0.05$) with a large effect size.

RQ3. *Does SEDE generate, for each RCC, a set of images sharing similar characteristics?*

Design and measurements.

This research question assesses if the images generated by SEDE for each RCC, present similar characteristics (i.e., similar values for a subset of the simulator parameters). If this is true, for each RCC, we should observe a subset of parameters with a variance that is significantly lower than the variance observed in randomly generated images.

For each RCC, instead of comparing the variance of each parameter p, which depends on the parameter range, we can compare the variance reduction rate (VRR), which can be computed as the ratio of the variance for a parameter p for a given RCC C_i over that of a set of randomly generated images:

$$VRR_{C_i}(p) = 1 - \frac{variance\ of\, p\ for\ the\ images\ in\ C_i}{variance\ of\ p\ for\ a\ set\ of\, random\ images} \tag{4.17}$$

For a parameter, a positive VRR indicates that its values are likely to be constrained to a smaller range than the one of the input domain. A 0.5 VRR means that the variance is reduced by 50%, which is considered to be a high reduction rate [18].

For each RCC, we compute VRR for each parameter. We positively answer our research question if, for a large number of RCCs, a subset of parameters presents a large VRR (> 0.5).

Experiment Results.

Fig. 4.11 provides boxplots capturing the distribution of the variance reduction for each of the 26 RCCs. Each data point in a boxplot captures the variance reduction of one parameter.

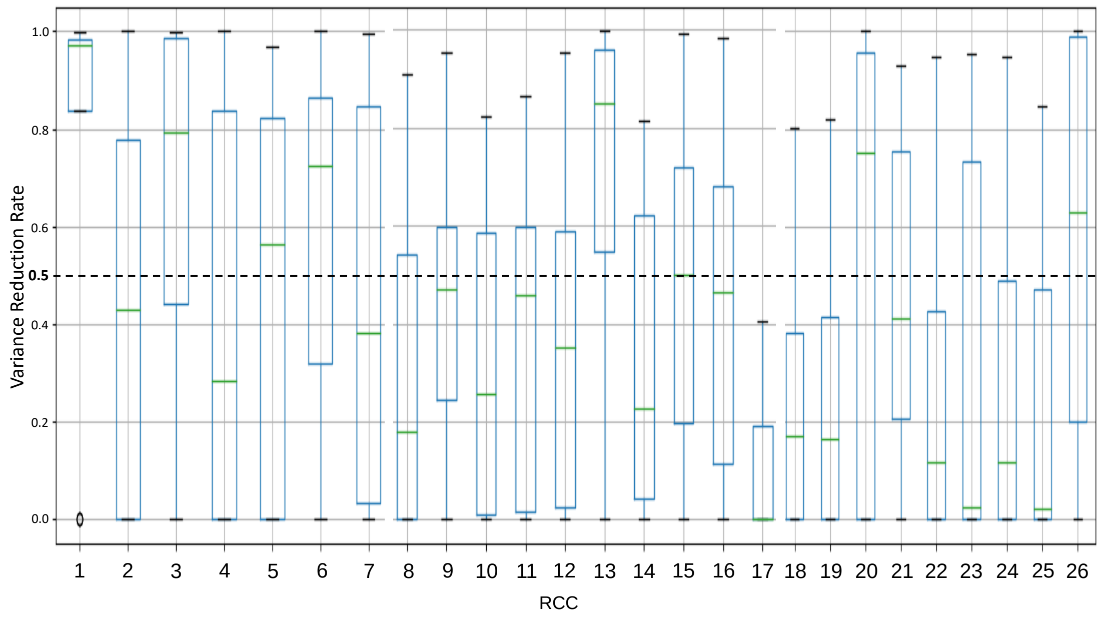


Figure 4.11: Variance reduction for all simulator parameters associated with generated images in the RCCs of HPD-F (boxplots 1 to 7), HPD-H (boxplots 8 to 17), and FLD (boxplots 18 to 26) DNNs.

Fig. 4.11 shows that for all the RCCs except boxplot 17, the top whisker is above 0.8; since the top whisker reports the max value (excluding outliers[3]) observed for a parameter, such result indicates that for all RCCs except one, we can observe at least one parameter with a high variance reduction, thus enabling us to positively answer our research question. Also, for 20 out of 26 clusters, the third quartile is above 0.5. Since it indicates the lowest value for the 25% data points having the highest variance reduction, it means that in 20 RCCs, 25% of the parameters present a variance reduction rate above 0.5. Therefore, for a large proportion of RCCs (77%), more than one parameter present a large variance reduction, which

[3]In our boxplots, $upperwhisker = min(max(x), Q_3 + 1.5 * IQR)$, $lowerwhisker = max(min(x), Q_1 - 1.5 * IQR))$, with IQR being the Inter Quartile Range and Q_x the x-th quantile.

indicates that the hazard-triggering event is captured by several parameters (i.e., a specific combination of parameter values is needed to make the DNN fail).

RQ4. *Do the RCC expressions identified by SEDE delimit an unsafe space?*

Design and measurements.

For each RCC, SEDE provides a set of expressions representing conjunctions of parameter ranges that characterize the unsafe images in the RCC. This research question evaluates if these expressions actually characterize an unsafe space, which implies that images whose parameters match such an expression are more likely to trigger a DNN failure than those that do not.

To address this research question, for each RCC, we generated 500 images matching the RCC expression and computed the DNN accuracy obtained with these images. To generate each image, for each simulator parameter, we selected a random value in the range provided in the expression. Also, we considered a random input set consisting of 500 images selected from the randomly generated simulator images used to test the fine-tuned DNN (see column Simulator-based Test Set in Table 4.3).

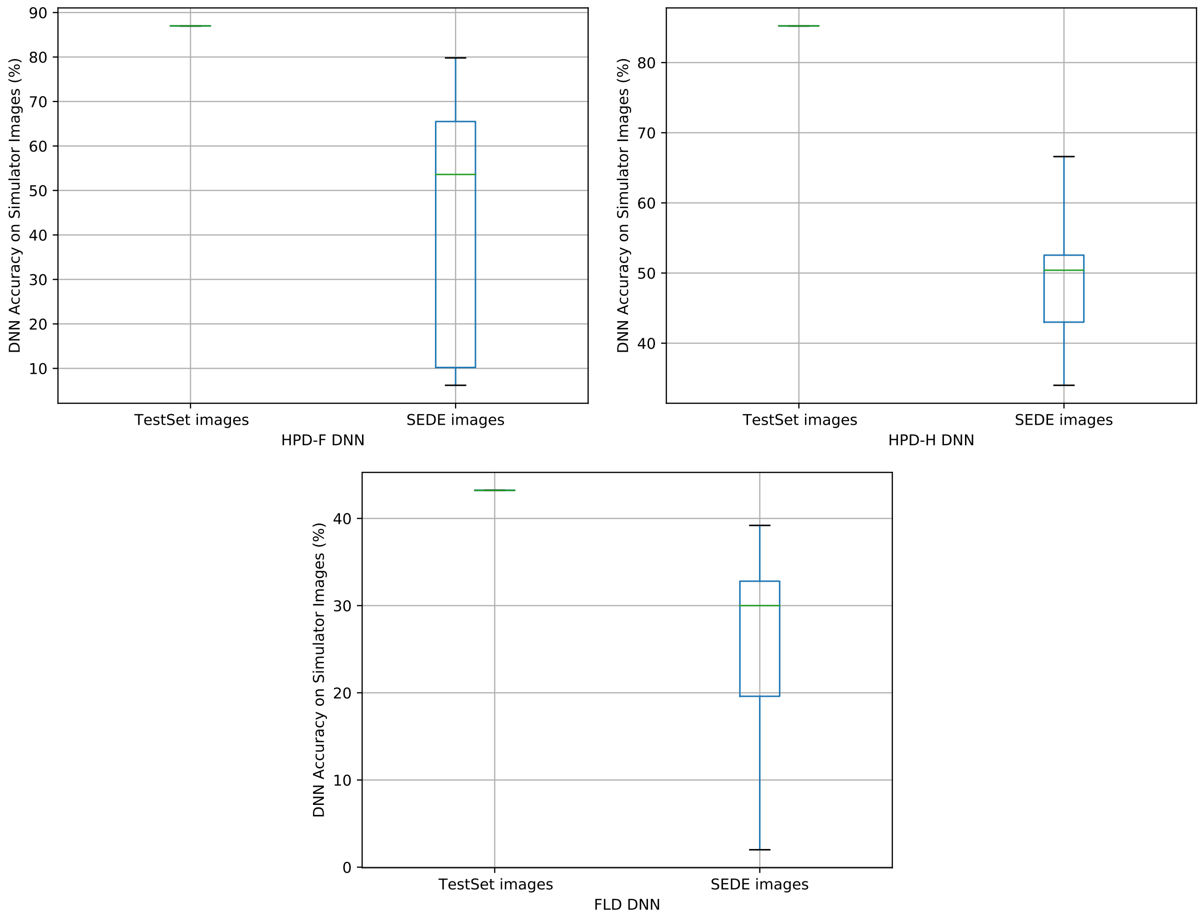


Figure 4.12: Percentage of correctly classified images observed on SEDE images compared to the random test set images for HPD-F, HPD-H, and FLD DNNs.

We can positively answer our research question if, for a large subset of RCCs, the images generated from RCC expressions lead to a significantly lower DNN accuracy [4] than randomly generated images, for

[4]Recall that accuracy is 100% minus the percentage of inputs leading to DNN failures (i.e., if 90% of the inputs generated by SEDE is failure-inducing, we will observe an accuracy of 10%).

each RCC. Since, for each RCC, we compare two image groups (i.e., 500 random images and 500 images generated by SEDE) labeled with a categorical variable (i.e., indicating if the DNN produces a correct output), we rely on the Fisher's exact test to assess differences in proportions of images leading to DNN failures.

Experiment Results.

Fig. 4.12 provides boxplots capturing the accuracy obtained with the random test set images and those generated according to RCC expressions. Each data point in the *SEDE images* boxplots corresponds to the DNN accuracy of one of the 26 RCC expressions.

The accuracy observed with the random test sets for HPD-F, HPD-H, and FLD DNNs is 87.0%, 85.2%, and 43.2%, respectively. In contrast, the images generated according to RCC expressions lead to much lower accuracy in the ranges [6.2%, 79.8%], [34.0%, 66.6%], and [2.0%, 39.2%], for HPD-F, HPD-H and FLD, respectively. Moreover, for 25 out of 26 clusters, SEDE generates images leading to an accuracy that, based on a Fisher's exact test (see Table 4.15), significantly differs from the accuracy obtained with random images. Since in 25 out of 26 RCCs (96.15%) the RCC expressions generated by SEDE clearly delimit an unsafe space, we can positively answer RQ4.

Table 4.15: RQ4: HPD-F (RCCs 1 to 7), HPD-H (RCCs 8 to 17), and FLD (RCCs 18 to 26) accuracy within the unsafe space identified by SEDE compared to the whole input space.

Case Study Subject	**RCC**	**DNN Accuracy**		**p-value** (Fisher's Exact)
		SEDE Unsafe Set	Random Input Set	
HPD-F	1	10.2%	87.0%	7.40e-73
	2	67.4%		9.25e-04
	3	6.2%		9.11e-88
	4	79.8%		0.047
	5	73.0%		0.019
	6	59.8%		2.28e-06
	7	53.6%		2.43e-09
HPD-H	8	39.4%	85.2%	1.26e-19
	9	50.6%		1.19e-10
	10	34.0%		4.84e-25
	11	49.0%		1.31e-11
	12	41.0%		7.47e-18
	13	51.8%		5.66e-10
	14	66.6%		1.73e-03
	15	52.8%		2.19e-09
	16	56.6%		2.32e-07
	17	50.2%		7.14e-10
FLD	18	39.2%	43.2%	0.30
	19	19.6%		4.77e-13
	20	30.0%		9.19e-88
	21	32.8%		4.88e-03
	22	19.6%		4.39e-13
	23	32.8%		4.12e-03
	24	27.0%		2.82e-06
	25	2.0%		2.49e-58
	26	31.0%		6.31e-04

RQ5. *How does SEDE compare to state-of-the-art DNN accuracy improvement practices?*

Design and measurements.

This research question aims to determine if SEDE performs better than state-of-the-art solutions. We compare SEDE with HUDD since it is the only approach that works with real-world images and addresses both root cause explanations and retraining (Chapter 3). Further, other retraining approaches [70, 123, 124, 71, 125, 126] generate retraining images through pixel value transformation (e.g., image contrast, image brightness, image blur, and image noise), affine transformation (e.g., image translation, image scaling, image shearing, and image rotation), or adversarial modification (e.g., Fast Gradient Signed Method (FGSM) [127], Projected Gradient Descent (PGD) [128], Carlini & Wagner (C&W) [129]). None of these approaches, however, aims to generate images similar to real-world, failure-inducing ones targeted by SEDE. In addition, DNNs might be inaccurate because either trained for a limited number of epochs or with a limited number of inputs; in such cases, retraining the DNN for additional epochs with a larger set of randomly selected inputs would suffice. For this reason, we consider a baseline approach which consists of retraining the DNN with an additional set of randomly generated images.

Regarding SEDE, we followed the procedure described in Section 4.3.3. The third column of Table 4.16 provides the size of the training set considered to retrain each DNN; precisely, we report the total number of simulator images generated according to SEDE expressions (*Unsafe Set*), the number of simulator images retained from the simulator training set (*Training Set Sim.*) and the number of real-world images retained from the set used for fine-tuning (*Training Set Real*). For each RCC, we have generated, with SEDE, 50 images to be used for retraining. As anticipated in Section 4.3.3, we retained 5% of the images in the original simulator-based training set, and all the real-world images used to fine-tune the DNN.

In the case of HUDD, we applied its selection algorithm to sample, from an improvement set, 50 images for each RCC (HUDD selects the images that are closer to the RCC centroid). We generated an improvement set with random face images (i.e., generated by randomly selecting simulator parameter values). To avoid bias, we selected for each case study DNN the same number of images as that generated by SEDE (i.e., 50 images for each RCC), which led to a total of 450 images for FLD, 350 for HPD-F, and 500 for HPD-H.

Concerning RBL, for each case study DNN, we generated a retraining set consisting of a number of randomly generated simulator images and real-world images. To avoid unfair comparisons, the randomly generated images match in number the images selected by SEDE.

Further, for all the three approaches described above (i.e., SEDE, HUDD, and random baseline), we followed the SEDE retraining process, which consists of retraining the DNN for multiple times and selecting the DNN yielding the best test set accuracy. In our experiments, since each retraining task takes approximately three hours, we performed three retraining tasks since they can be performed overnight, which is acceptable in practice. For all the approaches, we constructed the retraining data set in the same way as SEDE. It consists of the images selected by the approach under analysis, a random selection of images from the original simulator-based training set (5% of the whole set), and all the real-world images used to fine-tune the DNN. To account for randomness, for each approach, we repeated the experiment ten times (i.e., for ten times, we selected the best DNN generated out of three runs). For each approach, we generate ten DNNs.

To positively answer our research questions, SEDE should lead to retrained DNNs with an accuracy that is significantly higher than the one observed when retraining the DNN using either HUDD or the random baseline.

Experiment Results.

Columns five to seven (i.e., RBL, HUDD, and SEDE) in Table 4.16 show the average accuracy of the three approaches across ten runs along with the delta with respect to the original DNN model. Also, for SEDE, we report the delta with respect to the best competing approach (i.e., HUDD or random baseline). Finally, we report p-values using a non-parametric Mann–Whitney U-test, for statistical significance, and the $\hat{A}_{12}$ statistics, for effect size.

SEDE, on average, improves the DNN for FLD, HPD-F and HPD-H by $+6.07\%$, $+4.50\%$ and $+18.65\%$; the other two approaches, instead, do not improve the HPD-F and FLD DNNs but decrease their performance. For HPD-H, HUDD and the random baseline led to an improvement that is lower than SEDE's, $+9.62\%$ and $+4.54\%$, respectively. We believe that in the case of HPD-F and FLD, the retraining based on HUDD and random decrease the DNN performance because the simulator being used generates images that are less realistic; such characteristic leads to the generation of images that are unlikely to include hazard-triggering events and, in turn, the retraining set selected by HUDD and random is not likely to include unsafe images. Since the retraining is performed using (1) the images selected by the approach under analysis, (2) a subset of the original training set, and (3) the real-world training set, safe cases will be over-represented in the training set. In general, unsafe images that are present in the original training set may not be retained for retraining.

Similar to the above, we believe that, since it is complicated for IEE-Faces to generate unsafe images, in the case of HPD-F and FLD, SEDE led to a more limited improvement in accuracy than in the case of HPD-H: $+4.50\%$ and $+6.07\%$ vs $+18.65\%$.

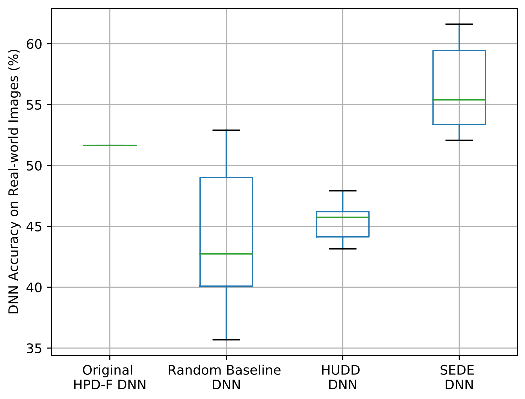
Finally, SEDE, on average, improves the DNN results by 9 percentage points over the best competing approach (i.e., $+6.19\%$ for FLD, $+10.35\%$ for HPD-F, and $+9.03\%$ for HPD-H). The difference is always statistically significant (*p-value* < 0.05) with a large effect size.

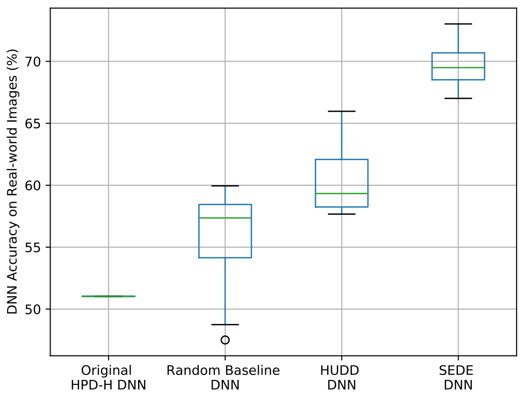
Table 4.16: RQ5: Unsafe set size and the accuracy improvement of SEDE compared to HUDD and random baseline (RBL).

DNN	Original Model Accuracy	Retraining Set Size		RBL Accuracy **(Gain)**	HUDD Accuracy **(Gain)**	SEDE		Stat. Sig.	
		Training Set (Sim. + Real)	Unsafe Set			Accuracy **(Gain)**	**Gain over best baseline**	*p-value*	$\hat{A}_{12}$
FLD	80.06%	150 + 6,000	450	77.41% (**-2.65%**)	79.94% (**-0.11%**)	86.14% (**+6.07%**)	**+6.19%**	1.8e-04	1.0
HPD-F	51.65%	1,075 + 476	350	44.33% (**-7.32%**)	45.80% (**-5.85%**)	56.15% (**+4.50%**)	**+10.35%**	4.4e-04	0.94
HPD-H	51.03%	900 + 476	500	55.57% (**+4.54%**)	60.65% (**+9.62%**)	69.68% (**+18.65%**)	**+9.03%**	1.1e-04	1.0

Figure 4.13 shows boxplots with the accuracy of the DNNs retrained using SEDE, HUDD, and the random baseline. Each data point captures the accuracy obtained in one of the ten retraining runs.

The boxplots of SEDE overlap with the boxplot of a competing approach only in one case (i.e., the max value for the random baseline), thus suggesting that SEDE performs significantly better with a strong





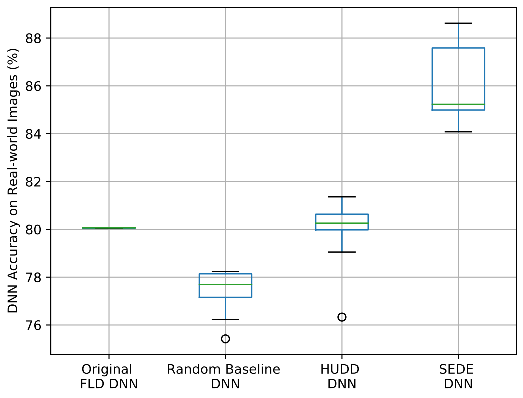


Figure 4.13: DNN accuracy on real-world test set images after retraining by SEDE compared to HUDD and random baseline for HPD-F, HPD-H and FLD case study DNNs.

effect size; indeed, competing approaches never lead to a better DNN than SEDE.

4.4.3 Threats to Validity

Internal Validity: In our work, we rely on clusters to capture different root causes of a DNN failure where the quality of clusters largely affects the characterization of an unsafe space. This threat is mitigated by empirical results demonstrating that HUDD clusters include images with similar characteristics [18].

External Validity: The selection of the case study DNNs and simulators may affect the generalizability of results. We alleviate this issue by selecting subject DNNs that implement classification and regressions tasks motivated by IEE business needs and addressing problems that are quite common in the automotive industry.

Moreover, we rely on two simulators that differ in their fidelity and number of configuration parameters. As we have seen, these characteristics affect the effectiveness of retraining and SEDE's capacity to identify hazard-triggering events. We focus on DNNs processing human faces because they are the focus of our project partners at IEE; we did not consider DNNs performing other tasks (e.g., processing road images) because an industrial case study on that topic was not available for our project.

Conclusion Validity: To avoid violating parametric assumptions in our statistical analysis, we rely on a non-parametric test and effect size measure (i.e., Mann Whitney U-test and the Vargha and Delaney's $\hat{A}_{12}$ statistics, respectively) to evaluate the statistical and practical significance of differences in results. When comparing classification results (i.e., RQ4) we applied the Fisher's exact test, which is commonly used in similar contexts.

Further, for RQ1, we executed the competing approaches for four runs, for each of the 31 RCCs under analysis, to account for randomness and eliminate bias. For RQ2 to RQ4, we compared the results obtained with a large number of images. For RQ2, we considered 25 SEDE images for each RCC and $5,470$ unsafe test set images. For RQ3, we considered 650 SEDE images and $1,500$ randomly generated images. For RQ4, we considered 500 SEDE images for each RCC and $1,500$ random test set images. For RQ5, because of the stochastic nature of SEDE (e.g., DNN retraining), the experiments were executed over thirty runs.

Construct Validity: In RQ1 we aim to evaluate if our approach achieves a higher diversity than a state-of-the-art approach. We rely on the chromosome distance computed for each pair of images generated for a RCC, which is based on the parameter values used to generate images with a simulator. By relying on simulator parameter values we can objectively measure diversity.

In RQ2 we evaluate if the images generated by SEDE are close to the medoid of the RCC under analysis. We rely on the heatmap distance since it is the distance metric adopted to generate RCCs.

For RQ3, since simulator parameters drive the selection of specific image characteristics (e.g., head direction), we rely on the variance reduction rate for parameter values to objectively assess the similarity across images belonging to a RCC; variance reduction has also been used to evaluate HUDD [18].

RQ4 evaluates if SEDE expressions are useful for delimiting an unsafe space. RQ5 evaluates the ability of SEDE to improve a DNN. For both, we relied on the accuracy metric (i.e., the percentage of correctly classified images), which is also suggested by safety standards as a mean to evaluate if DNNs can be used for safety-related tasks [15].

4.5 Conclusion

The identification of hazard-triggering events is a safety engineering practice that enables engineers to evaluate the risk associated to potentially hazardous behaviors of a system. In this chapter, we address the problem of characterizing hazard-triggering events affecting the correct execution of DNN-based systems. We introduced SEDE, a novel approach based on evolutionary algorithms, which generates expressions that characterize hazard-triggering events observed in real-world images processed by DNNs. Such expressions constrain the configuration parameters of a simulator capable of generating images similar to the real-world images under analysis. In turn, they characterize the unsafe portions of the input space.

To identify the unsafe portions of the input space from real-world images, SEDE relies on a state-of-the-art approach, HUDD, that generates clusters (i.e. RCCs) containing images sharing a common set of characteristics. Such commonalities capture the hazard-triggering events to be investigated by engineers. For each RCC, SEDE performs three distinct executions of evolutionary algorithms; as a result, it generates two sets of images belonging to the RCC, one leading to DNN failures while the other leading to correct DNN outputs. The two sets are then processed by PART, a rule extraction algorithm, to automatically derive SEDE expressions. The evolutionary algorithms employed by SEDE are a modified version of NSGA-II and PaiR, an algorithm introduced in this work to efficiently generate, using a simulator, images that belong to a RCC and are diverse.

Additionally, SEDE improves the DNN under analysis by retraining it with images that match the generated expressions.

Empirical results conducted with representative case studies in the automotive domain show that (a) our genetic algorithm, PaiR, can generate images for a larger number (87.1%) of RCCs, it generates a larger number of images for each RCC, and the generated images have a significantly higher diversity than those generated by NSGA-II (for all case studies) and DeepNSGA-II (for one case study), (b) the evolutionary searches employed by SEDE lead to images belonging to the RCCs and having similar characteristics, (c) the expressions generated by SEDE successfully characterize the unsafe input space (i.e., they lead to images showing a DNN accuracy decreased by at least 30% points, on average), and, (d) the retraining process employed by SEDE increases the DNN accuracy up to 18 percentage points with a gain over the best baseline of at least 6 percentage points.

Chapter 5

Safety Analysis based on Feature Extraction

In this chapter, we propose Safety Analysis based on Feature Extraction (SAFE), the first black-box approach to support safety analysis. The foremost objective is to not rely on internal information about the DNN or its modification, thus facilitating its adoption in practice. Indeed, engineers do not have access to such information in many contexts or are not sufficiently skilled to modify the DNN, whose development is often outsourced. Our approach targets DNNs that process images since they are the most common form of inputs for many DNN-based components in the automotive and other safety-critical domains (e.g., manufacturing robots).

5.1 Introduction

Although effective, HUDD presents a number of limitations. First, it can only analyze DNN implementations extended to compute LRP. Although LRP implementations for multiple DNN architectures relying on the TensorFlow framework are available [72], it might be particularly complex for engineers to integrate LRP into a different DNN architecture. Indeed, the relevance computation formula to be adopted for each layer depends on the layer type (e.g., input, normalization, spatial pooling, internal layer) and the presence of recursion [62].

Also, companies often acquire off-the-shelf DNNs which cannot be modified, thus preventing the computation of LRP and the application of HUDD. Moreover, computing a heatmap-based euclidean distance might become particularly expensive when layers are made of thousands of neurons and hundreds of failure-inducing images need to be processed. Finally, given that the neurons relevant for a specific DNN failure (i.e., the neurons with high relevance scores) might represent a small proportion of the neurons in a DNN layer, computing the euclidean distance considering all the items in a heatmap may potentially lead to imprecise clusters caused by noise (i.e., the sum of many differences that are almost zero).

For all the reasons above, although the safety analysis and improvement of DNNs through the automated identification of root cause clusters has demonstrated to be effective, achieving a wider adoption requires a black-box substitute for HUDD.

SAFE makes use of transfer learning-based feature extraction, dimensionality reduction, and unsupervised learning. This approach is an improvement over HUDD to avoid reliance on heatmap-based distance, which requires access to the DNN's internal information, and to improve the quality of the root cause clusters' identification. Transfer Learning transfers the knowledge from a generic domain to another specific domain using a pre-trained model. Besides a large amount of time saved by using these methods, it has been shown that starting from a pre-trained model may perform better than training from scratch even on a different problem [130, 131]. SAFE extracts the features from failure-inducing images based on convolutional layers in a pre-trained model instead of relying on heatmaps.

Further, compared to HUDD, SAFE also allows the detection of non-convex clusters [132]. A cluster is convex if, for every pair of points within this cluster, every point on the straight line segment that joins them is also within the cluster [132]. This kind of clusters is usually represented by its centroid. But in many practical cases, the data leads to clusters with arbitrary, non-convex shapes. Such clusters, however, cannot be properly detected by a centroid-based algorithm or even a hierarchical clustering algorithm, as they are not designed for arbitrary-shaped clusters.

To summarize, SAFE has the following merits compared to HUDD:

- It avoids the need for extending the DNN under analysis to compute LRP, which is achieved by relying on a transfer learning model that extracts features from the images.
- It applies a feature-based distance instead of a heatmap-based distance, thus saving training time and memory.
- It applies a density-based clustering algorithm to detect non-convex clusters, modeling the DNN failures' root causes.
- It relies on the non-convex root cause clusters to select an unsafe set for the retraining of the DNN.

5.2 The SAFE Approach

Figure 5.1 presents an overview of SAFE. It consists of six steps. In Step 1, root cause clusters are identified by relying on feature extraction-based clustering. Step 2 involves a visual inspection performed by engineers as required by functional safety analysis [82, 14]. In Step 3, a new set of images, referred to as the *improvement set*, is provided by the engineers to retrain the model. In Step 4, SAFE automatically selects a subset of images from the improvement set called the *unsafe set*. The engineers label the images in the unsafe set in Step 5. Finally, in Step 6, SAFE automatically retrains the model to enhance its prediction accuracy.

The main contributions of SAFE, when compared to HUDD, are in Step 1 and Step 4. Step 1 consists of four stages, namely (i) data acquisition and preprocessing, (ii) features extraction, (iii) dimensionality reduction, and (iv) clustering. Step 4 relies on a new method for the identification of the unsafe set that fits the clustering solution integrated in SAFE.

Similar to HUDD, SAFE includes some manual activities: Step 2 (visual inspection of images for functional safety analysis), Step 3 (generate new images), and Step 5 (label images). Such activities

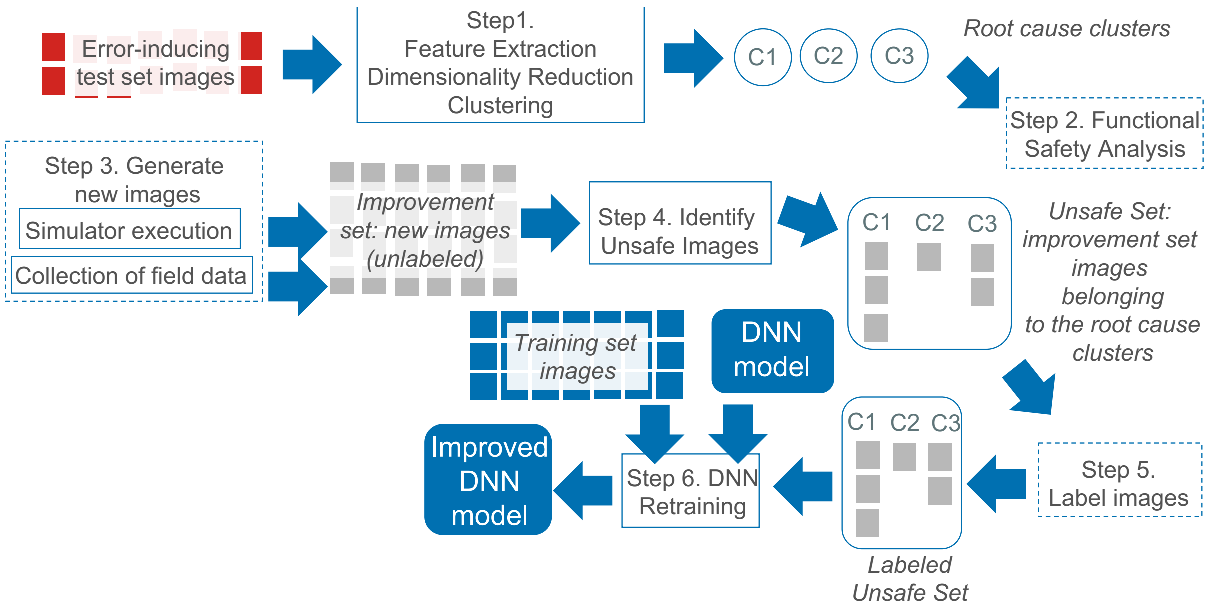

Error-inducing test set images
Step1. Feature Extraction Dimensionality Reduction Clustering
C1
C2
C3
Root cause clusters
Step 2. Functional Safety Analysis
Step 3. Generate new images
Simulator execution
Collection of field data
Improvement set: new images (unlabeled)
Step 4. Identify Unsafe Images
C1 C2 C3
Unsafe Set: improvement set images belonging to the root cause clusters
Training set images
DNN model
Improved DNN model
Step 6. DNN Retraining
C1 C2 C3
Step 5. Label images
Labeled Unsafe Set

Figure 5.1: Overview of SAFE

are, however, required also by state-of-the-practice approaches. Indeed, to debug and improve DNNs, engineers usually visually inspect all failure-inducing images, select an improvement set, and manually label the images to be reused for retraining the DNN.

However, both HUDD and SAFE significantly reduce the costs associated with these activities. Indeed, by inspecting a few representative images for each root cause clusters, instead of the whole set of failure-inducing images, engineers can save substantial effort; further, since clusters group similar images together and each cluster is considered, it is less likely for engineers to overlook characteristics and associated causes appearing in a small subset of the images. Also, Step 2 is required only for functional safety analysis (e.g., to determine the unsafe cases to be discussed in safety analysis documents); it is not necessary if engineers aim at automated DNN improvement only.

The effort required for the acquisition of the improvement set is the same for both HUDD, SAFE, and common practice (e.g., purchase an additional stock of real-world pictures); however, HUDD and SAFE require only the unsafe images to be labelled, thus reducing development costs. Finally, by relying on feature-extraction rather than heatmaps, SAFE eliminates the effort required to integrate LRP-based heatmap generation into the DNN under analysis, which is required by HUDD.

Figure 5.2 depicts the stages composing Step 1. We provide further details about each stage in the following subsections. Steps 2 and 4 are described last. Steps 3 and 5 are not further described because they are standard practice.

5.2.1 Step 1: Generation of Root-Cause Clusters (RCCs)

Step 1.1: Data Preprocessing

This step aims to downsample the image sizes to the size required by the transfer learning models. Since we rely on a VGG-16 model, which requires an input size of 224×224 pixels, the images have to be resampled to match this requirement. To downsample the size of each image we rely on the Python

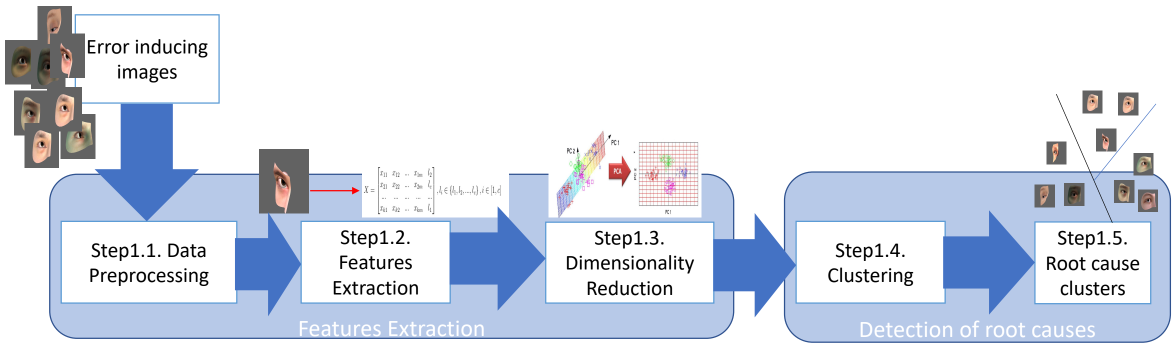

Error inducing images
Step1.1. Data Preprocessing
Step1.2. Features Extraction
Step1.3. Dimensionality Reduction
Step1.4. Clustering
Step1.5. Root cause clusters
Features Extraction
Detection of root causes

Figure 5.2: Generation of root cause clusters with SAFE

Numpy reshape function[1], which changes the shape of an array so that it has a new, compatible shape[2]. In our experiments, we downsampled images from (640×480) to (224×224). In this chapter, we exploited the configurations D and E, which are composed of 16 and 19 layers (see Section 2.1.3), respectively.

Step 1.2: Feature Extraction

Feature extraction aims to transform unstructured data into structured data for their exploitation by clustering algorithms [133].

To maximize the accuracy of DNNs in a cost-effective way, engineers often rely on the transfer learning approach, which consists of transferring knowledge from a generic domain, usually ImageNet [134], to another specific domain, (e.g., Safety Analysis, in our case). In other terms, we try to exploit what has been learned in one task and improve generalization in another task. Researchers have demonstrated the efficiency of transfer learning from ImageNet to other domains [135]. The hierarchical nature of convolutional neural networks (CNNs) encouraged the computer vision community to use this technique with distant datasets due to the similarities between the features extracted by the first CNN layers. Transfer learning saves training time, gives better performance in most cases, and reduces the need for a large dataset.

Transfer learning-based *Feature Extraction* is an efficient method to transform unstructured data into structured raw data to be exploited by any machine learning algorithm. In this method, the features are extracted based on a pre-trained CNN model [136].

Step 1.3: Dimensionality Reduction

Dimensionality reduction aims at approximating data in high-dimensional vector spaces [137].

This can be achieved using projections on hyperplanes. These methods, referred to as linear dimensionality reduction, include the well-known Principal Component Analysis (PCA) [45, 47] (see Section 2.1.3.

[1] https://sparrow.dev/numpy-reshape/

[2] The reshape function just changes the shape of the array containing the data (not the image itself) to match the VGG requirement. The new shape must include the same total number of elements as the original shape.

Step 1.4: Clustering

Clustering is an unsupervised learning technique that divides a set of objects into clusters: (a) objects in the same cluster must be as similar as possible, (b) objects in different clusters must be as different as possible.

Since the root cause clusters of images may have any shape and their number K cannot be determined a priori, we rely on an automatic K-determination clustering algorithm that can find clusters of arbitrary shapes. We use DBSCAN (density-based spatial clustering of applications with noise) [40], which is the most used density-based clustering algorithm [132, 41]. Intuitively, regions with high density are considered clusters and points with the most neighbors are referred to as the "core" of the clusters. The points with fewer neighbors are considered noise. The DBSCAN algorithm relies on two concepts: Reachability and Connectivity.

- Reachability: A point is reachable from another if the distance between them is inferior to a threshold ϵ.
- Connectivity: If two points p and q are connected (i.e., p is in the neighborhood of q based on ϵ) they belong to the same cluster.

DBSCAN introduces two parameters to apply these concepts: $MinPts$ and ϵ.

- $MinPts$: The minimum number of points that a region (a hypersphere with a diameter equal to ϵ) should have to be considered dense.
- Threshold ϵ: A threshold to determine if a point belongs to another point's neighborhood.

SAFE uses a common technique to choose optimal values of ϵ and $MinPts$. To select ϵ, SAFE relies on the elbow method [138]. For this step, unlike that to find an optimal $MinPts$, SAFE does not require the execution of DBSCAN.

More specifically, the optimal value for ϵ is selected as follows:

- First, we calculate the euclidean distance from each point to its closest neighbor.
- Then, we compute the average distance of every point to its closest neighbor and plot these distances in ascending order.
- Finally, we find the plot's elbow point [139], which is a point where there is a sharp change in the distance plot, which serves as a threshold. This point corresponds to the optimal ϵ value.

To find the optimal value for $MinPts$, we run DBSCAN with ϵ equal to the optimal value found above, and with different values for $MinPts$. We then select the clustering configuration with the highest Silhouette Index value [140]. The Silhouette index computes the compactness and the separateness of clusters. For a data point x_i assigned to cluster C_i, the Silhouette index is calculated as follow:

$$SI(i) = \frac{(b(i) - a(i))}{Max(b(i) - a(i))} \tag{5.1}$$

where $a(i)$ is the average distance between x_i and all the data points assigned to cluster C_i. $b(i)$ is the minimum average distance between x_i and the data points assigned to one of the other clusters C_j where $j = 1, .., K; j \neq i$. Based on the concepts described above, DBSCAN defines three types of points:

- Core point: It has at least $MinPts$ points within a distance of ϵ.
- Border point: It is not a core point, but it belongs to at least one cluster. That means that it lies within a distance ϵ from a core point.
- Noise point: It is a point that is neither a core point nor a border point.

The DBSCAN algorithm proceeds by sampling points randomly from the dataset until all points are selected. For each point, it determines if it is a Core, Border or Noise point based on the parameters ϵ and $MinPts$. The core points are considered representative of clusters. The clusters are then expanded by recursively repeating the neighborhood calculation for each point within the region. In DBSCAN, randomness affects only the selection of the point to be treated next; however, since each point ends up being assigned to the cluster containing the closest core point, randomness affects results only when border points are reachable from more than one cluster, which is unlikely [141]. If we run the algorithm several times with the same parameter settings, the same clusters will likely be obtained each time.

DBSCAN also helps identifying rare cases in the set of failure-inducing images. Indeed, if there are enough rare cases to form a cluster (i.e., there are at least *MinPts* data points within a hypersphere with a diameter equal to ϵ), they are grouped together. If rare cases cannot form a cluster, most of them are considered noise and excluded from the set of clusters returned to the end-user. Since we select *MinPts* and ϵ based on the analysis of the distribution of the failure-inducing images, rare cases are considered noise only if they occur in regions of low density (i.e., the region contain less images than $MinPts$). Figure 5.3 shows an example of a cluster containing images representing a rare case (the eye is in an unusual position). In this case, there are ten failure-inducing images with such characteristic. When $MinPts <= 10$, then all the ten images form a cluster. However, if $MinPts > 10$, the images will be considered noise. In practice, $MinPts$ is unlikely to be above 10 because its value is automatically derived considering all the dense regions, including the area with these rare cases. Further, if rare cases that are considered noise share similarities with images in other clusters, they will be assigned to the cluster with the most similar features. However, this case is unlikely to happen since such rare cases appear in sparse areas while clusters only appear in dense areas. In practice, a rare case assigned to a cluster might actually share the same root cause of the other images belonging to the cluster.

5.2.2 Step 2: Functional Safety Analysis

To analyze the clusters, safety engineers can use the same approach as in HUDD (see Chapter 3). For each cluster, a subset of elements is visually inspected to determine the associated unsafe conditions, as functional safety analysis requires. Similarities among images within each cluster may suggest the cause of failures of the DNN. In other words, engineers attempt to identify the root cause of each cluster to gain a better understanding of the DNN behavior.

Figure 5.4 shows examples of root cause clusters identified by SAFE for the Head Pose Detection case study subject considered in our empirical evaluation (see Section 5.3.2). We notice that in Cluster 1 the hidden eye seems to confuse the DNN and is a plausible reason for failure. The same observation can be made for Cluster 2, where, because the head is turned to the right, the right eye is not visible. Clusters 3 and 4 identify common causes of failure due to an incomplete training set, i.e., the absence of images with a head pose close to a borderline.

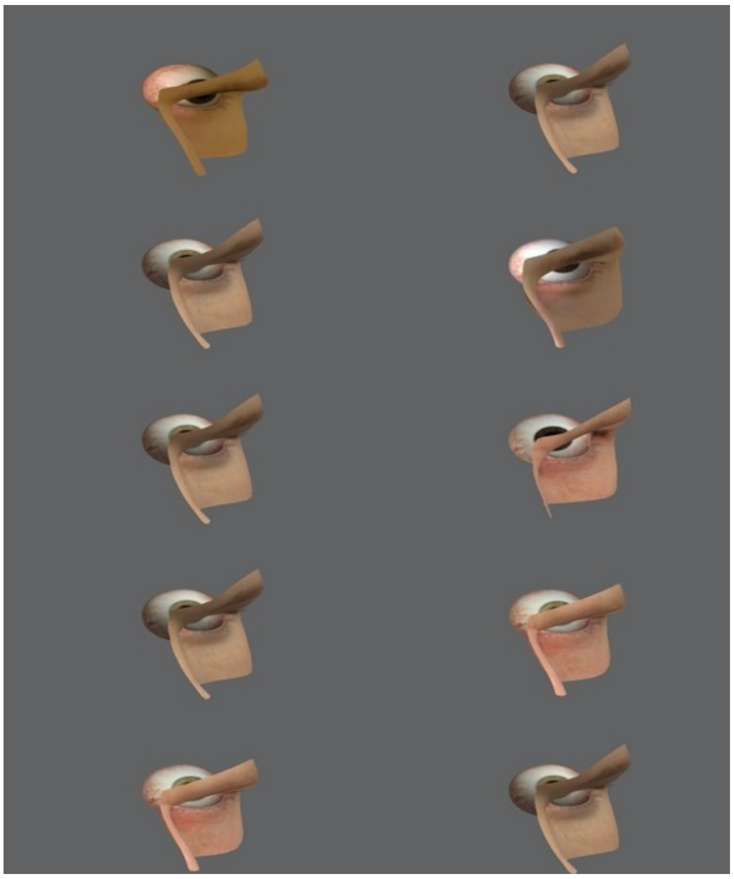


Figure 5.3: Example of a root cause cluster with rare cases

In general, SAFE generates clusters that differ for at least one common characteristic in the images. For example, the root causes presented by Cluster 1 and Cluster 2 are not the same. Indeed, Cluster 1 concerns the right eye while Cluster 2 concerns the the left eye. Engineers might be interested in knowing if the training set is lacking images with only the left eye being hidden or both; for this reason, we believe that separating such clusters is beneficial. Empirical results are reported in Section 5.3.2.

Similar to HUDD, SAFE can identify different root causes of failures, including (1) borderline cases (e.g., the gaze and head pose angle detected by Cluster 1 in Figure 5.4), (2) an incomplete training set (e.g., Clusters 3 and 4 in Figure 5.4), (3) an incomplete definition of the predicted classes (e.g., Cluster 1 in Figure 5.5 shows a cluster generated from our GD case study in Section 5.3.2, with eyes looking middle center, a class missing from our configuration) and (4) limitations in our capacity to control the simulator (e.g., unlikely face positions detected by Cluster 2 in Figure 5.5). In general, SAFE identifies commonalities (i.e., root causes) across images leading to failures. Based on a recent taxonomy [108], the cases described above concern training data quality; in general, SAFE can help engineers to discover any fault that affects the correctness of the DNN output (e.g., a missing model layer). However, we do not integrate mechanisms to automatically determine the underlying cause for the observed failures. In general, we suggest engineers to first inspect root cause clusters to determine major pitfalls (e.g., missing output class) then proceed with automated retraining (i.e., Steps 3-6); if the DNN accuracy is not sufficiently improved then it is necessary to modify the DNN (e.g., add layers or change architecture).

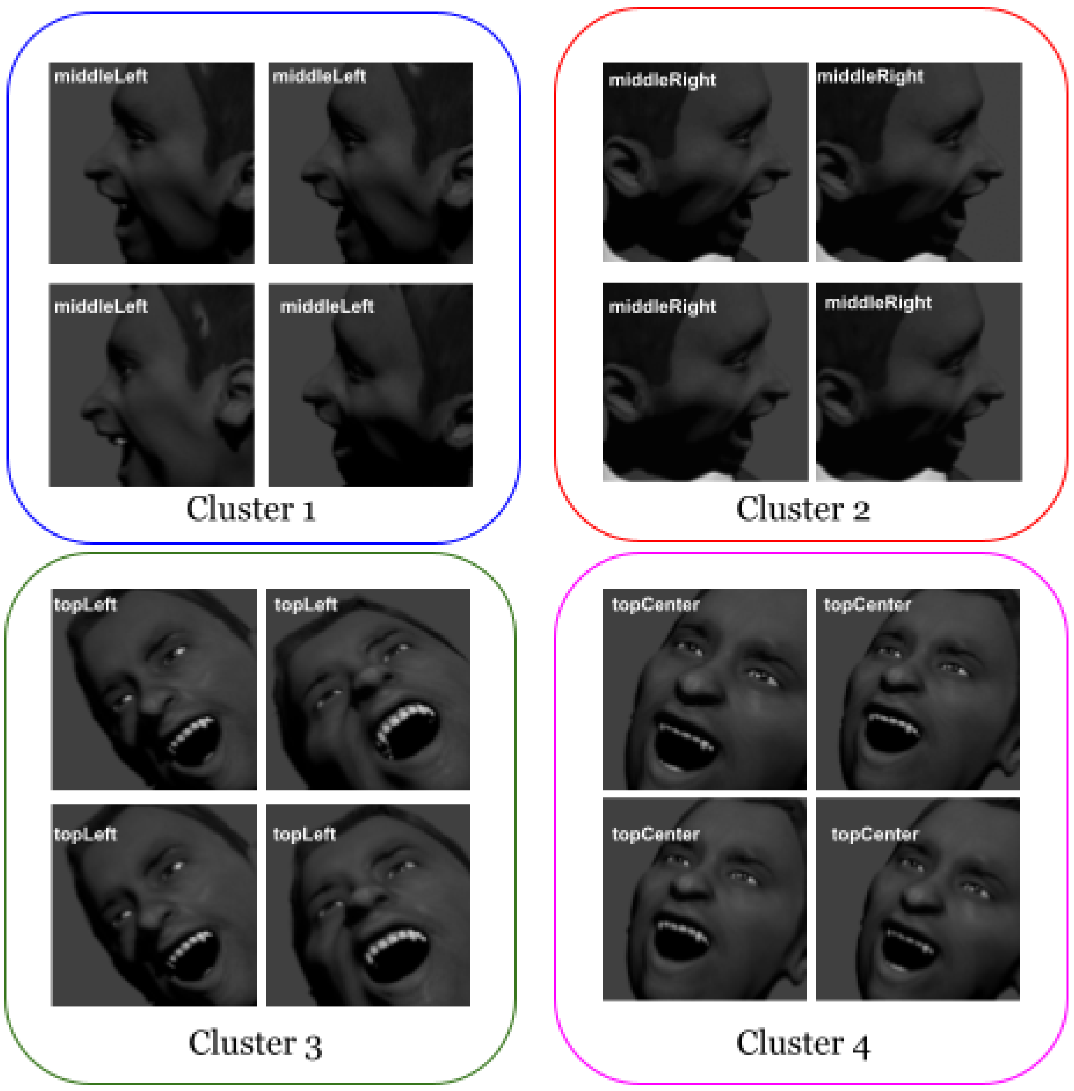


Figure 5.4: Examples of root cause clusters for Head Pose detection

5.2.3 Step 4: Identification of Unsafe Images

Since the manual labeling of images is expensive, it is necessary to automatically identify an unsafe set of reduced size containing images that can improve the DNN accuracy to achieve a cost-effective retraining process. An unsafe set is a subset of unlabelled images from the improvement set, to be labeled and used for retraining the model. Unlike HUDD, SAFE relies on a new clustering-based selection method to automatically select this unsafe set. We identify images from the improvement set that belong to a root cause cluster and select images that, within the same cluster, are representative of the different types of images in the cluster (e.g., different face shapes with a gaze angle that confuses a gaze classifier). Therefore, we assume that the selected images are representative of a cluster in the sense that they contain sufficient information to replace the other cluster members [142] and can be effectively used for retraining.

The key difference between SAFE and HUDD, in this step, is that the latter relies on a Hierarchical Clustering Algorithm. This method only finds spherical clusters. In addition, HUDD depends on the cluster medoids and the cluster ratio to determine if improvement set images belong to the root cause clusters. Unfortunately, such a method might be suboptimal if clusters are not spherical. With DBSCAN,

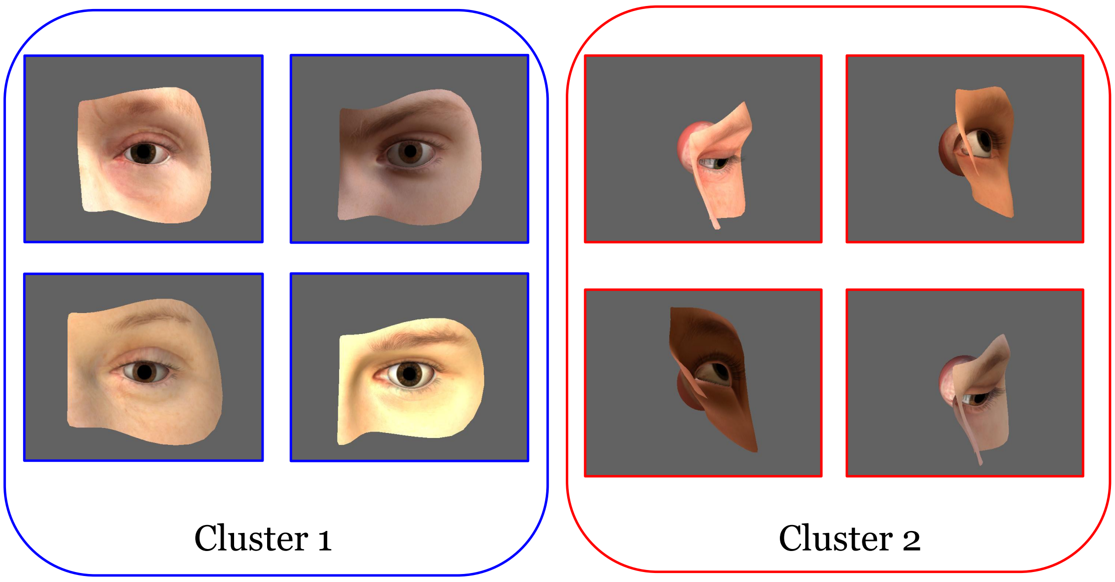


Figure 5.5: Examples of root cause clusters for Gaze detection

the identification of core points enables the determination of representative cluster members even in non-spherical clusters. It does not rely on cluster centroids but core points in each cluster, assuming that, taken together, these points represent the entire cluster. A point close to any core point will be assigned to the cluster represented by this core point.

In SAFE, the selection is performed according to Algorithm 1, which is detailed below. After selecting the unsafe set, we follow the same retraining process as HUDD. We retrain the DNN model by relying on a training set consisting of the union of the original training set and the manually labeled unsafe set. The original training set is reused to prevent reducing the accuracy of the DNN for parts of the input space that are safe. The retraining process is thus expected to lead to an improved DNN model.

Algorithm 1 SAFE Unsafe Set Selection Algorithm

```
    Input: improvement set images $\mathcal{X}$, core points detected for each cluster, a set of clusters $\mathcal{C}$.
    Output: unsafe set
1: for $x$ in $\mathcal{X}$ do
2:     Assign $x$ to the cluster corresponding to the closest core point.
3: Compute $N$, the number of selected images for the unsafe set based on Equation 5.2.
4: Compute the number of selected images $r_i$ from each cluster $c_i$, computed as $N$ over the proportion
   of images in the cluster.
5: for $c_i$ in $\mathcal{K}$ do
6:     Sort the images in ascending distance to their respective closest core point.
7:     From the sorted images, take the $r_i$ first images and add them to the unsafe set.
```

To minimize the retraining effort, and most particularly that related to labeling, we look for representatives images in each cluster. For that, we rely on the core points (see Section 5.2.1). The choice of the core points is motivated by the fact that they represent better the shape of a particular cluster than a centroid. As described in Section 5.2.1, a root cause cluster is typically represented by several core points. The border points that define the cluster's shape are localized around the core points. We assume that the points close to the core points contain enough information to replace the other border points. These points

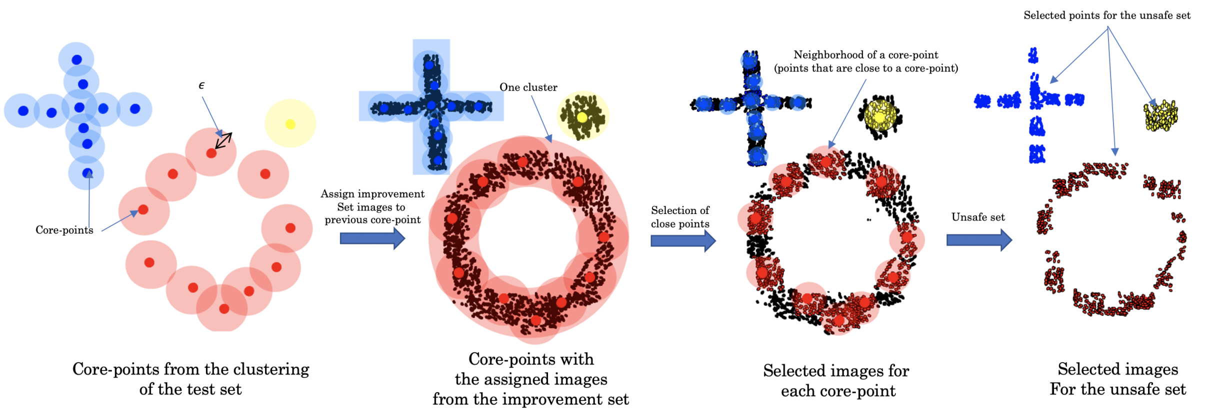

Selected points for the unsafe set
Neighborhood of a core-point (points that are close to a core-point)
One cluster
ϵ
Core-points
Assign improvement Set images to previous core-point
Selection of close points
Unsafe set
Core-points from the clustering of the test set
Core-points with the assigned images from the improvement set
Selected images for each core-point
Selected images For the unsafe set

Figure 5.6: The selection of Unsafe Set

usually take approximately the same shape as the cluster [132]. Recall that core points, unlike centroids, can represent a non-convex cluster with arbitrary shapes.

Algorithm 1 show the steps for the selection of the unsafe set for retraining. The algorithm requires the root cause clusters and their core points. It also requires an improvement set.

The algorithm starts by assigning every image in the improvement set to the closest core point in terms of euclidean distance (line 1, Algorithm 1).

In lines 2 and 3, SAFE computes N, the number of images to be selected for the unsafe set and the number of images to be selected from each root cause cluster i, denoted as r_i, where $r_i = N \times \frac{C_i}{C}$. C is the number of images in the improvement set and C_i is the number of such images assigned to cluster i. Unsafe set images across root cause clusters therefore preserve the distribution of the improvement set across such clusters.

As in HUDD, we assume that the distribution of failure-inducing images across clusters is similar in the improvement and test sets. We determine the number of images N selected from the improvement set to include in the unsafe set as follows:

$$N = (|TestSet| \times sf) \times (1 - TestSetAcc) \tag{5.2}$$

$|TestSet|$ is the size of the test set, while sf is a selection factor in the range $[0 - 1]$ (we use 0.3 in our experiments, same as HUDD to make the comparison fair). $TestSetAcc$ represent the accuracy of the original model on the test set of the case study subject. The term $(1 - TestSetAcc)$ indicates the proportion of failure-inducing images that are observed in the improvement set. $(|TestSet| \times sf) \times (1 - TestSetAcc)$ estimates the number of failure-inducing images that should be selected from the improvement set. The term $(|TestSet| \times sf)$ provides an upper bound for the unsafe set size as a proportion of the test set size.

In line 4, the images are sorted according to ascending distance to their respective closest core point. Finally, in line 5, for each cluster i, we select the first r_i sorted images to include in the unsafe set.

We also illustrate the algorithm steps in Figure 5.6. The first image represents the core points obtained from the clustering of the failure-inducing images (the dots represent the core points). In the second step, we assign the images in the improvement set to their closest core point, respectively. Then, we select a subset of points from the neighborhood of each core point based on Algorithm 1. The last image represents the selected unsafe set.

Our algorithm excludes from the unsafe set the images leading to DNN failures due to root causes not observed in the test set; indeed, such images will be distant from clusters' core points. Furthermore, such images will not help improve the DNN performance on the test set, which is our objective since the test set is assumed to be representative of real-world scenarios. Finally, when the improvement set does not include any image belonging to a root cause cluster then SAFE does not assign any image to the cluster; this is not the case for HUDD which, different from SAFE, will not prevent engineers from labeling useless images.

5.2.4 SAFE Running Example

This section presents an example of SAFE usage. It is based on the headpose detection DNN (HPD) considered in our empirical evaluation. HPD receives as input a picture taken from a camera positioned inside a car; the picture is automatically cropped to a size of 640×640 pixels.

To reduce development costs, HPD has been trained and tested using images generated by a simulator capable of generating pictures of human heads. The training and test sets consists of 16013 and 2825 images, respectively, both generated by randomly selecting simulator parameters' values. The training and the test set could also have included real-world images. After 30 epochs, we obtained an accuracy of 88.03%. From the test set, 1580 images were misclassified, they represent the failure-inducing images that should be investigated to determine the root causes of failures.

We implemented the SAFE pipeline as a Jupyter Notebook[3].

The SAFE pipeline starts by pre-processing the failure-inducing images to match our model's input requirement (SAFE Step 1.1, in Figure 5.2). It automatically converts the images into a NumPy[4] array and downsample them as explained in Section 5.2.1.

After preprocessing, the images are ready for feature extraction (SAFE Step 1.2, in Figure 5.2). The SAFE pipeline automatically extracts the features by relying on a pre-trained VGG model loaded by the Notebook. Precisely, for each image, SAFE stores the output of the second-last fully connected layer of the VGG model, which leads to an array of 512 features for each image.

The pipeline continues by applying the PCA dimensionality reduction method to reduce the number of features from 512 to 256 (SAFE Step 1.3, in Figure 5.2). The output of the PCA method is an 1580×256 array, where 1508 is the number of failure-inducing images and 256 is the number of features. The number of features (i.e., 256) has been empirically determined in preliminary experiments (see Section 5.2.1) and is not supposed to be further modified by end-users. This array is passed to DBSCAN as an input. We use DBSCAN from the SciKitLearn library[5].

Before performing clustering (SAFE Step 1.4), we first need to choose the optimal parameter settings. We apply the method explained in Section 5.2.2 to obtain $\epsilon = 0.9$ and $minPts = 9$. Using these parameters, we run our algorithm to generate the final root cause clusters, which are 20 in this case. The next step of the pipeline (i.e., SAFE Step 2) includes a procedure that, from the clusters generated by DBSCAN, generates several folders, each containing the images belonging to the cluster. It also generates an animated gif image (similar to a video) with the images belonging to each cluster, to help the user visualize it. The end-user is then expected to visualize a portion of the images appearing in each cluster (e.g., five according to our experimental results in Section 5.3.2) to perform the functional safety analysis step of SAFE (i.e., Step 2 in Figure 5.1). Example images are reported in Figure 5.4..

[3]http://jupyter-notebook.readthedocs.io
[4]http://numpy.org
[5]http://scikit-learn.org

Next, the end-user should provide an improvement set (SAFE Step 3). For our experiments with HPD, we rely on an improvement set of $4,103$ images generated with additional executions of the simulator. We could have also included real-world images collected by our industry partner in the field but they might have prevented replicability (e.g., they cannot be publicly shared because of privacy agreements).

In our execution, SAFE selected 154 images as an unsafe set for retraining. The number of selected images is computed automatically based on the selection factor (sf) configured by the end-user (see Section 5.2.3) These images need to be labelled by the end-user (SAFE Step 5) but for our case study subject, labels are automatically derived from simulator parameters. In case the improvement set includes real-world images, labelling is performed manually.

After obtaining the unsafe set, the end-user simply combines it with the training set and retrains the model according to the specific procedure of the DNN under analysis.

5.3 Evaluation

In this section, we aim to evaluate our approach. SAFE is expected to perform better than HUDD in terms of the quality of the clustering and DNN accuracy after retraining. We choose HUDD for comparison not only because SAFE aims to improve over it but because they are the only approaches in the literature that aim to support safety analysis, which is achieved through the identification of root cause clusters and the selection of images for retraining based on these clusters. To investigate whether if such expectations hold and SAFE is useful, we compare these two approaches following the experimental procedures described below addressing the following research questions.

RQ1 *Does SAFE enable engineers to identify the root causes of DNN failures?*

The clusters produced by SAFE should provide useful information to identify plausible causes of DNN failures in a form that is amenable to practical analysis.

RQ2 *Does SAFE enable engineers to more effectively and efficiently retrain a DNN when compared with HUDD and baselines approaches?* We expect SAFE to lead to a higher model accuracy after retraining

RQ3 *Does SAFE provides time and memory savings compared to HUDD?* SAFE's black-box nature should provide significant time and memory savings, compared to HUDD, which is a white-box approach.

To perform our empirical evaluation, we have implemented SAFE as a toolset that relies on the PyTorch [98] and SciPy [99] libraries. Our toolset, case study subjects, and results are available for download [25]. In our experiments, steps 1 to 5 were carried out on an Intel Core i9 processor running macOS with 32 GB RAM. Step 6 (retraining) was conducted on the HPC facilities of the University of Luxembourg (see http://hpc.uni.lu). We relied on a Dual Intel Xeon Skylake CPU (28 cores) and 128 GB of RAM.

5.3.1 Subject Systems

We rely on images generated using simulators as it allows us to associate each generated image to values of the simulator's configuration parameters. These parameters capture information about the characteristics of the elements in the image and can thus be used to objectively identify the likely root causes of DNN failures. Such simulators are increasingly common, and of higher fidelity in many domains [143], including automotive and aerospace.

We consider the same DNNs as the HUDD approach (see Section 3.3.1), which support gaze detection, drowsiness detection, headpose detection, and face landmarks detection systems under development at IEE Sensing, our industry partner.

Since DNN failures and, consequently, clustering results, depend on the initial training of the DNN under analysis, to deal with such randomness we repeated the initial training ten times for the case study subjects GD, OC, and HPD; each training execution relies on a different split of the training and the test data sets. Unfortunately, we could not repeat the training for the FLD DNN because it was provided by our industry partner along with the failure-inducing images. Further, we could not repeat the execution of HUDD ten times for each case study DNN because of the large amount of time required to compute distance matrices based on heatmaps (see Section 3.3.2). However, to discuss the statistical significance of the differences between HUDD and SAFE, for each of the metrics selected to address our research questions, we relied on a one-sample Wilcoxon signed rank test. The one-sample Wilcoxon signed rank test is a non-parametric statistical hypothesis test used to determine whether the median of a population (here, the SAFE median) is greater than a reference value (here, the HUDD result). It enables us to test the null hypothesis: *the SAFE median is equal to the HUDD result*.

Finally, to determine the number of components to be selected by PCA, which is an input parameter for SAFE, we conducted a set of experiments. Precisely, we considered a number of features between two and 256, considering all powers of two; then we applied DBSCAN and measured the quality of its result based on the Silhouette Index [140]. We performed the analysis on all the case studies and concluded that 256 is the number of features that provides the best results.

5.3.2 Measurements and Results

RQ1. *Does SAFE enable engineers to identify the root causes of DNN failures?*

We refine RQ1 into five complementary subquestions (RQ1.1, RQ1.2, RQ1.3, RQ1.4, and RQ1.5), which are described in the following, along with their corresponding experiment design and results.

RQ1.1: Is the number of generated clusters small enough for enabling visual inspection? *Design and measurements.*

Though this is to some extent subjective and context-dependent, we discuss whether SAFE finds a number of root cause clusters that is amenable to inspection by experts. To respond to this research question, we assume that experts visually inspect five images per root cause cluster to be able to make a decision. This assumption is supported by an experiment we conducted, as presented in later in this Section (RQ1.5). Under this assumption, we compare SAFE and HUDD in terms of the number of generated clusters and based on the ratio of failure-inducing images that should be visually inspected when relying on each method. This ratio is calculated as follows:

$$ratio = \frac{(k \times 5) \times 100}{n} \tag{5.3}$$

where k is the number of root cause clusters, and n is the number of failure-inducing images.

Experiment Results.

Table 5.1: Root cause clusters generated by SAFE.

Subject	SAFE					HUDD		
	# error inducing images	# of clusters		% of inspected images		# error inducing images	# of clusters	% of inspected images
	min/max/med	min/max/med	p-value	min/max/med	p-value			
GD	4967/6290/5602	14/31/25	0.004	1.41/2.46/2.24	0.004	5371	16	1.49
OC	409/557/492	21/33/26	0.002	25.67/29.62/26.15	0.002	506	14	13.83
HPD	1371/2089/1519	20/30/24	0.002	7.29/7.18/7.99	0.002	1580	17	5.38
FLD	1554	64	/	20.5	/	1554	71	22.84
OD	758/933/822	2/2/2	0.002	1.07/1.32/1.22	0.004	838	14	8.35
TS	2239/2698/2450	7/12/9	0.004	1.45/2.63/1.93	0.004	2317	20	4.31

Table 5.1 shows, for each case study subject, the total number of failure-inducing images belonging to the test set, the number of root cause clusters generated by SAFE and HUDD, and the ratio of failure-inducing images that should be visually inspected when using SAFE or HUDD.

For the respective DNNs, SAFE identifies 25 (GD), 26 (OC), 24 (HPD), 64 (FLD), 2 (OD), 9 (TS) root cause clusters (for GD, OC, HPD, OD, and TS we refer to median of the ten runs executed). In contrast, HUDD identifies 16 (GD), 14 (OC), 17 (HPD), 71 (FLD), 14 (OD), and 20 (TS) root cause clusters.

We notice that in 50% of the case study subjects, SAFE yields a lower number of clusters. This experiment shows both SAFE and HUDD find an acceptable number of clusters for visual inspection. Indeed, the ratios of failure-inducing images to be inspected are low (between 1.07 and 26.15, with a median of 5.12). These results suggest that using SAFE can save significant effort compared to the manual inspection of the entire set of images.

RQ1.2: Does SAFE generate root cause clusters with a significant reduction in variance for simulator parameters?

Design and measurements.

This research question investigates if SAFE achieves high within-cluster similarity concerning at least one simulator parameter. Indeed, since we rely on DNNs that are trained and tested with simulators, images assigned to the same cluster should present similar values for a subset of the simulator parameters. Within each cluster, the variance of these parameters should be significantly smaller than that computed on the entire test set.

Similarly to HUDD, for a cluster C_i, the rate of reduction in variance for a parameter p can be computed as in Equation 3.8. Table 3.3 provides the list of parameters considered in our evaluation.

Note that simulator parameters are only used to objectively evaluate the approach; they are not involved in the practical application of the approach. Section 5.3.2 addresses the application of SAFE to case study subjects for which a simulator is not available (i.e., TS and OD).

Since the number of parameters that capture common failure causes is not known a priori, we consider a significant variance reduction in at least one parameter to be enough for the cluster to be indicative of root causes. Therefore, we compute the percentage of clusters showing such a variance reduction for at least one of the parameters.

Consistent with HUDD, we compute the percentage of clusters with a variance reduction between 0% and 90%, with incremental steps of 10%. To answer our research question positively, a high percentage of the clusters should reduce variance for at least one of the parameters. We compare our results to those of

HUDD.

Experiment Results.

We report in Table 5.2 the maximum, minimum, and median percentage of clusters having at least one parameter showing a reduction rate above a threshold in the range $[0\% - 90\%]$, compared to the percentage obtained by HUDD. We also report the p-values resulting from performing a one-sample Wilcoxon signed rank test (see Section 5.3.1). We notice that, when the median obtained with SAFE is higher than the value obtained with HUDD (i.e., SAFE performs better), the p-values are always below 0.05. This implies that, in these cases, the median percentage of clusters with a reduced variance obtained with SAFE is significantly larger than the one obtained with HUDD.

Table 5.2: Minimum, Maximum and Median percentage of clusters with at least one parameter showing a reduction rate above a threshold in the range [10% - 90%], with the percentage obtained by HUDD and the p-value when comparing SAFE to HUDD.

Threshold		**GD**	**OC**	**HPD**	**FLD**
90%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	85%/100%/90%	90%
	HUDD	93%	29%	82%	85%
	p-value	0.001	0.001	0.004	-
80%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	85%/100%/92%	94%
	HUDD	93%	43%	100%	90%
	p-value	0.001	0.001	0.99	-
70%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	87%/100%/94%	97%
	HUDD	93%	50%	100%	94%
	p-value	0.001	0.001	0.99	-
60%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	99%
	HUDD	100%	57%	100%	94%
	p-value	1.0	0.001	1.0	-
50%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	100%
	HUDD	100%	57%	100%	96%
	p-value	1.0	0.001	1.0	-
40%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	100%
	HUDD	100%	64%	100%	99%
	p-value	1.0	0.001	1.0	-
30%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	100%
	HUDD	100%	71%	100%	99%
	p-value	1.0	0.001	1.0	-
20%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	100%
	HUDD	100%	79%	100%	100%
	p-value	1.0	0.001	1.0	-
10%	**SAFE** (min/max/median)	100%/100%/100%	100%/100%/100%	100%/100%/100%	100%
	HUDD	100%	100%	100%	100%
	p-value	1.0	1.0	1.0	-

To enable visual comparison, Figure 5.7 plots the median percentage of clusters with variance reduction for at least one of the simulator parameters, with different reduction rates, for both SAFE and HUDD.

Results show that SAFE yields a higher percentage for three out of the four case study subjects (i.e., GD, OC, FLD). For HPD, based on the p-values reported in Table 5.2, the differences between HUDD and SAFE are not significant (i.e., HUDD does not perform better than SAFE).

We report in Table 5.3 the minimum, maximum and median number of core points found by the clustering algorithm for each case study subject and the ratio of failure-inducing images being considered as core points. We recall that a cluster consists of several core points and that the core points determine the cluster shape. The ratio of failure-inducing images being considered as core points is therefore an

Table 5.3: Number of core-points and number of clusters for each case study subject

Case Study Subject	**# of core points** (min/max/median)	**# of clusters** (min/max/median)	**% of images considered as core points** (min/max/median)
GD	2389/5808/5080	14/31/26	46%/92%/88%
OC	366/495/441	21/33/26	89%/97%/90%
HPD	259/623/316	20/30/24	18%/30%/22%
FLD	622	64	40%

indicator of the complexity of the cluster shape. The larger this ratio, the more complex the cluster shape and the less likely it is to be convex.

Since non-convex clusters cannot be properly modeled by a centroid-based algorithm or even a hierarchical clustering algorithm (see Section 5.2), this partly explains the results presented in Figure 5.7.

For GD and FLD, SAFE performs slightly better than HUDD. With GD, out of ten executions, SAFE yields a median of 100% of the clusters with a variance reduction above 90% compared to 93% for HUDD. As for FLD, it obtained 90% of the clusters with variance reduction above 90%, compared to 85% for HUDD. These values are close because the number of clusters detected by both methods is similar for GD and FLD. Detecting the optimal number of clusters is crucial as it leads to root cause clusters grouping very similar images with less noise. Consequently, identifying the right root cause clusters will result in higher variance reduction. Nevertheless, the slight superiority shown by SAFE is explained by the fact that the latter finds root cause clusters with arbitrary shapes, compared to convex shapes found by HUDD. This is important since arbitrary-shaped clusters can find more homogeneous clusters (i.e., clusters with higher within-cluster similarity) with very similar images. In contrast, a convex cluster tends to be less dense and can group rather dissimilar images.

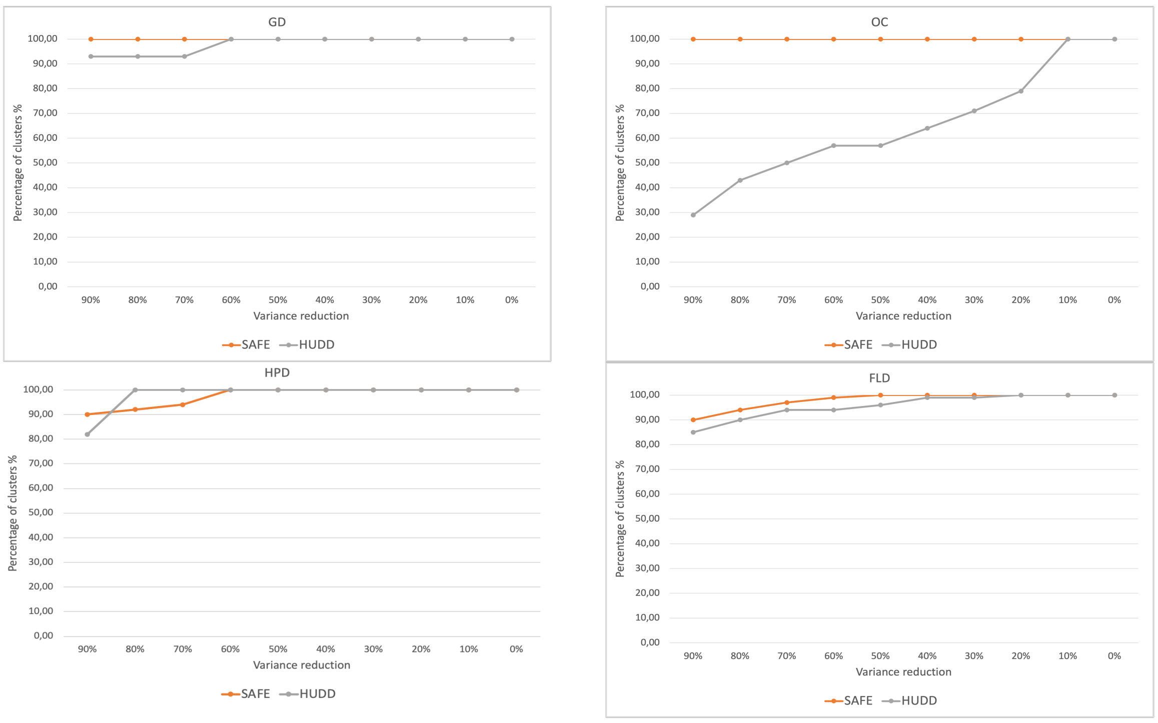


Figure 5.7: RQ1.2: median percentage of clusters with at least one parameter showing a reduction rate above a threshold in the range $[0\% - 90\%]$.

In the case of OC, the percentage of clusters with a given variance reduction obtained by SAFE is much higher than that obtained by HUDD. SAFE yielded 100% of the clusters (median of ten executions) with variance reduction above 90%, in contrast to 29% for HUDD. This is explained by the fact that SAFE found a much higher number of clusters than HUDD (26 for SAFE compared to 14 for HUDD). Also, 90% of the failure-inducing images are considered core points by SAFE (median), thus indicating complex cluster shapes, which may, in turn, explain the detection of more clusters. A larger number of clusters leads to root cause clusters with a lower number of images (15 images per cluster on average for SAFE with OC), which have higher chances to contain similar images.

For the HPD case study subject, HUDD and SAFE results are very close. SAFE yields 90% of the clusters (median of ten runs) with variance reduction above 90% compared to 82% for HUDD. At the same time, HUDD shows 100% of the clusters with variance reduction above 80%. Both methods yield 100% of the clusters with variance reduction above 60%. We notice that the number of clusters detected by both methods is pretty close (24 for SAFE compared to 17 for HUDD), thus explaining these results. Also, we observe that only 22% of the failure-inducing images are considered core points, hence indicating that clusters do not have complex shapes and are closer to being convex than in the OC case, for example.

In general, when the cluster shapes obtained by SAFE are relatively simple, the results of the two approaches can be expected to be similar.

In contrast, when clusters have arbitrarily complex shapes, there is a clear advantage in using SAFE, as illustrated by the OC results and to a lesser extent the GD results.

In general, in spite of being a black-box approach, SAFE tends to find a high percentage of clusters with at least one reduced parameter. For GD and OC, 100% of the clusters show parameters with a variance reduction above 90%. HPD and FLD yield both 90%. Based on these results, we can positively answer RQ1.2 since all clusters present at least one parameter with a positive, significant reduction rate ($> 50\%$ in Figure 5.7).

RQ1.3: Do parameters with high variance reduction represent a plausible cause for DNN failures?

Design and measurements.

This research question investigates whether SAFE helps engineers understand root causes of an failure.

We assume that DNN failures occur in specific areas of the simulator parameter space. Under this assumption, we identify a set of unsafe parameters and corresponding unsafe values around which a DNN failure is susceptible to occur. Table 3.4 provides the list of unsafe parameters (similarly to what is used by HUDD in Section 3.3.2, along with the unsafe values identified. For example, for the Gaze Angle parameter, unsafe values consist of the boundary values distinguishing labels pertaining to different gaze directions. These unsafe parameters were selected in a systematic manner for all the case study subjects. Precisely, we report as unsafe values all the values used to label different classes (e.g., eyes openness above/below 20 pixels) and values for borderline cases (i.e., cases in which portions of the face are hidden). Also, we have identified additional unsafe parameters that can cause a DNN failure because they lead to masked elements in images (e.g., distance between the pupil center and the iris center, distance between the pupil bottom and the bottom eyelid margin). Note that determining unsafe parameters and values is only required for experimental purposes here, as detailed below, and not for applying SAFE in practice.

Face images generated with such unsafe values may lead to DNN failures because the DNN either cannot distinguish two classes or because part of the human face is not present in the image. Therefore, we expect all the failure-inducing images having such characteristics to belong to an appropriate cluster;

precisely, they should belong to a cluster having (a) high variance reduction for the unsafe parameter and (b) an average value close to the identified unsafe value.

For our experiment, we consider that a root cause cluster is explanatory in terms of root causes if it satisfies two requirements: (1) It should have at least one unsafe parameter with a variance reduction above 50%, (2) the cluster average should be close to one unsafe value. For Gaze Angle, Openness, Headpose-X, and Headpose-Y, an average value is considered close to an unsafe value if the difference between them is below 25% of the length of the subrange including the average value. For DistToCenter, PupilToBottom, and PupilToTop, an average value is considered close to an unsafe value if it is below or equal to it. For the FLD case study subject, since the reason for not detecting a landmark cannot be related to a single simulator parameter but often depends on combinations of parameters (e.g., the position of the head and the illumination angle lead to shadows on the face), it is impossible to determine unsafe values and therefore such a set of explanatory parameters; as a result, FLD is omitted from this experiment.

Based on the above, we address this research question by computing the percentage of clusters that are explanatory according to our definition. The higher this percentage, the more evidence we have that clustering is useful for identifying causes of DNN failures.

Experiment Results.

Table 5.4 shows the percentage of the root cause clusters that are explanatory for both SAFE and HUDD. Since we repeat the execution SAFE with ten different DNN instances for each case study subject, we report the minimum, maximum and median of the percentages obtained. Across all three case study subjects, SAFE shows a higher percentage of explanatory root cause clusters than HUDD. The median results with GD, OC, and HPD are 86%, 100%, and 90%, respectively, compared to 86%, 57%, and 88%, respectively, with HUDD. Table 5.4 also reports the p-values resulting from performing a one-sample Wilcoxon signed rank test (see Section 5.3.1). The p-values are below 0.05 for OC and HPD, which indicates that the percentage of SAFE's root cause clusters that present at least one explanatory parameter is significantly larger than the one obtained with HUDD for OC and HPD. As for GD, the results are similar.

Table 5.4: Minimum, Maximum and Median percentage of root cause clusters that present at least one explanatory parameter with the percentage obtained by HUDD and the p-value when comparing SAFE to HUDD.

Case Study Subject	**SAFE**			**HUDD**	**p-value**
	Min.	**Max.**	**Median**		
GD	83%	100%	86%	86%	0.99
OC	93%	100%	100%	57%	0.002
HPD	84%	96%	90%	88%	0.02

These results show a large difference in the percentage of clusters that can be explained between SAFE or HUDD for the OC case study subject (43% difference). Indeed, for SAFE, all the clusters have a high reduction in variance, while this is only the case for 57% of the clusters for HUDD. As explained in the previous Section, this can be explained by the fact that OC clusters have a complex shape, more so than in other case study subjects.

As for GD and HPD, the SAFE median is close to the result obtained with HUDD (although for HPD the difference is significant with a significance level of 0.05). For the median, we observe a 2% difference for HPD and no difference for GD. These results confirm the results obtained in RQ1.2. For GD and HPD, both methods show 100% of the clusters with a parameter presenting a variance reduction above 50%. These results are however still slightly in favor of SAFE.

Once again, the above results are explained by the fact that the root cause clusters found by SAFE can take arbitrary shapes. Such clusters, as previously explained and in the general case, have better chances to group similar images than clusters with convex shapes.

RQ1.4: Does SAFE identify more distinct failure root causes than HUDD?

Design and measurements.

This research question investigates if SAFE identifies a larger number of possible causes of failures than HUDD. Specifically, we compare the two approaches in terms of the number of unsafe values being covered by at least one cluster. We say that an unsafe value v is covered by a cluster c, when c presents a parameter p with a high variance reduction and the parameter p has an average value close to the unsafe value v.

Since our simulators generate images having parameter values that are uniformly sampled within the input domain, every unsafe value has the same likelihood of being observed in the test set images. Therefore, ideally, we aim for the root cause clusters to cover all such values.

Experiment Results.

In Table 5.5, we report the minimum, maximum and median percentage of the unsafe values covered by the root cause clusters obtained when applying SAFE to our case study subjects. The clusters generated by SAFE with GD, OC, and HPD cover (median) 92%, 64%, and 80% of the unsafe values, respectively. The clusters generated by HUDD, instead, cover 71%, 50%, and 60% of the unsafe values, respectively. The p-values resulting from performing the one-tailed, one-sample Wilcoxon signed-rank test are always below 0.05, which implies that the median obtained with SAFE is significantly higher than the result obtained with HUDD.

Table 5.5: Minimum, Maximum and Median percentage of the unsafe values covered by the root cause clusters with the p-value when comparing SAFE to HUDD.

Case Study Subject	**SAFE**			**HUDD**	**p-value**
	Min.	**Max.**	**Median**		
GD	85%	100%	92%	71%	0.002
OC	64%	71%	64%	50%	0.002
HPD	60%	80%	80%	60%	0.004

Below, we discuss, more in detail, the differences between SAFE and HUDD. To exemplify our discussion, we report in Table 5.6 (GD, OC) and Table 5.7 (HPD) examples of unsafe values covered by the clusters generated in one of the runs of SAFE and HUDD.

Based on Table 5.5, for GD, SAFE identifies root cause clusters covering 92% unsafe values (median out of ten runs), compared to 71% for HUDD.

This is because SAFE relies on images represented with features extracted from convolutional layers, which provide a better representation than heatmaps and show aspects that are not captured by heatmaps, such as eye shape, edges, and corners.

For OC, SAFE covers a median of 65% unsafe values compared to 50% for HUDD. The uncovered unsafe values concern the parameter *Angle*. However, in the OC case study subject, we mainly focus on eye openness (*Distance* parameter) and the distance between the pupil and eyelid (*StrangeDist Top/Bot* parameter). All of the unsafe values for these two relevant parameters were covered by SAFE and HUDD.

Table 5.6: Coverage of the unsafe values by the root cause clusters (OC and GD case study subjects)

Parameter	**Unsafe value**	**GD**		**OC**	
		SAFE	**HUDD**	**SAFE**	**HUDD**
Angle	337.5	✓	✓	✓	✗
	22.5	✓	✓	✗	✗
	67.5	✓	✓	✗	✗
	112.5	✓	✓	✗	✗
	157.5	✓	✓	✗	✗
	202.5	✓	✓	✓	✗
	247.5	✓	✓	✓	✓
	292,5	✓	✓	✓	✓
Headpose-X	220	✓	✗	✓	✓
	160	✓	✓	✓	✓
Headpose-Y	20	✓	✗	✓	✓
	340	✓	✗	✓	✗
StrangeDist Top/Bot	-14	✓	✗	✓	✓
Distance	25	✓	✓	✓	✓
TOTAL Coverage		**14**	10	**10**	7

Table 5.7: Coverage of the unsafe values by the root cause clusters (HPD case study subject)

Parameter	**Unsafe value**	**HPD**	
		SAFE	**HUDD**
Headpose-X	-28	✓	✓
	-10	✓	✓
	0	✓	✓
	10	✓	✗
	21	✗	✗
HeadPose-Y	-88	✗	✗
	-10	✓	✓
	0	✓	✓
	10	✓	✓
	77	✓	✗
TOTAL Coverage		**8**	6

For the HPD case study subject, SAFE covers a median of 80% unsafe values compared to 60% for HUDD. None of the techniques covers values 21 for *H-Headpose* and -88 for *V-Headpose*. However, such values can be observed if we also consider parameters with a variance below 50% (which is an arbitrary threshold). These two values represent boundary values that we believe confuse SAFE as they correspond to situations where it is hard to see the eyes and the shape of the head. As a result, such images are sometimes clustered erroneously. Thus, these clusters show a lower variance reduction.

In addition, we report that 90% of the clusters cover a unique set of unsafe values (e.g., *Angle*=337.5*,* *H-Headpose*=220*,* *V-Headpose*=340*,* *Distance*=25). Concerning the remaining 10%, we observe that they are still unique but differ with respect to a parameter that is not unsafe. Therefore, we conclude that all the clusters generated by SAFE cover a unique set of parameter values that are useful to determine distinct failure causes.

RQ1.5: How many images are required to identify commonalities in a cluster?

Design and measurements.

We aim to determine how many images from a root cause cluster an engineer should inspect to correctly identify the root cause captured by the cluster. Recall that in our previous analysis (i.e., RQ1.1), we made an assumption regarding this point to estimate cost savings that can be expected from SAFE. To this end we conducted an online questionnaire-based experiment, following best practices [144]. Our population of participants consists of 19 PhD students at the University of Luxembourg and University of Ottawa.

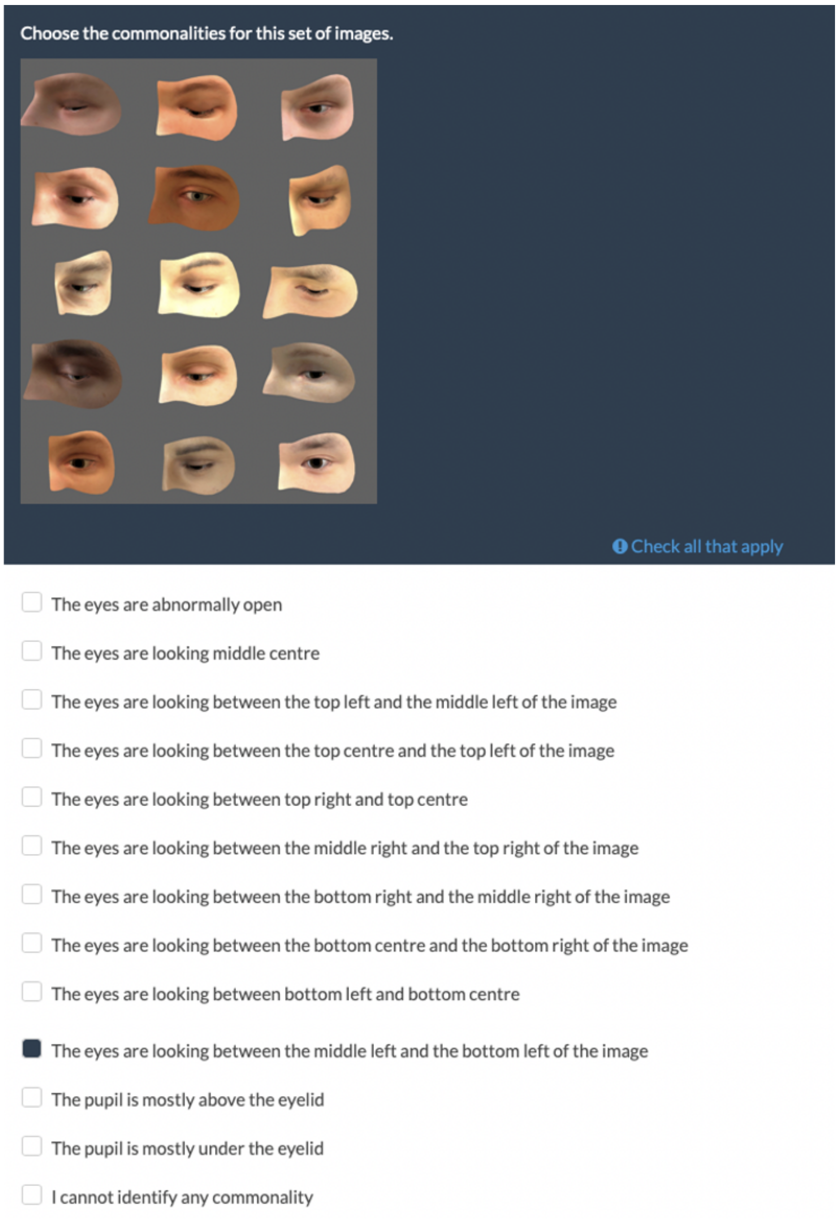
All the selected students hold a master's degree in computer science or a corresponding engineering degree; further, most have acquired fundamentals in machine learning and a few work on safety-related topics. Therefore, all the selected participants are competent to perform the experimental task, which does not require background knowledge, though they are less familiar with the problem domain than engineers would be in practice and therefore can be expected to make more mistakes.

In our experiment, we asked participants to identify the commonalities in a randomly selected subset of images belonging to randomly selected root cause clusters generated by SAFE for our case study subjects. Such task emulates what should be done by an engineer to understand the root causes for an failure. We asked each participant to perform the task three times, on different clusters of images. Each cluster of images included respectively 5, 10, and 15 images randomly selected from a root cause cluster belonging to a different case study subject. Also, for the same subject, each participant received a different random set of images, belonging to a randomly selected cluster. In short, each participant therefore received three questions with 5 images, 10 images, and 15 images, respectively, belonging to different clusters from different case study subjects.

We provided closed-ended questions to objectively compare the obtained answers with the ground truth (i.e., likely root causes for DNN failures). Similarly to RQ1.4, we considered only case study subjects performing classification tasks because, for these cases, the availability of the ground truth makes it possible to determine the reasons for DNN failure objectively.

Table 5.8: The descriptive sentence used in the survey for each unsafe value

Parameter	Descriptive sentence	Unsafe value
Headpose-Y	The face is straight forward	0°
	The face is partially not visible because it is inclined to the top	74.17°
	The face is partially not visible because it is inclined to the bottom	-88.10°
	The face is turned between the centre and the top of the image	10°
	The face is looking between the centre and the bottom of the image	-10°
Headpose-X	The face is partially not visible because it is turned to the left side of the image	-28.88°
	The face is partially not visible because it is turned to the right side of the image	21.35°
	The face is turned between the middle and the left side of the image	-10°
	The face is turned between the middle and the right side of the image	10°
Eye Gaze Angle	The eyes are looking between the top left and the middle left of the image	22.5°
	The eyes are looking between the top centre and the top left of the image	67.5°
	The eyes are looking between top right and top centre	112.5°
	The eyes are looking between the middle right and the top right of the image	157.5°
	The eyes are looking between the bottom right and the middle right of the image	202.5°
	The eyes are looking between the bottom centre and the bottom right of the image	247.5°
	The eyes are looking between bottom left and bottom centre	292.5°
	The eyes are looking between the middle left and the bottom left of the image	337.5°
Openness	The eyes are abnormally open	>64°
DistToCenter	The eyes are looking middle centre	<11.5°
PupilToTop	The pupil is mostly above the eyelid	<-16°
PupilToBottom	The pupil is mostly under the eyelid	>-16°


Choose the commonalities for this set of images.

Check all that apply

- [ ] The eyes are abnormally open
- [ ] The eyes are looking middle centre
- [ ] The eyes are looking between the top left and the middle left of the image
- [ ] The eyes are looking between the top centre and the top left of the image
- [ ] The eyes are looking between top right and top centre
- [ ] The eyes are looking between the middle right and the top right of the image
- [ ] The eyes are looking between the bottom right and the middle right of the image
- [ ] The eyes are looking between the bottom centre and the bottom right of the image
- [ ] The eyes are looking between bottom left and bottom centre
- [✓] The eyes are looking between the middle left and the bottom left of the image
- [ ] The pupil is mostly above the eyelid
- [ ] The pupil is mostly under the eyelid
- [ ] I cannot identify any commonality

Figure 5.8: Example question appearing in our questionnaire.

As commonalities to be identified by the participants, we considered all the unsafe values described in RQ1.4. For each unsafe value, we provided a descriptive sentence (see Table 5.8) to be selected by the participants from a checkbox list. Participants could select more than one option and we provided the option *None of the above*, for participants who did not find the set of provided answers to be satisfactory.

Each questionnaire was introduced by a short description of SAFE and a detailed description of the task to perform. Further, we provided an example answer from a different case that is simple to understand (classification of animal pictures).

Data collection was automated using Lime Survey[6] and its link was sent to the participants by email. Figure 5.8 shows an example question provided to the participants (in this case, the images belong to one root cause cluster generated for the GD DNN). An example of a questionnaire sent to the participants can be found in our replicability package.

[6]https://ulsurvey.uni.lu/

For each set of answers (i.e., the ones provided with 5, 10, or 15 images), we counted the number of answers that matched our ground truth. The number of images required to determine the root causes of a DNN failure is the minimal number leading to a high percentage of correct answers.

To summarize, based on the experimental design above, we prevented learning effects from one question to the next. We ensured that, for the same case study, the selected cluster and images were different for each participant and selected randomly, to avoid any form of systematic bias.

Experiment Results.

Table 5.9 presents the results obtained from analyzing the questionnaire data. Overall, we notice that when looking at 5, 10, and 15 images, 89%, 84%, and 74% of the participants found the correct commonality in a cluster, respectively. Given that participants performed that type of task for the first time, with limited initial training, we can consider 89% to be a good result and a lower bound of what experienced practitioners would achieve.

Table 5.9: Percentage of correct responses by inspecting 5, 10 and 15 images for each case study subject, based on the questionnaire study.

	Correct responses		
Case Study Subjects	**5 inspected images**	**10 inspected images**	**15 inspected images**
GD	80%	88%	67%
OC	83%	83%	86%
HPD	100%	83%	67%
Overall	89%	84%	74%

However, surprisingly, a larger number of inspected images does not improve results. On the contrary, with a larger number of images being inspected (e.g., 15 images), the percentage of correct answers tends to drop. This result may be due to the fact that a larger set of images leads to a higher cognitive load and is also more likely to include noisy images (i.e., images that do not present the same commonalities as most of the other images in the cluster). In the presence of noisy images, a user who looks for a commonality across all the images may identify characteristics that are not a correct explanation for the DNN failure.

Since we are dealing with proportions, we performed a Fisher exact test to determine if the differences observed between the pairs 5 *images vs* 10 *images* and 5 *images vs* 15 *images* are significant. The obtained p-values (i.e., 1 for '5 *vs* 10' and 0.4 for '5 *vs* 15') indicate that differences are not significant, which may be due in part to the small number of participants (19). However, our results clearly show that the inspection of five images does not lead to worse results than the inspection of a higher number of images, thus justifying our assumption in RQ1.1 (i.e., engineers inspect five images per cluster).

GD and OC lead to similar results, with 80% (GD) and 83% (OC) of the participants who inspected five images providing a correct answer. HPD leads to better results; indeed, 100% of the participants found the correct commonality by inspecting five images. Such difference between HPD and the other two cases is likely due to the fact that the simulator used to generate HPD images is more realistic (i.e., generates a whole face), while the simulator used for GD and OC simply generates eye bulbs and part of the forehead (see Figure 5.5), thus resulting in images that are more complicated to quickly understand by participants who are not familiar with the application domain. Despite such differences, which are to be expected across case studies, the generic trend is consistent: five images seem to be sufficient and not worse than larger sets of images. Therefore, we conclude that inspecting five images per cluster is an acceptable choice regardless of the case study DNN.

RQ2. *Does SAFE enable engineers to more effectively and efficiently retrain a DNN when compared with HUDD and baselines approaches?*

Design and measurements.

In this experiment, we investigate whether SAFE significantly improves the accuracy due to retraining the DNN, thanks to its selection method accounting for the shape of clusters. We compare these improvements to HUDD and two baselines, namely $B1$ and $B2$, which are depicted in Figure 5.9 and explained in the following:

$B1$: In this baseline, we use the failure-inducing images in the improvement set as an unsafe set. Precisely, we label[7] a random subset of the improvement set and execute the DNN to identify failure-inducing images. As for HUDD, this selected unsafe set

is augmented by applying bootstrap resampling (i.e., replicating samples in the unsafe set) in order to have a sufficiently large number of unsafe images to improve the DNN.

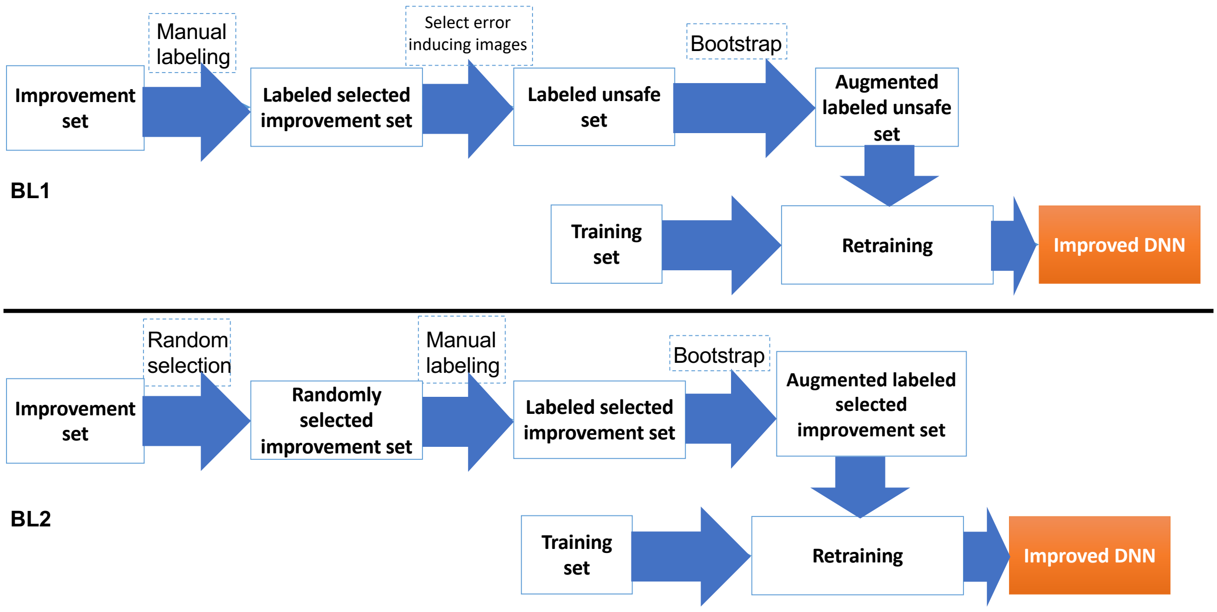

Manual labeling
Select error inducing images
Bootstrap
Improvement set
Labeled selected improvement set
Labeled unsafe set
Augmented labeled unsafe set
BL1
Training set
Retraining
Improved DNN
Random selection
Manual labeling
Bootstrap
Improvement set
Randomly selected improvement set
Labeled selected improvement set
Augmented labeled selected improvement set
BL2
Training set
Retraining
Improved DNN

Figure 5.9: Process of the two baselines used to compare with SAFE

$B2$: This baseline consists of randomly selecting a set of images from the improvement set, labeling them to obtain a *labeled selected improvement set*, and using them for retraining.

We rely on the same settings and environment to run the experiments for the four different retraining strategies (i.e., SAFE, HUDD, $B1$, and $B2$). These experiments are repeated ten times to account for randomness.

To retrain DNNs, we rely on the approach described in Section 5.2.

For fair comparisons with HUDD (see Chapter 3), $B1$, and $B2$, we configure bootstrap resampling to generate an *augmented labeled unsafe set* and an *augmented labeled selected improvement set* with the same size as the *balanced labeled unsafe set* for HUDD (see Figure 3.1).

[7]For our experiments, labeling comes for free because we either derive images using a simulator or we rely on available datasets. However, labeling comes at a high cost in industrial practice, where new images are collected from the field.

We compute the accuracy of the retrained models on the test sets and compare the accuracy improvement obtained by SAFE with those obtained by HUDD and the baselines. For this experiment, we consider all the case study subjects presented previously in Table 6.1.

The improvement sets for GD and OC have been generated through additional executions of UnityEyes. For HPD and FLD, they have been generated with additional executions of the IEE simulator, configured to use two new face models which were not used for generating the training and test sets. We selected images of the original datasets not used for the training and test sets for the other cases.

Experiment Results.

Table 5.10 provides an accuracy comparison between SAFE, HUDD, and the two baselines on the different case study DNNs. It also provides the size of the unsafe sets (number of images) selected by each method. Additionally, we report the significance of these results in Table 5.11, including the values of the Vargha and Delaney's $\hat{A}_{12}$ effect size and the p-values resulting from performing a Mann-Whitney U-test between the accuracy of SAFE and other improvement strategies. Recall that we run each strategy ten times. Typically, an $\hat{A}_{12}$ effect size above 0.56 is considered significant with higher thresholds for medium (0.64) and large (0.71) effects, thus suggesting the effect sizes between SAFE and other strategies are large across case study DNNs, in all but one case.

Table 5.10: RQ2: Unsafe set size and the accuracy improvement of SAFE compared to HUDD

DNN	**Original Model**	**SAFE**			**HUDD**			**BL1**	**BL2**
	Accuracy	Unsafe set size	Accuracy	Improvement over original model	Unsafe set size	Accuracy	Improvement over original model	Accuracy	Accuracy
GD	95.95%	1648	96.74%	**+0.79**	1615	96.23%	+0.28	95.23%	95.80%
OC	88.03%	153	96.29%	**+8.2**	160	94.41%	+6.38	91.65%	92.33%
HPD	44.07%	438	69.58%	**+25.51**	481	68.13%	+24.06	66.73%	66.30%
FLD	44.99%	659	79.17%	**+34.18**	502	75.23%	+30.24	72.02%	73.83%
TS	81.65%	597	93.47%	**+11.82**	704	93.03%	+11.38	92.63%	92.73%
OD	84.12%	212	97.14%	**+13.02**	258	97.04%	+12.92	97.04%	96.67%

Table 5.11: RQ2: p-values and VDA values when comparing SAFE to HUDD and the baselines

	GD			**OC**			**TS**			**OD**			**HPD**			**FLD**		
p-value	3e-3	3e-3	1e-3	1e-4	1e-4	1e-4	7e-3	3e-3	1e-3	0.22	2e-3	0.04	3e-3	2e-4	1e-4	1e-4	1e-4	1e-4
VDA	0.88	0.88	0.9	1.0	1.0	1.0	0.85	0.96	1.0	0.66	0.9	0.77	0.85	1.0	1.0	1.0	1.0	1.0

For SAFE, improvements in accuracy over the original model varies from 0.79 to 34.18, compared to lower ranges for HUDD (from 0.28% to 30.24%), $B1$ (from -0.18% to 27.03%) and $B2$ (-0.15% to 28.84%). Across case study DNNs, SAFE clearly and systematically yields better accuracy results compared to HUDD and the baselines, though to varying extents. Note that unsafe set sizes are comparable across improvement strategies and that differences in accuracy are therefore due to the strategy adopted for selecting unsafe images.

The lower improvement shown by the baselines can be explained by the fact they do not rely on root cause clusters to select images for the unsafe set, a strategy that appears to be beneficial in our context. SAFE always yields higher improvement than that of HUDD. This is because HUDD relies on the cluster's centroids to select images for the unsafe set without accounting for the cluster's shape. In contrast, SAFE relies on core points to choose the images for the unsafe set. This enables the handling of clusters with arbitrary shapes as the selected images take the shape of the cluster, as explained in Section 5.2.

Regarding SAFE results, we notice three types of improvement explained in the following:

For GD, the improvement is only $+0.79$. This is because the accuracy of the original model for this case study subject was already very high (95.95%), with limited room for improvement.

For HPD and FLD, the improvement in accuracy is $+26.16$ and $+34.18$, respectively. The original models for these two case study subjects had low accuracy and we therefore expected a large improvement. In fact, HPD and FLD represent two cases where retraining was very much needed.

For OC, TS and OD, the improvement in accuracy is $+8.2$, $+11.82$ and $+13.18$, respectively. Though this improvement is moderate compared to the previous case study DNNs, it is still significant given the original models' relatively high accuracy. After retraining, SAFE achieved high accuracy for these case study subjects with 96.29%, 93.47%, and 97.30%, respectively.

We notice in Table 5.11 that the p-values when comparing SAFE to the baselines are always below 0.05. As for HUDD, the p-values are lower than 0.05 in five out of six case study subjects. This implies that in most cases, the null hypothesis is rejected.

RQ3. ***Does SAFE provide time and memory savings compared to HUDD?***

Design and measurements.

As mentioned in the previous Section, SAFE proved its effectiveness in retraining the different case study subjects as it obtained significant improvements over the original models. SAFE not only performs better than HUDD but also, and perhaps more significantly, provides very high time and memory savings. This research question investigates whether SAFE, with its black-box nature, offers significant time and memory saving.

We compare the time required by SAFE and HUDD to perform their most expensive tasks, which are the time required to extract the features by SAFE and to generate the heatmaps by HUDD; we do not report the time required to perform the other steps of the two approaches because these steps are either shared or do not have any practical impact on performance (i.e., they took few seconds in our experiments). We also compare the memory allocation of the features and the heatmaps, which are the data types processed only by SAFE and HUDD, respectively.

Experiment Results.

Table 5.12 provide our results. We observe a large time and memory saving for SAFE compared to HUDD. Such performance considerations have significant practical implications. For example, we can observe that SAFE requires, in the worst case, only 3.5 minutes to generate RCCs and the unsafe set to be used for retraining. HUDD, instead, requires 65 minutes to achieve the same objective. Such difference has a huge impact on the practicality of the approach as, for SAFE, the analysis of RCCs can be performed shortly after observing DNN failures. HUDD may require up to one hour to do the same.

Such execution time savings allow engineers to conduct DNN safety analysis and improvements in a much shorter time. Memory savings also have significant implications since it prevents the need for expensive hardware to perform such analysis.

The explanation for the above results is that SAFE is using extracted features to represent the images instead of heatmaps. Feature extraction is less costly in time and memory than the computation of

Table 5.12: Execution time of the feature extraction of the improvement sets and the memory allocations of these features for SAFE compared to the execution time of the generation of the heatmaps and their memory allocation.

Subject	**Execution time (s)**		**Memory allocation (Mb)**	
	SAFE	**HUDD**	**SAFE**	**HUDD**
GD	176	3,920	74.2	78,641
OC	160	958	4.1	3,551
HPD	161	1,294	1.6	8,839
FLD	210	1,883	0.82	11,981
OD	149	1,059	13.2	5,989
TS	152	2,335	1.6	16,744

heatmaps. Indeed, the distance computed on features is less computationally complex than the one calculated on heatmaps.

Heatmaps also take a great deal of memory for storage. HUDD generates heatmaps for each layer. For instance, the heatmap for the eighth layer of AlexNet has a size of 169×256 (convolution layer), while the heatmap for the tenth layer has a size of $4096 \times C$. With this architecture, HUDD will generate eight heatmaps of size 169×256 and one heatmap of size $4096 \times C$ for every image in the dataset. In contrast, for SAFE, each image is represented by a 1×256 matrix (256 features for each image).

5.3.3 Threats to Validity

We discuss internal, conclusion, construct, and external validity according to conventional practices [145].

Internal Validity: A possible internal threat is the use of the feature extraction method on which we rely, which could negatively affect our results if inadequate. Indeed, clustering relies on the similarity computed in terms of the extracted features. To mitigate this threat, we have checked that some of the features extracted by our method are consistent within clusters, by visually inspecting them. Indeed, such features contain enough information on the images if the clusters are visually consistent, thus demonstrating that the features extraction method worked.

External validity: The selection of the case study DNNs could be a threat to validity. This work alleviates this issue by using six datasets with diverse complexity. Four subject DNNs out of six implement tasks motivated by IEE business needs that address problems that are quite common in the automotive industry. Also, the simulators used in our experiments, though being related to IEE in-car sensing business cases, vary in terms of characteristics; indeed, they range from high-fidelity simulation of specific body parts (in our case, the human eye) to whole human body simulations (we crop the face) with lower fidelity. In our experiments we test DNNs that process cropped images (e.g., human's head, traffic signs). Cropping, which a DNN can perform, limits the number of features appearing in the images to be processed, thus potentially simplifying the task to be performed by SAFE. However, cropping was justified in our context by the expected inputs of our subject DNNs. Finally, although we focus on data sets related to in-car sensing, we believe that SAFE will perform well with other data sets since the VGG model used for the feature extraction was pre-trained on Image-Net, which means that the model can capture features related to $1,000$ classes, including humans, animals, and objects. In the future, we aim to extend our work to include subjects from different domains (e.g., different types of classification tasks with non-cropped images).

Another threat to the generalizability of our results is the dependence on ImageNet. SAFE relies on VGG16 to extract features from images. VGG16 was pre-trained on ImageNet, which is a large image database. Therefore, SAFE is expected to work better with images containing objects recognized by ImageNet. However, we believe that this characteristic does not affect the practical applicability of SAFE in the automotive context since the ImageNet VGG recognizes $1,000$ different objects, including objects belonging to the automotive scenery; these objects include cars, faces, and eyes, for example [8].

Conclusion validity: To avoid violating parametric assumptions in our statistical analysis, we rely on a non-parametric test and effect size measure (i.e., Mann Whitney U-test, the Vargha and Delaney's $\hat{A}_{12}$ statistics, and the one-sample Wilcoxon signed rank test, respectively) to evaluate the statistical and practical significance of differences in results. We report both p-values and effect sizes.

Due to the stochastic nature of SAFE (e.g., DNN retraining), the experiments conducted with the SAFE method were executed over ten runs. We reported the descriptive statistics of those runs and discussed the statistical significance and effect size of differences across methods.

Construct validity: The constructs considered in our work are effectiveness and cost. Effectiveness is measured through complementary indicators, which include (1) within-cluster variance reduction for at least one parameter (for RQ1.2), (2) average values being close to unsafe values for parameters with high within-cluster variance reduction (for RQ1.3), (3) coverage of plausible causes of failures represented by unsafe values (for RQ1.4), (4) DNN accuracy improvement (for RQ2). Although the effectiveness regarding the analysis of root causes (i.e., RQ1.2 to RQ1.4) might be evaluated with user studies, such evaluation might be biased by the background and experience of the selected pool of users, which shall also be sufficiently large in number. In our context, end-users are engineers with background in safety analysis (e.g., to determine if an input is realistic or if a DNN failure may lead to a hazard) and machine learning. However, since safety experts are generally not trained to use DNNs whereas DNN experts (e.g., recently graduated students) are typically not safety experts, it would be difficult to select a large enough set of users for the study. For this reason, we preferred to rely on reflective indicators based on the information provided by simulators, thus enabling an objective evaluation. Moreover, the quality and usefulness of our results have been confirmed by experts at our industry partner, IEE Sensing, over multiple technical and management meetings. These experts included researchers with a Ph.D. in mathematics and machine learning who develop safety-critical software components, safety engineers integrating DNN-based components, and chief technology officers. To measure the effectiveness of DNN retraining, we relied on improvements in accuracy, which is common practice.

Concerning cost, we discussed the feasibility of root cause analysis by reporting the number of clusters generated by the approach and the number of images that are sufficient to determine commonalities across images, based on our experience and that of our industry partners. We also discussed the cost of DNN retraining by reporting the time required for retraining, which affects the feasibility of the approach in practice, and memory allocation costs, which affect hardware requirements.

Another threat is with RQ1.3 (Section 5.3.2) where we systematically select unsafe parameters for each case study to evaluate the clusters. This can prevent us from identifying other reasons for failures if,

[8]The full list of classes is available at https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

for any reason, we miss some of these parameters. However, whether this is the case or not, this does not prevent the identification of clusters and missing parameters would then lead to clusters without clear root causes, something we have not observed in our experiments.

5.4 Conclusion

In this chapter, we presented SAFE, a new black-box approach that automatically identifies the different situations in which a DNN is more likely to fail, without requiring any modification to the DNN or access to its internal information. Similar to *HUDD*, SAFE characterizes such situations by generating clusters of images that likely lead to a DNN failure because of the same underlying reason. We refer to these clusters as root cause clusters. Different from HUDD, SAFE is based on a pre-trained model to extract features from failure-inducing images. These features replace the heatmaps used by HUDD to generate the root cause clusters. Also, SAFE uses a density-based clustering algorithm to generate arbitrary-shaped clusters.

SAFE uses a new method to select an unsafe set for retraining based on the cluster's core points. More specifically, SAFE selects images close to each cluster's core points. These images are manually labeled and used to augment the training set to improve the DNN through retraining.

Empirical results show that SAFE derives root cause clusters that can effectively help engineers determine the root causes for DNN failures. Indeed, the number of generated clusters is low, thus making the visual inspection of a few representative images for each cluster feasible. They include images with similar characteristics that are likely related to the cause of the failure. Further, for our case study subjects, the generated clusters capture all the possible causes of failures. Compared to HUDD, SAFE generates clusters with more similar characteristics covering a larger set of failure causes. Moreover, the DNNs retrained by SAFE achieve a higher accuracy than that obtained with HUDD and baseline approaches.

Besides the benefits described above, SAFE also saves large amounts of execution time and memory due to its black-box nature and the reliance on the extracted features instead of heatmaps, thus making it more usable in practical contexts.

Chapter 6

Pipelines for Clustering-based DNN Explanation: an Empirical Assessment.

In the previous chapters, we proposed a white-box approach (HUDD, see Chapter 3) and a black-box approach (SAFE, see Chapter 5) to automatically characterize the root causes of DNN failures. Their objective was to identify clusters of similar images from a potentially large set of images leading to DNN failures. By visualizing the images in each cluster, the engineers can easily understand what are their commonalities and, therefore, determine the scenarios in which the DNN may fail. This is necessary in the context of safety analysis to enable the evaluation of risks associated with a DNN and identify countermeasures. The analysis pipelines for HUDD and SAFE were instantiated in specific ways according to best practices. However, several variants exist for each pipeline component.

In this chapter, we report on an empirical evaluation of 99 different pipelines for root cause analysis of DNN failures. They combine transfer learning, autoencoders, heatmaps of neuron relevance, dimensionality reduction techniques (PCA, UMAP), and different clustering algorithms (K-means, DBSCAN, HDBSCAN).

6.1 Introduction

Our previous work is the first application of unsupervised learning to perform root cause analysis targeting DNN failures. Precisely, we proposed two DNN explanation methods: *SAFE* (Safety Analysis based on Feature Extraction) [24] and *HUDD* (Heatmap-based Unsupervised Debugging of DNNs) [18]. They both process a set of failure-inducing images and generate clusters of similar images. Commonalities across images in each cluster provide information about the root cause of the failure. For example, applying our approaches to failure-inducing images for a DNN that classifies car seat occupancy may include a cluster of images with child seats containing a bag; such cluster may help engineers determine that bags inside child seat are likely to be misclassified and, therefore, the training set should be improved accordingly (e.g., more child seats with objects should be considered). Both SAFE and HUDD also support the identification of additional images to be used to retrain the DNN.

HUDD and *SAFE* differ with respect to the kind of data used to perform clustering and the pipeline of steps they rely on. *HUDD* applies clustering based on internal DNN information; precisely, for all failure-inducing images, it generates heatmaps capturing the relevance of DNN neurons on the DNN output. Finally, it applies a hierarchical clustering algorithm relying on a distance metric based on the generated heatmaps. *SAFE* is black-box as it does not rely on internal DNN information. It generates clusters based on the visual similarity across failure inducing images. To this end it relies on feature extraction based on transfer learning, dimensionality reduction, and the DBSCAN clustering algorithm.

SAFE and *HUDD* rely on a pipeline that has been configured in specific ways according to best practices. However, several variants exist for each component of both approaches (e.g., different transfer learning models, different clustering algorithms).

In this chapter, we aim to evaluate these pipeline variants for both SAFE and HUDD. Therefore, we propose an empirical evaluation of 99 alternative configurations for SAFE and HUDD (pipelines). These pipelines were obtained using different combinations of feature extraction methods, clustering algorithms, dimensionality reduction techniques, in addition, we assessed the effect of fine tuning the transfer learning models used by feature extraction methods.

For our empirical evaluation we considered six case study subjects, two of which were provided by our industry partner in the automotive domain, IEE Sensing [17]. Our subjects' applications include head pose classification, eye gaze detection, drowsiness detection, steering angle prediction, unattended child detection, and car position detection.

We present a systematic and extensive evaluation scheme for these pipelines, which entails generating failure causes that resemble realistic scenarios (e.g., poor lighting conditions or camera misconfiguration). Since the reason of failure in these scenarios are known a priori, such an evaluation scheme enables us to objectively analyze and evaluate the performance and robustness of these pipelines.

Our empirical results conclude that the best pipelines support and facilitate the process of functional safety analysis such that they 1) can generate RCCs that group together a very high proportion of images capturing a same root cause (94.2%, on average), 2) can capture most of the root causes of failures for all case study subjects (100.0%, on average), and 3) are robust to the rarity of failure instances in a data set (i.e., when some causes of failures affect less than 10% of the failure-inducing images).

6.2 The Proposed Pipelines

This section presents the different pipelines that can be used to implement variants of SAFE and HUDD. The evaluated pipelines differ from the original SAFE and HUDD variants with respect to four components: Feature Extraction, Dimensionality Reduction, Clustering, and Fine-Tuning. Each pipeline is a combination of a feature extraction method (FE), a dimensionality reduction technique (DR), and a clustering algorithm (CA). When feature extraction is based on transfer learning, we distinguish between models that are fine-tuned and not fine-tuned (FT/NoFT); feature extraction approaches not based on transfer learning cannot be fine-tuned.

We refer to each pipeline with the pattern *FE/{FT,NoFT}/DR/CA*, with each keyword being replaced with the name of the selected method. We depict in Figure 6.1 all the pipelines evaluated in our study; the different components are described in the following sections.

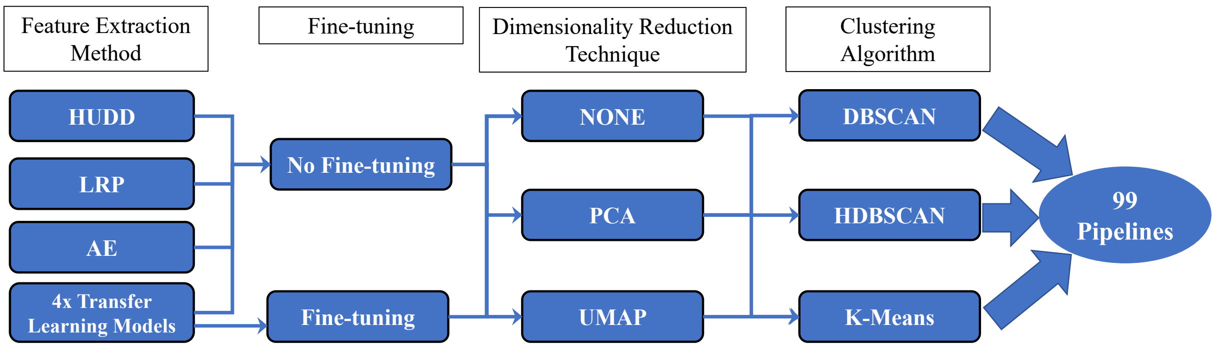


Figure 6.1: Pipelines evaluated in our experiments.

6.2.1 Feature Extraction

Feature Extraction based on Transfer Learning

Several DNN architectures to extract features based on transfer learning have been proposed: Inception-V3 [146], VGGNet [31], ResNet-50 [147], and Xception [148]. These DNNs were trained on ImageNet [149], which is a dataset with more than 14 millions annotated images. The number of extracted features depends on the selected DNN architecture; Inception-V3, VGGNet-16, ResNet-50, and Xception generate 2048, 512, 2048, and 2048 features, respectively. They are described in the following paragraphs.

- **VGG-16:** VGG-16 is a Convolution Neural Network (CNN) architecture and the winner of the ILSVR (Imagenet) competition in 2014. VGG-16 focuses on convolution layers of 3×3 filters with a stride of 1 and always uses the same padding and maxpooling layer of 2×2 with a stride of 2 instead of having a large number of hyper-parameters. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. VGG-16 has two fully connected layers followed by a softmax layer as an output. The network has an image input size of 224×224.

- **ResNet-50:** ResNet [147] is a CNN based on residual blocks. This architecture aims to solve vanishing gradient problems in deep neural networks. During the backpropagation process, the gradient diminishes dramatically in deep networks. Small values of gradients prevent the weights from changing their values, which slows the training process. To solve this issue, ResNet introduces residual blocks. These building blocks present skip connections between the previous convolutional layer's input and the current convolutional layer's output. Similar to VGG-16, the network has an image input size of 224×224.

- **Inception-V3:** Inception-V3 is a refined version of Inception [150]. This network proposes additional variants of Inception blocks to reduce the number of multiplications in the convolution and minimize computational complexity. These variants are based on two factorizations: factorization into smaller convolutions and factorization into asymmetric convolutions. The network has an image input size of 224×224.

- **Xception:** Xception is a pre-trained CNN that is 71 layers deep and can classify images into $1,000$ different classes such as animals, objects, and humans. This allowed the model to learn various feature representations for a wide range of images. The Xception's input size is 299×299.

Fine-tuning

Fine-tuning is a typical strategy for extracting features using transfer learning. Specifically, fine-tuning relies on taking a model that has already been trained for a particular task A as a starting point and then removing, adding, freezing, or unfreezing some layers to improve the performance for a similar task B. It aims to benefit from the knowledge gained from a source task and generalize it to a target task.

Fine-tuning consists of freezing the shallow layers (close to the input), which learn more generic features (edges, shapes, and textures), and retraining the deeper layers (i.e., we let the DNN algorithm update the weights of the layers close to the output), which learn more specific features from the input data [151]. To fine-tune a pretrained DNN model, we follow four steps:

1. Create a new model whose layers (along with their weights) are cloned from the pre-trained model, except for the output layer.

2. Add a new fully connected output layer with a number of outputs equal to the number of classes in the target dataset, and initialize its weights with random values.

3. Freeze shallow layers in the network, which are responsible for the feature extraction process (to guarantee that all the important features, previously learned by the pre-trained model, are not eliminated).

4. Start training the new model on the target dataset, where the weights of all the non-frozen layers will keep updating using the backpropagation process. For the termination criterion, we use 100 epochs or until the loss stops improving (whichever criterion is met first).

Feature Extraction based on Autoencoders

As explained in Section 2.1.3, an *Encoder* plus a *Decoder* make up an autoencoder (AE). The input is compressed by the *Encoder*, and the *Decoder* reconstructs the input using the *Encoder*'s compressed version (at the *Bottleneck*).

Since AEs extracts only the few input features necessary to aid the reconstruction of the output, the encoder might ignore other features which are not prioritized. For example in case of face images, the AE can discard the color of the skin because it is a non-prioritized feature to the AE.

However, the encoder often learns useful properties of the data [152]. The model can then receive input data from any domain, and a fixed-length feature vector obtained at the *Bottleneck* can be used for clustering. Such a vector offers a compressed version of the input data representation containing sufficient information about this data.

Feature Extraction based on Heatmaps

In our work, we rely on heatmaps as an additional method for feature extraction. Since heatmaps represent the relevance of each neuron on DNN outputs, failure-inducing inputs sharing the same underlying cause should show similar heatmaps.

Precisely, we rely on two different methodologies for extracting features using heatmaps, we refer to them as *LRP* and *HUDD*, according to the name of the technique driving feature extraction. LRP and HUDD have been introduced in Chapters 2.4 and 3.2. Feature extraction based on LRP, which generates heatmaps for internal layers but does not integrate a mechanism to select the most informative layer, considers the heatmap computed by the LRP technique for the input layer. Feature extraction based on HUDD, instead, considers the heatmap generated for the DNN internal layer selected by HUDD as the best for clustering.

6.2.2 Dimensionality Reduction Techniques

Several dimensionality reduction techniques exist in the literature. In this chapter, we rely on two state-of-the-art techniques: Principal Component Analysis (PCA) [45] and Uniform Manifold Approximation and Projection (UMAP) [46]. PCA is used for its simplicity of implementation and because it does not require much time and memory resources. UMAP is used for its effectiveness when applied before clustering. UMAP groups data points based on relative proximity, which optimizes the clustering results. PCA and UMAP are described in detail in Chapter 2.1.3.

Principal Component Analysis (PCA)

In our context, we reduce the features for all our evaluated pipelines to 10 components. We empirically obtained this number in a preliminary investigation conducted with one of our case study subjects (i.e., HPD). Precisely, we executed a clustering algorithm (K-means) multiple times; each execution was performed with a set of features obtained by applying PCA with a different number of components. We evaluated all the clustering solutions using the Silhouette Index [140] (see Section 6.2.3) and chose the number of components yielding the highest index value.

Uniform Manifold Approximation and Projection (UMAP)

Since UMAP can keep the structure of the data, even in a 2-dimensional space, we reduce the number of features to 2 components.

6.2.3 Clustering Algorithms

In this study, we rely on three well-known clustering algorithms, K-means [39], DBSCAN [40], and HDBSCAN [41] described in Chapter 2.1.3. These three clustering algorithms were chosen after preliminary experiments including also the Hierarchical Agglomerative Clustering (HAC) [92] and the Mean Shift algorithm [153]. When generating clusters for one of our subjects (i.e., HPD, see Section 6.3), HAC and Mean Shift yielded much lower values of the Silhouette Index than the DBSCAN, HDBSCAN, and K-means algorithms; therefore, we discarded HAC and Mean Shift from our selection. Since a clustering algorithm may require the manual selection of parameters' values, such as the number of clusters (K-means) or the minimum distance between data points (DBSCAN), we rely on an internal evaluation metric (the Silhouette Index [140]) and the *knee-point method* [87] to automate the selection of such values.

The *Silhouette Index* is a standard practice in cluster analysis that maximizes cohesion (i.e., how closely related objects are in a cluster) and separation (i.e., how well-separated a cluster is from other clusters).

The *knee-point method* automates the *elbow method* heuristics [90] by fitting a spline to the raw data using univariate interpolation, normalizing min/max values of the fitted data, and selecting the knee-points

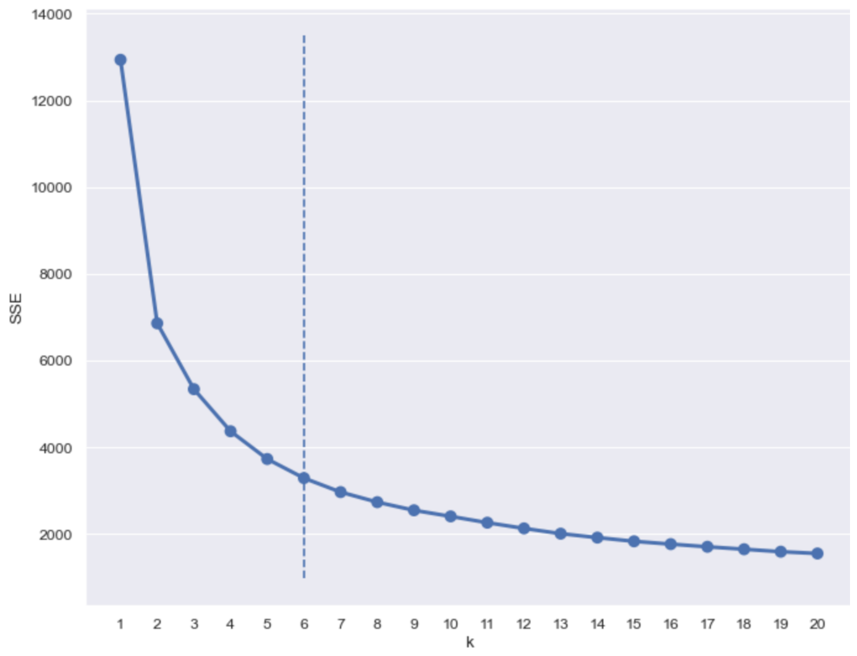


Figure 6.2: Approximating the optimal number of clusters K using the Knee-point method. In this case the optimal K is equal to six.

at which the curve most significantly deviates from the straight line segment that connects the first and last data point. We rely on the *knee-point method* to automatically select the optimal number of clusters for the K-means algorithm.

K-means:

To select an optimal value of K, we rely on the knee-point method. Precisely, we cluster the data with different values of K (in our evaluation, we consider the range $[5-35]$). For each clustering result, we compute the within-cluster sum of squared errors (SSD), which is the sum of the distances of each point to its cluster center. We then apply the knee-point approach to these SSDs and their respective K.

Figure 6.2 shows an example of K-approximation using the knee-point method.

DBSCAN:

For the identification of the values for ϵ and *MinPts*, we rely on the same strategy integrated in SAFE, described below.

We determine the optimal value for ϵ by first computing the Euclidean distance from each data point to its closest neighbor. Then, we identify the optimal ϵ value as the knee-point of the curve obtained by considering those distances in ascending order.

To select an optimal *MinPts* value, we execute DBSCAN multiple times with varying $MinPts$ values and with an ϵ equal to the optimal value determined above. We then select the clustering configuration that corresponds to the highest Silhouette Index value.

6.3 Evaluation

In this Section, we aim to evaluate the pipelines presented in Section 6.2. A pipeline leads to the generation of clusters of images that are visually inspected by safety engineers to determine the root cause captured by each. We assume that a root cause can be described in terms of the commonalities across the images in a cluster; each root cause is thus a distinct scenario in which the DNN may fail (hereafter, *failure scenario*). The pipeline that best support such process should be the one requiring minimal effort towards accurate identification of root causes. Therefore, the best pipeline is the one that generates clusters having a high proportion of similar images (to facilitate the identification of the root cause, based on analyzing similarities across images in a cluster), enable the detection of all the root causes of failures, and be robust to the rarity of a particular root cause (to avoid ignoring infrequent but unsafe failure causes). Based on the above, we defined three research questions to drive our empirical evaluation:

RQ1 *Which pipeline generates root cause clusters with the highest purity?*

We define a *pure* cluster as one that contains only images representing the same failure scenario. Such clusters are expected to be easier to interpret; indeed, the engineer should more easily determine the root cause of failures if all the images share the same characteristics. Therefore, the best pipeline is the one that leads to clusters with the highest degree of purity. The purity of a cluster is computed as the maximum proportion of images belonging to a same failure scenario in this cluster.

RQ2 *Which pipelines generate root cause clusters covering more failure scenarios?*

This research question investigates to which extent the different pipelines miss failure failure scenarios. Ideally, all failure scenarios should be captured by one or more clusters. We say that a failure scenario is covered by a cluster if a majority of the images in the cluster belong to the scenario; indeed, commonalities shared by most of the images in a cluster should be noticed by engineers during visual inspection. We aim to determine which pipeline maximizes such coverage.

RQ3 *How is the quality of root cause clusters generated affected by infrequent failure scenarios?*

Some failure scenarios may be infrequent but are nevertheless important to identify. Ideally, a pipeline should be able to produce high-quality clusters even when a small number of images belong to one or more failure scenarios. In this research question, we vary the number of images belonging to failure scenariosand study how the effectiveness of pipelines (purity and coverage of the generated clusters) is affected.

To perform our empirical evaluation, we have implemented the transfer learning models using Tensorflow [154] and Keras [155]. The clustering algorithms and the dimensionality reduction methods were implemented using the Scikit-Learn library [156]. All the experiments were carried out on an Intel Core $i9$ processor running macOS with $32GB$ RAM. Additionally, in our experiments, we relied on the LRP implementation provided by LRP authors [85] for well-known types of layers (i.e., MaxPooling, AvgPooling, Linear, and Convolutional layers). Further, we extended the implementation of LRP to include DNN models implemented in PyTorch [98], Tensorflow, and Keras libraries.

6.3.1 Subject Systems

To evaluate our pipelines, we consider four different DNNs that process synthetic images in the automotive domain. These DNNs support gaze detection, drowsiness detection, headpose detection, and unattended child detection, which are subjects of ongoing innovation projects at IEE Sensing, our industry partner developing sensing components for automotive. Additionally, we consider two DNNs that process real-world images to support autonomous driving: steering angle prediction and car position detection.

The gaze detection DNN (GD) performs gaze tracking; it can be used to determine a driver's focus and attention. It divides gaze directions into eight categories: TopLeft, TopCenter, TopRight, MiddleLeft, MiddleRight, BottomLeft, BottomCenter, and BottomRight. The drowsiness detection DNN (OC) has the same architecture as the gaze detection DNN and relies on the same dataset, except that it predicts whether the driver's eyes are open or closed.

The head-pose detection DNN (HPD) is an important cue for scene interpretation and computer remote control, such as in driver assistance systems. It determines the pose of a driver's head in an image based on nine categories: straight, rotated left, rotated left, rotated top left, rotated bottom right, rotated right, rotated top right, tilted, and headed up.

The unattended child detection DNN is trained with the *Synthetic dataset for Vehicle Interior Rear seat Occupancy* detection (SVIRO) [157]. SVIRO is a dataset generated by IEE that represents scenes in the passenger compartment of ten different vehicles. The dataset has been used by IEE to train DNNs performing rear seat occupancy detection using a camera system. We use it to train a DNN for unattended child detection. We consider a seat empty when there is an object, an empty infant/child seat, or nothing. We consider the presence of a child/infant as a class and the presence of an adult as another class. As a result, we have labelled the dataset with three classes (i.e., *empty seats*, *children/infants not accompanied by adults*, and *presence of an adult*).

For Steering Angle Prediction (SAP), we rely on the pre-trained Autumn DNN model [158], which follows the DAVE-2 architecture [159] provided by NVIDIA. It is a DNN to automate steering commands of self-driving vehicles [160]; it predicts the angle at which the wheel should be turned. It has been trained on a dataset of road images captured by a dashboard camera mounted in the car.

Car Position Detection (CPD) DNNs are used by most Advanced-Driver Assistance Systems (ADAS) to predict the positions of nearby cars. We rely on the CenterNet DNN [161], which is an accurate DNN used by most competition-winning approaches for object detection [162]. It has been trained on images from the ApolloScape dataset [163] collected using a dashboard camera to estimate the absolute position of vehicles with respect to the ego-vehicle.

For each subject DNN, we apply our pipelines to a set of failure-inducing images. Such sets consist of (1) images belonging to a provided test set and leading to a DNN failure and (2) test set images that were not leading to a DNN failure but had been modified to cause a DNN failure; the latter are images with injected root causes of failures and are described in Section 6.3.2.

In classifier DNNs (i.e., OC, GD, HPD, and SVIRO) a failure occurs in the presence of an image being incorrectly classified. For SAP and CPD, which are regression DNNs, we set a threshold to determine DNN failures. For SAP, we observe a DNN failure when the squared error between the predicted and the true angle is above 0.18 radian (10.3°), which is deemed to be an acceptable error in related work [3]. For CPD, since it tackles a multi-object detection problem, we report a DNN failure when the result contains at least one *false positive* (i.e., the distance between the predicted position of the car and the ground truth

Table 6.1: Case Study Systems

DNN	Data Source	Training Set Size	Test Set Size (Accuracy)	Failure inducing images	# M[1]	# N[2]	# H[3]	# B[5]	# SG[6]	# EG[7]	# EO[8]	# S[9]	# D[10]	# NF[11]
GD	UnityEyes	61,063	132,630 (96%)	5,371	-	80	-	80	-	-	-	80	80	80
OC	UnityEyes	1,704	4,232 (88%)	506	-	20	-	20	-	-	-	20	20	20
HPD	Blender	16,013	2,825 (44%)	1,580	90	90	90	90	90	90	-	90	90	90
SVIRO	Blender	15,489	3,427 (74%)	884	-	30	-	30	-	-	30	30	30	30
SAP	Autopilot [165]	33,808	45,406 (84%)	7,169	-	90	-	90	-	-	-	90	90	90
CPD	Apollo [163]	5,208	4,996 (91%)	444	-	90	-	90	-	-	-	90	90	90

[1] Mask [2] Noise [3] Hand [5] Blurriness [6] SunGlasses [7] EyeGlasses [8] Everyday Object [9] Scaling [10] Darkness [11] No Injected Fault

is above 10 pixels [164]).

In Table 6.1, we provide details about the case study subjects used to evaluate our pipelines. For each subject, we report the source of the data set (e.g., the simulator used to generate the data), the training and test set sizes, the accuracy of the DNN on the original test set, the number of failure-inducing images and the number of images for each injected root cause (they are detailed in Section 6.3.2).

We fine-tune the pipelines relying on transfer learning using the test sets of the respective case studies. We use the resulting fine-tuned model to extract the features from the failure-inducing sets. We train on the test sets because the number of images in each set is sufficient for the model to learn the features. Also, we train the autoencoders on the training set, and use the test set of the respective case study to validate the results. The termination criterion is 50 epochs unless we reach an early stopping point (the model stops improving). After training, we use only the encoder part to extract the features from the images in the failure inducing set.

6.3.2 Injected Failure Scenarios

To assess the ability of different pipelines to generate clusters that are pure and cover all the root causes of failures, we need to know the root causes of failures in the test set. Since such root causes may vary (e.g., lack of sufficient illumination, presence of a shadow in a specific part of the image) and it is not possible to objectively demonstrate that a failure cause has been correctly captured by a cluster (e.g., some readers may not agree that certain images show lack of sufficient illumination), to avoid introducing bias and subjectivity in our results, we modify a subset of the provided test set images so that they will fail because of known root causes of failures. In total, we considered nine different root causes to be injected in our test set images and refer to them as *injected failure scenarios* (i.e., failure scenarios with injected root causes).

We derive an image belonging to an injected failure scenario by modifying a test set image according to the specific root cause we aim to inject; for example, by covering the mouth of a person with a mask. To ensure that a modified image leads to a DNN failure because of the injected root cause, we modify only test set images that, before modification, lead to a correct DNN output.

Figure 6.3 illustrates the different injected failure scenarios. Below, we describe the nine root causes considered in our study:

- **Hand:** The presence of a hand blocking the full view of the driver's face could affect the DNN result, leading it to mispredict the driver's head direction. We simulate a hand that is partially covering the face appearing in the image.

- **Mask:** Similar to *Hand*, the presence of a mask covering the nose or the mouth may affect a DNN that recognizes the driver's head pose. Using image key points, we drew the shape of a white mask to simulate a mask covering the nose and the mouth.

- **Sunglasses:** As for the *Mask*, we use the eyes' key points to draw sunglasses covering the driver's eyes.

- **Eyeglasses:** Different from the *Sunglasses*, we draw glasses with the eyes being still visible.

- **Noise:** A noisy image is one that contains random perturbations of colors or brightness. This failure scenario has been considered in related work to evaluate DNN robustness against adversarial attacks [166]. In real-world automotive systems, such a failure scenario resembles a defective camera or a high signal-to-noise ratio (SNR) in the communication channel between different electronic control units (ECUs), resulting in a noisy input. We use the Scikit-Image library [167] to add Gaussian Noise, a statistical noise with a probability density function equal to a normal distribution, also known as Gaussian Distribution.

- **Blurriness:** As for *Noise*, this failure scenario was used to evaluate DNN robustness [3]. We use the Pillow library [168] to add blurriness to images using a radius of 30 pixels.

- **Darkness:** Once again, this failure scenario was used in related work to evaluate DNN robustness [169] . In practice, poor lighting conditions could make the DNN fail because it cannot clearly recognize what is depicted in an image. We use the Pillow library [168] to decrease the brightness of images by a factor of 0.3; we selected 0.3 because it is the lowest value introducing failures in our subject results.

- **Scaling:** Such a failure scenario mimics the situation where a camera is misconfigured, leading to rescaled images being fed to the DNN. We reduce the size of an image by a value based on the image size (i.e., large $1200px \times 1200px$ images are scaled by $400px$, small $320px \times 320px$ images by $70px$) and insert a black background using the Pillow library [168].

- **Everyday Object:** For the SVIRO dataset, we introduce, in the car's rear seat, an object (e.g., a washing machine or a handbag) never observed in the training set, thus simulating the effect of an incomplete training dataset.

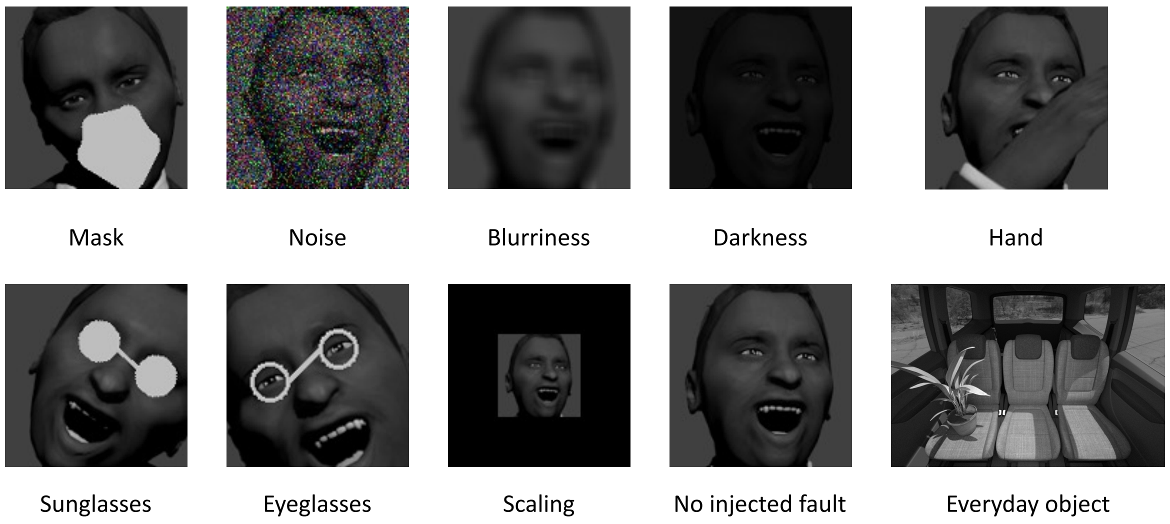


Figure 6.3: Injected failure scenarios in our study.

For regression DNNs (SAP and CPD), we randomly selected 90 images to be copied and modified for each failure scenario. For classifier DNNs, for each failure scenario, we randomly selected 10 images for each class label.

Please note that in addition to the injected failure scenarios explained above, our DNNs are affected by other natural failure causes (e.g., borderline images that are misclassified because they are very similar to the ones belonging to another class). Such cases are observed with any machine learning model since it is usually not possible to achieve perfect accuracy through training. We refer to these images as belonging to *natural failure scenarios*. In our analysis, we include a number of images belonging to natural failure scenarios equal to those with injected failure scenarios. This is because natural failure scenarios are usually observed with any DNN and, therefore, should be considered when generating RCCs.

6.3.3 Measurements and Results

RQ1. *Which pipeline generates root cause clusters with the highest purity?*

Design and measurements.

A pure cluster includes only images presenting the same root cause (i.e., common cause leading to a DNN failure); for example, a hand covering a person's mouth. Pure clusters simplify root cause analysis because they should make it easier for an engineer to determine the commonality across images and therefore the cause of failures.

Since the likely root cause of the failure in our *injected failure scenarios* is known, we focus on these scenarios to respond to RQ1. For each RCC, we compute the proportion of images belonging to each injected failure scenario. Therefore, we measure the purity P of a cluster C (hereafter, P_C) as the highest proportion of images belonging to one injected failure scenario $f \in F$ assigned to cluster C, where F is the set of all failure scenarios. P_C is computed as follows:

$$P_C = \max_{f \in F} \left(\frac{C_f}{|C|} \right) \tag{6.1}$$

The proportion of a failure scenario f in a cluster C is computed as the number of images belonging to f assigned to cluster C (C_f), divided by the size of cluster C.

Clusters that do not include any image belonging to an injected failure scenario are assumed to capture root causes due to natural failure scenarios and, consequently, are excluded from our analysis.

We study the purity distribution across RCCs generated for the different case study subjects. Since, ideally, we would like to obtain pure clusters, the best pipeline is the one that maximizes the average purity across the generated RCCs.

Experiment Results.

Figure 6.4 depicts a regression tree illustrating how the different components of a pipeline (feature extraction methods, fine-tuning, dimensionality reduction techniques and clustering algorithms) determine the purity of the clusters generated by a pipeline. We use the Conditional Inference Tree (CTree) algorithm [170] to generate this decision tree with a maximum depth set to 4 (we have four components in a pipeline) and a minimum split set to 10 (i.e., the weight of a node to be considered for splitting). The dataset used to build the tree consists of the components of each pipeline as attributes, and the purity of the generated clusters as the predicted outcome. The dataset size is equal to 99, the number of pipelines.

Each node of the tree represents a feature of the pipeline. Leaves (terminal nodes) depict box plots representing distributions of the average purity across RCCs generated by the pipelines belonging to each leaf. Each point in the box plot is the average purity of one pipeline (i.e., the average of the purity of all the RCCs generated across all case study subjects). To split a node, the CTree algorithm first identifies the feature with the highest association (covariance) with the response variable (purity, in our case). Precisely, it relies on a permutation test of independence (null hypothesis) between any of the features and the response [171]; the feature with the lowest significant p-value is then selected ($alpha = 0.05$, in our case). Once a feature has been selected, a binary split is then performed by identifying the value that maximizes the test statistics across the two subsets. Since we are in the presence of multiple hypothesis (assume m, for each node), to prevent a Type I error, for each feature j, CTree computes its Bonferroni-adjusted [172] p-value$_j$ as

$$\text{adjusted } p\text{-value}_j = 1 - (1 - p\text{-value}_j)^m$$

In Figure 6.4, we notice that the pipelines with fine-tuned models (Node 3 and 4) generate lower-purity clusters than those without any fine-tuning (Node 6 and 7). This is because these models were fine-tuned on a test set that did not include any injected root cause (i.e., only natural failure scenarios); recall that fine tuning is performed with labeled images (e.g., training set) and since our injected root causes capture scenarios not foreseen at training time, it would be unrealistic to consider such scenarios for fine tuning. Fine-tuning a model on a set of images means that it will learn all the features of those images. Therefore, clustering based on fine-tuned models will generate clusters based on the features observed during training, excluding injected features (i.e, the injected root causes). As a result, in our experiments, images are clustered based only on their natural fault (e.g., borderline class) instead of the injected faults.

The pipelines using non-fine-tuned transfer learning models as a feature extraction method (Node 7) generate purer clusters (min = 50%, median = 80%, max = 96%) than the pipelines using an autoencoder model, HUDD, or LRP (Node 6) (min = 50%, median = 65%, max = 70%). The purpose of the Autoencoder model is to provide a condensed representation of the image to be used for reconstruction. This is done by ignoring the features that the model considers insignificant and only keeping the features that help the encoder reconstruct the image accurately. Therefore, since the autoencoder is trained on the training set, the injected faults are ignored. Given that clustering is based on the condensed representation, the generated clusters are less pure than the ones generated by the pipelines with transfer learning models.

As for HUDD and LRP, it seems that their main limitation is that heatmaps do not well capture the presence of root causes affecting all the pixels in an image (i.e., the result of noise, blurriness, darkness, scaling). Heatmaps mainly capture which pixels of the image drive the DNN output, thus leading clustering to group images where the same pixels affected the output.

For instance, the DNN's response to a blurred image with a shadow on the mouth could be different from that of another blurred image with a shadow on the eyes, thus leading to different clusters for these images although they represent the same injected failure scenario (blurriness).

Finally, we notice that the pipelines using HDBSCAN and DBSCAN (Node 3) as a clustering algorithm yield purer clusters (min = 25%, median = 40%, max = 80%) than those using K-means (Node 4, min = 22%, median = 27%, max = 29%). This is because K-means faces difficulty dealing with non-convex clusters. A cluster is convex if, for every pair of points belonging to it, it also includes every point on the straight line segment between them [132], which gives the cluster a spherical form. Nevertheless, in many practical cases, the data leads to clusters with arbitrary, non-convex shapes. Such clusters, however, cannot be appropriately detected by a centroid-based algorithm (e.g., K-means), as they are not designed for arbitrary-shaped clusters.

DBSCAN and HDBSCAN are density-based clustering algorithms. They consider high-density regions as clusters (see Section 2.1.3). The root cause clusters generated by DBSCAN and HDBSCAN are arbitrary-shaped and more homogeneous (i.e., clusters with higher within-cluster similarity) with very similar images. In contrast, a convex cluster generated by K-means tends to be less dense and can group rather dissimilar images. As a result, a convex cluster is less pure than a non-convex one.

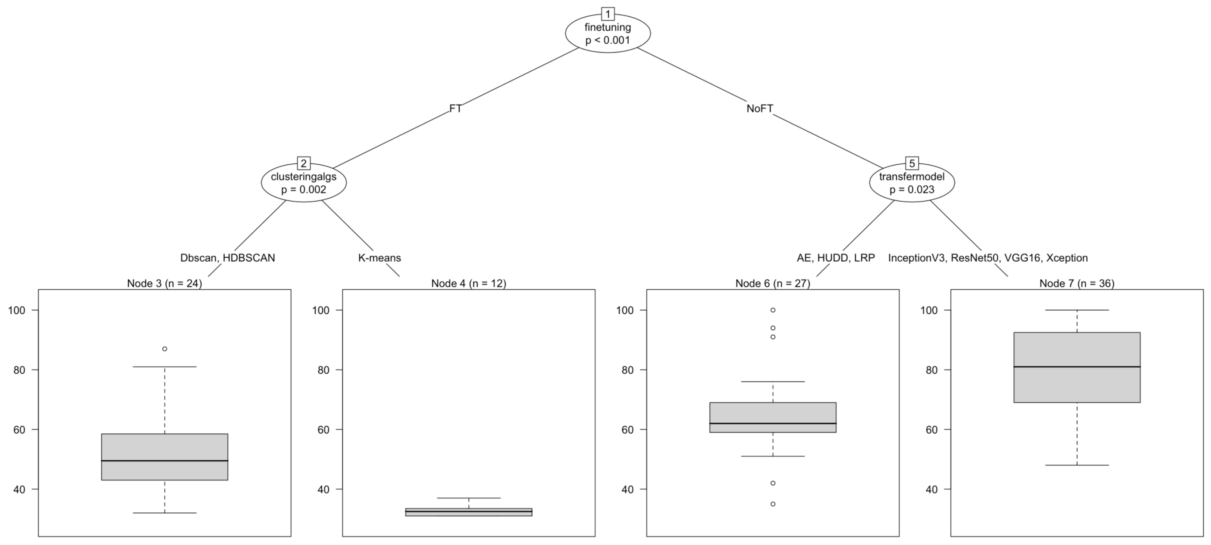


Figure 6.4: Decision Tree illustrating how the different features of a pipeline determine the average purity of root-cause clusters.

We report the significance of these results in Table 6.2, including the values of the Vargha and Delaney's $\hat{A}_{12}$ effect size and the p-values resulting from performing a Mann-Whitney U-test to compare the average purity of the pipelines using transfer learning models (Node 7 in the decision tree) and the

Table 6.3: RQ1: Pipelines with a purity greater than 90%. The last column represents the average of averages.

Pipelines					**Avg. purity across RCCs**						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
19	VGG-16	NO	None	K-Means	91.7%	92.1%	95.5%	82.5%	97.3%	99.7%	93.2%
25	VGG-16	NO	UMAP	K-Means	97.6%	84.4%	93.7%	82.4%	90.3%	97.8%	91.0%
26	VGG-16	NO	UMAP	DBSCAN	99.0%	93.0%	99.6%	79.7%	98.1%	96.6%	94.3%
39	ResNet-50	NO	None	HDBSCAN	96.4%	100.0%	100.0%	78.8%	87.5%	100.0%	93.8%
43	ResNet-50	NO	UMAP	K-Means	99.4%	93.0%	82.3%	79.6%	99.6%	97.7%	91.9%
44	ResNet-50	NO	UMAP	DBSCAN	100.0%	95.8%	95.8%	79.0%	99.7%	99.3%	94.9%
62	Inception-V3	NO	UMAP	DBSCAN	93.4%	95.2%	98.1%	76.6%	97.4%	83.1%	90.7%
100	eHUDD	NO	UMAP	DBSCAN	97.4%	98.3%	99.7%	75.9%	96.8%	97.8%	94.2%

^{FE} Feature Extraction ^{FT} Fine-tuning ^{DR} Dimensionality Reduction ^{CA} Clustering Algorithm

pipelines represented by the other nodes. Typically, an $\hat{A}_{12}$ effect size above 0.56 is considered practically significant with higher thresholds for medium (0.64) and large (0.71) effects [122], thus suggesting the effect sizes between the pipelines using transfer learning models and other pipelines are large. Further, p-values suggest these differences are statistically significant.

Table 6.2: RQ1: p-values and and effect size values when comparing the results of the pipelines with the best purity of clusters (according to the decision tree) to the other pipelines.

	Node 3	Node 4	Node 6
p-value	7e-11	2e-7	5e-6
$\hat{A}_{12}$	1.00	1.00	0.80

Finally, in Table 6.3, we report the pipelines that generated clusters with an average purity above 90% across all case study subjects, along with the purity obtained for each subject; the complete results obtained for all pipelines appear in Appendix A.1, Table A.1. An average purity of 100% means that all the clusters generated by the pipeline are pure. Interestingly, all the pipelines in Table 6.3 belong to Node 7 in Figure 6.4, thus confirming our main finding. Five of these seven best pipelines, rely on UMAP, without fine-tuning but with a transfer learning model, which is therefore our suggestion to perform root cause analysis. The best result is obtained with ResNet-50 combined with UMAP and DBSCAN.

RQ2. *Which pipelines generate root cause clusters covering more failure scenarios?*

Design and measurements.

This research question investigates the extent to which our pipelines identify all failure scenarios. We compare pipelines in terms of the percentage of injected failure scenarios being covered by at least one RCC. A failure scenario is covered by an RCC if it enables the engineer to determine the root cause of the failure. Precisely, when images belonging to a failure scenario f represent a sufficiently large share of images in a cluster C, it is easier for an engineer to notice that f is a likely root cause. Therefore, we assume that an injected failure scenario f is covered by a cluster C if it contains at least 90% of images with f. Since this threshold is relatively high, our results can be considered conservative.

Given that our injected failure scenarios are represented by the same number of images in the failure-inducing test set, every failure scenario has the same likelihood of being observed. Therefore, we expect to obtain RCCs corresponding to each failure scenario.

Experiment Results.

Figure 6.5 shows a decision tree illustrating how the different components of a pipeline determine the coverage of failure scenarios.

As for RQ1, we used the Conditional Inference Tree CTree algorithm to generate this decision tree (with the same parameter settings).

Each leaf node depicts a box plot with the distribution of the percentages of failure scenarios covered by the set of pipelines that include the components listed in the decision nodes.

For instance, Node 9 provides the distribution of the percentage of failure scenarios covered by the RCCs generated by pipelines using UMAP as a dimensionality reduction technique and non-fine-tuned transfer learning models as feature extraction methods (12 pipelines). Ideally, the root-cause clusters generated by a pipeline should cover 100% of the failure scenarios.

The decision tree in Figure 6.5 confirms RQ1 results. The pipelines without fine-tuning (Nodes 6, 8 and 9) outperform the pipelines with fine-tuning (Nodes 3 and 4). The pipelines with transfer learning models (Nodes 8 and 9) generate clusters that cover more failure scenarios than those generated by the pipelines using HUDD, LRP, and AE (Node 6). Also, the pipelines using the DBSCAN and HDBSCAN clustering algorithms (Node 3) yield better results than the ones using K-means (Node 4).

Further, the decision tree in Figure 6.5 gives us more insights into which dimensionality reduction method is more effective. We notice that the root-cause clusters generated by the pipelines using UMAP (Node 9) lead to a better distribution (min = 45%, median = 85%, max = 100%) than the pipelines using PCA or not using any dimensionality reduction (Node 8, min = 25%, median = 55%, max = 90%). This is because UMAP yields a better separation of the clusters (i.e., less clusters overlap) compared to PCA. When using UMAP, all the data points converge towards their closest neighbor (the most similar data point). Therefore, neighboring data points in higher dimensions end up in the same neighborhood in lower dimensions, resulting in a compact and well-separated clusters where it is easier for the clustering algorithms to distinguish them.

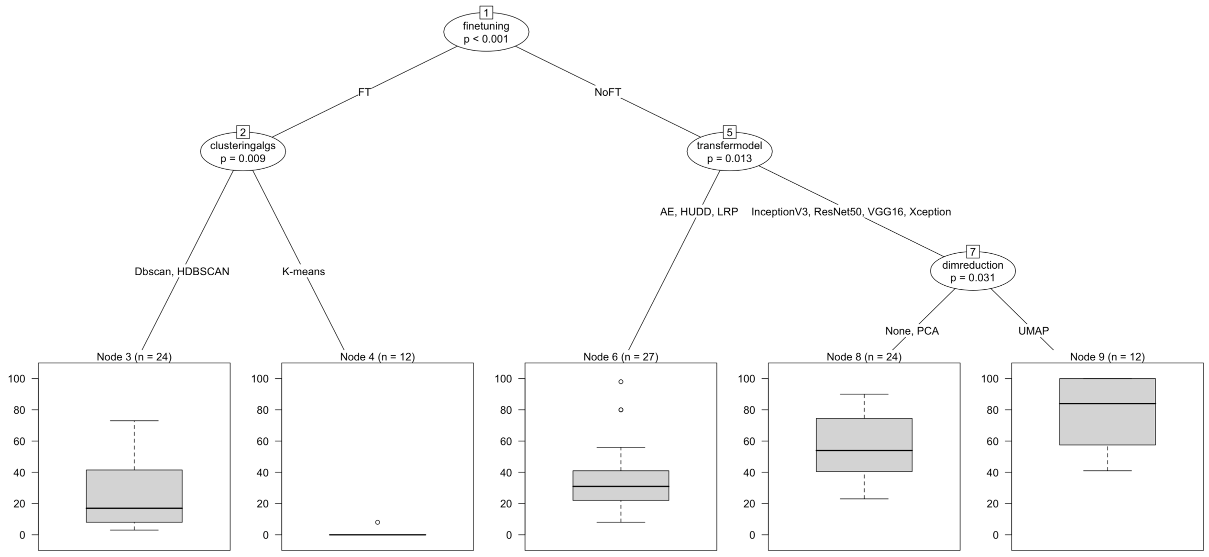


Figure 6.5: Decision Tree illustrating how the different features of a pipeline determine the coverage of the failure scenarios.

We report the significance of these results in Table 6.4, including the values of the Vargha and

Table 6.5: RQ2: Pipelines with a coverage greater than 90%. The last column represents the average of averages.

Pipelines					Percentage of covered faulty scenarios						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
26	VGG-16	None	UMAP	DBSCAN	100.0%	100.0%	100.0%	80.0%	100.0%	100.0%	96.7%
44	ResNet-50	None	UMAP	DBSCAN	100.0%	100.0%	100.0%	60.0%	100.0%	100.0%	93.3%
62	Inception-V3	None	UMAP	DBSCAN	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
80	Xception	None	UMAP	DBSCAN	100.0%	100.0%	100.0%	40.0%	100.0%	100.0%	90.0%
100	eHUDD	None	UMAP	DBSCAN	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

^{FE} Feature Extraction ^{FT} Fine-tuning ^{DR} Dimensionality Reduction ^{CA} Clustering Algorithm

Delaney's $\hat{A}_{12}$ effect size and the p-values resulting from performing a Mann-Whitney U-test to compare the percentages of covered failure scenarios resulting from the pipelines using UMAP (Node 9 in the decision tree in Figure 6.5), and the other pipelines. Table 6.4 shows that the p-values, when comparing the pipelines using UMAP to the other pipelines, are always below 0.05. This implies that in all the cases, differences are statistically significant with large effect sizes (above 0.77).

Table 6.4: RQ2: p-values and and effect size values when comparing the results of the pipelines with the best coverage of the faulty scenarios (according to the decision tree) to the other pipelines.

	Node 3	Node 4	Node 6	Node 8
p-value	1e-5	1e-5	4e-5	8e-3
$\hat{A}_{12}$	0.95	1.00	0.91	0.77

In Table 6.5, we report the pipelines that generated clusters covering at least 90% of the failure scenarios across all case study subjects, along with the coverage obtained for each case study subject (complete results for all the pipelines are reported in Appendix A.2, Table A.2). If the coverage is equal to 100%, all the failure scenarios are covered by the RCCs. Unsurprisingly, the pipelines in Table 6.5 belong to Node 7 in Figure 6.5: they rely on a non-fine-tuned transfer learning model for feature extraction, and UMAP for dimensionality reduction. Further, they all use DBSCAN for clustering. These pipelines consistently yielded the best results for all individual case studies (confirming the results obtained in RQ1).

Such findings are further supported by the results in Table A.1 and Table A.2, where we notice that the combination of UMAP with DBSCAN always achieves higher purity and coverage (in bold) than its alternatives, regardless of the used feature extraction method.

RQ3. *How is the quality of root cause clusters generated affected by infrequent failure scenarios?*

Design and measurements.

We study the effect of infrequent failure scenarios on the quality of the RCCs generated by the pipelines. We consider a failure scenario infrequent when it is observed in a low proportion of the images in the failure-inducing set. To be practically useful, a good pipeline should be able to generate root-cause clusters even for infrequent failure scenarios; indeed, in safety-critical contexts, infrequent failure scenarios may lead to hazards and thus should be detected when testing the system. For instance, if only five out of hundred failure-inducing images belong to a failure scenario and we have three failure scenarios in total, a robust pipeline should still generate an RCC containing only the images of the infrequent failure scenario.

We generate 10 different failure-inducing sets for each case study subject (a total of 60 failure-inducing sets). To construct a failure-inducing set, for each root cause that might affect the case study (see Table 6.1, Page 123), we generate a number n of images affected by the injected root cause. We randomly select a number n that is lower than the number of images selected for the same root cause in RQ1. Further, for classifier DNNs, we select a value higher than the number of classes of the corresponding case study (we enforce one root cause of failures for one image per class, at least); for regression DNNs, we select a value above 2. Since n is randomly selected (uniform distribution), we obtained failure-inducing sets containing failure scenarios whose number vary. In addition, we also include a randomly selected number of images belonging to natural failure scenarios, to mimic what happens in practice (see RQ1). The number of images belonging to natural failure scenarios varies between two and the total number of injected failure scenario images.

In total, we generated 60 failure-inducing sets (10 $\times$ 6 subject DNNs). For each failure-inducing set, we randomly selected the number of images representing a failure scenario (injected or natural scenario). Such number should be higher than the number of classes (to ensure that there is at least one scenario for each class) and lower than the total number of images representing a failure scenario generated in RQ1 (see Table 6.1). In the case of regression DNNs, the minimum number of images representing a failure scenario is set to 2 since a cluster is formed by grouping at least two images. For instance, the number of images representing a failure scenario for each failure-inducing set of the HPD case study (9 classes) is randomly selected between 9 and 90 (see Table A.3, Appendix A.3).

Since we aim to study the effect of infrequent failure scenarios on the quality of the generated RCCs, we categorize our 290 failure scenarios into *infrequent* and *frequent*. Infrequent failure scenarios are the ones that include a proportion of injected images that is lower than the median proportion in all the generated failure-inducing sets (equals to 18% in our study). For example, noise is frequent in the dataset GD_1 ($64 > 18$) but infrequent in the dataset OC_2 ($4 < 18$).

We consider only the best pipelines resulting from the experiments in RQ1 and RQ2 (i.e., having purity or coverage above 90% as shown in Tables 6.3 and 6.5); they are pipeline 26 (*VGG16/DBSCAN/UMAP/NoFT*), 44 (*ResNet50/DBSCAN/UMAP/NoFT*), 62 (*InceptionV3/DBSCAN/UMAP/NoFT*), 19 (*VGG16/K-means/None/NoFT*), 25 (*VGG16/K-means/UMAP/NoFT*), 39 (*ResNet50/HDBSCAN/None/NoFT*), 43 (*ResNet50/K-means/UMAP/NoFT*), and 80 (*Xception/DBSCAN/UMAP/NoFT*). The first three pipelines (i.e., 26, 44, 62) were the best for both RQ1 and RQ2, the next four (i.e., 19, 25, 39, 43) were selected based on RQ1 results while the latter (i.e., 80) based those of RQ2. We compute the purity and coverage of the RCCs generated by each of these pipelines, following the same procedures adopted for RQ1 and RQ2. We then compare the distribution of purity and coverage for infrequent and frequent failure scenarios. The most robust pipelines are the ones being affected the least, in terms of purity and coverage, by infrequent failure scenarios.

Experiment Results.

In Figure 6.6, for each selected pipeline, we report the average purity across all the RCCs[1] with the injected failure scenarios having a certain frequency. The x-axis reports the proportion of images for

[1]As discussed in Section 6.3.3. The red vertical line represents the median frequency of failure scenarios. We say that an RCC is associated with (or captures) an injected failure scenario f when the majority of the images in the cluster belong to scenario f.

Table 6.6: RQ3: p-values and effect size values when comparing the purity of the best pipelines with the frequent and infrequent failure scenarios.

Pipelines	26	44	62	39	80	19	25	43
Average Purity for infrequent failure scenarios	94%	87%	91%	79%	87%	76%	70%	65%
Average Purity for frequent failure scenarios	100%	100%	100%	92%	99%	96%	96%	93%
p-value	4e-6	2e-10	1e-6	2e-9	8e-9	8e-5	2e-10	3e-14
$\hat{A}_{12}$	0.58	0.64	0.59	0.60	0.63	0.68	0.70	0.75

Table 6.7: RQ3: p-values and effect size values when comparing the best pipeline in Table 6.6 (i.e., Pipeline 26, *VGG16/Dbscan/UMAP/NoFT*) to the other pipelines based on the average purity of the clusters associated to infrequent failure scenarios.

Pipelines	44	62	39	80	19	25	43
p-value	0.002	0.51	4e-5	0.006	3e-12	2e-14	4e-21
$\hat{A}_{12}$	0.57	0.51	0.60	0.56	0.69	0.71	0.77

failure scenarios whereas the y-axis reports the average purity of the RCCs associated to each failure scenario.

Figure 6.6 shows that when the frequency of the failure scenarios is below the median (infrequent scenario), the cluster purity obtained by pipelines tends to significantly lower and decrease rapidly as the frequency decreases. This is expected because when a failure scenario is infrequent, the clustering algorithm tends to either cluster its images as noise or distribute them over the other clusters. For density-based clustering algorithms, images belonging to infrequent scenarios may not become core points when the identification of a core point requires more data points in their neighborhood. In such case, images belonging to infrequent scenarios will be either labeled as noise points or border points (belonging to other clusters). The same is true for K-means, where these points are usually spread across other clusters because they cannot form a cluster.

To strengthen our findings, in Table 6.6, we report the results when comparing the purity of the selected pipelines for frequent and infrequent failure scenarios; further, we report the Vargha and Delaney's $\hat{A}_{12}$ effect size and the p-values resulting from performing a Mann-Whitney U-test. We notice that for all pipelines, the difference between frequent and infrequent scenarios are significant (p-value < 0.05). However, the effect sizes for Pipelines 26, 62, 45, and 80 are small, while they are medium for Pipelines 19 and 44, which indicates that pipelines including DBSCAN (i.e., Pipelines 26, 62, 45, and 80) are much more robust to infrequent scenarios than others (i.e., the difference between frequent and infrequent scenarios is less pronounced). Actually, the pipelines using DBSCAN fare better than the rest also in the general case. Indeed, almost all the injected failure scenarios with frequency above 18% have 100% purity (see Figure 6.6); further for infrequent failure scenarios they include less data points below 100% than the other pipelines. This is because DBSCAN tends to find clusters with different sizes if these clusters are dense enough; K-means, instead, derives clusters that are of similar size.

Further, we notice that the purity of the clusters generated by Pipeline 26 (*VGG16/Dbscan/UMAP/NoFT*), for infrequent failure scenarios, is higher (average is 94%) than the purity of the clusters generated by the other pipelines; differences are significant (see Table 6.7), thus suggesting Pipeline 26 might be the best choice.

Table 6.8: RQ3: Fisher exact test values when comparing the coverage of the lowly represented and highly represented faulty scenarios by the clusters generated by the best pipelines.

Pipelines	26	44	62	39	80	19	25	43
Average Coverage for infrequent failure scenarios	85%	71%	82%	66%	73%	51%	46%	34%
Average Coverage for frequent failure scenarios	100%	98%	99%	86%	98%	86%	87%	77%
Fisher's Exact test	1e-5	1e-5	1e-5	1e-5	1e-5	2e-4	1e-5	1e-5

Table 6.9: RQ3: Fisher exact test values when comparing the best pipeline "VGG16/Dbscan/UMAP/NoFT" to the other pipelines based on the coverage of the infrequent failure scenarios.

Pipelines	44	62	39	80	19	25	43
Fisher's Exact test	4e-2	0.55	1e-5	2e-3	0.018	1e-5	1e-5

Concerning *coverage*, Figure 6.7 shows, for each pipeline, histograms with the average coverage obtained for failure scenarios having proportions of failure inducing images within specific ranges. In general, we observe that coverage is higher for frequent scenarios. This is due to the correlation between pure clusters and coverage; the less pure the generated clusters, the fewer failure scenarios they cover. When the failure scenarios are infrequent, their images are distributed over the other clusters, reducing their purity and, thus, reducing the probability of these scenarios being covered. To demonstrate the significance of the difference between coverage results obtained with frequent and infrequent scenarios, we apply the Fisher's Exact test[2] to compare the coverage of frequent and infrequent scenarios for the clusters generated by the selected pipelines. We report the p-values resulting from the Fisher's Exact test in Table 6.8 and observe that differences are statistically significant thus indicating that pipelines perform better with frequent failure scenarios.

Further, Figure 6.7 shows that pipeline 62 (*InceptionV3/DBSCAN/UMAP/NoFT*) is the one performing best with the least frequent scenarios (i.e., range $0-5\%$) but no pipeline fares well in that range. Pipeline 26 (*VGG16/DBSCAN/UMAP/NoFT*) is the one performing best with infrequent scenarios in the range 5% to 20%; indeed, it is the only pipeline providing an average coverage above 90% for that range. To further demonstrate the significance of the difference in performance between Pipeline 26 and the other pipelines, we apply Fisher's exact test to the coverage obtained for infrequent scenarios. We report the p-values resulting from this test in Table 6.9. We notice that all the p-values are below 0.05 except when Pipeline 26 is compared to Pipeline 62; indeed, the results of these two pipelines are similar as visible in Figure 6.7), even though Pipeline 26 performs slightly better on average.

In conclusion, infrequent failure scenarios affect both purity and coverage; however, some pipelines fare better than others. Our results suggest that the pipeline (26) relying on a non-fine-tuned VGG16 model, with UMAP and DBSCAN (Pipeline 26) is the best choice since it yields significantly higher purity and coverage than the other pipelines. Pipeline 26 is also less negatively affected by infrequent failure scenarios since coverage is above 90% when the frequency is above 5%, which is not the case for all the other pipelines.

[2]The Fisher's Exact test [173] is a statistical test used to determine if there is a non-random association between two categorical variables [174].

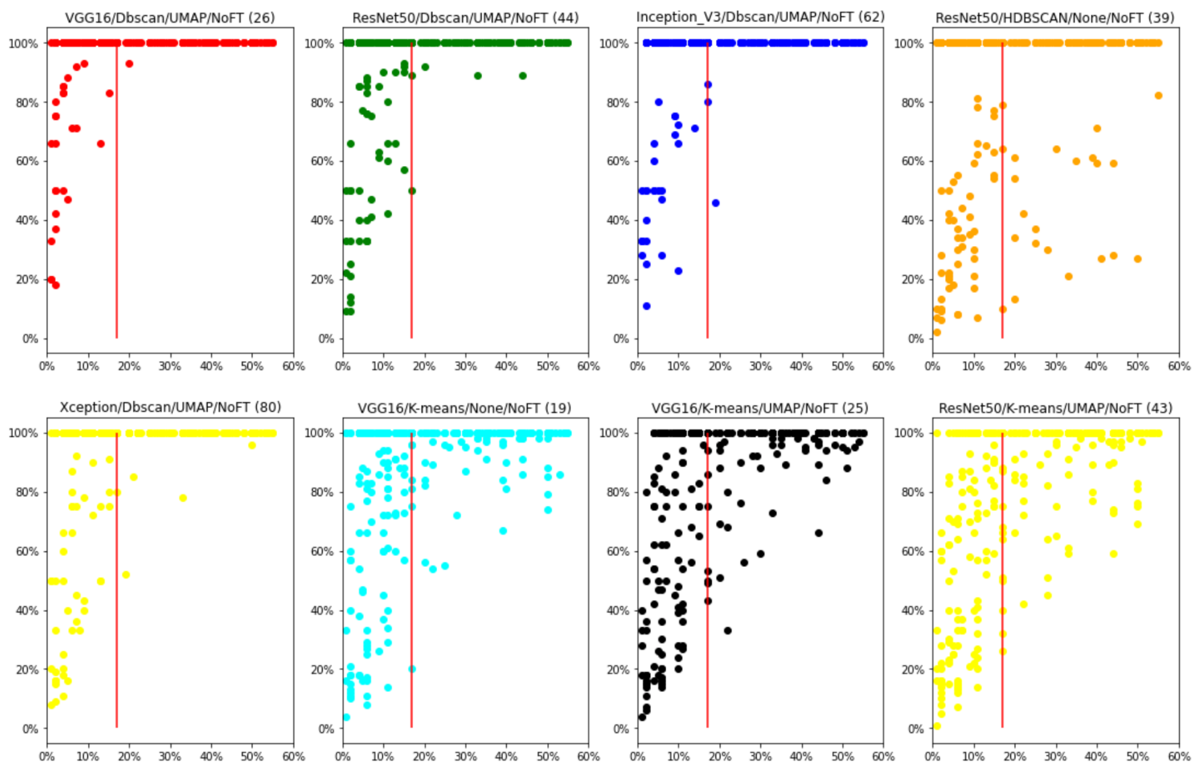


Figure 6.6: Purity of the clusters associated with frequent and infrequent failure scenarios. The x-axis captures the frequency of a failure scenario (i.e., proportion of failure-inducing images for a failure scenario). Each data point is the average of all the RCCs associated to one distinct failure scenario. The red vertical line represents the median frequency of failure scenarios.

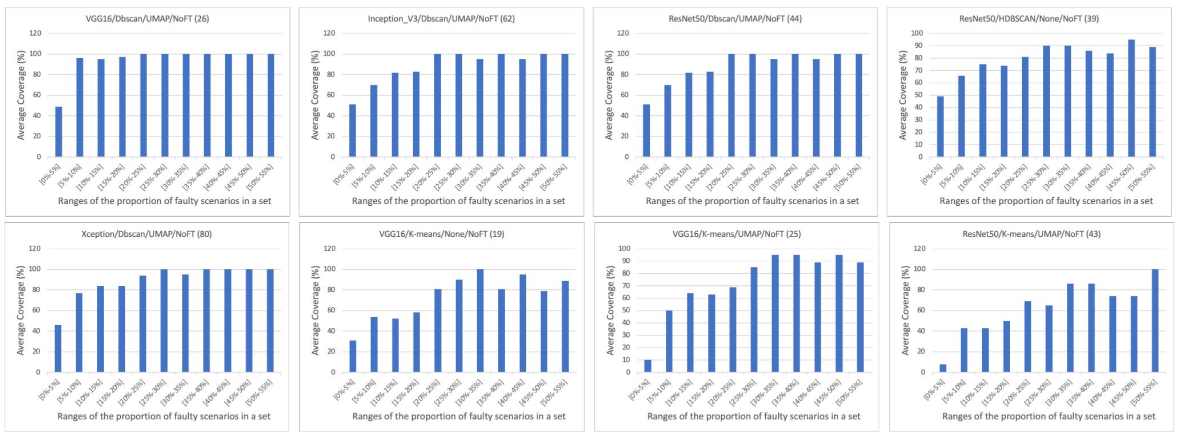


Figure 6.7: Comparing the percentage of coverage across different ranges of proportions of failure scenarios in each set.

Discussion

The results of RQ1 and RQ2 show that there is a family of pipelines leading to higher purity (i.e., they simplify the identification of root causes) and coverage (i.e., they enable the identification of all root causes). Such pipelines rely on transfer learning, UMAP for dimensionality reduction, DBSCAN for clustering, and are not using fine tuning. Among such pipelines, considering that it is reasonable to expect unsafe scenarios to be infrequent, based on the results of RQ3, we suggest to use the pipeline relying on

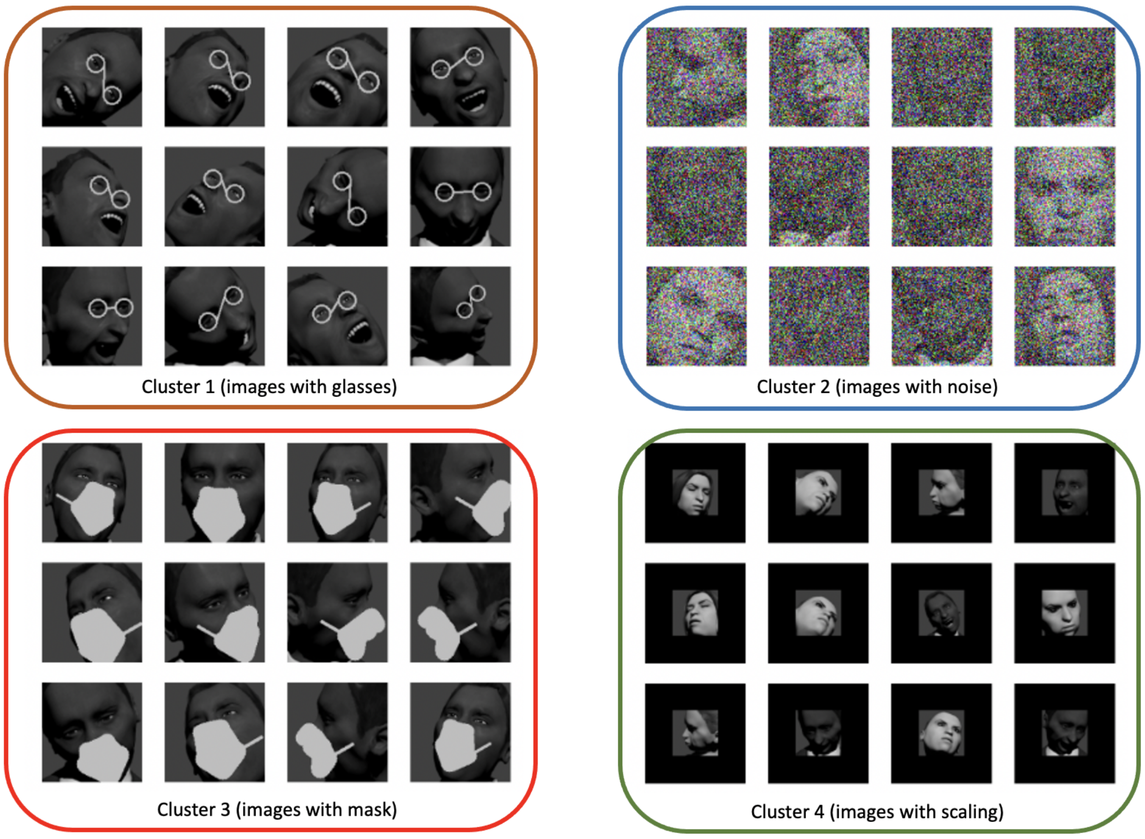

Cluster 1 (images with glasses)
Cluster 2 (images with noise)
Cluster 3 (images with mask)
Cluster 4 (images with scaling)

Figure 6.8: Examples of clusters generated by Pipeline 26 for the HPD case study subject.

VGG16 (Pipeline 26) as transfer learning model.

In our study, we focused on effectiveness, not cost; indeed, our main purpose is to identify the pipeline that generates clusters that do not confuse the end-user (i.e., they are pure) and is likely to help identify all the root causes of failures (i.e., they have high coverage). In contrast, cost is related to the number of clusters being inspected. However, our root cause analysis toolset [19] includes the generation of animated GIFs, one for each cluster, thus enabling the quick visualization of all the images in a cluster. With such toolset we conjecture that the number of clusters' images does not strongly impact cost as all the images are, anyway, quickly visualized. What is important, instead, is the purity of clusters as with low purity the end-user will not find it easy to determine commonalities among images.

Nevertheless, to further discuss cost, we measure the number of clusters to be inspected for each pipeline considering the dataset used for RQ1 and RQ2. We count only clusters capturing the injected failure scenarios since for the others we cannot precisely determine what is the expected number of clusters. A lower number of clusters should indicate lower cost and, since a number of clusters higher than the number of failure scenarios to be discovered implies the presence of redundant clusters, we compute the degree of redundancy as:

$$redundancy\ ratio = \frac{number\ of\ clusters}{covered\ failure\ scenarios}$$

Finally, to discuss how well each pipeline improves current practice in industry, we estimate the degree of savings with respect to the such practice, which entails the visual inspection of all images. To do so, we assume that inspecting a single cluster using animated gifs is as inexpensive as visualizing one

Table 6.10: The number of redundant clusters generated by the best pipelines for each case study subject and across all of them. The last columns represent the number and the percentage of failure scenarios covered by the pipelines, the redundancy ratio, and the savings.

Pipelines	**Number of generated clusters**							**Covered failure scenarios (percentage %)**	**Redundancy ratio**	**Savings**
	GD	HPD	OC	SVIRO	CPD	SAP	TOTAL			
19	3	5	3	3	3	2	19	17 (59%)	1,12	0,99
25	4	8	2	4	3	3	24	20 (69%)	1,20	0,99
43	3	4	3	2	2	4	18	15 (52%)	1,20	0,99
26	26	77	13	8	13	37	174	28 (97%)	6,21	0,91
44	42	51	5	10	27	44	179	27 (93%)	6,63	0,91
62	28	60	7	9	15	42	161	29 (100%)	5,55	0,92
80	33	30	9	2	9	14	97	25 (86%)	3,88	0,95
39	14	171	15	7	74	3	284	24 (83%)	11,83	0,86

single image. Indeed, though clusters involve several images, through animation, they actually make it easier to quickly identify commonalities rather than guessing root causes from a single image. Figure 6.8 shows four example clusters where all the images present a commonality (i.e., the root cause of the DNN failure) that is easy to determine when visualizing all the images in a sequence. Therefore, we estimate savings as:

$$savings = 1 - \frac{number\ of\ clusters}{number\ of\ images}$$

Table 6.10 shows our results; it reports the number of RCCs generated for each case study DNN and across all of them. Further, it reports the *percentage and number of failure scenarios* covered by each pipeline (used to compute redundancy and providing information about the effectiveness of a pipeline), along with *redundancy ratio* and *savings*. We report only the results for the best pipelines identified when addressing RQ1 to RQ2 because there is no reason to select pipelines that do not achieve high purity and coverage.

The number of clusters generated by the selected pipelines ranges between 18 and 284. The pipelines leading to the lowest number of clusters are the ones including K-means: *ResNet50/K-means/UMAP/NoFT* (18), *VGG16/K-means/None/NoFT* (19), and *VGG16/K-means/UMAP/NoFT* (24). Pipelines with DB-SCAN and HDBSCAN lead to a much higher number of clusters. To discuss the practical impact of such a high number of clusters, we focus on the *redundancy ratio*, which ranges between 1.12 and 11.8; the redundancy ratio indicates that the pipeline with the highest number of clusters (i.e., *ResNet50/HDBSCAN/None/NoFT*), on average, presents 11 redundant clusters for each identified failure scenario. Given that, in the presence of pure clusters, understanding the scenario captured by one pipeline is quick with animated gifs, we consider that inspecting 11 redundant clusters per fault has a limited impact on cost. Finally, if we focus on *savings*, we can observe that respect to current practice, all the pipelines except (*ResNet50/HDBSCAN/None/NoFT*) lead to savings above 90%, thus showing that their adoption is highly beneficial.

Although the pipelines including K-means lead to the lowest cost, their coverage is particularly low for infrequent scenarios (see Table 6.8, with coverage below 35% for the range $[0 - 5]$, and below 60% for the range $[5 - 10]$), which is bound to be a common situation in practice. Since pipelines leading to a small number of clusters can be highly ineffective in realistic safety-critical contexts (i.e., when some failure causes are infrequent), assuming that redundant clusters are easy to manage, we conclude that the best choice are the pipelines that maximize purity and coverage, as discussed above (i.e., Pipeline 26, *VGG16/DBSCAN/UMAP/NoFT*). A possible tradeoff is Pipeline 80 (*Xception/DBSCAN/UMAP/NoFT*), which is among the best performing for RQ3 (e.g., coverage above 40% for the range $[0-5]$, and above 70% for the range $[5-10]$) and leads to 3.6 redundant clusters only, on average.

6.3.4 Improvements to the white-box pipeline (extended-HUDD)

Since the results discussed in the previous section show that white-box approaches, despite using internal information about the DNN, perform worse than black-box approaches, we extended HUDD to fully exploit the potential of a white-box analysis. We refer to such method as extended-HUDD (eHUDD). HUDD's extensions concern two aspects:

- RCC generation, which is based on additional information (i.e., neuron activations);
- Identification and removal of impure clusters, which is not present in HUDD.

Below, we provide an overview of eHUDD extensions and compare its results with the ones obtained by the other pipelines. We extended the assessment of RQ1, RQ2, and RQ3 to include also eHUDD.

eHUDD extensions.

RCC generation eHUDD takes in consideration also (forward-propagated) neuron activations for each layer, in addition to the back-propagated relevance generated by LRP. The extension concern Step 1 of HUDD (see Section 3.2.1), as shown in Figure 6.9. It is worth noting that eHUDD does not introduce additional cost because forward-propagated neuron activations are already stored to compute LRP.

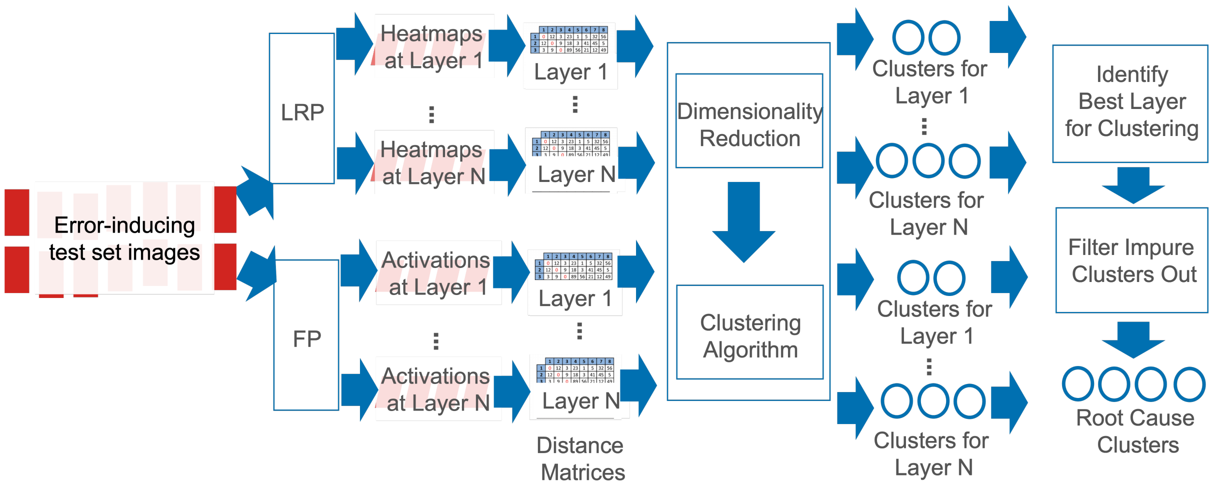


Figure 6.9: The eHUDD workflow.

As shown in Figure 6.9, during DNN testing, for each failure-inducing input image the DNN should store the neuron activations observed for each neuron of every layer; one *activation matrix* is generated for each layer. Then, we normalize each matrix using min-max normalization and prune out the least 50% neurons. By normalizing the matrices, we can ensure that each neuron's activation value is represented proportionally to its importance to the DNN's output. Additionally, pruning out the least important neurons can reduce the dimensions of the matrices, which can simplify the clustering process by reducing the computational complexity. Moreover, it can also help remove any irrelevant or noisy information from the matrices, which can improve the overall performance of the clustering. Subsequently, the dimensions of these *activation matrices* are reduced, and a distance matrix is computed for each layer. The provided distance matrix for each layer is then used to generate clusters. Note that the same process is performed for heatmaps, with the distance matrices generated from the reduced dimensions of the normalized pruned heatmap, and clusters generated accordingly.

The best layer is then selected based on the clustering results obtained with both heatmaps and activation matrices. Precisely, to select the best layer, we adopt the same strategy used by HUDD (see Section 3.2.1), which is to compute the weighted intra-cluster distance of each layer. The layer with the highest similarity score (lowest intra-cluster distance) is then selected for clustering.

Identification and removal of impure clusters. To further reduce the effort of inspection by engineers during safety analysis process, eHUDD includes a solution to improve the purity of the generated clusters. Although clusters' purity can be correctly computed only when ground truth information is available, which is what we do in empirical evaluations, we assume that internal evaluation metrics for clustering, in particular the Silhouette score, can provide a useful estimate for the purity of a cluster. Indeed, the Silhouette score measures how well each data point in a cluster fits with the other data points in the same cluster as opposed to data points in other clusters. Therefore, if the Silhouette score for a set of clusters is low, it is likely that the clusters are relatively impure, contain dissimilar data points or overlapping.

Precisely, we compute the average Silhouette index of the datapoints in each cluster (please refer to Equation 5.2.1). Then we remove the clusters with an average Silhouette score below the 25^{th} percentile of the average Silhouette score across clusters. Note that the Silhouette index is computed based on the distances between images pixels, rather than the internal information of the DNN used for clustering.

Removing impure clusters is particularly useful in the context of safety-critical systems, where even a human error when performing safety analysis can have significant consequences. For this reason having too many clusters to inspect can be overwhelming and increase the likelihood of a human error. By filtering out clusters with lower quality, engineers can focus on clusters with a clear interpretation. If the removal of clusters do not decrease the coverage of failure scenarios, which is demonstrated in our experiments (see Section 6.3.4), the quality of safety analysis could be improved while reducing the overall cost of inspection.

eHUDD assessment.

To assess eHUDD, we rely on UMAP as a dimensionality reduction technique and DBSCAN as a clustering algorithm since such pipeline has been shown to outperform other alternatives with respect to purity and coverage, regardless the used feature extraction method. We rely on the same experiment design and metrics adopted for RQ1, RQ2, and RQ3.

Table 6.11: Purity and Coverage obtained by the clusters of the white-box approach (eHUDD). The last column represents the average of averages.

Pipelines					Avg. purity across RCCs						
#	FE	FT	DR	CA	GD	OC	HPD	SVIRO	SAP	CPD	Avg.
100	eHUDD	NO	UMAP	DBSCAN	97.4%	98.3%	99.7%	75.9%	96.8%	97.8%	94.2%
Pipelines					Percentage of covered failure scenarios						
100	eHUDD	NO	UMAP	DBSCAN	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

^{FE} Feature Extraction ^{FT} Fine-tuning ^{DR} Dimensionality Reduction ^{CA} Clustering Algorithm

The results of the experimental evaluation of the extended version of HUDD (eHUDD) are highly promising and demonstrate significant improvements over the previous white-box approach. In particular, the combination of generating forward-propagation activations for each layer and filtering out redundant clusters using the silhouette index represents a substantial improvement to the approach, resulting in improved purity, coverage and robustness.

Table 6.11 reports the results for RQ1 and RQ2, they show that eHUDD achieves similar or even better results than the best black-box approach (Pipeline 26). Specifically, the results show that eHUDD achieves an average purity of 94.2% compared to Pipeline 26's purity of 94.3% for RQ1 (*p-value* $= 0.54$), and achieves a perfect coverage of 100.0% compared to Pipeline 26's coverage of 96.7% for RQ2 (*p-value* $= 0.042$).

Concerning RQ3, we report in Figure 6.10 the purity of the clusters and the coverage associated with frequent and infrequent failure scenarios. For infrequent failure scenarios, eHUDD performs better than Pipeline 26 in terms of coverage, with 95% of the failures scenarios being covered versus 85% for Pipeline 26 (*p-value* $= 0.031$). This is a relevant difference because, in safety contexts maximizing the number of failure scenarios being identified is critical thus justifying the higher cost required by white-box analysis. The higher coverage comes at the cost of a lower purity, with eHUDD having a purity of 92% compared to Pipeline 26's purity of 94% (*p-value* $= 0.57$); however, a few percentage points indicate that a very limited number of images (around one image per cluster, given that the average cluster size is 13 images) inappropriately appear in each cluster, which should have a very limited impact on the cost of the visual inspection of clusters.

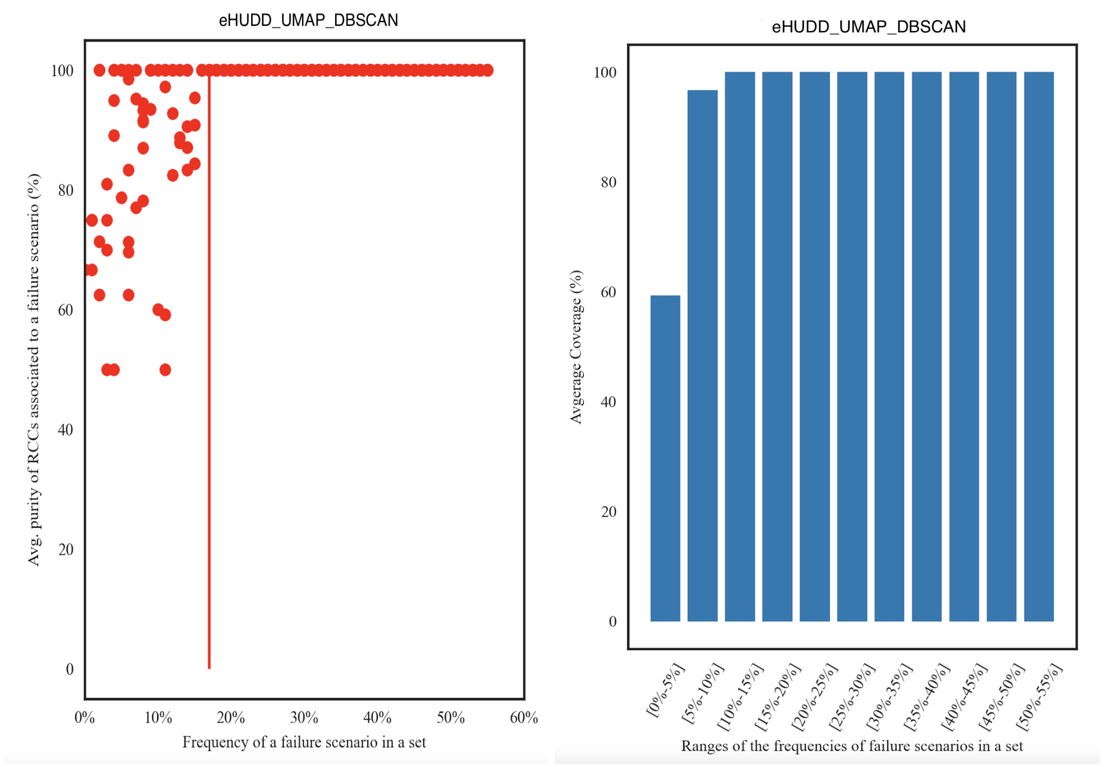


Figure 6.10: Purity of the clusters (left) and the coverage (right) associated with frequent and infrequent failure scenarios. The x-axis captures the frequency of a failure scenario (i.e., proportion of failure-inducing images for a failure scenario). Each data point is the average of all the RCCs associated to one distinct failure scenario. The red vertical line (left) represents the median frequency of failure scenarios.

Table 6.12: The number of redundant clusters generated by the white-box approach (eHUDD) for each case study subject and across all of them. The last columns represent the number and the percentage of failure scenarios covered by the pipelines, the redundancy ratio, and the savings.

Pipeline	**Number of generated clusters**							**Covered failure scenarios (percentage %)**	**Redundancy ratio**	**Savings**
	GD	HPD	OC	SVIRO	CPD	SAP	TOTAL			
100	19	56	13	25	38	34	185	29 (100%)	6,37	0,91

Finally, we report in Table 6.12 the number of clusters generated by eHUDD for each case study. The eHUDD pipeline generates, in total, across all the case study subjects, 11 more clusters than Pipeline 26 (185 vs 174); however, such difference is practically negligible. Indeed, assuming five images are needed to inspect each cluster and one minute to inspect 100 images – given that our tool generates clusters in the form of GIFs where many images can be quickly inspected (see Sections 5.3.2 and 3.2.2) – this lead to an average difference of 20 seconds per case study subject.

Note that filtering redundant clusters out using the Silhouette index can be applied to both white-box and black-box approaches. This strategy has been shown to significantly improve the purity of the clustering results by removing clusters with lower similarity scores. By only retaining the most representative and informative clusters, we can obtain a more accurate identification of failure scenarios. Although the strategy improves the purity of clusters when applied to the black-box approach (from 94.3% to 95.7%), it negatively affects the coverage (from 96.7% to 92.5%). This discrepancy in performance could be attributed to the fact that, in Pipeline 26, the Silhouette index is calculated using the same criterion as the clustering process, whereas eHUDD employs clustering based on the internal information of the model. As a result, the Silhouette index may complement the clustering in eHUDD, leading to a different outcome.

To conclude, eHUDD results show that the combination of generating forward-propagation activations for each layer and filtering out redundant clusters using the Silhouette index represents a significant improvement to the white-box approach, and the improvement in coverage may justify its adoption despite higher execution costs mainly related to the generation and storage of heatmaps (see Section 3.3.2).

6.3.5 Threats to Validity

Internal validity Since 72 of our 100 pipelines use a Transfer Learning pre-trained model to extract the features, a possible internal threat is that this model can negatively affect our results if inadequate. Indeed, clustering relies on the similarity computed on the extracted features. To mitigate this threat, we visually inspected the clusters to check their consistency. Having consistent clusters means the features extracted by the models contain enough information to cluster the images based on their similarity.

Another potential threat might be that the dataset (with the injected faults) was created with the proposed approach in mind. Therefore, there might be a risk of bias. To mitigate this risk, all the methods used in our pipelines (feature extraction methods, clustering algorithms, dimensionality reduction techniques) are independent of the data. These methods do not require any a priori knowledge on the data. We also publish our data to further mitigate this risk. All the experiments can be reproduced with any injected faulty scenario.

External validity To alleviate the threats related to the choice of the case study DNNs, we use six well-studied datasets with diverse complexity. Four out of six subject DNNs implement tasks motivated by IEE business needs. These DNNs address problems that are quite common in the automotive industry. The other two DNNs are also related to the automotive industry and were used in many Kaggle challenges [162, 175].

Although our pipelines were only tested on case study DNNs related to the automotive industry, we believe they will perform well with other data sets. This is because the models used for the feature extraction were pre-trained on ImageNet, which means that the model can capture features related to $1,000$ classes, including humans, animals, and objects. As for AE, it can learn the aspects of any data set during training and provide high-quality clusters. Finally, for HUDD and LRP, the extraction of heatmap-based features is performed on well-known layer types that are part of any DNN model, regardless of the task at hand (i.e., they can be extended to DNNs that were not studied in this work).

Construct validity The construct considered in our work is effectiveness. We measure the effectiveness through complementary indicators as follows:

For RQ1, we evaluate the effectiveness of our pipelines by computing the purity of the generated clusters. The purity of a cluster is measured as the maximum proportion of images representing one faulty scenario in this cluster.

For RQ2, we evaluate the effectiveness of our pipelines based on the coverage of the injected faulty scenarios by the root cause clusters. A faulty scenario is covered by a cluster if at least 90% of the images in this cluster represent such faulty scenario.

Finally, for RQ3, we consider both the purity and the coverage to measure the robustness of the top-performing pipelines to rare faulty scenarios.

Conclusion validity Conclusion validity addresses threats that impact the ability to conclude appropriately. To mitigate such threats and to avoid violating parametric assumptions in our statistical analysis, we rely on a non-parametric test and effect size measure (i.e., Mann Whitney U-test and the Vargha and Delaney's $\hat{A}_{12}$ statistics, respectively) to assess the statistical significance of differences in our results. Additionally, we applied the Fisher's exact test when comparing coverage results related to different distributions of faulty scenarios (i.e., RQ3), which is commonly used in similar contexts. All results were reported based on both purity and coverage parameters, and six datasets were analyzed during our experiments.

6.4 Conclusion

In this chapter, we presented an large-scale empirical evaluation of 99 different pipelines for root cause analysis of DNN failures. Our pipelines receive as input a set of images leading to DNN failures and generate as output cluster of images sharing similar characteristics. As demonstrated by our previous work, by visualizing the images in each cluster, an engineer can notice commonalities across the images in each cluster; such commonalities represent the root causes of failures, help characterize failure scenarios and, thus, support engineers in improving the system (e.g., by selecting additional similar images to retrain the DNN or by introducing countermeasures in the system).

We considered 99 pipelines resulting from the combination of five methods for feature extraction, two techniques for dimensionality reduction and three clustering algorithms. Our methods for feature extraction include white-box (i.e., heatmap generation techniques) and black-box approaches (i.e., fine-tuned and non-finetuned transfer learning models). Additionally, we rely on PCA and UMAP for dimensionality reduction and K-means, DBSCAN, and HDBSCAN for clustering.

We evaluated our pipelines in terms of clusters' purity and coverage of failures based on failure scenarios widely varying in terms of frequency, thus analyzing the impact of rare scenarios on our best pipelines. Based on six case study subjects in the automotive domain, our results show that the best results are obtained with pipelines relying on eHUDD (extended version of HUDD) as a feature extraction model, leveraging UMAP as a dimensionality reduction technique, and using DBSCAN as clustering algorithm. When the failure scenarios are equally distributed, the best pipeline achieved a purity of 94.2% (i.e., almost all the images in RCCs present the same failure scenario) and a coverage of 100.0%. The same pipeline also performs well with rare failure scenarios; indeed, when images belonging to failure scenarios represent between 5% and 10% of the total number of images, it still can cover 97% of the failure scenarios with a cluster purity above 88%.

Chapter 7

Conclusion & Future Prospects

In this PhD Dissertation, we examined automated debugging and repair of AI-based automotive software systems through a comprehensive study conducted in collaboration with our industrial partner in the automotive domain, IEE Sensing. We introduced three tools that use white-box, black-box, and search-based techniques to explain and debug the misbehaviors of DNN-based automotive systems. We also evaluated various supervised and unsupervised learning techniques and developed interpretability experiments to extract reasoning from DNNs.

Chapter 3 presented *HUDD*, an approach that automatically identifies situations where an image-processing DNN is likely to produce erroneous results. HUDD generated clusters of misclassified input images that shared a common set of characteristics that are plausible causes for errors. This was achieved through a hierarchical agglomerative clustering algorithm applied to heatmaps capturing the relevance of neurons across different DNN layers on the result. HUDD minimized the effort required to select and label additional images to augment the training set and improved the DNN by automatically selecting images that are close to members of root cause clusters and thus unsafe. Empirical evaluation with simulator images showed that HUDD could generate clusters of images that share similar values for some of the simulation parameters driving the generation of images. These clusters can then serve as a useful instrument for the identification of root causes of DNN errors and can be effectively used to select new images for retraining.

Chapter 4 presented *SEDE*, a novel approach based on evolutionary algorithms that generates expressions characterizing hazard-triggering events observed in real-world images processed by DNNs. These expressions constrain a simulator's configuration parameters that generate images similar to the real-world images under analysis. Empirical results showed that the genetic algorithm employed by SEDE could generate images for a larger number of clusters and with a higher diversity than other methods. Additionally, the expressions generated by SEDE successfully characterized the unsafe input space, and the retraining process increased the DNN accuracy by up to 18 percentage points.

Chapter 5 presented *SAFE*, a black-box approach that automatically identifies the situations where a DNN is more likely to fail without requiring any modification to the DNN or access to its internal information. SAFE characterized such situations by generating clusters of images that likely lead to a

DNN error due to the same underlying reason. SAFE uses a pre-trained model to extract features from error-inducing images and a density-based clustering algorithm to generate arbitrary-shaped clusters. Empirical results showed that SAFE generates clusters that can effectively help engineers determine the root causes for DNN errors, and the DNNs retrained by SAFE achieved higher accuracy than alternatives.

Chapter 6 presented an empirical evaluation of the proposed methods. The study examined different pipelines for identifying RCCs in automotive scenarios and showed that the best results were obtained using an extended version of HUDD (eHUDD) as a feature extraction method, UMAP for dimensionality reduction and DBSCAN for clustering. The best pipeline achieved a purity of 94.2% and a coverage of 100.0%, and also performed well with rare failure scenarios, covering 95% of them with a cluster purity above 88%.

Future Prospects We have developed automated techniques for functional safety analysis that we demonstrated being effective in identifying root causes of failures and improving accuracy. These contributions are particularly relevant for the increasing use of DNNs in other sectors dealing with the development of safety-critical systems. However, there are opportunities for further work to extend our results.

For instance, extending the developed automated techniques for functional safety analysis to system-level testing involves integrating the DNNs within the broader safety-critical system context, incorporating realistic operational conditions, conducting fault injection testing, and engaging with domain experts. By embracing these approaches, the reliability, robustness, and safety of DNN-based systems can be effectively evaluated, leading to improved safety-critical applications across various sectors.

One potential direction is to evaluate the effectiveness and efficiency of the proposed techniques in a broader range of applications and domains. This could involve using real-world and simulated datasets to evaluate the performance of the techniques in healthcare and aviation applications. While this thesis has focused on automotive applications, it is reasonable to believe that they perform similarly for other vision-based tasks (e.g., controlling robots in factories) but there might be application scenarios that may require further developments. Indeed, the level of detail in the images processed by DNNs can impact the results of both black-box and white-box approaches. In our experiments, we used images similar to those processed by IEE for in-car sensing, where the number of subjects in each image is limited. However, other DNNs may present new challenges, particularly when processing images in low-illumination environments such as mines or space. Further research in these areas is necessary to fully evaluate the proposed techniques and their potential applicability across different domains.

Another potential direction for future work (which is also a part of an ongoing work) would be to characterize clusters of error-inducing images using concepts generated through an extended Automatic Concept-based Explanations (ACE) algorithm [176] (an algorithm that groups image segments without the need for human intervention). Integrated to HUDD, such approach may help the engineer to focus directly on the problematic concepts.

In addition to characterizing clusters of failure-inducing images, another promising avenue for future work is to present heatmaps to engineers for inspection and utilize augmented images as useful guidance for developers. By integrating these techniques into the existing system, engineers can gain valuable insights into the specific regions or features of the input images that contribute to errors or failures in the DNN predictions.

Finally, an intriguing prospect would be to investigate the potential for using the proposed techniques to detect failures at run-time. This could involve the use of our proposed clustering approaches to detect failures observed during testing in the field. By monitoring the distribution of images within the cluster, we can check if the frequency of failures is higher than expected, which may indicate a distribution shift and potential safety risks. This can serve as an early warning system to trigger appropriate responses and mitigate risks before they become more severe. Such a framework could be a valuable contribution to the field of DNNs and functional safety, as it could help to improve the reliability and safety of DNN-powered systems in safety-critical applications by detecting and addressing failures at run-time.

Ultimately, it would be beneficial to engage regulatory bodies in discussions regarding the proposed approaches to evaluate their feasibility for inclusion in safety guidelines. This would help to ensure that the proposed approaches align with safety standards and best practices in the relevant domain.

Appendix A

Chapter 6

A.1 Additional material for RQ1

Table A.1: Comparing the clusters generated by the different pipelines based on the average of the purity across root cause clusters. The last column represents the average of averages.

Pipelines					Case Study Subjects						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
1	HUDD	NO	None	K-Means	51.3%	36.6%	40.7%	62.4%	78.9%	39.9%	51.6%
2			None	DBSCAN	56.2%	53.4%	43.1%	53.0%	80.6%	63.7%	58.3%
3			None	HDBScan	68.5%	61.7%	43.7%	45.4%	51.2%	27.4%	49.6%
4			PCA	K-Means	49.1%	56.4%	40.8%	74.8%	79.7%	27.4%	54.7%
5			PCA	DBSCAN	43.4%	54.6%	48.7%	48.3%	80.6%	27.4%	50.5%
6			PCA	HDBScan	68.5%	61.7%	35.5%	45.4%	74.1%	26.9%	52.0%
7			UMAP	K-Means	56.7%	47.6%	42.3%	54.3%	61.1%	32.2%	49.0%
8			UMAP	DBSCAN	69.6%	58.7%	68.3%	59.3%	68.7%	53.9%	**63.1%**
9			UMAP	HDBScan	68.5%	61.7%	33.3%	45.4%	74.1%	27.4%	51.7%
10	LRP	NO	None	K-Means	42.9%	36.6%	56.5%	33.8%	82.8%	73.6%	54.4%
11			None	DBSCAN	39.5%	28.7%	69.8%	50.7%	96.5%	35.4%	53.4%
12			None	HDBScan	69.0%	58.3%	56.2%	47.8%	42.4%	27.3%	50.2%
13			PCA	K-Means	54.2%	56.4%	54.9%	35.7%	82.9%	73.6%	59.6%
14			PCA	DBSCAN	47.2%	20.6%	71.8%	48.9%	79.7%	35.4%	50.6%
15			PCA	HDBScan	52.5%	58.3%	25.5%	47.8%	46.8%	26.2%	42.8%
16			UMAP	K-Means	55.9%	47.6%	54.7%	31.8%	67.0%	32.2%	48.2%
17			UMAP	DBSCAN	67.0%	62.8%	68.9%	49.7%	80.3%	33.5%	**60.4%**
18			UMAP	HDBScan	69.0%	58.3%	56.2%	47.8%	51.6%	26.4%	51.5%
19	VGG-16	NO	None	K-Means	91.7%	92.1%	95.5%	82.5%	97.3%	99.7%	93.2%
20			None	DBSCAN	87.0%	85.9%	96.7%	57.3%	98.0%	100.0%	87.5%
21			None	HDBSCAN	52.6%	99.0%	30.7%	73.4%	77.8%	54.5%	64.7%
22			PCA	K-Means	90.5%	87.6%	58.3%	87.7%	94.2%	92.2%	85.1%
23			PCA	DBSCAN	90.7%	94.9%	81.0%	71.6%	96.0%	91.8%	87.7%
24			PCA	HDBSCAN	45.6%	95.1%	56.2%	93.5%	100.0%	76.0%	77.7%

Pipelines					Case Study Subjects						
#	FE	FT	DR	CA	GD	OC	HPD	SVIRO	SAP	CPD	Avg.
25	VGG-16	NO	UMAP	K-Means	97.6%	84.4%	93.7%	82.4%	90.3%	97.8%	91.0%
26			UMAP	DBSCAN	99.0%	93.0%	99.6%	79.7%	98.1%	96.6%	**94.3%**
27			UMAP	HDBSCAN	78.0%	96.7%	56.2%	88.9%	44.1%	79.9%	74.0%
28		YES	None	K-Means	26.2%	33.9%	15.8%	24.3%	27.9%	25.5%	25.6%
29			None	DBSCAN	26.4%	38.5%	18.2%	32.8%	29.6%	25.2%	28.4%
30			None	HDBSCAN	23.4%	51.1%	14.3%	48.1%	25.7%	53.2%	36.0%
31			PCA	K-Means	25.4%	37.7%	16.7%	24.5%	26.8%	25.8%	26.1%
32			PCA	DBSCAN	29.2%	51.4%	23.4%	46.6%	32.3%	29.0%	35.3%
33			PCA	HDBSCAN	22.7%	43.7%	13.8%	41.5%	25.3%	23.3%	28.4%
34			UMAP	K-Means	26.3%	33.6%	18.0%	25.3%	26.2%	26.2%	25.9%
35			UMAP	DBSCAN	45.4%	42.8%	27.0%	39.1%	43.8%	44.0%	40.4%
36			UMAP	HDBSCAN	23.5%	40.4%	14.1%	36.6%	22.0%	24.2%	26.8%
37	ResNet-50	NO	None	K-Means	84.2%	84.6%	74.0%	61.2%	86.0%	83.7%	78.9%
38			None	DBSCAN	63.5%	84.6%	87.5%	72.6%	75.5%	72.0%	76.0%
39			None	HDBSCAN	96.4%	100.0%	100.0%	78.8%	87.5%	100.0%	93.8%
40			PCA	K-Means	67.4%	79.6%	61.3%	53.4%	85.8%	75.6%	70.5%
41			PCA	DBSCAN	79.7%	72.9%	51.1%	45.0%	89.8%	80.3%	69.8%
42			PCA	HDBSCAN	40.8%	79.6%	56.2%	32.0%	42.5%	49.7%	50.2%
43			UMAP	K-Means	99.4%	93.0%	82.3%	79.6%	99.6%	97.7%	91.9%
44			UMAP	DBSCAN	100.0%	95.8%	95.8%	79.0%	99.7%	99.3%	**94.9%**
45			UMAP	HDBSCAN	82.6%	87.3%	38.8%	60.0%	30.5%	69.4%	61.4%
46		YES	None	K-Means	26.7%	37.4%	19.3%	30.0%	26.2%	25.6%	27.5%
47			None	DBSCAN	47.2%	40.9%	32.1%	33.7%	35.2%	39.4%	38.1%
48			None	HDBSCAN	55.0%	46.7%	15.4%	45.2%	26.4%	24.9%	35.6%
49			PCA	K-Means	29.5%	37.1%	17.8%	39.5%	26.6%	26.2%	29.5%
50			PCA	DBSCAN	40.1%	45.6%	23.8%	41.5%	39.4%	39.4%	38.3%
51			PCA	HDBSCAN	23.7%	50.7%	15.7%	48.2%	24.6%	23.3%	31.1%
52			UMAP	K-Means	25.5%	34.6%	17.3%	25.7%	27.5%	25.6%	26.0%
53			UMAP	DBSCAN	37.8%	54.8%	35.9%	48.4%	41.7%	50.6%	44.9%
54			UMAP	HDBSCAN	23.9%	44.5%	15.1%	23.3%	23.7%	24.2%	25.8%
55	Inception-V3	NO	None	K-Means	84.6%	86.0%	95.1%	69.2%	91.2%	87.8%	85.7%
56			None	DBSCAN	100.0%	63.2%	80.9%	17.9%	98.4%	60.4%	70.1%
57			None	HDBSCAN	62.6%	96.0%	99.8%	77.9%	62.6%	95.6%	82.4%
58			PCA	K-Means	66.5%	74.5%	80.9%	68.6%	88.9%	75.7%	75.8%
59			PCA	DBSCAN	97.9%	83.8%	86.0%	96.5%	92.0%	82.2%	89.7%
60			PCA	HDBSCAN	92.5%	86.1%	53.4%	70.8%	69.4%	34.3%	67.8%
61			UMAP	K-Means	94.1%	87.4%	95.0%	67.0%	90.1%	71.3%	84.2%
62			UMAP	DBSCAN	93.4%	95.2%	98.1%	76.6%	97.4%	83.1%	**90.7%**
63			UMAP	HDBSCAN	74.0%	83.3%	55.9%	80.1%	65.8%	70.0%	71.5%
64		YES	None	K-Means	26.6%	34.4%	15.6%	23.6%	26.7%	25.2%	25.4%
65			None	DBSCAN	50.0%	38.3%	24.7%	17.0%	51.0%	44.1%	37.5%
66			None	HDBSCAN	49.9%	48.2%	15.4%	44.4%	53.0%	53.8%	44.1%
67			PCA	K-Means	24.4%	31.5%	16.7%	25.6%	27.2%	25.2%	25.1%
68			PCA	DBSCAN	32.1%	50.6%	29.1%	38.5%	35.1%	38.6%	37.3%
69			PCA	HDBSCAN	46.7%	44.3%	15.1%	45.2%	25.0%	22.9%	33.2%
70			UMAP	K-Means	26.1%	31.0%	17.1%	25.8%	25.2%	27.2%	25.4%
71			UMAP	DBSCAN	45.7%	50.4%	23.4%	45.1%	44.4%	52.5%	43.6%
72			UMAP	HDBSCAN	47.2%	26.2%	15.3%	37.8%	24.7%	25.0%	29.4%
73	Xception	NO	None	K-Means	85.2%	90.4%	88.8%	68.4%	95.2%	72.6%	83.4%
74			None	DBSCAN	60.3%	35.2%	82.6%	34.7%	73.6%	57.0%	57.2%

Pipelines					**Case Study Subjects**						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
75	Xception	NO	None	HDBSCAN	88.9%	91.7%	85.6%	71.5%	100.0%	92.2%	88.3%
76			PCA	K-Means	75.5%	73.3%	67.2%	57.6%	83.6%	44.4%	66.9%
77			PCA	DBSCAN	78.8%	87.7%	83.6%	71.2%	93.6%	35.2%	75.0%
78			PCA	HDBSCAN	75.8%	84.5%	47.5%	75.5%	62.0%	32.7%	63.0%
79			UMAP	K-Means	93.1%	90.2%	89.8%	66.3%	92.6%	66.7%	83.1%
80			UMAP	DBSCAN	94.7%	92.7%	98.7%	62.4%	99.0%	85.4%	**88.8%**
81			UMAP	HDBSCAN	77.9%	86.3%	36.7%	78.1%	99.4%	62.4%	73.5%
82		YES	None	K-Means	25.0%	35.6%	17.1%	27.3%	25.4%	26.2%	26.1%
83			None	DBSCAN	32.3%	34.9%	22.7%	17.8%	20.2%	55.2%	30.5%
84			None	HDBSCAN	34.7%	46.2%	15.9%	33.4%	25.2%	54.0%	34.9%
85			PCA	K-Means	27.0%	35.6%	17.0%	26.2%	25.4%	36.8%	28.0%
86			PCA	DBSCAN	44.3%	31.1%	21.4%	28.2%	35.4%	35.2%	32.6%
87			PCA	HDBSCAN	46.8%	56.3%	14.8%	37.6%	24.9%	26.6%	34.5%
88			UMAP	K-Means	26.4%	32.8%	17.1%	28.7%	27.0%	24.6%	26.1%
89			UMAP	DBSCAN	36.9%	42.8%	26.7%	50.4%	46.2%	45.5%	41.4%
90			UMAP	HDBSCAN	23.1%	26.2%	14.9%	40.9%	24.7%	26.4%	26.1%
91	AE	NO	None	K-Means	40.9%	56.1%	47.0%	41.6%	72.5%	73.0%	55.2%
92			None	DBSCAN	35.2%	60.8%	11.5%	16.9%	20.3%	21.1%	27.6%
93			None	HDBSCAN	36.9%	67.8%	0.0%	67.5%	0.0%	63.0%	39.2%
94			PCA	K-Means	62.5%	55.1%	51.7%	52.9%	60.9%	77.2%	60.0%
95			PCA	DBSCAN	20.4%	73.3%	68.2%	63.6%	89.1%	76.7%	65.2%
96			PCA	HDBSCAN	73.5%	60.8%	25.7%	68.1%	76.1%	27.8%	55.3%
97			UMAP	K-Means	43.7%	46.2%	36.4%	37.6%	69.2%	66.0%	49.9%
98			UMAP	DBSCAN	65.3%	61.8%	59.0%	51.9%	69.1%	70.6%	**62.9%**
99			UMAP	HDBSCAN	28.4%	60.5%	45.4%	47.2%	41.6%	41.7%	44.1%

^{FE} Feature Extraction ^{FT} Fine-tuning ^{DR} Dimensionality Reduction ^{CA} Clustering Algorithm

A.2 Additional material for RQ2

Table A.2: Percentage of faulty scenarios covered by the root cause clusters generated for each pipeline. The last column represents the average of averages.

Pipelines					Case Study Subjects						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
1	HUDD	NO	None	K-Means	0.0%	0.0%	0.0%	20.0%	25.0%	0.0%	7.5%
2			None	DBSCAN	25.0%	25.0%	0.0%	80.0%	25.0%	25.0%	30.0%
3			None	HDBScan	100.0%	25.0%	0.0%	0.0%	0.0%	0.0%	20.8%
4			PCA	K-Means	0.0%	25.0%	0.0%	40.0%	50.0%	0.0%	19.2%
5			PCA	DBSCAN	0.0%	25.0%	0.0%	20.0%	50.0%	0.0%	15.8%
6			PCA	HDBScan	100.0%	25.0%	0.0%	0.0%	100.0%	0.0%	37.5%
7			UMAP	K-Means	25.0%	0.0%	0.0%	0.0%	0.0%	0.0%	4.2%
8			UMAP	DBSCAN	100.0%	50.0%	75.0%	60.0%	75.0%	100.0%	**76.7%**
9			UMAP	HDBScan	100.0%	25.0%	0.0%	0.0%	100.0%	0.0%	37.5%
10	LRP	NO	None	K-Means	0.0%	0.0%	12.5%	0.0%	50.0%	25.0%	14.6%
11			None	DBSCAN	0.0%	0.0%	37.5%	20.0%	50.0%	0.0%	17.9%
12			None	HDBScan	100.0%	25.0%	12.5%	20.0%	0.0%	0.0%	26.2%
13			PCA	K-Means	25.0%	25.0%	12.5%	0.0%	50.0%	25.0%	22.9%
14			PCA	DBSCAN	0.0%	0.0%	12.5%	20.0%	50.0%	0.0%	13.8%
15			PCA	HDBScan	0.0%	25.0%	0.0%	20.0%	0.0%	0.0%	7.5%
16			UMAP	K-Means	25.0%	0.0%	25.0%	0.0%	50.0%	0.0%	16.7%
17			UMAP	DBSCAN	75.0%	100.0%	75.0%	40.0%	100.0%	0.0%	**65.0%**
18			UMAP	HDBScan	100.0%	25.0%	12.5%	20.0%	0.0%	0.0%	26.2%
19	VGG-16	NO	None	K-Means	50.0%	50.0%	87.5%	80.0%	100.0%	100.0%	77.9%
20			None	DBSCAN	50.0%	50.0%	75.0%	20.0%	100.0%	100.0%	65.8%
21			None	HDBSCAN	0.0%	75.0%	0.0%	60.0%	50.0%	0.0%	30.8%
22			PCA	K-Means	50.0%	50.0%	12.5%	60.0%	100.0%	75.0%	57.9%
23			PCA	DBSCAN	50.0%	50.0%	37.5%	40.0%	75.0%	75.0%	54.6%
24			PCA	HDBSCAN	0.0%	75.0%	12.5%	100.0%	75.0%	50.0%	52.1%
25			UMAP	K-Means	75.0%	75.0%	87.5%	80.0%	75.0%	100.0%	82.1%
26			UMAP	DBSCAN	100.0%	100.0%	100.0%	80.0%	100.0%	100.0%	**96.7%**
27			UMAP	HDBSCAN	50.0%	75.0%	12.5%	60.0%	0.0%	50.0%	41.2%
28		YES	None	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
29			None	DBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
30			None	HDBSCAN	0.0%	25.0%	0.0%	20.0%	0.0%	100.0%	24.2%
31			PCA	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
32			PCA	DBSCAN	0.0%	25.0%	0.0%	20.0%	0.0%	0.0%	7.5%
33			PCA	HDBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
34			UMAP	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
35			UMAP	DBSCAN	25.0%	0.0%	0.0%	0.0%	50.0%	25.0%	16.7%
36			UMAP	HDBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
37	ResNet-50	NO	None	K-Means	50.0%	50.0%	25.0%	40.0%	50.0%	50.0%	44.2%
38			None	DBSCAN	25.0%	50.0%	37.5%	40.0%	50.0%	25.0%	37.9%
39			None	HDBSCAN	50.0%	100.0%	100.0%	60.0%	75.0%	100.0%	80.8%
40			PCA	K-Means	25.0%	50.0%	12.5%	20.0%	50.0%	25.0%	30.4%
41			PCA	DBSCAN	50.0%	50.0%	12.5%	0.0%	50.0%	25.0%	31.2%
42			PCA	HDBSCAN	0.0%	100.0%	12.5%	0.0%	0.0%	0.0%	18.8%
43			UMAP	K-Means	100.0%	75.0%	50.0%	40.0%	100.0%	100.0%	77.5%
44			UMAP	DBSCAN	100.0%	100.0%	100.0%	60.0%	100.0%	100.0%	**93.3%**

Pipelines					**Case Study Subjects**						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
45		NO	UMAP	HDBSCAN	50.0%	75.0%	0.0%	20.0%	0.0%	25.0%	28.3%
46			None	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
47			None	DBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
48			None	HDBSCAN	50.0%	0.0%	0.0%	0.0%	0.0%	0.0%	8.3%
49	ResNet-50		PCA	K-Means	0.0%	0.0%	0.0%	20.0%	0.0%	0.0%	3.3%
50		YES	PCA	DBSCAN	0.0%	0.0%	0.0%	20.0%	0.0%	0.0%	3.3%
51			PCA	HDBSCAN	0.0%	0.0%	0.0%	40.0%	0.0%	0.0%	6.7%
52			UMAP	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
53			UMAP	DBSCAN	0.0%	50.0%	0.0%	40.0%	25.0%	100.0%	35.8%
54			UMAP	HDBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
55			None	K-Means	75.0%	75.0%	87.5%	40.0%	100.0%	50.0%	71.2%
56			None	DBSCAN	75.0%	25.0%	50.0%	0.0%	100.0%	25.0%	45.8%
57			None	HDBSCAN	25.0%	100.0%	87.5%	100.0%	25.0%	100.0%	72.9%
58			PCA	K-Means	50.0%	50.0%	37.5%	40.0%	75.0%	25.0%	46.2%
59		NO	PCA	DBSCAN	50.0%	75.0%	62.5%	40.0%	75.0%	25.0%	54.6%
60			PCA	HDBSCAN	25.0%	75.0%	12.5%	60.0%	25.0%	0.0%	32.9%
61			UMAP	K-Means	75.0%	75.0%	87.5%	40.0%	75.0%	50.0%	67.1%
62			UMAP	DBSCAN	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	**100.0%**
63			UMAP	HDBSCAN	25.0%	100.0%	12.5%	100.0%	25.0%	25.0%	47.9%
64	Inception-V3		None	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
65			None	DBSCAN	0.0%	0.0%	0.0%	0.0%	25.0%	0.0%	4.2%
66			None	HDBSCAN	75.0%	25.0%	0.0%	40.0%	75.0%	100.0%	52.5%
67			PCA	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
68		YES	PCA	DBSCAN	0.0%	25.0%	0.0%	0.0%	0.0%	0.0%	4.2%
69			PCA	HDBSCAN	25.0%	0.0%	0.0%	20.0%	0.0%	0.0%	7.5%
70			UMAP	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
71			UMAP	DBSCAN	25.0%	50.0%	0.0%	20.0%	25.0%	75.0%	32.5%
72			UMAP	HDBSCAN	50.0%	0.0%	0.0%	0.0%	0.0%	0.0%	8.3%
73			None	K-Means	50.0%	75.0%	87.5%	40.0%	100.0%	75.0%	71.2%
74			None	DBSCAN	25.0%	0.0%	37.5%	0.0%	25.0%	25.0%	18.8%
75			None	HDBSCAN	100.0%	100.0%	62.5%	60.0%	50.0%	100.0%	78.8%
76			PCA	K-Means	50.0%	50.0%	37.5%	0.0%	75.0%	0.0%	35.4%
77		NO	PCA	DBSCAN	50.0%	75.0%	50.0%	0.0%	75.0%	0.0%	41.7%
78			PCA	HDBSCAN	100.0%	50.0%	0.0%	80.0%	25.0%	0.0%	42.5%
79			UMAP	K-Means	75.0%	75.0%	62.5%	40.0%	100.0%	50.0%	67.1%
80			UMAP	DBSCAN	100.0%	100.0%	100.0%	40.0%	100.0%	100.0%	**90.0%**
81			UMAP	HDBSCAN	50.0%	75.0%	0.0%	60.0%	100.0%	25.0%	51.7%
82	Xception		None	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
83			None	DBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	25.0%	4.2%
84			None	HDBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	100.0%	16.7%
85			PCA	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
86		YES	PCA	DBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
87			PCA	HDBSCAN	75.0%	50.0%	0.0%	0.0%	0.0%	0.0%	20.8%
88			UMAP	K-Means	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
89			UMAP	DBSCAN	0.0%	0.0%	0.0%	60.0%	50.0%	0.0%	18.3%
90			UMAP	HDBSCAN	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
91			None	K-Means	0.0%	25.0%	12.5%	20.0%	50.0%	50.0%	26.2%
92	AE	NO	None	DBSCAN	0.0%	25.0%	0.0%	0.0%	0.0%	0.0%	4.2%
93			None	HDBSCAN	0.0%	25.0%	0.0%	40.0%	0.0%	25.0%	15.0%
94			PCA	K-Means	0.0%	25.0%	12.5%	0.0%	50.0%	50.0%	22.9%

Pipelines					Case Study Subjects						
#	**FE**	**FT**	**DR**	**CA**	**GD**	**OC**	**HPD**	**SVIRO**	**SAP**	**CPD**	**Avg.**
95	AE	NO	PCA	DBSCAN	0.0%	25.0%	0.0%	20.0%	50.0%	50.0%	24.2%
96			PCA	HDBSCAN	100.0%	50.0%	0.0%	80.0%	50.0%	0.0%	46.7%
97			UMAP	K-Means	0.0%	0.0%	0.0%	0.0%	25.0%	50.0%	12.5%
98			UMAP	DBSCAN	50.0%	50.0%	37.5%	40.0%	50.0%	100.0%	**54.6%**
99			UMAP	HDBSCAN	0.0%	25.0%	0.0%	20.0%	0.0%	0.0%	7.5%

^{FE} Feature Extraction ^{FT} Fine-tuning ^{DR} Dimensionality Reduction ^{CA} Clustering Algorithm

A.3 Additional material for RQ3

Table A.3: Distribution of faults for the different failure inducing sets for each case study subject.

Case Study	Dataset	Faulty Scenarios									
		N	B	S	D	M	H	SG	EG	EO	NF
GD	GD_1	64	40	48	72	-	-	-	-	-	56
	GD_2	48	32	24	16	-	-	-	-	-	64
	GD_3	24	64	16	40	-	-	-	-	-	8
	GD_4	72	16	24	48	-	-	-	-	-	40
	GD_5	40	24	72	16	-	-	-	-	-	8
	GD_6	56	32	48	16	-	-	-	-	-	24
	GD_7	64	32	8	56	-	-	-	-	-	24
	GD_8	72	24	16	8	-	-	-	-	-	64
	GD_9	40	64	48	16	-	-	-	-	-	56
	GD_10	56	8	64	24	-	-	-	-	-	48
OC	OC_1	18	6	10	4	-	-	-	-	-	8
	OC_2	4	18	12	2	-	-	-	-	-	14
	OC_3	2	14	10	6	-	-	-	-	-	16
	OC_4	2	4	6	10	-	-	-	-	-	8
	OC_5	6	4	16	18	-	-	-	-	-	10
	OC_6	8	6	10	12	-	-	-	-	-	16
	OC_7	18	8	16	6	-	-	-	-	-	2
	OC_8	16	18	14	10	-	-	-	-	-	4
	OC_9	14	16	4	10	-	-	-	-	-	2
	OC_10	10	2	14	8	-	-	-	-	-	18
HPD	HPD_1	45	72	54	9	36	63	81	18	-	27
	HPD_2	27	81	45	18	54	63	72	36	-	9
	HPD_3	54	81	27	63	18	45	9	36	-	72
	HPD_4	36	18	63	72	9	81	54	27	-	45
	HPD_5	27	63	18	72	36	9	45	81	-	54
	HPD_6	45	36	54	63	81	9	72	27	-	18
	HPD_7	63	45	81	36	27	72	18	54	-	9
	HPD_8	72	9	63	27	36	18	81	54	-	45
	HPD_9	72	63	18	27	45	9	81	54	-	36
	HPD_10	54	81	63	27	45	72	18	9	-	36
SVIRO	SVIRO_1	6	12	18	21	-	-	-	-	24	15
	SVIRO_2	9	24	6	12	-	-	-	-	15	3
	SVIRO_3	15	18	21	3	-	-	-	-	27	9
	SVIRO_4	21	9	12	24	-	-	-	-	6	27
	SVIRO_5	27	21	3	9	-	-	-	-	18	24
	SVIRO_6	3	27	24	6	-	-	-	-	21	12
	SVIRO_7	24	6	15	18	-	-	-	-	3	21
	SVIRO_8	15	3	27	24	-	-	-	-	12	6
	SVIRO_9	12	15	3	27	-	-	-	-	9	18
	SVIRO_10	18	21	9	15	-	-	-	-	21	18
CPD	CPD_1	87	74	55	28	-	-	-	-	-	44
	CPD_2	43	56	27	22	-	-	-	-	-	74
	CPD_3	6	32	4	62	-	-	-	-	-	35
	CPD_4	49	22	88	34	-	-	-	-	-	5
	CPD_5	24	69	37	57	-	-	-	-	-	86

Case Study	Dataset	Faulty Scenarios									
		N	**B**	**S**	**D**	**M**	**H**	**SG**	**EG**	**EO**	**NF**
CPD	CPD_6	13	69	58	54	-	-	-	-	-	25
	CPD_7	3	32	51	9	-	-	-	-	-	59
	CPD_8	77	62	12	53	-	-	-	-	-	4
	CPD_9	85	27	78	30	-	-	-	-	-	62
	CPD_10	65	46	66	89	-	-	-	-	-	40
SAP	SAP_1	22	33	54	48	-	-	-	-	-	72
	SAP_2	75	22	48	17	-	-	-	-	-	57
	SAP_3	22	4	57	42	-	-	-	-	-	81
	SAP_4	74	21	40	36	-	-	-	-	-	42
	SAP_5	15	14	86	74	-	-	-	-	-	51
	SAP_6	73	60	2	83	-	-	-	-	-	72
	SAP_7	58	57	47	83	-	-	-	-	-	43
	SAP_8	6	75	26	16	-	-	-	-	-	70
	SAP_9	89	86	66	32	-	-	-	-	-	68
	SAP_10	67	77	14	4	-	-	-	-	-	55

^{N} Noise ^{B} Blurriness ^{S} Scaling ^{D} Darkness ^{M} Masks ^{H} Hands ^{SG} SunGlasses ^{EG} EyeGlasses ^{EO} Everyday Objects ^{NF} Non-injected Fault $^{(-)}$ Not Applicable

Bibliography

[1] World Health Organization. Road Traffic Injuries, 2022. http://www.who.int/mediacentre/factsheets/fs358/en/.

[2] European Road Safety Observatory. Annual Statistical Report, 2021. https://road-safety.transport.ec.europa.eu/statistics-and-analysis/data-and-analysis/annual-statistical-report_en.

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

[4] Zhong Li, Minxue Pan, Tian Zhang, and Xuandong Li. Testing dnn-based autonomous driving systems under critical environmental conditions. In *International Conference on Machine Learning*, pages 6471–6482. PMLR, 2021.

[5] Satti RG Reddy, GP Varma, and Rajya Lakshmi Davuluri. Deep neural network (dnn) mechanism for identification of diseased and healthy plant leaf images using computer vision. *Annals of Data Science*, pages 1–30, 2022.

[6] A Kousar Nikhath, MD Abdul Rab, N Venkata Bharadwaja, L Goutham Reddy, K Saicharan, C Venkat Manas Reddy, et al. An intelligent college enquiry bot using nlp and deep learning based techniques. In *2022 International Conference for Advancement in Technology (ICONAT)*, pages 1–6. IEEE, 2022.

[7] Ekin EKİNCİ, Hidayet TAKCI, and Sultan ALAGÖZ. Poet classification using ann and dnn. *Electronic Letters on Science and Engineering*, 18(1):10–20, 2018.

[8] Mihalj Bakator and Dragica Radosav. Deep Learning and Medical Diagnosis: A Review of Literature. *Multimodal Technologies and Interaction*, 2(3), 2018.

[9] Jian Huang, Junyi Chai, and Stella Cho. Deep learning in finance and banking: A literature review and classification. *Frontiers of Business Research in China*, 14(1):13, 2020.

[10] S Ribu Hassini, T Gireesh Kumar, and S Kowshik Hurshan. A Machine Learning and Deep Neural Network Approach in Industrial Control Systems. In Simon Fong, Nilanjan Dey, and Amit Joshi, editors, *ICT Analysis and Applications*, pages 525–536, Singapore, 2022. Springer Nature Singapore.

[11] Diogo V Carvalho, Eduardo M Pereira, and Jaime S Cardoso. Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832, 2019.

[12] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[13] David Gunning and David Aha. Darpa's explainable artificial intelligence (xai) program. *AI magazine*, 40(2):44–58, 2019.

[14] International Organization for Standardization. ISO, ISO26262-1:2018, Road vehicles: Functional safety, 2020.

[15] International Organization for Standardization. ISO/PAS 21448:2019, Road vehicles: Safety of the intended functionality, 2020.

[16] International Organization for Standardization. ISO, ISO-14971-2019, Application of risk management to medical devices, the European Forward, 2019.

[17] IEE. IEE sensing solutions. www.iee.lu, 2020.

[18] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. Supporting deep neural network safety analysis and retraining through heatmap-based unsupervised learning. *IEEE Transactions on Reliability*, 70(4):1641–1657, 2021.

[19] Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. Hudd: A tool to debug dnns for safety analysis. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings*, ICSE '22, page 100–104, New York, NY, USA, 2022. Association for Computing Machinery.

[20] Hazem Fahmy, Mojtaba Bagherzadeh, Fabrizio Pastore and Lionel Briand. HUDD: toolset and replicability package, 2020. https://github.com/SNTSVV/HUDD-toolset/.

[21] Hazem Fahmy, Fabrizio Pastore, Lionel Briand, and Thomas Stifter. Simulator-based explanation and debugging of hazard-triggering events in dnn-based safety-critical systems. *ACM Trans. Softw. Eng. Methodol.*, oct 2022.

[22] Hazem Fahmy, Fabrizio Pastore, Lionel Briand, and Thomas Stifter. SEDE: replicability package, 2022. https://figshare.com/s/a7fd8e713e038ee0d86c.

[23] Hazem Fahmy, Fabrizio Pastore, Lionel Briand, and Thomas Stifter. SEDE code repository, 2022. https://github.com/SNTSVV/SEDE.

[24] Mohammed Oualid Attaoui, Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. Black-box safety analysis and retraining of dnns based on feature extraction and clustering. *ACM Trans. Softw. Eng. Methodol.*, jul 2022. Just Accepted.

[25] Mohammed Oualid Attaoui, Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. SAFE: toolset and replicability package. https://zenodo.org/record/6619279, 2022.

[26] Mohammed Oualid Attaoui, Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. Dnn explanation for safety analysis: an empirical evaluation of clustering-based approaches, 2023.

[27] Mohammed Oualid Attaoui, Hazem Fahmy, Fabrizio Pastore, and Lionel Briand. DNN Explanation for Safety Analysis: an Empirical Evaluation of Clustering-based Approaches - Replicability package. https://figshare.com/projects/DNN_Explanation_for_Safety_Analysis_an_Empirical_Evaluation_of_Clustering-based_Approaches/157973, 2023.

[28] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[29] Yu-Dong Zhang, Suresh Chandra Satapathy, David S Guttery, Juan Manuel Górriz, and Shui-Hua Wang. Improved breast cancer classification through combining graph convolutional network and convolutional neural network. *Information Processing & Management*, 58(2):102439, 2021.

[30] Sandeep Sony, Kyle Dunphy, Ayan Sadhu, and Miriam Capretz. A systematic review of convolutional neural network-based structural condition assessment techniques. *Engineering Structures*, 226:111347, 2021.

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

[32] Christoph Molnar. *Interpretable Machine Learning*. LuLu, 2011. https://christophm.github.io/interpretable-ml-book/.

[33] Edward Kim, Divya Gopinath, Corina Păsăreanu, and Sanjit A. Seshia. A programmatic and semantic approach to explaining and debugging neural network based object detectors. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11125–11134, 2020.

[34] Fitash Ul Haq, Donghwan Shin, Lionel C. Briand, Thomas Stifter, and Jun Wang. Automatic test suite generation for key-points detection dnns using many-objective search (experience paper). In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2021, page 91–102, New York, NY, USA, 2021. Association for Computing Machinery. https://doi.org/10.1145/3460319.3464802.

[35] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel Briand. Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems. *Empirical Softw. Engg.*, 26(5), sep 2021.

[36] Eibe Frank and Ian H. Witten. Generating accurate rule sets without global optimization. In J. Shavlik, editor, *Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann, 1998.

[37] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart Russell, editors, *Machine Learning Proceedings 1995*, pages 115–123. Morgan Kaufmann, San Francisco (CA), 1995.

[38] Ronald S. King. *Cluster Analysis and Data Mining: An Introduction*. Mercury Learning & Information, USA, 2014.

[39] J. MacQueen. Some methods for classification and analysis of multivariate observations. *Proceedings of BSMSP*, pages 281–297, 1967.

[40] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

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

[42] Jr. Ward, J. H. Hierarchical grouping to optimize an objective function. *American Statistical Association Journal*, 58:236–244, 1963.

[43] R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958.

[44] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *WIREs Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[45] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin philosophical magazine and journal of science*, 2(11):559–572, 1901.

[46] Leland McInnes, John Healy, Nathaniel Saul, and Lukas Grossberger. Umap: Uniform manifold approximation and projection. *The Journal of Open Source Software*, 3(29):861, 2018.

[47] Jonathon Shlens. A tutorial on principal component analysis. *arXiv:1404.1100*, 2014.

[48] Florent Forest, Mustapha Lebbah, Hanene Azzag, and Jérôme Lacaille. Deep embedded self-organizing maps for joint representation learning and topology-preserving clustering. *Neural Computing and Applications*, 33(24):17439–17469, 2021.

[49] Erroll Wood, Tadas Baltrušaitis, Louis-Philippe Morency, Peter Robinson, and Andreas Bulling. Learning an appearance-based gaze estimator from one million synthesised images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, ETRA '16, pages 131–138, New York, NY, USA, 2016. ACM.

[50] Hal Daumé III. *A Course in Machine Learning*. Available Online, 2020. http://ciml.info/.

[51] Annibale Panichella, Fitsum Meshesha Kifetew, and Paolo Tonella. Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering*, 44(2):122–158, 2018.

[52] Fitash Ul Haq, Donghwan Shin, Shiva Nejati, and Lionel C. Briand. Comparing offline and online testing of deep neural networks: An autonomous car case study. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 85–95, 2020.

[53] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing advanced driver assistance systems using multi-objective search and neural networks. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 63–74, 2016.

[54] Raja Ben Abdessalem, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, pages 1016–1026, New York, NY, USA, 2018. ACM.

[55] Mohamed El Mostadi, Hélène Waeselynck, and Jean-Marc Gabriel. Virtual test scenarios for adas: Distance to real scenarios matters! In *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 836–841, 2022.

[56] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[57] Bingdong Li, Jinlong Li, Ke Tang, and Xin Yao. Many-objective evolutionary algorithms: A survey. *ACM Comput. Surv.*, 48(1), sep 2015.

[58] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pages 811–822, 2022.

[59] Rafael Garcia, Alexandru C. Telea, Bruno Castro da Silva, Jim Torresen, and Joao Luiz Dihl Comba. A task-and-technique centered survey on visual analytics for deep learning model engineering. *Computers and Graphics*, 77:30 – 49, 2018.

[60] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2018.

[61] Piotr Dabkowski and Yarin Gal. Real time image saliency for black box classifiers. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS?17, pages 6970–6979, Red Hook, NY, USA, 2017. Curran Associates Inc.

[62] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus Robert Müller. *Layer-Wise Relevance Propagation: An Overview*, pages 193–209. Springer International Publishing, Cham, 2019.

[63] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 618–626, Oct 2017.

[64] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[65] J.T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. In *ICLR (workshop track)*, 2015.

[66] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, June 2016.

[67] D. Gopinath, H. Converse, C. Pasareanu, and A. Taly. Property inference for deep neural networks. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 797–809, 2019.

[68] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proceedings of the '18 AAAI Conference on Artificial Intelligence*, 2018.

[69] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. Mode: Automated neural network model debugging via state differential analysis and input selection. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 175–186, New York, NY, USA, 2018. ACM.

[70] Jinhan Kim, Robert Feldt, and Shin Yoo. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 41st International Conference on Software Engineering*, ICSE '19, pages 1039–1049. IEEE Press, 2019.

[71] Hasan Ferit Eniser, Simos Gerasimou, and Alper Sen. Deepfault: Fault localization for deep neural networks. In Reiner Hähnle and Wil van der Aalst, editors, *Fundamental Approaches to Software Engineering*, pages 171–191, Cham, 2019. Springer International Publishing.

[72] Maximilian Alber, Sebastian Lapuschkin, Philipp Seegerer, Miriam Hägele, Kristof T. Schütt, Grégoire Montavon, Wojciech Samek, Klaus-Robert Müller, Sven Dähne, and Pieter-Jan Kindermans. innvestigate neural networks! *Journal of Machine Learning Research*, 20(93):1–8, 2019. https://github.com/albermax/innvestigate.

[73] TorchRay. DNN Explanation, 2020. https://github.com/facebookresearch/TorchRay.

[74] Yue Zhao, Hong Zhu, Kai Chen, and Shengzhi Zhang. Ai-lancet: Locating error-inducing neurons to optimize neural networks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 141–158, 2021.

[75] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

[76] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K. Muller. Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems*, 28(11):2660–2673, Nov 2017.

[77] Gregoire Montavon, Sebastian Lapuschkin, Alexander Binder, Wojciech Samek, and Klaus-Robert Muller. Explaining nonlinear classification decisions with deep taylor decomposition. *Pattern Recognition*, 65:211 – 222, 2017.

[78] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models, 2018.

[79] Raja Ben Abdessalem, Shiva Nejati, Lionel C Briand, and Thomas Stifter. Testing vision-based control systems using learnable evolutionary algorithms. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pages 1016–1026. IEEE, 2018.

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

[81] Vincenzo Riccio and Paolo Tonella. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2020, page 876–888, New York, NY, USA, 2020. Association for Computing Machinery.

[82] International Organization for Standardization. ISO, ISO-24765-2017, Systems and software engineering - Vocabulary, 2020.

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

[84] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 483–499, Cham, 2016. Springer International Publishing.

[85] Gregoire Montavon. WIFS 2017 Tutorial on Methods for Understanding DNNs and their Predictions, 2019.

[86] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd edition, 2011.

[87] V. Satopaa, J. Albrecht, D. Irwin, and B. Raghavan. Finding a "kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, pages 166–171, 2011.

[88] Z. Li, C. Luo, Y. Zhao, Y. Sun, K. Sui, X. Wang, D. Liu, X. Jin, Q. Wang, and D. Pei. Generic and robust localization of multi-dimensional root causes. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, pages 47–57, 2019.

[89] L. Jendele, M. Schwenk, D. Cremarenco, I. Janicijevic, and M. Rybalkin. Efficient automated decomposition of build targets at large-scale. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, pages 457–464, 2019.

[90] Robert L Thorndike. Who belongs in the family? *Psychometrika*, 18(4):267–276, 1953.

[91] J. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability - Vol. 1*, pages 281–297. University of California Press, Berkeley, CA, USA, 1967.

[92] Fionn Murtagh and Pierre Legendre. Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion? *Journal of Classification*, 31(3):274–295, Oct 2014.

[93] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, Cambridge, UK, 2008.

[94] M. Inaba, N. Katoh, and H. Imai. Variance-based k-clustering algorithms by voronoi diagrams and randomization. *IEICE Transactions on Information and Systems*, 83:1199–1206, 2000.

[95] Andrea Vattani. k-means Requires Exponentially Many Iterations Even in the Plane. *Discrete & Computational Geometry*, 45(4):596–616, 2011.

[96] Bengt Fornberg. Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comp.*, pages 699–706, 1988.

[97] Saeideh Bakhshi, David A. Shamma, Lyndon Kennedy, Yale Song, Paloma de Juan, and Joseph 'Jofish' Kaye. Fast, cheap, and good: Why animated gifs engage us. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 575–586, New York, NY, USA, 2016. Association for Computing Machinery.

[98] PyTorch. PyTorch DNN framework, 2020. `https://pytorch.org`.

[99] SciPy. Pyton framework for mathematics, science, and engineering., 2020. `https://scipy.org/`.

[100] Blender. Blender 3D simulation and rendering engine, 2020. `https://www.blender.org/`.

[101] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, May 2017.

[102] MakeHuman community. MakeHuman computer graphics middleware for the prototyping of humanoids. , 2020. `http://www.makehumancommunity.org`.

[103] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 3730–3738, 2015.

[104] INI. Traffic Sign Dataset, 2020. `http://benchmark.ini.rub.de/?section=gtsrb&subsection=dataset`.

[105] András Vargha and Harold D. Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2):101–132, 2000.

[106] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *Proceedings of the 33rd International Conference on Software Engineering*, ICSE '11, pages 1–10, New York, NY, USA, 2011. ACM.

[107] S. Varrette, P. Bouvry, H. Cartiaux, and F. Georgatos. Management of an academic hpc cluster: The ul experience. In *Proc. of the 2014 Intl. Conf. on High Performance Computing & Simulation (HPCS 2014)*, pages 959–967, Bologna, Italy, July 2014. IEEE.

[108] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. Taxonomy of real faults in deep learning systems. In *Proceedings of the 42nd International Conference on Software Engineering*, New York, NY, USA, 2020. Association for Computing Machinery.

[109] Jordan J. Bird, Diego R. Faria, Anikó Ekárt, and Pedro P. S. Ayrosa. From simulation to reality: Cnn transfer learning for scene classification. In *2020 IEEE 10th International Conference on Intelligent Systems (IS)*, pages 619–625, 2020.

[110] Jiman Kim and Chanjong Park. End-to-end ego lane estimation based on sequential transfer learning for self-driving cars. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1194–1202, 2017.

[111] Tadanobu Inoue, Subhajit Choudhury, Giovanni De Magistris, and Sakyasingha Dasgupta. Transfer learning from synthetic to real images using variational autoencoders for precise position detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 2725–2729, 2018.

[112] Gordon Fraser and Andrea Arcuri. Whole test suite generation. *IEEE Transactions on Software Engineering*, 39(2):276–291, 2013.

[113] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. Deepmetis: Augmenting a deep learning test set to increase its mutation score. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering*, ASE '21, page 355–367. IEEE Press, 2021.

[114] Vedat Toğan and Ayşe T. Daloğlu. An improved genetic algorithm with initial population strategy and self-adaptive member grouping. *Computers & Structures*, 86(11):1204–1218, 2008.

[115] Hisao Ishibuchi, Yuji Sakane, Noritaka Tsukamoto, and Yusuke Nojima. Evolutionary many-objective optimization by nsga-ii and moea/d with large populations. In *2009 IEEE International Conference on Systems, Man and Cybernetics*, pages 1758–1763, 2009.

[116] Steve Dias Da Cruz, Bertram Taetz, Thomas Stifter, and Didier Stricker. Autoencoder attractors for uncertainty estimation. In *Proceedings of the IEEE International Conference on Pattern Recognition (ICPR)*, 2022.

[117] ManuelBastioniLAB. Character creation tool for Blender, 2016. `https://github.com/animate1978/MB-Lab`.

[118] Gabriele Fanelli, Matthias Dantone, Juergen Gall, Andrea Fossati, and Luc Van Gool. Random forests for real time 3d face analysis. *Int. J. Comput. Vision*, 101(3):437–458, February 2013.

[119] Microsoft. Kinect, 2010. `https://developer.microsoft.com/en-us/windows/kinect/`.

[120] FaceShift GmbH. Online modeling for real-time facial animation, U.S. Patent 9378576, May 2016.

[121] DLIB Team. C++ toolkit containing machine learning algorithms and tools for creating complex software., 2022. `http://dlib.net/`.

[122] Barbara Kitchenham, Lech Madeyski, David Budgen, Jacky Keung, Pearl Brereton, Stuart Charters, Shirley Gibbs, and Amnart Pohthong. Robust Statistical Methods for Empirical Software Engineering. *Empirical Software Engineering*, 22(2):579–630, 2017.

[123] Xiang Gao, Ripon K. Saha, Mukul R. Prasad, and Abhik Roychoudhury. Fuzz testing based data augmentation to improve robustness of deep neural networks. In *Proceedings of the 42nd International Conference on Software Engineering*, ICSE '20, New York, NY, USA, 2020. ACM.

[124] Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. Exploring the landscape of spatial robustness. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1802–1811, Long Beach, California, USA, 09–15 Jun 2019. PMLRG.

[125] Jingyi Wang, Jialuo Chen, Youcheng Sun, Xingjun Ma, Dongxia Wang, Jun Sun, and Peng Cheng. Robot: Robustness-oriented testing for deep learning systems. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 300–311. IEEE Press, 2021.

[126] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, Yang Liu, Jianjun Zhao, Bo Li, Jianxiong Yin, and Simon See. Deephunter: A coverage-guided fuzz testing framework for deep neural networks. *ISSTA 2019 - Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 158–168, 2019.

[127] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–11, 2015.

[128] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings*, pages 0–28, 2018.

[129] N. Carlini and D. Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57, Los Alamitos, CA, USA, may 2017. IEEE Computer Society.

[130] Mohamed Loey, Gunasekaran Manogaran, Mohamed Hamed N Taha, and Nour Eldeen M Khalifa. A hybrid deep transfer learning model with machine learning methods for face mask detection in the era of the covid-19 pandemic. *Measurement*, 167:108288, 2021.

[131] Zitong Wan, Rui Yang, Mengjie Huang, Nianyin Zeng, and Xiaohui Liu. A review on transfer learning in eeg signal analysis. *Neurocomputing*, 421:1–14, 2021.

[132] Hans-Peter Kriegel, Peer Kröger, Jörg Sander, and Arthur Zimek. Density-based clustering. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 1(3):231–240, 2011.

[133] Gábor Gosztolya, Róbert Busa-Fekete, Tamás Grósz, and László Tóth. Dnn-based feature extraction and classifier combination for child-directed speech, cold and snoring identification. In *INTERSPEECH*. International Speech Communication Association (ISCA), 2017.

[134] Stanford Vision Lab. ImageNet, image database organized according to the wordnet hierarchy. https://www.image-net.org. Accessed: 2021-12-07.

[135] Muhammed Talo. Automated classification of histopathology images using transfer learning. *Artificial Intelligence in Medicine*, 101:101743, 2019.

[136] Nassima Dif, Mohammed Oualid Attaoui, Zakaria Elberrichi, Mustapha Lebbah, and Hanene Azzag. Transfer learning from synthetic labels for histopathological images classification. *Applied Intelligence*, pages 1–20, 2021.

[137] Alexander N Gorban and Andrei Y Zinovyev. Principal graphs and manifolds. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 28–59. IGI Global, 2010.

[138] Purnima Bholowalia and Arvind Kumar. Ebk-means: A clustering technique based on elbow method and k-means in wsn. *International Journal of Computer Applications*, 105(9), 2014.

[139] Nadia Rahmah and Imas Sukaesih Sitanggang. Determination of optimal epsilon (eps) value on dbscan algorithm to clustering data on peatland hotspots in sumatra. In *IOP conference series: earth and environmental science*, volume 31, page 012012. IOP Publishing, 2016.

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

[141] Erich Schubert, Jörg Sander, Martin Ester, Hans Peter Kriegel, and Xiaowei Xu. Dbscan revisited, revisited: Why and how you should (still) use dbscan. *ACM Trans. Database Syst.*, 42(3), jul 2017.

[142] Levoy Marc, Marc Levoy, Rusinkiewicz Szymon, Weyrich Tim, Pfister Hanspeter, Amenta Nina, Wu Jianhua, Barthe Loïc, Zwicker Matthias, Kobbelt Leif, et al. 2-the early history of point-based graphics. In *Point-Based Graphics*, pages 8–16. Elsevier, 2007.

[143] Rajaditya Mukherjee, Qingyang Li, Zhili Chen, Shicheng Chu, and Huamin Wang. Neuraldrop: Dnn-based simulation of small-scale liquid flows on solids. *arXiv:1811.02517*, 2018.

[144] Johan Linaker, Sardar Muhammad Sulaman, Martin Höst, and Rafael Maiani de Mello. Guidelines for conducting surveys in software engineering v. 1.1. *Lund University*, 2015.

[145] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*, volume 9783642290. Springer Berlin, 2012.

[146] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[147] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[148] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017.

[149] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[150] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[151] Raghav Bali Dipanjan Sarkar. *Transfer Learning in Action*. Manning Publications, 2021. https://livebook.manning.com/book/transfer-learning-in-action/chapter-1/v-1/69 (visited 2022-02-19).

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

[153] Keinosuke Fukunaga and Larry Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on information theory*, 21(1):32–40, 1975.

[154] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org/, 2015.

[155] Francois Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[156] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[157] Steve Dias Da Cruz, Oliver Wasenmuller, Hans-Peter Beise, Thomas Stifter, and Didier Stricker. Sviro: Synthetic vehicle interior rear seat occupancy dataset and benchmark. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 973–982, 2020.

[158] Autumn pre-trained model. Udacity self-driving car challenge 2. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/autumn, 2017.

[159] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[160] Udacity self-driving challenge 2. https://github.com/udacity/self-driving-car/tree/master/challenges/challenge-2. Accessed: 2022-08-16.

[161] Kaiwen Duan, Song Bai, Lingxi Xie, Honggang Qi, Qingming Huang, and Qi Tian. Centernet: Keypoint triplets for object detection. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 6568–6577. IEEE Computer Society, 2019.

[162] Peking University/Baidu - autonomous driving. https://www.kaggle.com/c/pku-autonomous-driving. Accessed: 2022-08-16.

[163] Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The ApolloScape open dataset for autonomous driving and its application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2702–2719, oct 2020.

[164] Xibin Song, Peng Wang, Dingfu Zhou, Rui Zhu, Chenye Guan, Yuchao Dai, Hao Su, Hongdong Li, and Ruigang Yang. Apollocar3d: A large 3d car instance understanding benchmark for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5452–5462, 2019.

[165] Sully Chen. Autopilot-Tensorflow. https://github.com/SullyChen/Autopilot-TensorFlow, 2016.

[166] Bin Liang, Hongcheng Li, Miaoqiang Su, Xirong Li, Wenchang Shi, and Xiaofeng Wang. Detecting adversarial image examples in deep neural networks with adaptive noise reduction. *IEEE Transactions on Dependable and Secure Computing*, 18(1):72–85, jan 2021.

[167] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, jun 2014.

[168] Alex Clark. Pillow (PIL Fork) Documentation. https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf, 2015.

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

[170] Torsten Hothorn, Kurt Hornik, and Achim Zeileis. ctree: Conditional inference trees. *The comprehensive R archive network*, 8, 2015.

[171] Helmut Strasser and Christian Weber. On the asymptotic theory of permutation statistics. WorkingPaper 27, SFB Adaptive Information Systems and Modelling in Economics and Management Science, WU Vienna University of Economics and Business, 1999.

[172] S Paul Wright. Adjusted p-values for simultaneous inference. *Biometrics*, pages 1005–1013, 1992.

[173] Graham JG Upton. Fisher's exact test. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 155(3):395–402, 1992.

[174] Eric W. Weisstein. Fisher's exact test. https://mathworld.wolfram.com/FishersExactTest.html, 2022. Accessed: 2022-11-15.

[175] Advay Patil. Car Object Detection. https://www.kaggle.com/code/advaypatil/car-object-detection, 2022.

[176] Amirata Ghorbani, James Wexler, James Zou, and Been Kim. Towards automatic concept-based explanations, 2019.