A Semantic Testing Approach for Deep Neural Networks Using Bayesian Network Abstraction

Thesis submitted in accordance with the requirements of the University of Liverpool for the degree of Doctor in Philosophy by

Amany Alshareef

November 2023

Dedication

To my faithful mother, Joman Alqasimi
Who has always been the epitome of strength, love, and sacrifice. Throughout my life, she has been my guiding light and unwavering support. Her lifelong belief in my abilities inspired me to pursue this PhD journey of knowledge and to reach this significant accomplishment.

To my gentle father, Dr Fahad Alshareef
Who has been my pillar of wisdom and encouragement since the beginning. His guidance, patience, and perseverance have shaped me into the person I am today.

To my beloved husband, Dr Mohammad Alabbasi
Who has stood by my side through thick and thin. His love, encouragement, and understanding have been my driving force throughout this challenging academic journey.

To my dearest parents-in-law, Hamza and Awatif
Who have embraced me as their own child and provided unwavering love and support.

To my precious children Lana, Dana , and Hamza
Who have been patient throughout this tough academic pursuit and through the late nights I spent on my computer completing this research.

To my siblings: Eman, Somaya, Yasir, Samah, Abdulrahman, Fatima, Amal, Duaa, and Ahmad
Who have been my cheerleaders, my sounding boards, and my inspiration. Their constant presence and belief in my capabilities have pushed me to strive for excellence.

Acknowledgements

I would like to express my deepest gratitude and appreciation to all those who have contributed to the completion of this PhD thesis. This work would not have been possible without the support, guidance, and encouragement of numerous individuals and institutions. First and foremost, I am immensely grateful to my primary supervisor, Prof. Xiaowei Huang, for his invaluable mentorship, constructive criticism, thought-provoking discussions, and suggestions, which have immensely enriched my research and strengthened the quality of this thesis. Prof. Xiaowei supports me through challenging periods in my PhD, during my pregnancy, and Covid-19. There are no words that express my deep gratitude for his unlimited support. I would also like to express my profound gratitude to my second supervisor, Prof. Sven Schewe, for his expertise, dedication, and insightful feedback, which have been instrumental in shaping my research and academic growth.

I would also like to extend my heartfelt thanks to Dr. Nicolas Berthier, who was a great supervisor for two years of my PhD, and a professional advisor after he finished his postdoctoral period at the University of Liverpool. Thanks for his time, expertise, valuable cooperation, and all that I have learned under his guidance. I am truly fortunate to have had the opportunity to work under the guidance of such supervisors.

I am grateful to my family and friends for their unwavering love, encouragement, and faith in me, which have given me the strength to overcome obstacles and strive for greatness. Their belief in me and their constant motivation have been my driving forces during stressful PhD moments. Special thanks to the Liverpool's friends: Leena Abu Hussein, Khadija Alawfi, Wdha Alshayul, Futun Alhamidi, Mariah Hafez, Nawal Almutairi, Souad Alotaibi, Hanaa Khan, Amal Alzahrani, Nora Bakhsh, Hasna Halika, and Nouf Basha. Last but not least, I gratefully acknowledge the generous funding provided by the Saudi Arabian government since the beginning and until I achieved this milestone. Thanks to the Saudi Arabian Cultural Bureau in London for their financial support throughout my PhD journey.

Abstract

The studies presented in this thesis are directed at investigating the internal decision process of Deep Neural Networks (DNNs) and testing their performance based on feature importance weights. Deep learning models have achieved state-of-the-art performance in a variety of machine learning tasks, which has led to their integration into safety-critical domains such as autonomous vehicles. The susceptibility of deep learning models to adversarial examples raises serious concerns about their application in safety-critical contexts. Most existing testing methodologies have failed to consider the interactions between neurons and the semantic representations formed in the DNN during the training process. This thesis designed weight-based semantic testing metrics that first modelled the internal behaviour of the DNNs into Bayesian networks and the contribution of the hidden features to their decisions into importance weight. Moreover, it measured the test data coverage according to the weight of the features. These approaches were followed to answer the main research question, *"Is it a better measure of trustworthiness to measure the coverage of the semantic aspect of deep neural networks and treat each internal component according to its contribution value to the decision when testing these learning models' performance than relying on traditional structural unweighted measures?"*.

This thesis makes three main contributions to the field of machine learning. First, the thesis proposes a novel technique for estimating the importance of a neural network's latent features through its abstracted behaviour into a Bayesian Network (BN). The algorithm analysed the sensitivity of each extracted feature to distributional shifts by observing changes in BN distribution. The experimental results showed that computing the distance between two BN probability distributions, clean as well as perturbed by interval-shifts or adversarial attacks, can detect the distribution shift wherever it exists. The hidden features were assigned weight scores according to the computed sensitivity distances. Secondly, to further justify the contribution of each latent feature to the classification decision, the abstract scheme of the BN was extended to perform a prediction. The performance of the BN in predicting input classification labels was shown to be a decent approximator of the original DNN. Moreover, feature perturbation on the BN classifier demonstrated that each feature influenced prediction accuracy differently, thereby validating the presented feature importance assumption. Lastly, the developed feature importance measure was used to assess the extent to which a given test dataset exercises high-level features that have been

learned by hidden layers of the DNN, taking into account significant representations as a priority when generating new test inputs. The evaluation was conducted to compare the initial and final coverage of the proposed weighting approach with normal BN-based feature coverage. The testing coverage experiments indicated that the proposed weight metrics achieved higher coverage compared to the original feature metrics while maintaining the effectiveness of finding adversarial samples during the test case generation process. Furthermore, the weight metrics guaranteed that the achieved testing percent covered the most crucial components, where the test generation algorithm was directed to synthesise new input targeting features with higher importance scores. Hence, the evidence of DNNs' trustworthy behaviour is subsequently furthered through this study.

Contents

List of Figures

List of Tables

Notations

The following notations of abbreviations and symbols are found throughout the thesis:

List of abbreviations:

DNNs	Deep Neural Networks
CNNs	Convolutional Neural Networks
NNs	Neural Networks
FI	Feature Importance
BNs	Bayesian Networks
DAG	Directed Acyclic Graph
CPT	Conditional Probability Table
PCA	Principal Component Analysis
ICA	Independent Component Analysis
KDE	Kernel Density Estimation
BFCov	BN Feature Coverage
BFdCov	BN Feature Dependence Coverage
BFxCov	BFCov + BFdCov
WFCov	Weighted Feature Coverage
WFdCov	Weighted Feature Dependence Coverage
WFCovTot	WFCov + WFdCov

List of symbols:

$\mathcal{N}$	neural network
$\mathcal{B}$	Bayesian network
$\mathbb{D}_X$	input domain
$\mathbb{D}_Y$	output domain
$f_{\mathcal{N}}(x)$	predicted label of x according to the network $\mathcal{N}$
l_i	the i-th layer
$n_{i,j}$	the j-th neuron on the i-th layer
$\hat{n}$	the neuron activation
h_i	the set of neurons on the i-th layer
$\hat{h}_i$	the set of neurons activation on the i-th layer
Λ	the feature mapping
$\mathsf{Discr}^\sharp$	the discretisation function
$\mathcal{P}_i$	the unconditional probability
$\mathcal{CP}_i$	the conditional probability
random_shift	the feature's intervals shifting
d_p	a given distance metric
P_{ref}	the BN reference probabilities
P'_f	the BN probabilities after perturbing feature f
$\mathbf{r}$	the discretised representation of input $\mathbf{x}$
Y	the prediction node
π_Y	the parent nodes of Y

Chapter 1

Introduction

1.1 Overview

Machine Learning (ML) systems have shown impressive performance, which has led to them being broadly applied to various domains. Recent advances in deep learning have involved *Deep Neural Networks (DNNs)*, which facilitate a variety of innovative applications of machine learning. DNNs have become highly expressive models that have achieved close to human-level capability on a wide range of tasks, including speech and visual recognition, natural language processing, and game playing. With this promising performance, DNNs are indeed increasingly integrated into safety-critical applications such as autonomous driving vehicles, disease diagnostics, and secure authentication. As they are being executed on these critical types of programs, the software employing them must be systematically tested and certified to ensure their reliability and safety.

Despite their tremendous capabilities, neuronal networks are, however, complex models working as black boxes where they learn their decision rule through training on a large dataset by gradually optimising parameters until they achieve the required accuracy. Therefore, they do not have a specific control-flow structure, which makes it difficult to analyse their behaviours and define suitable test criteria. Moreover, neural networks designed for regression and classification tasks do not capture model uncertainty, which means that they cannot provide information about the reliability of their predictions. This lack of reliable confidence estimates and other robustness issues can make them vulnerable to adversarial attacks.

Various DNN's testing techniques have been established, inspired by the notion of soft-

ware testing in traditional software engineering. Neuron activation [73], which is equivalent to the code coverage in traditional testing, and other structural coverage techniques, such as Modified Condition/Decision Coverage (MC/DC) [90], that are defined based on the syntactic model components have proven to be less effective in validating the safety behaviour of these intelligent systems [93]. Neural networks have different components that need to be considered when testing them including, neurons, syntactic connections between neurons, activation functions, and relationship between layers. Consequently, structural coverage such as the activation patterns of individual neurons may not be sufficient to capture the full range of behaviours and interactions between neurons that occur in the DNN. Moreover, DNNs have been criticised for their vulnerability to input perturbation, which leads to misclassified results with high confidence. This critical shortcoming of DNNs makes their prediction accuracy scores drop significantly when there is an adversarial shift in data distributions. Small perturbations, including adversarial perturbations, random noise, and geometric transformations, applied to the input samples should not cause significant losses in the classifier's performance.

As Bayesian Networks (BNs) present a principled way to capture relationships between variables that are explicitly represented using a directed graph, Bayesian probabilistic models gain more perspective and insights into their utility within ML. A recent approach combining neural network models and the Bayesian paradigm introduced by Berthier et al. [9] performs a dimensionality reduction technique using feature extraction algorithms to abstract the behaviour of a neural network into a Bayesian Network (BN). The authors identified hidden features learned by hidden layers of the DNN and associated each feature with a node of the BN. The BN is therefore defined based on high-level features, rather than on low-level neurons. This probabilistic abstraction of deep neural networks gives an ideal framework for investigating the DNNs internal representation through analysing the extracted hidden features.

The remainder of this introductory chapter is structured as follows. Section 1.2 explains the motivation for conducting this research. The main research question and its associated research issues are discussed in Section 1.3. In Section 1.4, the adopted research methodology to address the research questions is presented. Section 1.5 summarises the research contributions followed, in Section 1.6, the structure of the rest of the thesis is outlined.

1.2 Motivation

To address the aforementioned concerns, the main motivation for the work discussed in this thesis is defining a testing approach that uses semantic aspects of neural networks. The majority of existing testing methods are directed at the syntactic component of the neural networks, in particular, the testing adequacy is measured based on neuron-level properties. Intuitively, when statement coverage is achieved through traditional program testing, it implies new functionality is exercised. However, a neuron being activated does not have the same implication. It is possible that a model's semantics lies in the distribution of the entire activation vector instead of a set of discrete events. The fundamental idea presented in this thesis is to design advanced high-level testing metrics that are compatible with the working mechanism of neural networks and their behaviour in decision-making.

During the training process, DNNs learn to represent the input data in a lower-dimensional latent feature space, which captures the input's essential features in a more abstract and general form [33]. The core concept to be integrated into the suggested metrics is the Feature Importance (FI) measure. The notion of feature importance exists in explainable AI, which uses the relative importance of features in an *input test sample* to explain learning models' predictions. The method assigns to each feature an importance value which represents how much that particular feature was important for the prediction under analysis. A variety of methods have been developed to interpret neural network predictions by providing feature importance maps, such as Local Interpretable Model-agnostic Explanations (LIME) [78], Deep Learning Important FeaTures (DeepLIFT) [85], Integrated Gradients [91], and SHapley Additive exPlanation (SHAP) [61]. In contrast to these methodologies, the proposed importance scores are derived from latent features extracted from a neural network's layers.

After performing the interior analysis on the network's internal representation, the semantic mechanism can be modelled as a quantity of weights. The suggested testing criteria are based on the hypothesis that the contributions of features to a model's classification decision represent the most reasonable basis to measure the test data coverage and report the quality of the model performance.

1.3 Research Questions

Given the previous motivations, this thesis seeks to address the following primary research question:

"Is it a better measure of trustworthiness to measure the coverage of the semantic aspect of deep neural networks and treat each internal component according to its contribution value to the decision when testing these learning models' performance than relying on traditional structural unweighted measures?"

Providing an answer to this research question involves resolving the following Subsidiary Research Questions (SRQs):

1. **SRQ1:** Is a Bayesian abstraction of a neural network able to systematically analyse a DNN's interior decisions?
2. **SRQ2:** How can the importance of the latent features of a deep neural network be quantified using an abstracted Bayesian Network?
3. **SRQ3:** How to demonstrate the impact of the *feature importance* measure on the classification decisions based on the abstraction?
4. **SRQ4:** Do existing testing metrics guarantee the coverage of a model's critical internal regions, as well as direct the test case generation algorithm to target the most relevant features?
5. **SRQ5:** Do the proposed coverage metrics deliver a reliable testing measurement in terms of reporting the coverage that prioritises the important internal representation of the model?
6. **SRQ6:** Does the generated test dataset from the feature weight directed concolic testing provide a *trustworthy* measure of a model's performance?

1.4 Research Methodology

To provide answers to the research questions listed above, the place to start was to review existing work related testing deep neural networks where coverage criteria and test case

generation are applied. Identifying their shortcomings and limitations is the first step in the search for a solution. Since my aim is to investigate a high-level testing strategy for deep learning models, Bayesian principles were the first option to address this issue. The reason for choosing Bayesian networks as a framework to analyse the DNNs' internal representation is its clear characterisation of the connection between the variables, which gives it the strength to model and abstract the internal behaviour of the DNNs. Moreover, Bayesian networks allow probabilistic inference, which can contribute to identifying uncertainties in the model or potential areas where further refinement is needed. These capabilities make it a robust tool for analysing complex systems, such as deep learning models. Hence, the Bayesian network scheme was intensively studied to identify an appropriate method of analysing the DNN's features, which are represented with BN's nodes, sensitivity to perturbation.

Figure 1.1 outlines the proposed methodology to design a weight-based semantic testing metrics for neural networks using the Bayesian network abstraction model of Berthier et al. [9]. The testing framework illustrates three main phases (four, five, and six), each explored in a specific chapter corresponding to the marked number. The first phase is calculating the DNN's latent features importance weights based on the distance between two probability distributions represented by the abstracted BN. The suggested method involves measuring the DNN's features sensitivity to perturbations, which may capture the actual contribution made by each feature to the decision. This process results in a matrix of various computed weights according to given distance metrics. The second step is performing BN-based inference to extract supportive feature weights by observing the change in BN prediction after perturbing the latent features. This BN prediction weight is then compared with the weights produced from different distance metrics in the first phase, which are highly correlated with it, to be passed on to the next stage. Finally, developing weight-based feature coverage metrics that measure the test data's adequacy, taking into account the feature weight. The primary testing objective is ensuring that maximum test coverage is obtained from the presence of important features that have a dominating influence on the classification decision. Thereafter, the test generation algorithm will be guided by the importance scores to synthesise new test cases targeting the highest-weight feature. Each iteration of the concolic testing either generates a new test input that passes the test oracle, an adversarial input that has the wrong classification label, or the attempt fails and the algorithm moves on to the next iteration. The coverage is updated properly according to the outcome of each iteration.

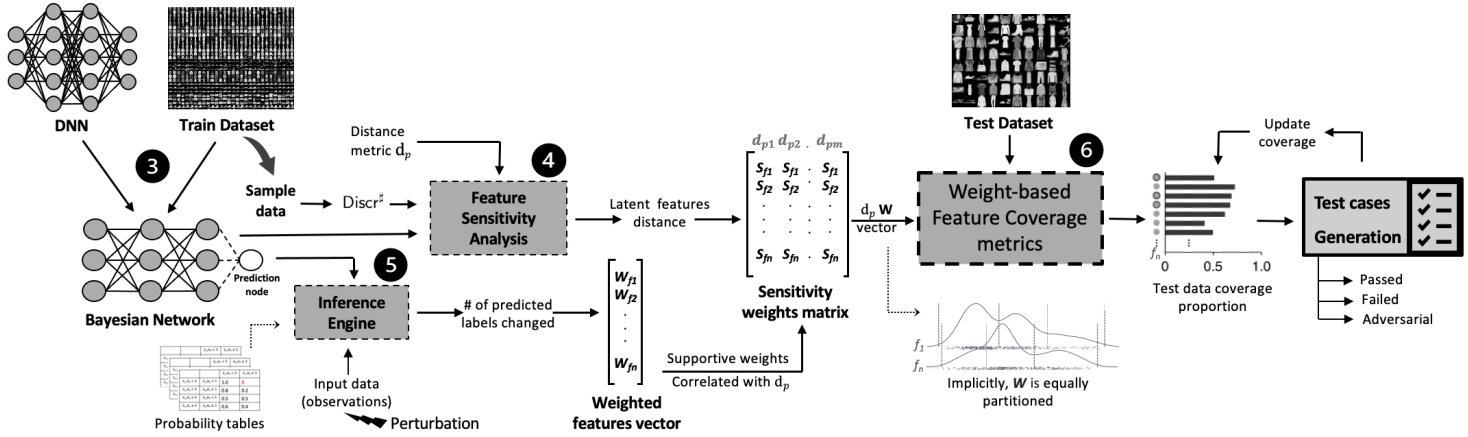


Figure 1.1: The outline of the proposed approach explains the overall framework for the weight-based test dataset generation. The methodology involves three phases; each dashed rectangle is a contribution presented in a separate chapter. The number on the top of each rectangle indicates the chapter in which each contribution is addressed. Number three represents the Bayesian Network construction step that is discussed in Chapter 3.

1.5 Contributions

The work presented in this thesis makes the following contributions to the neural network and Bayesian network fields. The corresponding chapter in which each contribution is addressed is indicated in parentheses.

1. Introducing a method to quantify the importance of latent features in neural networks by implementing the Bayesian network sensitivity analysis algorithm (Chapter 4).
2. Detecting adversarial distribution shift using the developed BN-based feature sensitivity analysis technique (Chapter 4).
3. Advancing the Bayesian network abstraction model with the ability to perform the original neural network classification task (Chapter 5).
4. Demonstrating that the constructed BN predictor is a good approximator of the original DNN and exhibits a smaller gap between accuracy over clean data and accuracy of perturbed data (Chapter 5).
5. Extracting supportive feature weights through perturbing each feature and observing the effect on the classification output (Chapter 5).
6. Designing new BN-based feature coverage metrics that use the developed importance weights (Chapter 6).
7. Generating test inputs targeting uncovered feature with the highest weight using the concolic test case generation algorithm (Chapter 6).

1.6 Thesis Structure

The remainder of this thesis is structured as follows:

Chapter 2: Deep Neural Networks: Background and Related Work. This chapter provides background and relevant literature to the work presented in this thesis. The chapter presents the deep neural networks definitions and their formal notations. It then discusses related safety concerns and some other issues, emphasising the adversarial perturbations in further detail. This is followed by a review of existing testing techniques

and their limitations. The chapter closes with general approaches designed to explore the DNN's internal decisions.

Chapter 3: Bayesian Network Abstraction: Definition and Preliminaries. The fundamentals and preliminary concepts of Bayesian networks are presented in this chapter. The chapter discusses how researchers utilised Bayesian network modelling to deal with neural networks' issues. It then introduces the abstracted Bayesian network schemes, which will be the basis for algorithms, models, and coverage metrics presented later in the thesis.

Chapter 4: Features Sensitivity Analysis. This chapter quantifies the importance of latent features in neural networks and defines the feature importance measure. It develops the BN sensitivity analysis algorithm to estimate the importance of a neural network's latent features by analysing an associated Bayesian network's sensitivity to distributional shifts. The work related to this chapter is published at the AAAI Workshop on Artificial Intelligence Safety (SafeAI 2022), as a technical paper entitled "Quantifying the Importance of Latent Features in Neural Networks", see Alshareef et al. [2].

Chapter 5: Bayesian Network Prediction. This chapter extends the abstracted Bayesian network to act as a classifier and perform prediction through probabilistic inference. The approach was adding an auxiliary prediction node at the end of the Bayesian network and calculating its conditional probability table according to the BN nodes' distribution. The BN prediction is used to compute features weights by perturbing the intervals of each feature for every sample in the test set and measure how many outputs have changed their value. The work related to this chapter is published at the International Conference on Machine Learning and Cybernetics (ICMLC) 2023, see Alshareef et al. [3].

Chapter 6: Weight-based Testing Metrics. This chapter designs weighted-based testing metrics using the developed feature importance weights to evaluate the coverage of a test set. Two main metrics are defined, with a third combining both. The chapter also presents the Concolic test cases generation algorithm that produce additional test inputs prioritising the higher weighted feature. The results from an extensive evaluation using the Fashion-MNIST and CIFAR-10 datasets and four testing metrics are presented. The work related to this chapter is published at the IJCAI Workshop on Artificial Intelligence

Safety 2023 (AISafety 2023), as a technical paper entitled "Weight-based Semantic Testing Approach for Deep Neural Networks", see Alshareef et al. [4].

Chapter 7: Conclusion and Future Work. The last chapter concludes the thesis with a summary of the discussed materials and the main findings with respect to the main research question and its subsidiary questions. The chapter closes with a discussion of potential future research directions.

Appendices: The appendices section contains further supplementary materials that support the experiments presented in the thesis. It includes the structure of the used DNNs models, additional figures, and resulting data tables.

References: This section contains a list of the references cited throughout the thesis.

1.7 Publications

A number of academic publications have cited the work provided in this thesis. The published and submitted papers are listed below, with a brief description of each publication that highlights its relevance in relation to this thesis.

1. Nicolas Berthier, **Amany Alshareef,** James Sharp, Sven Schewe, and Xiaowei Huang. Abstraction and symbolic execution of deep neural networks with Bayesian approximation of hidden features. arXiv preprint arXiv:2103.03704, 2021. This paper presents a novel abstraction method that abstracts a DNN and a dataset into a Bayesian network (BN). This suggested dimensionality reduction on the DNN's hidden features can improve its scalability and the size of the dataset as well, which allow feather analysis on its behaviour. I contributed to this paper through the first couple years of my PhD study as a co-author. The content of this paper is presented in Chapter 3, and it will be used as the foundation for the work investigated in the next chapters.

2. **Amany Alshareef,** Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Quantifying the importance of latent features in neural networks. In the AAAI Workshop on Artificial Intelligence Safety Proceedings, volume 3087, 2022. This technical paper investigates how the distribution of a neural network features in their latent space

changes in the presence of distortions. This achieved through abstracting the neural network into a Bayesian Network introduced in the paper 1. The method estimates the importance of each feature by analysing the sensitivity of the abstraction to targeted perturbations. The work conducted in this paper and the evaluation results are included in Chapter 4.

3. **Amany Alshareef,** Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Robust Bayesian abstraction of neural networks. In proceedings of the International Conference on Machine Learning and Cybernetics (ICMLC), 2023. This conference paper extends the BN model to make predictions, and it argues that the designed BN classifier is a good approximator of the original DNN. The paper demonstrated that the BN inference approximates the deep neural network prediction performance while outperforming it in an adversarial setting. The content of Chapter 5 is derived from this paper, where the BN classifier is used to examine the extent to which a perturbation on a DNN hidden feature may affect the classification decisions.

4. **Amany Alshareef,** Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Weight-based semantic testing approach for deep neural networks. In the IJCAI Workshop on Artificial Intelligence Safety, volume 3505, 2023. This technical paper proposes weight-based testing metrics that use feature importance weights to measure the coverage of the test set and facilitate the generation of additional test cases targeting higher weights' features. The suggested semantic metrics addressed the limitations of the current DNN's testing methodologies, which focus on structural testing coverage and ignore the semantic representation that forms in a DNN during the training process. The published outcomes of the weight testing metrics experiments were part of Chapter 6 content.

1.8 Summary

This chapter has introduced a background overview of the essential ideas of this thesis. The chapter discussed the motivation to carry out this research, the research questions, the proposed research methodology, and the main contributions made through this thesis. It then closes with the publication papers that have been published from the delivered contribution. The following chapter presents the background of the DNNs and a literature

review investigated to provide more detail regarding their current testing methods and their limitations.

Chapter 2

Deep Neural Networks Testing: Background and Related Work

2.1 Introduction

The widespread adoption of deep neural networks (DNNs) has led to significant breakthroughs in various fields such as computer vision, speech recognition, and natural language processing. However, the complexity and non-linearity of DNNs make them challenging to debug, and validate. In particular, the lack of robustness in DNNs under adversarial attacks and the difficulty of measuring their prediction uncertainty are limit their applicability in safety-critical applications and areas where explainability is essential.

Artificial Neural Networks are a set of algorithms modelled loosely to adopt the notion of a human brain that is designed to cluster, classify, and recognise patterns [69]. Deep Neural Networks, denoted as DNNs, consist of multiple simple, connected computing units named neurons $\left\{n_{i,1}, \ldots, n_{i,|l_i|}\right\}$ organised in interconnected layers. Each neuron performs two main tasks: the computation of a weighted sum of its inputs and the execution of an activation function. The weighted sum is computed by multiplying the input values by corresponding weights and adding them together. The activation function introduces non-linearity to the neuron's output, enabling the network to capture complex relationships and learn non-linear mappings. The commonly used activation functions are Sigmoid, Tanh, and Rectified linear unit (ReLu), which are the most popular activation functions. Mathematically, the computation within a neuron can be described as follows:

1. The weighted sum: $\hat{n}_{i,j} = W_{i,j}\left(n_{i-1,1}, \ldots, n_{i-1,|l_{i-1}|}\right) + b_i$.
2. The activation function: $\text{ReLU}(\hat{n}) = \max\{0, \hat{n}\}$.

The weight $W_{i,j}$ of the connection between neurons and the bias b_i are learned and adjusted during the training process by minimising a cost function over the training data. The majority of the current DNNs are trained with a gradient descent algorithm using backpropagation process, which used to compute the partial derivatives of the gradient. The process of training a DNN involves feeding it with a large amount of labeled data, allowing it to learn from the input-output patterns and adjust its internal parameters accordingly. Thus, the DNN's layers are responsible for transforming input data through a series of mathematical operations, gradually extracting and abstracting relevant features from raw data.

A Convolutional Neural Network (CNN) is a specialised type of DNN primarily designed for processing and analysing visual data such as images. Unlike traditional fully connected networks, CNNs are consist of layers that detect local patterns via convolution operations [36]. These layers are designed to automatically and adaptively learn spatial hierarchies from the input image, which allows the network to recognize complex visual features ranging from simple edges to sophisticated patterns. CNNs often comprise convolutional layers, pooling layers, and fully connected layers.

By leveraging the power of deep architectures and the ability to automatically learn representations, DNNs have achieved state-of-the-art performance in numerous domains and have gained popularity in critical applications, which means ensuring their reliability and trustworthiness becomes paramount [45]. Testing plays a crucial role in assessing the performance, robustness, and generalisability of DNNs [88]. It enables vulnerability detection at an early stage. The objective of testing techniques is to provide assurance by exposing the system to a comprehensive set of test cases; particularly, coverage-guided testing generates test cases based on a predefined set of coverage metrics. Intuitively, high coverage indicates that the majority of the DNN's behaviours have been tested, reducing the chance that the DNN contains undiscovered bugs.

The remainder of this chapter is structured as follows. The challenges and concerns associated with neural networks are discussed in Section 2.2. This includes issues such as the black box nature of DNNs, uncertainty, overfitting, overconfidence and adversarial attacks. Section 2.3 highlights the DNN properties that need to be tested before deployment, including accuracy, robustness, fairness, interpretability, and trustworthiness. Section 2.4

explains the concept of trustworthiness of DNNs in further details and the factors that contribute to it. Section 2.5 reviews the current DNNs testing metrics and test case generation algorithms. Last section 2.6 discussed several methods to explore the internal representation of the DNN models. Finally, the chapter is concluded, in Section 2.7, with a brief summary.

2.2 Challenges and Concerns

As stated earlier, several challenges and concerns associated with DNNs need to be carefully addressed. This section discusses specific issues that raise serious safety considerations.

Adversarial Perturbations. A key safety property of neural networks relates to the robustness of a model with respect to noise. The robustness property states that all inputs within some region of the input space have to be given the same class label. This means that small perturbations applied to an input sample should not cause significant losses to the network's performance. However, it is well-known that deep neural networks are vulnerable to adversarial examples, which are carefully perturbed inputs that confuse the deep learning models so they provide entirely different predictions. The mechanism of adversarial attacks is applicable by performing input-space gradient descent that maximises the loss of random samples at testing time. A variety of techniques for generating adversarial images have appeared in recent years. Here is a brief review of three typical state-of-the-art adversarial attacks that are utilised throughout the thesis experiments:

- The Fast Gradient Sign Method (FGSM) of Goodfellow et al. [35] is a one-step attack method seeking to craft the adversarial perturbation by moving in the opposite direction to the gradient of the loss function w.r.t. a given input image;
- The Projected Gradient Descent (PGD) of Madry et al. [65] is a multiple-step perturbation attack projecting an initial point that maximises the loss back into the ε-ball around the original input until it finds an adversarial example;
- The DeepFool algorithm of Moosavi-Dezfooli et al. [68] performs an iterative search for the minimal norm perturbation that can change the decision of the classifier. It uses geometry theory to direct the search.

Example 1. *An example of adversarial perturbations is shown in Figure 2.1 [88]. Suppose an image of a traffic light is attacked by making subtle, imperceptible changes using one of*

the methods above (adding a small amount of carefully crafted noise). These perturbations can force the model to misclassify it as a completely different object, like lipstick in the figure. This highlights the vulnerability of DNNs to adversarial inputs.

Figure 2.1: An example of an adversarial perturbation that results in predicting the traffic light input image with a lipstick. Adapted from Sun et al. [88]

Out-of-Distribution Inputs. DNNs struggle to handle inputs that lie outside the training data's distribution. When faced with unfamiliar inputs, DNNs may produce unreliable or erroneous outputs. This poses risks in situations where the model encounters novel or rare scenarios and can lead to poor decision-making and potential safety hazards. It is crucial to guarantee a DNN's safety by estimating whether a given input sample is drawn from the same data distribution for which the DNN was trained. Many Out-of-distribution detection methods have been developed to determine if the behavior of an input is similar to the behavior of the training samples of the DNN under investigation [25, 58].

Example 2. *In autonomous vehicles, DNNs are used for object detection and recognition. When faced with unusual road conditions that fall outside the scope of the classes the network has been trained to recognise, e.g., heavy fog, extreme weather, or objects not present in training data, the DNN may struggle to correctly identify objects or make safe driving decisions. This situation is considered an out-of-distribution input for the model.*

Uncertainty. As DNNs are not probabilistic in nature, they are incapable of providing confident predictions, which makes it difficult to grasp the uncertainty of their predictions. Providing an uncertainty estimate is crucial for reliable decision-making, risk assessment, and model deployment in real-world applications. Several approaches have been developed to capture and quantify uncertainty in DNNs, including Bayesian neural networks [75, 53], Monte Carlo dropout [29], and ensemble methods [66].

Non-interpretable logic. DNNs are considered black-box models that hide their logic in their internal processes, making it difficult to understand how they arrive at their predictions or decisions. Lack of interpretability and explainability may hinder the system's trust, especially if it is installed within a safety-critical application and cannot clarify the model's decision-making process. This has led to the emergence of a considerable number of studies and experiments to enhance the deep learning models interpretability. Popular methods are gradients explanation technique [86], the Local Interpretable Model-agnostic Explanations (LIME) method [78], the Class Activation Maps (CAMs) [108] and its extension versions Grad-CAM and Grad-CAM++.

Overfitting and Generalisation. Another issue that may occur during neural network training is overfitting. The model has learned the training set features exceptionally well, so that the loss metric is driven to a minimal value. However, when provided with new data that slightly deviates from the exact data used during training, the model will fail to generalise and accurately predict the output. In other words, overfitting occurs when the model performs better in the training data than in the test data. That primarily happens when the training data set is not sufficient or the model is too complicated for the data, so it also fits the noise, which are irrelevant features, of the training data. Different approaches may be applied to reduce the overfitting, such as data augmentation and dropout regularisation techniques [41, 55].

Example 3. *The two trained models in the figures below illustrate overfitting (Figure 2.2) and the optimal balanced fit (Figure 2.3). In the overfitting case, the DNN becomes increasingly specialised in memorising the training data rather than generalising from it.*

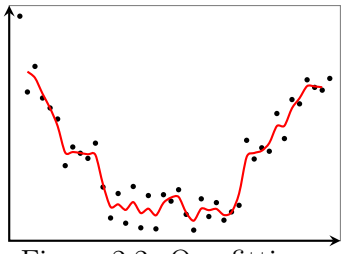


Figure 2.2: Overfitting

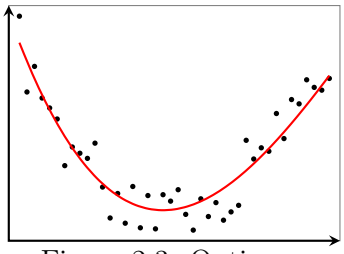


Figure 2.3: Optimum

2.3 DNN Properties to Be Tested

Deep neural networks are complex models that require thorough testing to ensure their effectiveness and reliability. The testing properties refer to the "what" function to be tested. They comprise both primary functional requirements, such as correctness and model relevance, as well as non-functional requirements, including efficiency, reliability, robustness, fairness and interpretability. By evaluating these properties, a comprehensive understanding of the DNN's performance can be gained, ensuring its suitability for the intended application.

Testing for correctness involves validating that the DNN produces accurate and reliable outputs for a given set of inputs. This includes determining whether the model's predictions align with the ground truth labels or expected outcomes. Techniques such as cross-validation, model accuracy measurement, and precision-recall evaluation can be employed to assess the correctness of a DNN. Regarding model relevance evaluation, it focuses on assessing whether the DNN is capturing and utilising the relevant features in the data and ensuring that the adopted machine learning algorithm is not over-complex than is needed. Poor model relevance often leads to overfitting or underfitting issues [51].

Model robustness is defined as how well a learning system is able to function correctly in the presence of invalid inputs or a hostile environment [1]. Ensuring the robustness of a DNN model is particularly vital in safety-critical applications, where the failure of the model might cause serious consequences. Adversarial robustness testing such as CLEVER score [94] is a popular method for evaluating the neural networks' robustness. Reliability is the probability that a model will achieve its required functions in the provided environments without failure for a specified period of time with a specified level of confidence. Furthermore, machine learning learns from data provided by humans. Therefore, checking the fairness and mitigating any biases or discriminatory behaviour exhibited by these models are necessary, especially in decision-making such as income prediction. Lastly, testing the ability of the model to produce explainable and interpretable predictions. Systems whose decisions cannot be well explained are hard to integrate into critical fields. Eventually, the trustworthiness of deep learning models became a crucial property for their widespread adoption and integration into various real-world applications. It refers to the ability of these models to produce reliable and consistent results while maintaining transparency and interpretability. The next section provides a further explanation of it, highlighting key factors and considerations that contribute to the overall trustworthiness of deep learning

models.

2.4 Trustworthiness of Deep Learning Models

One can trust intelligent systems if they have proven to function correctly and safely and if any of their behaviours can be clearly explained [46]. In safety-critical applications, the provision of safety evidence for the embedded learning systems' trustworthiness is required, where the consequences of errors can be severe [12].

The first process to address the model trustworthiness is related to verification, testing, adversarial attack and defence. Deep learning models requires comprehensive evaluation and testing methodologies to verify their performance, assess their vulnerabilities, and develop defenses against potential attacks. Testing, which is the main focus of this thesis, considers the primary instruments for evaluating the quality of the model's performance and providing safety assurance for its trustworthy and robust behaviour. Measuring the model's predictive capabilities on test datasets using evaluating metrics such as accuracy, precision, recall, and F1-score is insufficient and cannot provide the required assurance. In safety domains, it is necessary to systematically test the learning models and synthesise a new set of test inputs to prove the model's generalisability and robustness across different scenarios and input distributions.

Overall, addressing the vulnerability of the learning-enabled systems to adversarial attacks, ensuring the quality and representativeness of the training data, forcing higher testing coverage of testing data, and enhancing the interpretability of deep learning models are essential steps towards ensuring the trustworthiness of these models and increasing confidence about their behaviour.

2.5 Existing DNN Testing Techniques

Deep Neural Networks testing is an active research area where the safety-critical applications being deployed with them. This section reviews the literature of the current DNN's testing techniques. Software testing is one of the most commonly used validation techniques to detect software defects that cause unexpected behaviour. Deep learning systems are tested through generating a set of well-studied test cases and then feeding these test data to them. The intention of that is to validate their run-time behaviour's accuracy by conducting a comparison between expected outputs and actual outputs. Hence, the per-

formance of a developed DNN can be more reliable when it passes the test cases. Usually, the generation of test cases and evaluation of test adequacy is guided by coverage metrics, which defined a set of requirements to be satisfied. Numerous testing approaches are proposed to address the issues associated with these learning models, including test coverage criteria, test generation, and test oracles.

2.5.1 Testing Metrics and Coverage Criteria

Testing metrics are methods that test neural networks by measuring the extent to which the test inputs cover different regions of the input space. Coverage metrics are used to measure the adequacy of testing by providing a quantitative evaluation of how thoroughly a deep neural network has been tested according to specific criteria.

Definition 2.5.1 (Testing Coverage Metric)**.** *A test coverage metric for a deep neural network $\mathcal{N}$, is a function Cov:$\mathcal{N} \times \mathcal{T} \times C \rightarrow$ [0,1], that quantifies the degree of adequacy, ranges over 0 to 1, to which a network $\mathcal{N}$ is tested by a test suite $\mathcal{T}$ with respect to a coverage criteria C [87].*

Intuitively, the coverage percentage denotes the ratio of test criteria that have been satisfied over a set of test cases.

Coverage-guided deep neural network testing techniques are a class of testing methods that aim to increase the coverage of the network during testing, with the goal of covering different regions of the input space and revealing as many potential bugs and unexpected behaviours as possible. Enforcing higher coverage during the testing process makes the network under investigation more likely to be robust and reliable. The existing related research work in the literature are divided into two categories:

(i) **Structural coverage metrics** that are defined based on the syntactic characteristics of the neural networks. Syntactic characteristics refer to the architectural aspects of a neural network, such as its neurons, connections between neurons, layers, and the mathematical operations executed within the network. These aspects are concerned with how the network is physically organised and how data flows through it. Thus, structural coverage, measures how much of this physical architecture is exercised or covered during testing.

(ii) **High-level semantic coverage metrics** that are concentrated on the semantic representations created by neural networks. It is concerned with the high-level features

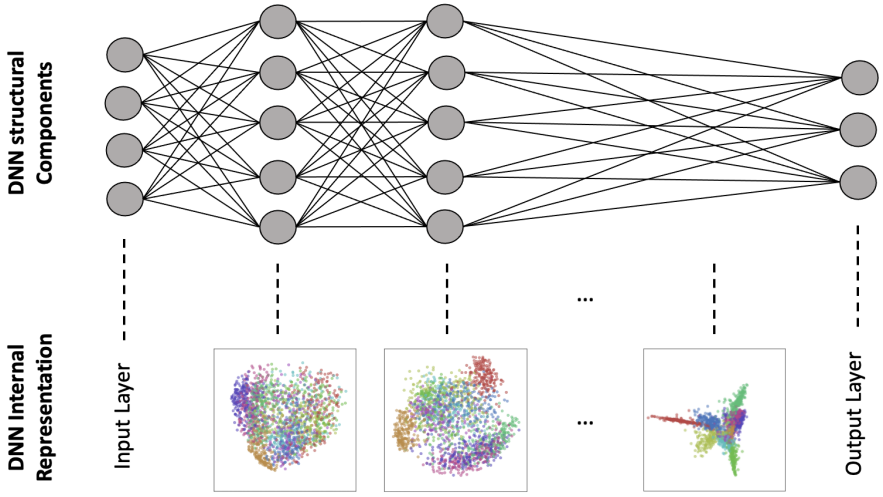


Figure 2.4: A deep neural network shows its structural components and the internal semantic representations that the network learned from a given data.

or patterns that the network extracts and understands from the data.

Example 4. *Consider a deep neural network in Figure 2.4. Structural coverage testing would ensure that each neuron, or at least a significant proportion of neurons in the network are activated during testing. It also could involve evaluating whether different combinations of layers and architectures are thoroughly tested to guarantee that the network's structural components are well covered. For semantic coverage, instead of just examining whether neurons are activated, the semantic coverage is concerned with whether the test images span a wide range of learned features that the network is supposed to recognise. For instance, if the network is trained to distinguish between cats and dogs, the test may cover various breeds, colours, sizes, and postures of cats and dogs. Doing so ensures that the network's learned representation of cats and dogs is being thoroughly tested. In simpler terms, structural coverage is concerned with HOW the network is built and connected (the top part of the diagram), while semantic coverage is concerned with WHAT the network understands or learns from data (the bottom part of the diagram).*

Most proposed testing approaches have been focused on the *structural testing coverage* to measure the coverage of the dataset relied on the individual neuron activation. These techniques are based on the idea of activating as many neurons in the network as possible

during the testing phase. The more neurons that are activated with a specific value, the more complete the testing of the network is considered to be.

Neuron activation was first introduced by Pei et al. [73] as a systematic metric which calculates the number of activated neurons (w.r.t. ReLU activation function) during the testing. They proposed DeepXplore, which is a white box differential testing algorithm for generating test inputs that can discover inconsistencies between multiple DNNs. It employs a technique called mutual exclusiveness, which guarantees the test inputs do not activate the same neurons as previously generated inputs, further increasing the diversity of the test inputs. Tian et al. [92] proposed DeepTest, a testing tool that automatically generates test cases relying on neuron coverage as a guidance mechanism for exploring different parts of the DNN logic. They concentrated on detecting erroneous behaviours of DNN-driven autonomous vehicles. Guo et al. [38] improve the neuron coverage by combining the fuzzing testing into the deep learning systems. Although neuron coverage has been shown to be effective at finding hidden bugs and been used to test real-world deep neural networks, investigates by Sun et al. [87] were demonstrated that neuron coverage is too coarse and easy to achieve.

Further approaches have been developed to extend neuron coverage with a focus on various activation value factors. Ma et al. [62] presented DeepGauge, a set of five coverage criteria based on layer-level and multi-granularity coverage. Their method followed a similar principal as neuron coverage, however, they intended to identify specific values of neuron activation for their metric such as neuron boundary. Sun et al. [87] introduced several coverage criteria inspired by Modified Condition/Decision Coverage (MC/DC) that are captured the causal changes in the test data by observing a sign, value, or distance change of a neuron activation value. They implemented the DeepCover tool that generates test cases based on linear programming.

The quantitative projection coverage, suggested by Cheng et al. [19], considered different criteria, each with its own weight, that described the operation conditions. They then partitioned the input domain according to these criteria and weighted each distinct class based on its relative importance. More testing metrics, i.e., safety coverage [95] and surprise coverage [50], have been designed based on the activation functions and the syntactic connections between neurons in successive layers.

Unfortunately, neuron activation and other structural coverage techniques, that are defined based on the syntactic model components have proven to be less effective in validating the safety behaviour of the intelligent systems. A study by Li et al. [60] showed

that there is no correlation between the number of misclassified natural input tests and their structural coverage on the corresponding neural networks. There is still considerable ambiguity on how such coverage criteria directly relates to the decision logic of black-box machine learning systems. Especially given that the semantic relationship between layers is ignored. The semantics of DNN models differ significantly from the semantics of the traditional programs. Simply because learning model behaviours are continuous, whereas program behaviours are discrete. Additionally, structural coverage has a limited correlation with network robustness, where high neuron coverage does not imply the network is robust to all possible inputs or it will behave well on unseen data [27, 98].

There are relatively few testing strategies that address the semantic aspects of DNN's internal representation. One recent effort is the Bayesian network-based feature coverage that will be studied in Chapter 3. Those higher-level testing criteria [9] are defined based on a combination of a dimensionality reduction technique, and the construction of a Bayesian Network, which capture how a given test dataset exercises high-level features that have been learned by hidden layers of the DNN.

2.5.2 Test Cases Generation Algorithms

Generating effective test cases for DNNs is a crucial process in order to report reliable testing accuracy. The existing set of test case generation algorithms for DNNs can be categorised according to the techniques utilised as follows:

1. Input-based Techniques, which rely on manipulating the input data to create test cases. These techniques are sub-divided into:
    - Mutation-based Methods, that modify specific characteristics of existing inputs to generate new test cases.
    - Randomised Testing (Fuzzing), that focuses on feeding random inputs to a DNN to explore its behaviour.
2. Formal Methods, which include techniques that have roots in formal verification:
    - Symbolic Execution, that evaluates the programme paths with symbolic values.
    - Concolic Testing, which is a hybrid approach that combines both concrete and symbolic execution.

A detailed description of each test input generation approach is provided below.

Input Mutation Test Input Synthesis. Input mutation methods generate new test inputs, either natural or adversarial, by altering the original data using transformation rules. The DeepXplore search algorithm in [73] used a gradient search to find a modified input that aims to discover inconsistent behaviours between multiple DNNs models. They introduced neuron coverage as a systematic metric to drive test generation under domain-specific constraints. Following that, DeepTest [92] applied greedy search to automatically generate test cases through implementing various transformations on seed images. On the other hand, the adversarial inputs using Generative Adversarial Networks (GANs) are widely used, i.e., [50, 62] to produce new test input.

Fuzzing Test Input Generation. Fuzz testing is an automated testing technique that generates random input data to detect faults and vulnerabilities in a model. The DLFuzz [38] uses a combination of coverage-guided fuzzing and differential testing to generate test cases for deep neural networks. In TensorFuzz [71], the inputs are randomly mutated based on user-specified constraints, and the guided coverage is measured by a fast approximate nearest neighbour algorithm.

Symbolic Execution Test Input Generation. Symbolic execution is an analysis technique that treats software variables as symbolic values to test whether specific inputs cause each part of a system to be executed. The work in [77] discusses an approach for testing intelligent systems that combines symbolic reasoning and statistical methods. The authors used decision procedures to generate test cases and statistical hypothesis testing to analyze the results. The paper provides examples of how this approach can be applied to testing image classification systems and autonomous vehicles. The results suggest that the proposed testing methodology can improve testing coverage and the accuracy of identifying errors in intelligent systems.

Concolic Testing is a testing technique in which concrete execution directs the symbolic analysis to generate a high coverage test suite. The DeepConcolic approach introduced in [89] is the first proposed concolic testing approach for DNNs that combines program execution and symbolic analysis to explore different execution paths in the network. The DeepConcolic algorithm alternates between:

- **Concrete executions** that evaluate a given coverage requirement's satisfaction of a test input within a test suite to be selected as a concrete input and encoded symbol-

ically in the next stage.

- **Symbolic analyses,** which synthesise new test inputs based on the chosen concrete test target, the process of finding a new input is equivalent to finding a new activation pattern. Therefore, the symbolic analysis method involves linear programming that encodes the activation value of each neuron of the concrete input as a linear constraint, which produces a set of inputs that exhibit the same ReLU behaviour as encoded. To check the usefulness of the obtained test case, an optimisation objective is added to the LP problem that minimises the distance between the two tests.

2.5.3 Test Case Evaluation

A **Test oracle** is a reference or ground truth used to evaluate the performance of the DNNs and ensure they behave correctly for a test case. The test oracle provides the expected output for a given input and is used to compare the output of the system to determine its accuracy. In the context of deep neural networks, the test oracle can be human-generated labels or ground truth data. In other cases, it can be the output of a simpler or more well-established model that serves as a reference for evaluating the performance of the new model.

Overall, structural coverage criteria, such as neuron coverage, focuses on the patterns that appear in the outputs of ReLU activation functions; while semantic coverage is a higher-level criteria that focuses on the features that have been learned by hidden layers of the DNN. The proposed semantic testing metrics are based on the model-internal representations and their roles in the input/output behaviour. Therefore, it is essential to first analyse and identify the features or factors that drive the model's decision and then build the testing metric based on the analysis results. The following section discusses prevalent techniques for exploring the DNN's interior decision-making mechanism.

2.6 Methods for Exploring the DNN's Inner Decisions

Since the proposed testing metrics target the semantic aspect of DNNs, understanding their decision-making processes is crucial for gaining insights into their inner workings. Exploring the inner decisions of DNNs involves identifying the factors, features, or patterns that influence the model's output and obtaining knowledge into the learned representations.

Backpropagation techniques analyse the gradients of the model's output and backpropagating them into the input data domain to obtain the regions of the input that have the most influence on the model's prediction. Popular methods are gradient-based methods [86], Gradient-weighted Class Activation Mapping (Grad-CAM) [84], Deconvnet [100], and Layer-wise Relevance Propagation (LRP) [6], etc. These methods either modify the algorithm of the gradient computation or the neural network architecture to produce saliency maps or heatmaps. By achieving this, they can visualise the gradients, which means identifying the important features or regions of an input that contribute to the decision.

Figure 2.5: Examples of the produced saliency maps using a single back-propagation pass through a ConvNet classification neural network [86].

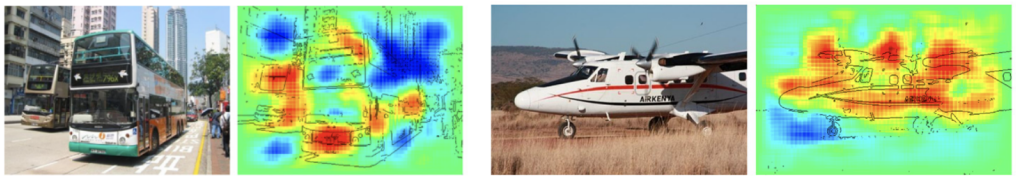


Figure 2.6: Examples of the computed heatmaps using the Layer-wise Relevance Propagation (LRP) [6].

Example 5. *In Figure 2.5, a visualisation of three saliency maps is extracted from input images using the gradient-based method. The gradients illustrate how much each pixel in an input image needs to change to impact the model's prediction. In the figure, the computed gradient in each aliency map indicates which pixels in the image were essential for the model to make its decision. Another example of interpreting classification decisions is*

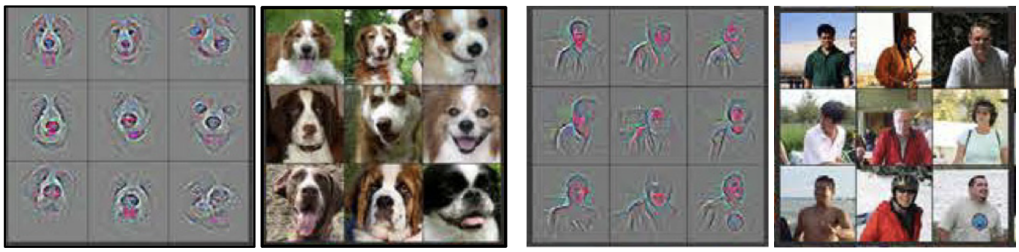


Figure 2.7: Visualisation of DNN intermediate feature layers [100].

illustrated in Figure 2.6, which visualises the pixel-wise decomposition process as heatmaps using layer-wise relevance propagation. The figure shows the pixel-wise predictions overlaid with prominent edges derived from an input image. This process involves using the image's feature vector representation and calculating relevance scores.

Perturbation-based methods such as Occlusion [100], LIME [78], RISE [74], and Extremal perturbations [28] that investigate the properties of DNN models by perturbing their input and observing output changes.

Example 6. *Figure 2.7 exemplifies intermediate features of different DNN's layers. To explore how discriminative these features are in each layer, an analysis is conducted through occlusion sensitivity experiments [100] . The method involves systematically occluding different portions of the input image with a grey square and observing the classifier's output.*

These methods are most frequently used in model interpretability, where they provide visual explanations of the DNN's important input elements. Nevertheless, these methods cannot be *easily* adapted to the purpose of the proposed semantic testing. For example, the perturbation methods modify pixels in an image or words in a text, which cannot be directly applied to modify the latent features. Quantifying the DNN's latent features importance is the first essential step to explore the DNN's inner workings.

2.7 Summary

This chapter has provided the essential background about issues concerning DNN and current testing techniques. The chapter commenced by presenting the DNN terminologies and mathematical notations used to construct them. It then highlighted various concerns that arise due to the complexity, ambiguity and inexplicability of DNNs. The chapter then

discussed different DNN aspects that need to be tested, emphasising the Trustworthiness property. Finally, a comprehensive review of the DNN existing testing techniques was presented, highlighting their limitations. Testing the behavior of DNN is a critical stage to ensure their effectiveness, reliability, and safety. The following chapter discusses Bayesian principles and how they became an integral part of machine-learning models.

Chapter 3

Bayesian Network Abstraction: Definition and Preliminaries

3.1 Introduction

There is an increasing adoption of Bayesian principles for addressing complex issues in neural networks [72]. This chapter provides a brief introduction to Bayesian networks, which are widely used for modelling complex systems. It begins by defining the basic concepts of Bayesian networks, such as nodes, edges, conditional probability tables, and Bayesian inference. It then reviews a number of BN modelling strategies for solving several issues of neural networks. The last section introduces the Bayesian Network abstraction model and the designed feature coverage metrics, which are essential parts for understanding the rest of the thesis.

A Bayesian network is a probabilistic graphical model that represents the relationships between variables and a set of Conditional Probability Tables (CPTs) that specify the probabilities of each variable given its parents in the graph.

Definition 3.1.1 (Bayesian Network)**.** *A Bayesian network over a set of random variables* $\{X_1, X_2, ..., X_n\}$ *is a pair (G,P), where G is a directed acyclic graph (DAG) G = (V, E) with n nodes* $V = \{X_1, X_2, ..., X_n\}$ *representing discrete variables and edges* $E \subseteq V \times V$ *representing the dependencies between the variables. P is a set of conditional probability distributions,* $P = \{P_1(X_1|\pi_{X_1}), ..., P_n(X_n|\pi_{X_n})\}$ *gives the probabilities of* $X_i \in V$*, given the values of the variables in its parent set* π_{X_i} *[21]. The joint probability distribution is*

factorised according to the chain rule of probability as:

$$P(X_1, X_2, ..., X_n) = \prod_{i=1}^{n} P(X_i|\pi_{X_i}) \tag{3.1}$$

where π_{X_i} denotes the set of parent nodes of X_i in the DAG. This factorisation allows efficient computation of the posterior distribution of the variables given evidence or observations.

Inference in Bayesian networks involves calculating the joint probability distribution of the variables using the network structure and CPTs, and then computing the posterior probability distribution of a set of variables given evidence or observations.

The Bayesian view of statistics treats the latent parameters as random variables and seeks to learn a distribution of these parameters conditional on what is observed in the training data. The Bayesian theory, therefore, is concerned with the parameter posterior instead of a point estimate which often leads to overconfident predictions as in DNNs. Converting a neural network from a graphical function representation to a graphical probability representation in the form of a Bayesian network can bring numerous improvements to DNNs. Bayesian networks provide an interpretable model, allowing users to understand how the model makes predictions and identify the most important features. They can also provide uncertainty estimates for predictions, which is useful in decision-making processes. Moreover, Bayesian networks are more robust to noisy and incomplete data than neural networks.

3.2 Utilisation of Bayesian Networks within Neural Networks

Bayesian theories have the capability of solving machine learning concerns as discussed by [72]. Bayesian networks are now widely used in deep neural networks for the purpose of Explainable AI (XAI) [106], DNN robustness and causality [52, 101], uncertainty quantification [24], DNN structure learning [80], and dimensionality reduction [70]. This section presents a short overview of recent work in this area.

The current field of deep learning systems focuses on constructing learning models whose behaviour is understandable and explainable. Zhao et al. [106] introduced the BayLIME method (Bayesian Local Interpretable Model-Agnostic Explanations) that can be used to improve the transparency and interpretability of machine learning models. They inte-

grated the Bayesian weighted sum into the LIME framework, one of the most widely used approaches to XAI, to obtain prior knowledge about the behaviour of the learning model. The results of the paper show that BayLIME outperforms the state-of-the-art XAI methods, such as LIME, SHAP, and Grad-CAM, in terms of consistency, robustness, and explanation fidelity.

For the problem of robustness of deep learning models, Zhang et al. [101] used Bayesian networks to build NNs robust against input manipulations. For this, the authors propose the deep CAusal Manipulation Augmented model (deep CAMA). The method exploits the idea that any input data is caused by its label, latent variables (e.g. writing styles) and interventions (e.g. rotations, adversarial attacks etc.). One then trains the distribution described with such a causal Bayesian network, and uses it as a robust classifier. The paper shows that the model outperforms discriminative NNs in terms of robustness against unseen manipulations, and argues that this is due to the lack of causal reasoning in discriminative NNs. Such a procedure is useful in various applications where robustness against input manipulations is important, such as image classification, speech recognition, and medical diagnosis. Similarly, Kristiadi et al. [52] discussed the overconfidence problem in ReLU classification NNs and how approximate Bayesian inference can improve predictive uncertainty. The authors theoretically analyse approximate Gaussian distributions on the weights of ReLU NNs and show that even a simplistic Bayesian approximation can fix these issues. Point estimates of ReLU classification NNs can yield arbitrarily high confidence far away from the training data, making the architecture neither calibrated nor robust. "Being Bayesian, Even Just a Bit" can be a sufficient condition for calibrated uncertainty in ReLU NNs.

In many other areas, Bayesian networks proved to be a successful approach. Daxberger et al. [24] proposed subnetwork inference in the task of Bayesian deep learning, tackling the high computational cost of Bayesian inference with neural networks. The framework involves performing inference over a small subset of neural network weights while keeping the other weights as point estimates. The proposed method compares favourably to ensembles and less expressive posterior approximations over full networks, resulting in better uncertainty calibration and robustness to distribution shift. Rohekar et al. [80] tackled the problem of neural network structure learning as a problem of Bayesian network structure learning. They did this by constructing a Bayesian network as a hierarchy of independent features in the input distribution. The proposed algorithm learns the structure of NNs in an unsupervised manner at a low computational cost. Thus, the method proves useful for

applications with limited computational resources or where the interpretability of the network structure is important. Njah et al. [70] employed Bayesian networks for the purpose of data dimensionality reduction as an antidote to the curse of dimensionality. They developed the Interpretable Bayesian Network Abstraction (IBNA) framework, which describes high-dimensional data features as a set of latent variables. They first partitioned the data into clusters, such that every cluster was described with one latent variable connected with original data features in a Bayesian network. The method shows good performance compared to categorical PCA, Multiple Correspondence Analysis, and other compression algorithms. Finally, the authors argued that IBNA can be generalised for continuous variables as well.

Considering the aforementioned recent studies, the Bayesian paradigm in deep learning has been shown to be crucial for building robust, explainable, safe, and efficient ML. Bayesian networks, with their causal structure, fit a wide range of applications, and thus their influence will certainly grow in the future. The following section provides a comprehensive review of the BN abstraction model that serves as an underlying basis for the investigations of the next chapters.

3.3 Bayesian Network (BN) Abstraction Model

This section introduces the terminology of the Bayesian Network abstraction model, donated as BN, that will be used as the foundational framework of the following analysis. Due to the scalability problem that arises when analysing neural networks, Berthier et al. [9] used a statistical analysis of activations at network layers, and abstracted the behaviours of the DNN using a Bayesian Network. The abstraction method identified the hidden features that had been learned by hidden layers of the DNN, and associated each feature with a node of the BN. This abstraction scheme is defined as a probabilistic model based on high-level features rather than low-level neurons. Hence, these extracted latent features considered the minimal semantic components that can be analysed to understand the behaviour of the feature space and the internal logic of the analysed DNN.

The process of constructing the BN from a trained DNN involves three main stages:

1. **Extraction of hidden features.** This stage constructs a mapping from a high-dimensional space into a feature space. Here, the hidden features learned by the DNN layers are identified by using feature extraction techniques on neuron activation values induced by a given training set. The authors implemented two linear feature extraction approaches: Principal Component Analysis (PCA) and Independent Com-

ponent Analysis (ICA), and one non-linear technique: radial basis functions (RBF) kernel-PCA;

2. **Feature space discretisation.** Since the extracted features range over a continuous space, each feature component is discretised into a finite set of feature intervals according to different strategies, i.e., Kernel Density Estimation (KDE), uniform-based and quantile-based discretisation;
3. **Bayesian network construction.** This consists of representing the probabilistic distribution of each extracted feature with a node in the BN. Each node is associated with either a marginal probability table for hidden features of the first (shallowest) layer, or a conditional probability table (CPT) for hidden or output layers. A simple CPT representation is possible due to the previous discretisation step.

The remainder of the section comprises four sub-sections. Sub-section 3.3.1 commences with a brief explanation of the DNN latent features. It then describes the mechanism for performing the DNN dimensionality reduction using two feature extraction algorithms. Sub-section 3.3.2 discusses the discretisation techniques employed to divide the extracted features into specific number of intervals. The process of construction the Bayesian Network model is considered in Sub-section 3.3.3. Lastly, Sub-section 3.3.4 presents the Bayesian network feature coverage metrics, which are defined in accordance with the BN model.

3.3.1 DNN Hidden Feature Extraction

A significant factor contributing to the DNN's effectiveness is its ability to autonomously learn relevant features from raw data. These autonomously learned features are referred to as hidden, **latent features** because they are not directly provided as input or explicitly taught to the network. Instead, they emerge as a result of the network's attempt to model and generalise the data's underlying patterns and structures [37, 56]. Latent features are the intermediate and learned non-explicit representations of data within a DNN. These features capture the essential and often hidden structures and patterns within the input data. Yu et al. [99] demonstrates that DNNs transform raw input features into more invariant and discriminative representations through multiple layers of nonlinear processing. As raw input data is transformed layer-by-layer through the network, the DNN learns hierarchical representations: simpler features in earlier layers like edges in images, and more complex high-level features in deeper layers like the shape of an object or even its semantic meaning.

Dimensionality reduction approaches leverage the DNN latent features to extract and

represent the hidden and essential structures of high-dimensional data. The purpose of dimensionality reduction techniques is to compute a mapping from a high-dimensional space into a much lower-dimensional space, called the feature space. Computing methods for such mappings usually rely on statistical principles and operate on a given sample of high-dimensional data. This section describes the mechanism of two well-known feature extraction algorithms called Principal Component Analysis (PCA) and Independent Component Analysis (ICA).

PCA is a widely used dimensionality reduction technique that can be applied to DNNs to reduce the input space dimensionality. PCA aims to find the orthogonal directions, called principal components, that capture the maximum variance in the data by projecting it onto a lower-dimensional subspace. To demonstrate its functioning and assumptions, a simple example is given in Figure 3.1 which shows a 2D data distribution in (x_1, x_2) space, with blue samples representing draws from the distribution. The distributions are clearly elongated in one particular direction, marked with a black line. The idea of reducing the dimensionality of this dataset is that the variance in the direction of the red line is not large and might be negligible. With this illustration, the core of the PCA can be stated

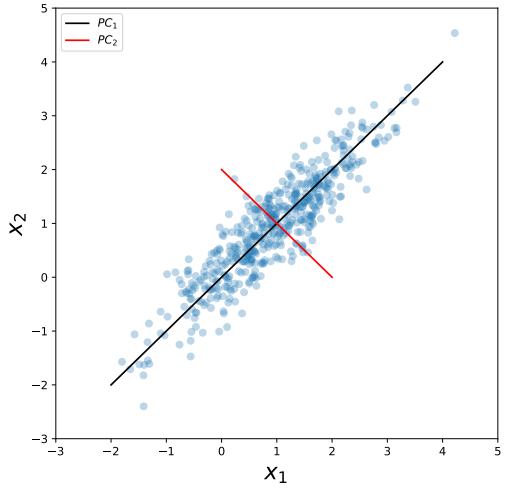


Figure 3.1: Example of the Principle Component Analysis in 2D. Blue points show a data distribution. Black and red lines show first and second principle component, respectively.

as the following: find N directions in the data that have the largest dispersion, and ignore others by projecting all points on these N dimensions. In the case of Figure 3.1, we want to lower the dimensionality of the data from 2D to 1D. In order to do that, we calculate with the PCA black and red directions (i.e. principal components, PCs). Finally, all the data on the first PC (the black line) is projected, ignoring dispersion in the second PC.

It's important to keep in mind that the main assumption of the algorithm is that the data is relatively Gaussian-distributed, and that principal components can be calculated with a simple linear transformation of the data. To calculate such compression of the data, firstly, the mean and covariance matrix is estimated:

$$\boldsymbol{\mu} = \frac{1}{M}\sum_i \boldsymbol{x}_i\,, \tag{3.2}$$

$$\Sigma_{mn} = \frac{1}{M-1}\sum_i (x_{im}-\mu_m)(x_{in}-\mu_n)\,. \tag{3.3}$$

Here, $\boldsymbol{x}_i$ labels i-th sample of the full vector of data, while x_{im} and x_{in} represent its m-th and n-th component. In the matrix form, if one writes *mean-removed* data as (D, M) matrix X, where D is the dimensionality of the data, and M number of samples, covariance matrix is then simply

$$\Sigma = \frac{1}{M-1} X\, X^T\,. \tag{3.4}$$

By writing the Singular Value Decomposition (SVD) of the matrix

$$X = U\, S\, V^T\,, \tag{3.5}$$

covariance matrix follows as:

$$\Sigma = U\, \frac{S^2}{M-1}\, U^T\,. \tag{3.6}$$

Here U contains all Eigenvectors, and can be viewed as a rotation matrix. $S^2/(M-1)$ contains variances in principal components, i.e. their Eigenvalues. By picking N Eigenvectors in the directions of the largest eigenvalues, the $\widetilde{U}$ is constructed and used for PCA dimensionality reduction. Compression is then simply obtained as $Y = \widetilde{U}^T X$, with Y being a compressed "projected" data.

ICA is another effective method for extracting features from DNN activation. PCA and ICA are both linear transformation techniques, but they are grounded in different

mathematical concepts and objectives. ICA seeks to find a linear representation of the data such that the transformed components are statistically independent from each other by maximising the non-Gaussianity of the data [47]. The primary objective of using ICA in conjunction with DNNs is to transform the high-dimensional activation vectors from the network into a set of statistically independent components, which are referred to as features. These features are intended to represent the most relevant and non-redundant information present in the original activations.

To compute the ICA, the data should have a zero mean, and then it can be transformed such that it has an identity covariance matrix:

$$\mathbf{x}_{\text{centered}} = \mathbf{x} - \mu\,, \tag{3.7}$$

$$C = E\Lambda E^T. \tag{3.8}$$

Here, C is the covariance matrix, E contains the Eigenvectors, and Λ is a diagonal matrix of Eigenvalues. The processed data $\mathbf{x}$ can be expressed as:

$$\mathbf{x} = \Lambda^{-\frac{1}{2}} E^T \mathbf{x}_{\text{centered}}. \tag{3.9}$$

The ICA decomposition goal is to find a matrix $\mathbf{W}$ such that $\mathbf{y} = \mathbf{W}\mathbf{x}$ yields components $\mathbf{y}$ that are statistically independent. Since the objective of ICA is to maximize the non-Gaussianity of the transformed components, measures like kurtosis or approximations such as negentropy are used to quantify non-Gaussianity. The optimization problem can be posed as:

$$\mathbf{W}^* = \arg\max_{\mathbf{W}} negentropy(\mathbf{y}) \tag{3.10}$$

After obtaining the independent components using ICA, the top k components (out of the total n components) can be chosen to reduce dimensionality. If S is the matrix of independent components, the reduced data S_{reduced} can be derived by selecting the top k components.

Example 7. *This example demonstrates how the PCA algorithm is utilised to extract latent features from DNN's layers. A set of input data is passed through a pre-trained DNN model, i.e., $\mathcal{N}_{sm}$ from Appendix A, to obtain the activations from the desired layers. These activations are then flattened into a 2D matrix and the data is standardised/scaled by making the mean of the data 0 and standard deviation as 1 using StandardScaler() function. PCA*

algorithm is implemented on the scikit-learn library, therefore, the decomposition.PCA() is used to decompose the standardised activation matrix into its principal components. Finally, the fit function calculates matrix $\widetilde{U}$ to obtain the transformed data that contains principal vectors.

Two extracted principal components from the $\mathcal{N}_{sm}$ layer `dense` *are shown below (the example shows the first 10 elements for each PC):*

```
1  [[-0.013956366, -0.00208691,-0.030311087, -0.004051855, -0.004236995,
      -0.0021598344, -0.00056796166, -0.0094258245, 0.009113272,
      0.01830883, ...]
2   [0.008219812, -0.000732703, 0.028722573, -0.0017270423, -0.0041757757,
      -0.0008947816, -0.0003945681, 0.0045744264, 0.029157056,
      0.02091162, ...]]
```

Here, each principal component represents a hidden feature, which is donated as $\mathbb{F}$ in the next step that performs the discretisation process.

3.3.2 Discretisation Techniques

Feature extraction techniques result in mappings that range over a continuous and potentially infinite domain. However, the BN-based abstraction technique relies on the construction of probability tables, where each entry associates a set of distinct hidden feature values with a probability. For this construction to be relevant, each hidden feature component is therefore discretise into a finite set of sub-spaces called intervals.

The BN approach employs predefined strategies for both; extracting hidden features using feature extraction algorithms to identify a number of mappings that correspond to relevant hidden features, and constructing a discretisation for each of them. One standard technique to perform the discretisation is the density-based discretisation that is empirically approximating a probability distribution over a given domain; thus, $\mathbb{F}$ component. The density estimates given in Figure 3.2 are obtained using Kernel Density Estimation (KED). The plots will be explained in the next example. Another useful approach is a partitioning-based discretisation that includes:

k-bins-uniform: The set of intervals partitions a given distribution into a specified positive number k of *bins*, all of the same width;

k-bins-quantile: The set of intervals is created as above, except that their respective width is calculated so that every interval holds a similar amount of individual projections from the $\mathbb{F}$ elements.

Example 8. *Continuing Example 7, Figure 3.2 illustrates what the discretisation process is produced. The two plots show the respective distribution values of the two hidden feature components* $\mathbb{F}_{\texttt{dense},0}$ *and* $\mathbb{F}_{\texttt{dense},1}$ *of the layer* `dense` *(the clouds of dots). Given such a component* $\mathbb{F}$*, the discussed discretisation strategies can be used to find a partition of its distribution. In the plots, the discretisation process partitions each horizontal axis into a set of distinct regions using the KED.*

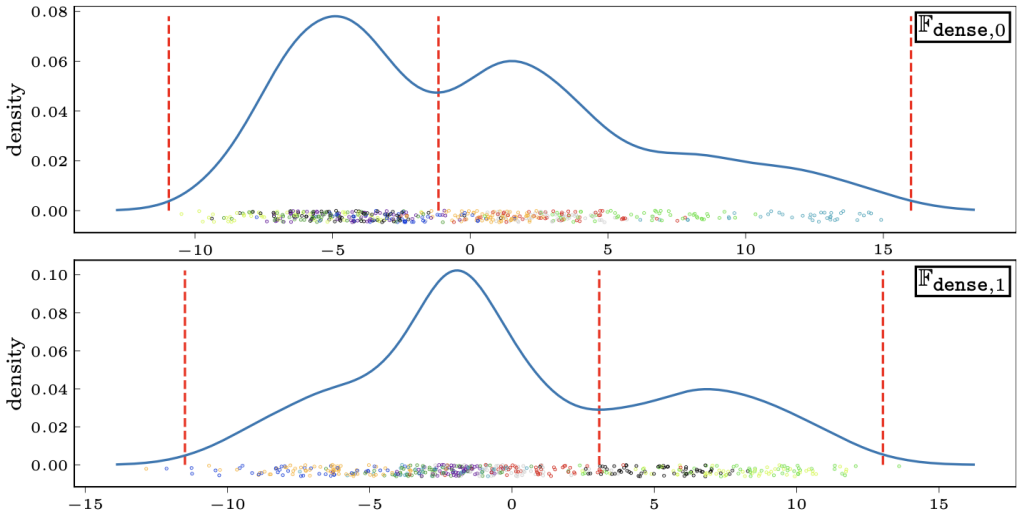
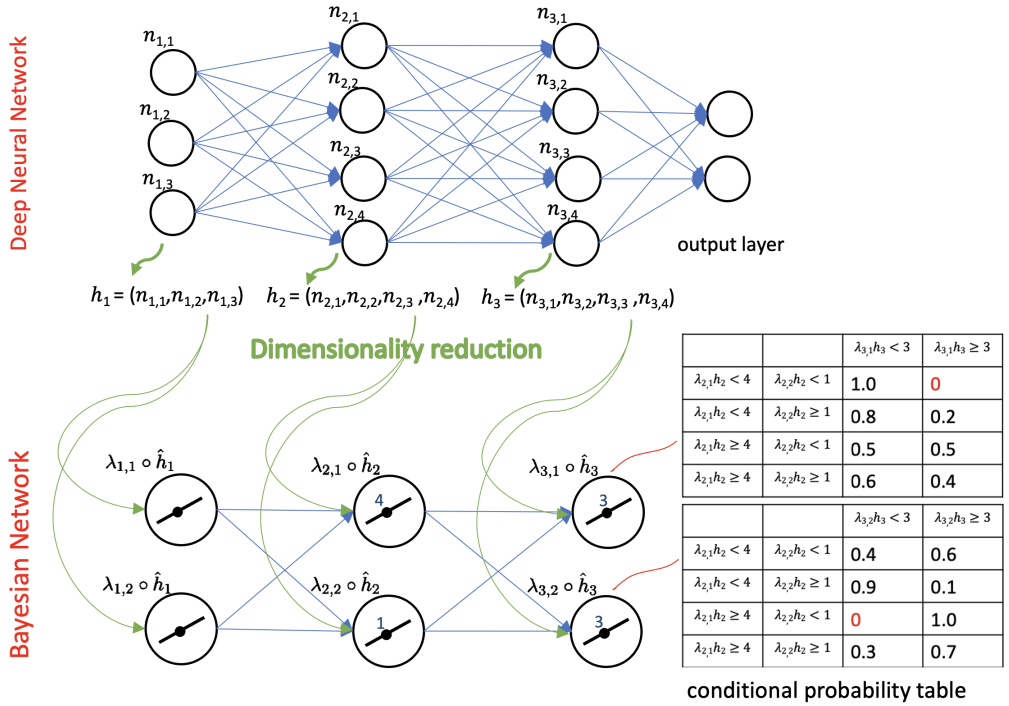


Figure 3.2: Projection onto two hidden feature components $\mathbb{F}_{\texttt{dense},0}$ and $\mathbb{F}_{\texttt{dense},1}$ of neuron values induced by a sample of training data X_{train}, associated density estimates (solid lines), and interval boundaries for discretisation (dashed vertical lines).

3.3.3 Bayesian Network Construction

Preliminaries. Let $\mathcal{N}$ be a DNN of a given architecture. For a learning model, the pair (X, Y) is used to denote the training data, where X is a set of inputs and Y is a corresponding set of outputs such that $|X| = |Y|$. Let $\mathbb{D}_X$ be the input domain and $\mathbb{D}_Y$ be the output domain, i.e. a set of labels. Hence, $X \subset \mathbb{D}_X$. The network $\mathcal{N}$ consists of a sequence of layers $\mathsf{Layers} = (l_1, \ldots, l_K)$, where every layer $l_i \in \mathsf{Layers}$ contains a set of $|l_i|$ *neurons* $h_i = \{n_{i,1}, \ldots, n_{i,|l_i|}\}$. Each neuron $n_{i,j}$ in h_i computes a value $\hat{n}_{i,j}$ in some domain that is usually assumed to be the set of Real numbers $\mathbb{R}$, and this value is typically computed as a function of the outputs of neurons in layer l_{i-1}. The valuation of all neuron values $\hat{n}_{i,1}, \ldots, \hat{n}_{i,|l_i|}$ builds up the *valuation space* $\mathbb{L}_i$ of all neurons for layer l_i. The feature



		$\lambda_{3,1}h_3 < 3$	$\lambda_{3,1}h_3 \geq 3$
$\lambda_{2,1}h_2 < 4$	$\lambda_{2,2}h_2 < 1$	1.0	0
$\lambda_{2,1}h_2 < 4$	$\lambda_{2,2}h_2 \geq 1$	0.8	0.2
$\lambda_{2,1}h_2 \geq 4$	$\lambda_{2,2}h_2 < 1$	0.5	0.5
$\lambda_{2,1}h_2 \geq 4$	$\lambda_{2,2}h_2 \geq 1$	0.6	0.4

		$\lambda_{3,2}h_3 < 3$	$\lambda_{3,2}h_3 \geq 3$
$\lambda_{2,1}h_2 < 4$	$\lambda_{2,2}h_2 < 1$	0.4	0.6
$\lambda_{2,1}h_2 < 4$	$\lambda_{2,2}h_2 \geq 1$	0.9	0.1
$\lambda_{2,1}h_2 \geq 4$	$\lambda_{2,2}h_2 < 1$	0	1.0
$\lambda_{2,1}h_2 \geq 4$	$\lambda_{2,2}h_2 \geq 1$	0.3	0.7

Figure 3.3: Structure of the Bayesian Network abstraction after reducing each h_1, h_2, h_3 into two features $\lambda_{i,1} \circ \hat{h}_i$ and $\lambda_{i,2} \circ \hat{h}_i$ with two intervals each. The conditional probability tables are shown for features $\lambda_{3,1}$ and $\lambda_{3,2}$.

extraction technique, namely PCA, is then used to analyse these neurons values and produce a set of feature mappings $\Lambda_i = \{\lambda_{i,j}\}_{j \in \{1,\ldots,|\Lambda_i|\}}$, where each $\lambda_{i,j} \colon \mathbb{L}_i \to \mathbb{F}_{i,j}$ maps the neuron valuation space $\mathbb{L}_i$ into the j-th component of the feature space $\mathbb{F}_i$ for layer l_i. Further, the value range of each feature, *i.e.*, $F^{\sharp}_{i,j} = \{f^{\sharp 1}_{i,j}, \ldots, f^{\sharp m}_{i,j}\}$, the j-th extracted feature from layer l_i, is partitioned into a finite set of m intervals. The $^{\sharp}$ exponents is denoted the discrete spaces.

As an abstract model of $\mathcal{N}$ and its training dataset X, a Bayesian Network (BN) is a directed acyclic graph $\mathcal{B}_{\mathcal{N},X} = (V, E, P)$, where V are nodes, E are edges that indicate dependencies between features in successive layers, and P maps each node in V to a probability table representing the conditional probability of the current feature over its parent features *w.r.t.* X.

Example 9. *Figure 3.3 gives a simple neural network of 2 hidden layers and its Bayesian Network abstraction.* $\hat{h}_i$ *is a function that gives the neuron activations at layer* l_i *from any*

given input sample, and $\lambda_{i,j}$ is a feature mapping from the set $\Lambda_i = \{\lambda_{i,j}\}_{j\in\{1,\dots,|\Lambda_i|\}}$. Each random variable $\lambda_{i,j} \circ \hat{h}_i$ in the BN represents the j-th component of the value obtained after mapping $\hat{h}_i$ into the latent feature space. Since each function $\lambda_{i,j} \circ \hat{h}_i$ ranges over a continuous space, the respective feature components—which are the codomains of the $\lambda_{i,j}$'s—are discretised into a finite set of feature intervals.

Each node in BN abstractions represents an extracted feature: $V_{\mathcal{N},X} = \left\{ \left(\!\left| f^{\sharp}_{i,j} \right|\!\right) \;\middle|\; \mathbb{F}^{\sharp}_{i,j} \in \mathbb{F}^{\sharp}_{\mathcal{N}} \right\}$. Each node is associated with either a marginal probability table for hidden features of layer l_1, or a conditional probability table for hidden or output layers. In Figure 3.3, the conditional probability table for the feature component $\lambda_{3,1}$ is defined for each feature interval $\{(-\infty, 3[,\ [3, +\infty)\}$ for layer l_3, *w.r.t.* each combination of the parent feature intervals from previous layer l_2.

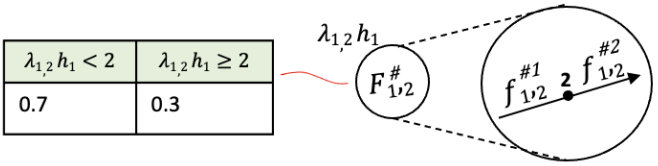


Figure 3.4: Illustration of probability tables and feature intervals with a Bayesian network node.

Example 10. *Figure 3.4 illustrates an example node in a BN, which corresponds to the second extracted feature from the first NN layer, i.e., $i = 1, j = 2$. The set $F^{\sharp}_{1,2}$ contains two intervals, $f^{\sharp 1}_{1,2}$ and $f^{\sharp 2}_{1,2}$, which partition the real line. The node is denoted as a random variable named $\lambda_{1,2} \circ \hat{h}_1$, which is associated with a probability table. The probability table is a marginal probability table because the features on the first layer do not have parent features. The table says that this feature has probability 0.7 to have a value smaller than 2 and probability 0.3 to have a value no less than 2.*

Notation. This part summaries the mathematical notations of constructing the BN in simple steps to make the developments in the next chapters clearer.

- At first, take a data input $\mathbf{x} \in \mathbb{D}_X$, where $\mathbb{D}_X$ is the input space.
- Pass the $\mathbf{x}$ into a neural network $\mathcal{N}$, the $\hat{h}_i(\mathbf{x})$ computes the neurons activation at layer l_i: $(\hat{n}_{i,1}, \dots, \hat{n}_{i,|l_i|}) \in \mathbb{L}_i$ where $\mathbb{L}_i$ is the valuation space of the i-th layer.

- Perform a feature mapping $\Lambda_i = \{\lambda_{i,j}\}_{j\in\{1,\ldots,|\Lambda_i|\}}$, where each $\lambda_{i,j}\colon \mathbb{L}_i \to \mathbb{F}_{i,j}$ maps the valuation space into j-th component of the feature space $\mathbb{F}_i \stackrel{\text{def}}{=} \prod_{j\in\{1,\ldots,|\Lambda_i|\}} \mathbb{F}_{i,j}$.

- Discretise each component of the feature space $\mathbb{F}_i$ into finite number of intervals to get $\mathbb{F}_i^\sharp = \prod_{j\in\{1,\ldots,|\Lambda_i|\}} \mathbb{F}_{i,j}^\sharp$. Here, $\mathbb{F}_{i,j}^\sharp$ is the discretised j-th component of the feature space for layer i. All in all, a full discretised feature space amounts to $\mathbb{F}_{\mathcal{N}}^\sharp = \left\{\mathbb{F}_{1,1}^\sharp, \mathbb{F}_{1,2}^\sharp, \ldots, \mathbb{F}_{K,|\Lambda_K|}^\sharp\right\}$. Note that, in the next chapters, the creation of the representation elements $\mathbf{r} = \text{Discr}^\sharp(\Lambda(\hat{h}(\mathbf{x}))) \in \mathbb{F}_{\mathcal{N}}^\sharp$ goes through all three previous processes, where $\hat{h}$ gives the activation values of all layers of the neural network, Λ is a feature space extraction and $\text{Discr}^\sharp$ is a discretisation process,

- Calculate the probability distribution of the BN

$$P_{\mathcal{N},X}\left(f_{i,j}^\sharp\right) \stackrel{\text{def}}{=} \begin{cases} \mathcal{P}_1\left(f_{1,j}^{\sharp k}\right) & \text{if } i = 1 \\ \mathcal{CP}_i\left(f_{i,j}^{\sharp k} \mid \mathbb{F}_{i-1}^\sharp\right) & \text{otherwise} \end{cases}$$

where the unconditional probability and conditional probability table are defined as:

$$\begin{aligned} \mathcal{P}_i\left(f_{i,j}^{\sharp k}\right) &\stackrel{\text{def}}{=} \Pr\left(\mathbf{x} \rightsquigarrow f_{i,j}^{\sharp k}\right), \\ \mathcal{CP}_i\left(f_{i,j}^{\sharp k} \mid F_{i-1}^\sharp\right) &\stackrel{\text{def}}{=} \Pr\left(\mathbf{x} \rightsquigarrow f_{i,j}^{\sharp k} \mid \mathbf{x} \rightsquigarrow \mathbb{F}_{i-1}^\sharp\right). \end{aligned}$$

Here $x \rightsquigarrow f_{i,j}^{\sharp k}$ denotes that the input x exercises the interval $f_{i,j}^{\sharp k}$. In practice, this amounts to iterating over the training set $\mathcal{D}_{\text{train}}$:

$$\begin{aligned} \mathcal{P}_i\left(f_{i,j}^{\sharp k}\right) &= \frac{\sum_{\mathbf{x}\in\mathcal{D}_{\text{train}}} \mathbb{1}\left(\mathbf{x} \rightsquigarrow f_{i,j}^{\sharp k}\right)}{|\mathcal{D}_{\text{train}}|}, \\ \mathcal{CP}_i\left(f_{i,j}^{\sharp k} \mid F_{i-1}^\sharp\right) &= \frac{\sum_{\mathbf{x}\in\mathcal{D}_{\text{train}}} \mathbb{1}\left(\mathbf{x} \rightsquigarrow f_{i,j}^{\sharp k} \wedge \mathbf{x} \rightsquigarrow \mathbb{F}_{i-1}^\sharp\right)}{\sum_{\mathbf{x}\in\mathcal{D}_{\text{train}}} \mathbb{1}\left(\mathbf{x} \rightsquigarrow \mathbb{F}_{i-1}^\sharp\right)}. \end{aligned}$$

Here, $\mathbb{1}$ returns 1 if the condition holds, or 0 otherwise.

3.3.4 Bayesian Network-based Coverage Metrics

Two testing coverage metrics are defined based on the above BN abstraction: the BN-based feature coverage (BFCov) and the BN-based feature-dependence coverage (BFdCov). These

metrics give the proportion of features or causal relationships between features that are adequately exercised by a set of inputs.

Definition 3.3.1 (BN-based Feature Coverage)**.** *Given a trained DNN $\mathcal{N}$, the* BN-based feature coverage *of a* non-empty *set of inputs $X \subset \mathbb{D}_X$ is obtained via the BN abstraction $\mathcal{B}_{\mathcal{N},X}$ as*

$$\mathrm{BFCov}\left(\mathcal{B}_{\mathcal{N},X}\right) \stackrel{\text{def}}{=} \frac{1}{\left|V_{\mathcal{N},X}\right|} \sum_{(\!|f_{i,j}^{\sharp}|\!) \in V_{\mathcal{N},X}} \frac{\left|\left\{ f_{i,j}^{\sharp k} \in \mathbb{F}_{i,j}^{\sharp} \,\middle|\, \mathcal{P}_i\left(f_{i,j}^{\sharp k}\right) \geq \varepsilon \right\}\right|}{\left|\mathbb{F}_{i,j}^{\sharp}\right|}. \tag{3.11}$$

Informally, $\mathrm{BFCov}\left(\mathcal{B}_{\mathcal{N},X}\right)$ ranges over $[0,1]$, and gives the percentage of features that are adequately exercised by X. The interpretation is that the coverage metric accounts for the number of marginal probabilities for every interval $f_{i,j}^{\sharp k} \in \mathbb{F}_{i,j}^{\sharp}$ in the BN's node appear with a probability bigger than ε.

Definition 3.3.2 (BN-based Feature-dependence Coverage)**.** *Given a trained DNN $\mathcal{N}$, the* BN-based feature-dependence coverage *of a* non-empty *set of inputs $X \subset \mathbb{D}_X$ is obtained via the BN abstraction $\mathcal{B}_{\mathcal{N},X}$ as*

$$\mathrm{BFdCov}\left(\mathcal{B}_{\mathcal{N},X}\right) \stackrel{\text{def}}{=} \frac{1}{\left|V_{\mathcal{N},X}^{+}\right|} \sum_{(\!|f_{i,j}^{\sharp}|\!) \in V_{\mathcal{N},X}^{+}} \frac{\left|\left\{ (f_{i,j}^{\sharp k}, F_{i-1}^{\sharp}) \in \left| \begin{array}{c} \mathcal{CP}_i\left(f_{i,j}^{\sharp k} \,\middle|\, F_{i-1}^{\sharp}\right) \geq \varepsilon \\ \mathbb{F}_{i,j}^{\sharp} \times \mathbb{F}_{i-1}^{\sharp} \end{array} \right| \begin{array}{c} \\ \vee \quad \mathcal{P}_i\left(f_{i,j}^{\sharp k}\right) < \varepsilon \end{array} \right\}\right|}{\left|\mathbb{F}_{i,j}^{\sharp} \times \mathbb{F}_{i-1}^{\sharp}\right|} \tag{3.12}$$

where $V_{\mathcal{N},X}^{+} \stackrel{\text{def}}{=} \left\{ (\!|f_{i,j}^{\sharp}|\!) \in V_{\mathcal{N},X} \,\middle|\, i > 1 \right\}$ are all nodes in $\mathcal{B}_{\mathcal{N},X}$ with conditional probabilities tables. The $\mathrm{BFdCov}\left(\mathcal{B}_{\mathcal{N},X}\right)$ *gives the percentage of assumed causal relationships between features of successive layers that are adequately exercised by X.*

Intuitively, a test dataset X satisfies the high-level BN-based criterion if all node values in the BN appear with a high probability. In other words, if there is no hidden feature interval is rare w.r.t. the BN, and each combination of intervals is exhibited by X. This indicates that the DNN's behaviours represented by the BN are well tested by X. Moreover, the two above metrics can be combined to deliver a coverage that is consistent and based on all probability entries of the BN (marginal and conditional distributions) as following:

$$\mathrm{BFxCov}\left(\mathcal{B}_{\mathcal{N},X}\right) \stackrel{\text{def}}{=} \mathrm{BFCov}\left(\mathcal{B}_{\mathcal{N},X}\right) \times \mathrm{BFdCov}\left(\mathcal{B}_{\mathcal{N},X}\right). \tag{3.13}$$

In practise, however, these coverage metrics do not achieve 100% coverage, especially in complex DNN models. This reduces the reliability of the testing, as a fatal error may hide in a trivial untested regions. Yet, these metrics do not consider the contribution role of each feature and treat them with equal importance.

3.4 Summary

The fundamental knowledge required for understanding the Bayesian network abstraction model has been presented in this chapter. The chapter commenced by defining the Bayesian Network's main notions, which integrate graph and probability theory to provide a probabilistic approach to making an inference. Numerous investigations conducted over the past few years have demonstrated the effectiveness of Bayesian networks in resolving existing issues in deep neural networks. These studies have concentrated on different DNNs' concerns; however, testing issues are not considered in the context of the Bayesian perspectives, except in the detailed feature coverage discussed in the last section. These BN-based feature metrics are defined based on the abstraction of neural network behaviours subject to a given dataset. Further analysis is needed to explore the contribution of these identified features to the DNN's decision and enhance the testing metrics using the outcome information.

Chapter 4

BN-based Features Sensitivity Analysis

4.1 Introduction

As discussed in the previous chapters, the level of understanding of a DNN's underlying decision processes often lies far below what can reasonably be accepted for standard safety assurance. Thus, the primary objective of this chapter is to acquire insight into the high-level representations learnt by neural networks and their semantic mechanisms of decision making. The study presented in this chapter is primarily concerned with answering the subsidiary research questions SRQ1 and SRQ2, which were posed in Chapter 1:

1. **SRQ1:** Is a Bayesian abstraction of a neural network able to systematically analyse a DNN's interior decisions?

2. **SRQ2:** How can the importance of the latent features of a deep neural network be quantified using an abstracted Bayesian Network?

The current trend of investigating the internal logic of a DNN model in the context of (for example), eXplainable Artificial Intelligence (XAI) [67], semantic-level robustness [39, 97], and exhibiting their internal working mechanism through test cases [43], is concentrating on the learnt features as a start point. These learnt features are representations of the data that the DNN has discovered through its training process, and they are used to make predictions or classifications. Since the BN model presented in Chapter 3 abstracts

the latent representations of a DNN as a Bayesian network, it can be utilised to study the behaviour of the extracted features. The general aim ofthe work presented in this chapter is to evaluate the performance of DNNs by developing weight-based test metrics that are similar to those described in Subsection 3.3.4 of Chapter 3. However, instead of directly using the extracted hidden features to measure test coverage, the idea is to first compute a weight value, w_i, for the i-th latent feature, by analysing the BN's sensitivity to controlled noise applied to this feature. The test coverage will then be reported based on the weight of the hidden features.

The weighted scoring mechanism is commonly applied in statistics when certain selected criteria are assigned more importance than others. In the machine learning context, **Feature Importance (FI)** is a measure that describes how much each feature influences the classifier's decision, which in turn indicates the importance of the feature for the classification. Contrary to the existing notion of feature importance in explanation models [10], where importance values are assigned to the features that belong to the input space, *e.g.,* age, sex, education. Instead, the proposed approach targets the learning models' latent feature space and examines how much their deep representation relies on a specific hidden features to affect their prediction. Applying changes to a vector in the latent space are semantic (high-level) in terms of the original representation [13]. Therefore, the introduced **FI** measure would provide rich characterizations of model-internal representations and their roles in input/output behavior.

Contribution. The key contribution of this chapter is proposing a method that estimates the importance of a neural network's latent features by analysing an associated Bayesian network's sensitivity to distributional shifts. This then allows for the definition of semantic testing metrics and the identification of distributional shifts in the feature space through the effect a latent feature has on the random variables in BNs.

The chapter also reports on a number of analyses conducted using BN abstraction to estimate the relative impacts and sensitivity of the latent features. The first analysis, presented in Section 4.4, was directed at measuring the relative impact one feature has on another for all feature pairs by estimating how a controlled noise impacts the BN's probability distributions. As an advanced approach, the next analysis estimated the sensitivity of a given latent feature by comparing the probability distributions of training samples before and after the feature has been perturbed. Figure 4.1 outlines the proposed feature sensitivity analysis approach.

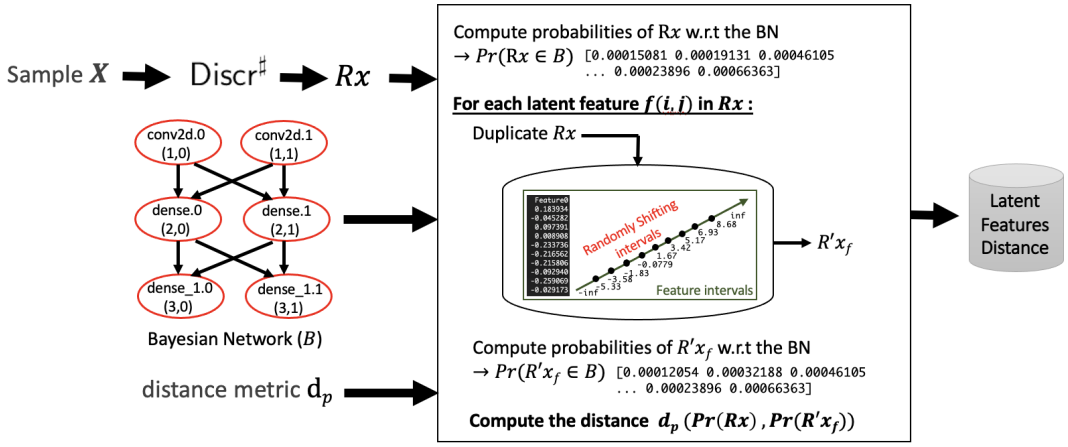


Figure 4.1: Illustration of the proposed BN analysis technique to compute the sensitivity of extracted latent features.

The structure of the BN in Figure 4.1, is built based on a *parameterisable abstraction scheme* that defines a series of DNN layers to consider (conv2d, dense and dense_1 in the Figure), a feature extraction technique to identify a given number of latent features for each one of these layers (2 in the example), and a discretisation strategy that determines the granularity at which values of latent features are aggregated into distinct intervals of indistinguishable values. Combined with the considered feed-forward nature of the DNNs, this scheme allows derivation of the structure of a BN, as shown in Figure 4.1. The BN in Figure 4.1 is constructed from the MNIST CNN model $\mathcal{N}_{sm}$ whose structure is demonstrated in Appendix A.

In addition, this scheme provides a *discretisation function* $\mathsf{Discr}^\sharp$, that transforms a set of inputs X into a low-dimensional, discretised representation R_X. In the Figure, the vector of inputs X is transformed into R_X, which associates each input $x \in X$ with six feature intervals, one for each latent feature represented by the BN. e.g., $\mathsf{Discr}^\sharp(x)$ may produce $R_X = [1, 2, 1, 1, 0, 2]$. The six values in the vector are the computed feature intervals *w.r.t.* the input x resulting from the feature mapping and discretisation step. As the BN assigns a probability to an input sample that belongs to the distribution it represents, the probabilities of a sample under a given BN before and after a perturbation can be observed. To do so, an interior analysis on R_X is conducted by calculating the probability of each sample under the BN B. After that, the process iterates over all considered latent feature f, and *shift* the associated intervals in R_X to produce a modified R'_{X_f} *w.r.t.* the feature

f, and calculates its probability belonging to the BN B distribution. The term **intervals shifting** refers to a technique used to artificially simulate a controlled distribution shift by randomly shifting intervals in the selected feature space. To identify the impact of a perturbation, a distance between the original probability vector and the probability vector obtained from the perturbed features is computed.

The rest of this chapter is organised as follows. Section 4.2 gives a brief background on the core concepts needed to understand the work presented in this chapter. Next, Section 4.3 introduces the preliminaries operations required to proceed with the next sections. This is followed in Section 4.4 by a detailed description of the proposed Bayesian network features sensitivity analysis algorithm, which is based on the BN abstraction scheme and is designed to analyse the internal representation of a DNN. A feature importance measure is introduced at the end of the section. In Section 4.5 the results of an extensive evaluation concerning latent features sensitivity to perturbation and adversarial distribution shift are presented. The final two sections of this chapter raise some discussion about the used methodology and conclude the work presented.

4.2 Background and Related Work

This section provides fundamental background details concerning DNN latent features intended for analysis in this chapter. First part explains the concept of a DNN latent feature space and provides a visualisation of what latent features look like within that space. The second part discusses the robustness of DNN internal representations, referred to as latent features. Finally, the third part reviews related work on performing BN sensitivity analysis, highlighting the benefits of sensitivity analysis and the different approaches employed to execute it.

Latent Feature Space. The neural networks' latent feature space refers to the internal, hidden representations learned by the network. These representations are not directly observable, but rather are formed as a result of the network processing training data and capture their important patterns [8]. It is considered compressed data, in which similar data points are closer together in the representation space. This fundamental aspect of DNNs, where they capture complex relationships and higher-level abstractions in the data, are difficult or impossible to model with traditional machine learning algorithms. The term "latent" implies the fact that these features are not explicitly defined or labeled in the input

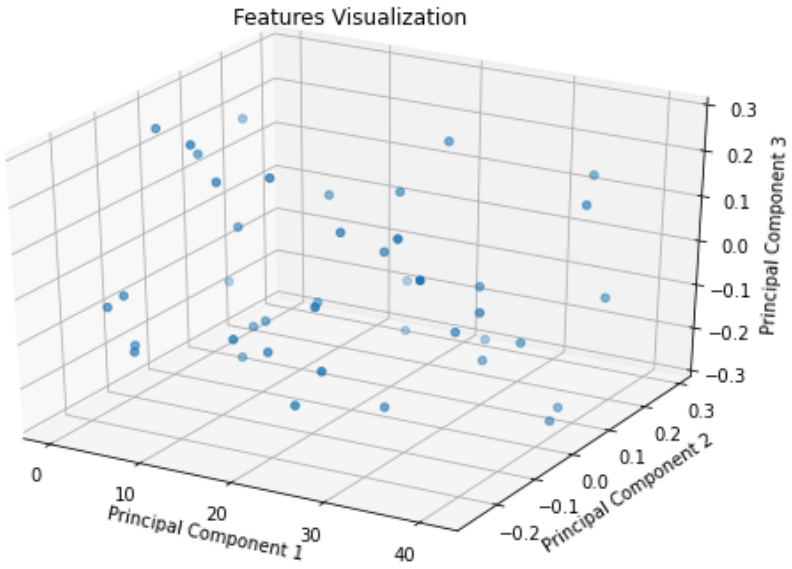


Figure 4.2: A visualisation of three extracted features using PCA from one CNN layer.

data, but rather emerge through the learning process of the network. For example, in a DNN for image classification, the latent features in the hidden layers might represent abstract concepts such as edges, corners, or textures that are relevant to distinguishing between different classes of images. Figure 4.2 illustrates three features components extracted from a convolutional neural network's dense layer using a Principal Component Analysis algorithm.

The latent feature space is a critical component to be investigated when analysing machine learning decision systems because it encodes the meaningful internal representation of the observed data. Since it is a lower-dimensional space that captures the essential features of the input data, it is often used as a starting point for further analysis or processing.

Robustness of Neural Network Latent Features. Features play an essential role in the field of image processing and classification. They are considered to be the basic conceptual components of the semantics of an image. Similar to this feature behaviour study, there are recent researches concentrated on the feature space to study the hidden semantic representation of intelligent models. Ilyas et al. [48] categorised useful features in the input-space into robust and non-robust features. They demonstrate that adversarial perturbation can arise from flipping non-robust features in the data that are useful for

the classification of regular inputs in the standard setting. They further argue that ML models are highly vulnerable to adversarial examples due to the presence of these useful non-robust features. This was further emphasised by Madaan et al. [63], who showed that the factor causing the adversarial vulnerability is the distortion in the latent feature space. They therefore concluded that properly learning the compact low-dimensional space, that captures the underlying structure and patterns of the input data makes DNNs achieve high levels of accuracy and generalisation of performance on a wide range of tasks. Going beyond this existing work, which justify the need for considering features (instead of pixels or neurons), the next section study the causality relation between features by constructing a formal model, namely a Bayesian network.

Bayesian Networks Sensitivity Analysis. Sensitivity analysis in Bayesian networks is concerned with understanding how a small change in the network parameter values may affect the final results drawn based on the network. Through sensitivity analysis, one can obtain essential insights into the model's behaviour and its response to varying inputs [11]. The key aspect of performing a sensitivity analysis on a Bayesian network can be listed as follows:

- Quantify the impact of different nodes on a target node;
- Discover important features that have significant influence on the classifier decision;
- Determine sensitive parts of the network that might cause network vulnerability.

Several Bayesian network sensitivity analysis techniques have been developed to assess the impact of changes in input variables on the model's outputs or predictions. Castillo et al. [16] studied the sensitivity of the conditional probabilities of a Bayesian network to small changes in parameter values using symbolic probabilistic inference. Their aim was to examin the algebraic and dependency structures of the BN probabilities of a target node given an evidence. Castillo and Kjærulff [15] analysed sensitivity in Gaussian Bayesian networks using symbolic propagation. They demonstrated how changes in parameter and evidence values affected marginal and conditional probabilities. The authors in [34] also addressed the sensitivity in Gaussian Bayesian networks, however, their method was based on the Kullback–Leibler divergence. While these Bayesian network sensitivity methods are focused on varying one of the network's parameter probabilities and keeping all other parameters fixed, [17] considered multiple parameter changes.

Nevertheless, sensitivity analysis cannot be directly applied in the traditional sense using abstracted BNs, where the sensitivity is performed on a BN with parametric probability distributions by changing the BN parameter from the input space and observing how that influences the final decision. Instead, since the proposed BN analysis targets the latent features in the low-dimensional space with discrete BN distributions, the method measures how sensitive are the BN probability distributions to changes in the values of hidden features. This process gives insight into how such perturbations impact the inner Bayesian network distribution and hence reflect the ground truth of the neural networks' behaviour in the presence of adversarial inputs.

Throughout this chapter, the Bayesian Network probability distribution is used to study neural networks latent features and analyse their deep representations.

4.3 Preliminaries

At the core of the approach lies the proposed BN-based latent feature analysis algorithm, the preliminaries presented in this section introduce how the BNs that have been discussed in Chapter 3 are utilised as an explainable abstraction of DNNs' latent features.

4.3.1 Data Abstraction Through a Bayesian Network

A set of probability tables can be fitted in a BN abstraction $\mathcal{B}$ by using a training sample X. This process first transforms X by means of the discretisation function to obtain a vector of elements from the discretised latent feature space $R_X = \mathsf{Discr}^\sharp(X)$. It then updates the probability tables in $\mathcal{B}$ in such a way that the joint probability distribution it represents fits the distribution of R_X. The fitted BN $\mathcal{B}$ can then be queried for the probabilities of the discretised input sample $R_{X'}$. This query operation is denoted as $\Pr(R_{X'} \in \mathcal{B})$, and may abuse this notation by defining $\Pr(X' \in \mathcal{B}) = \Pr\big(\mathsf{Discr}^\sharp(X') \in \mathcal{B}\big)$.

4.3.2 Perturbation of Latent Features

A **feature** is formally defined as a pair (i, j), where i indexes a layer $l_i \in \mathcal{L}$, and j identifies a component of the extracted feature space for layer l_i, *i.e.,* $j \in \{1, \ldots, |\Lambda_i|\}$. Note that, the layer index i can be replaced with the explicit layer name for some experiment clarification. The developments in the next Section rely on the application of a controlled change of a feature (i, j) in an element R_x of the latent feature space. This operation simulates

a distortion in single targeted component of the latent feature space by substituting its associated interval with an adjacent one, it is assumed that each latent feature component is partitioned into at least two intervals. When an interval has two neighbours, the function chooses uniformly at random between them. This operation is denoted with the function random_shift(R_x, i, j), which replaces the feature interval $f_{i,j}^{\sharp k}$ of R_x with either $f_{i,j}^{\sharp k-1}$ or $f_{i,j}^{\sharp k+1}$. For instance, assuming two hidden feature components extracted from activations at two layers of a DNN, each component being discretised into small-enough intervals, *i.e.,* 10 intervals, random_shift$((f_{0,0}^{\sharp 4}, f_{0,1}^{\sharp 7}, f_{1,0}^{\sharp 1}, f_{1,1}^{\sharp 9}), 1, 0)$ returns either $(f_{0,0}^{\sharp 4}, f_{0,1}^{\sharp 7}, f_{1,0}^{\sharp 0}, f_{1,1}^{\sharp 9})$ or $(f_{0,0}^{\sharp 4}, f_{0,1}^{\sharp 7}, f_{1,0}^{\sharp 2}, f_{1,1}^{\sharp 9})$.

4.4 BN-based Latent Feature Analysis

This section develops several BN-based analysis approaches employed to gain insights on latent features. The first approach produces a pairwise comparison matrix that exhibits the relative impact the latent features have on each other. Next, the BN is leveraged to estimate the sensitivity of an individual feature to a controlled distribution shift. It then describes how the sensitivity analysis technique can be applied to define feature importance based on a generic definition of weights on features. Finally, the last part formalises a concrete definition of weights based on the BN-based feature sensitivity. Each is discussed in further detail in the following three Sub-sections.

4.4.1 Pairwise Comparison

The pairwise comparison was designed to assess the degree to which the extracted features can affect each other by comparing the parallelised Conditional Probability Tables (CPTs) of a sample, under a BN, with the CPTs of the same sample after perturbing the features intervals of the BN. The pairwise comparison method is created to make a recursive comparison. It begins by extracting a set of inputs X from training data, and computing its feature intervals with $\mathsf{Discr}^{\sharp}$. This produces a sample R_X of intervals w.r.t. X. To generate the probability tables, the Bayesian network is fitted with R_X, which gives the clean reference probability tables $CPTs(R_X)$. Figure 4.3-(a) shows the CPT for feature $(3, 0)$, which is the first extracted feature from the third NN layer, named dense_1 in the BN drawn in Figure 4.1.

To extract knowledge about a given feature's independence and robustness, a controlled

a) CPT for the extracted feature (3,0).

```
'dense_1.0' :
['0', '0', '0', '0.056548412910109355']
['0', '0', '1', '0.9434515870898906']
['0', '0', '2', '0.0']
['0', '1', '0', '0.6184374222443394']
['0', '1', '1', '0.3815625777556606']
['0', '1', '2', '0.0']
['0', '2', '0', '0.9809251311397235']
['0', '2', '1', '0.01907486886027658']
['0', '2', '2', '0.0']
['1', '0', '0', '0.00013865779256794238']
['1', '0', '1', '0.5339711591791458']
['1', '0', '2', '0.46589018302828616']
['1', '1', '0', '0.06998935211729052']
['1', '1', '1', '0.9102301580801049']
['1', '1', '2', '0.019780489802604637']
['1', '2', '0', '0.42971393417410025']
['1', '2', '1', '0.5702860658258997']
...

['2', '2', '2', '0.07201309328968904']
```

b) CPT for the feature (3,0) after perturb feature (2,0)

```
'dense_1.0' :
['0', '0', '0', '0.02878870179250407']
['0', '0', '1', '0.7421238457360131']
['0', '0', '2', '0.22908745247148285']
['0', '1', '0', '0.38215232373210817']
['0', '1', '1', '0.609613005831419']
['0', '1', '2', '0.008234670436472876']
['0', '2', '0', '0.740080971659919']
['0', '2', '1', '0.25991902834008096']
['0', '2', '2', '0.0']
['1', '0', '0', '0.0002478929102627663']
['1', '0', '1', '0.4779375309866138']
['1', '0', '2', '0.5218145761031234']
['1', '1', '0', '0.06409668043445005']
['1', '1', '1', '0.8900107082759676']
['1', '1', '2', '0.045892611289582386']
['1', '2', '0', '0.3662551440329218']
['1', '2', '1', '0.6219135802469136']
...

['2', '2', '2', '0.06774193548387099']
```

Figure 4.3: A toy example, with only three intervals for each feature, illustrates the conditional probability table for the first extracted feature from layer dense_1, named $(3, 0)$, before and after shifting intervals of feature $(2, 0)$ in the dataset used to fit the BN.

change is applied to the targeted feature f, by using the random_shift operation to shift f's intervals in R_X to obtain R'_X. Then the BN's probabilities are re-fitted with R'_X, which gives the modified probability tables $CPTs(R'_X)$ *w.r.t.* the perturbed feature f, examplified in Figure 4.3-(b). To identify the impact, we use the Mean Squared Error (MSE) between each corresponding table in the reference $CPTs(R_X)$ and generated $CPTs(R'_X)$.

Example 11. *The results of pairwise comparison of latent features perturbation are illustrated in the following example. The investigation concentrates on the BN given in the Figure 4.1.*

Table 4.1 reports a pairwise comparison matrix outcome, where the perturbed features are arranged in the first column and compute their impact on each feature (i, j)*'s probability tables. The numbers reported in this matrix represent the change in the probability values. For instance, the controlled perturbation of feature* $(2, 0)$ *intervals has an impact on features* $(3, 0)$ *and* $(3, 1)$ *values. More specifically, the MSE between the* $(3, 0)$ *probability tables for feature* $(3, 0)$*, given in Figure 4.3 (a) before and (b) after perturbing feature* $(2, 0)$*, is* $0.011\,291$.

perturbed feature	$(1,0)$	$(1,1)$	$(2,0)$	$(2,1)$	$(3,0)$	$(3,1)$
$(1,0)$	0.003012	0.000000	0.008584	0.008075	0.000000	0.000000
$(1,1)$	0.000000	0.002569	0.007831	0.008654	0.000000	0.000000
$(2,0)$	0.000000	0.000000	0.014261	0.000000	0.011291	0.008066
$(2,1)$	0.000000	0.000000	0.000000	0.010236	0.008892	0.011429
$(3,0)$	0.000000	0.000000	0.000000	0.000000	0.022857	0.000000
$(3,1)$	0.000000	0.000000	0.000000	0.000000	0.000000	0.016100

Table 4.1: Pairwise comparison matrix for six extracted features. Each cell describes the extent to which a feature (rows) affects the others (columns).

Discussion. Suppose the diagonal line of the pairwise matrix is set to zeros since the change is made from the feature itself. In that case, one can observe that the perturbations are not affecting the probability of features from the previous layer (parent features) or the same layer as expected. On the other hand, random shifting only influenced the immediate features in the next layer. The largest difference occurred on feature $(3,1)$ when perturbing feature $(2,1)$. Although this impact is relatively small, can be observed (as expected) the dependencies between latent feature values of the BN model. Note that, the perturbations do not change the features' probability for deeper layers, *e.g.,* features of Layer 3 are not affected by the perturbation made on features of Layer 1. This is because of the inherent structure of the original DNN, from which the Bayesian Network is built, where features from the DNN layer l_i are influenced only by features in layer l_{i-1}.

4.4.2 Feature Sensitivity Analysis

The core benefit of relying on a Bayesian network is to have a model that exhibits the relevant theoretical aspects of Bayesian analysis. And, as stated, Bayesian Sensitivity analysis helps identify influential variables and quantify their effects. To estimate the sensitivity of the abstraction scheme on a given latent feature, we measure the impact of artificially perturbing the intervals representing the selected feature on the probability distribution represented by the BN. In this algorithm, the BN is already fitted using a training dataset, and the distribution it represents does not change.

The feature sensitivity analysis is given in Algorithm 1. This procedure receives an input sample X, taken from the training dataset, and first performs the feature projection and discretisation step with $\mathsf{Discr}^\sharp$ to obtain the associated feature intervals R_X. Not that; the number of features and intervals were specified in the BN construction phase. It then

Algorithm 1 BN-based Feature Sensitivity Analysis

Input: Bayesian network $\mathcal{B}$ and associated feature mapping & discretisation function $\mathsf{Discr}^\sharp$, training dataset X, distance metric d_p.

Output: Mapping associating a distance measure with each considered latent feature

1: Compute the feature intervals *w.r.t.* X:
   $R_X = \mathsf{Discr}^\sharp(X)$
2: Compute the reference probabilities of R_X *w.r.t.* $\mathcal{B}$:
   $P_{ref} = \Pr(R_X \in \mathcal{B})$
3: **for** each considered feature $f = (i, j)$ **do**
4: $\quad P'_f = \langle \Pr(\text{random\_shift}(R_x, i, j) \in \mathcal{B}) \rangle_{R_x \in R_X}$
5: $\quad d_f = d_p\Big(P_{ref}, P'_f\Big)$
6: **end for**
7: **return** distances d_f, for all f

calculates the probability of each element of R_X *w.r.t.* the BN $\mathcal{B}$; this gives the vector of reference probabilities P_{ref}, that associates a probability with each set of abstracted latent features that are elicited by each x in X. Then, for each extracted feature f, a random perturbation is performed in R_X via the random_shift function introduced in the preceding Sub-Section 4.3.2. This leads to a second vector, that holds the probabilities of the resulting R'_{X_f} *w.r.t.* the BN $\mathcal{B}$. The given distance d_p between these two probability vectors for the perturbed feature f is eventually computed.

The author chooses to make the feature sensitivity analysis algorithm parametric in the distance metric p for the purposes of easing further experimental use of the FI measure. The considered distances are:

- L_p's with different norms, typically 0, 1, 2, or ∞;
- *corr* is the correlation distance;
- *cos* is the cosine distance;
- *KL* is the Kullback-Leibler divergence;
- *JS* is the Jensen-Shannon distance, that is a metric that measures the similarity between two probability distributions based on entropy computations;
- *W1* is the first Wasserstein distance between two probability distributions;

- *MSE* is the *mean squared error*;
- *RMSE* is the *root mean squared error*;
- *MAE* is the *mean absolute error*;
- *AF* is a special purpose *anti-fit* divergence, which we define based on the *coefficient of determination* R^2. R^2 is a score that is typically used as a "goodness-of-fit" measure for regression models, and we refer to it as score_{R^2}. While the maximal score is 1 (indicating a perfect fit), the score decreases with the amount of variance in P that is not in Q and can take negative values. With this we define $d_{AF}(P,Q) = 1 - \mathsf{score}_{R^2}(P,Q)$.

  The rationale of using score_{R^2} as a basis for measuring the divergence is that we can view the probability vectors for perturbed features as output by a model. Divergence will be large when the effect of the perturbation is significant, and small when the model is not (very) sensitive to the perturbation.

Example 12. *The plot in Figure 4.4 gives an example distributions of probabilities in vectors obtained from the BN abstraction of the MNIST model with two extracted features per layer and five discretised intervals (the BN in Figure 4.1). Each one of these plots is annotated with various measures of distances between the reference probabilities P_{ref} that is generated using a sample from the training data set, and the respective six perturbed features probabilities P'_f. The shown difference between these two probability distributions illustrates the internal change in the distribution represented by the BN. For instance, when applying the* random_shift *on the first feature that is extracted from the first selected layer i.e., perturbed feature (1,0), the calculated probability distribution $P'(1,0)$, coloured with blue, shows a change on probabilistic causal relation that implies the change on the probability represented by the BN.*

4.4.3 Feature Importance

Each extracted feature $f_{i,j} \in F^\sharp$ is associated with a weight $w_{f_{i,j}}$ based on the set of measured sensitivity distances as follows:

$$w_{f_{i,j}} = \frac{d_{f_{i,j}}}{\sum_{f_{i,j} \in F^\sharp} d_{f_{i,j}}} \tag{4.1}$$

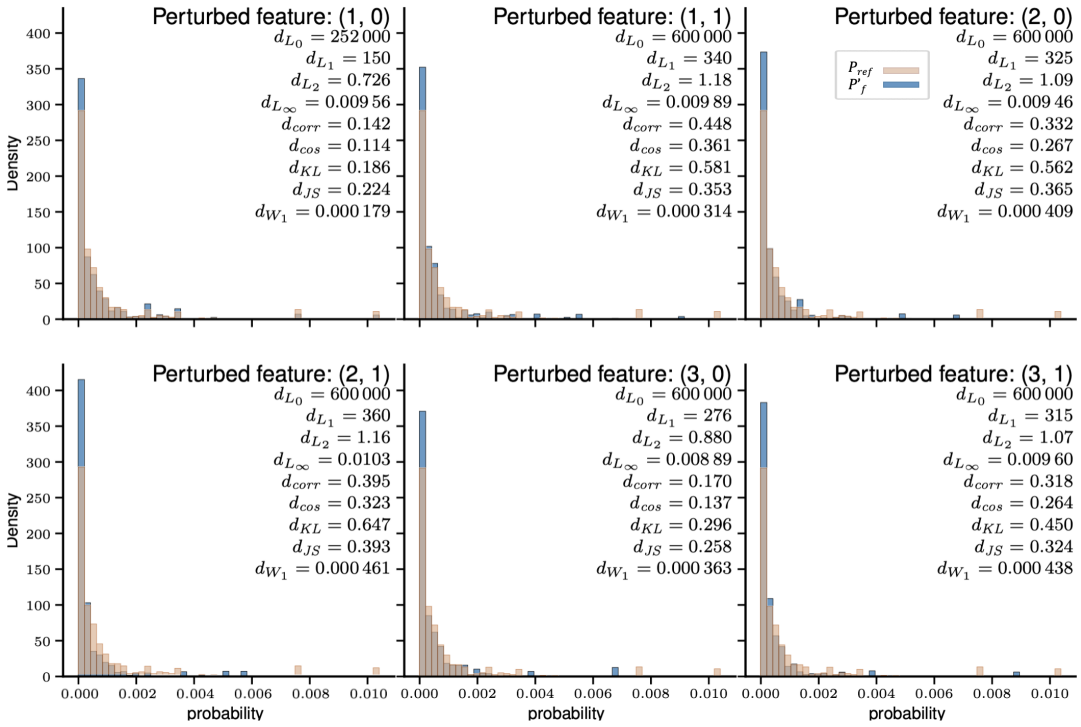


Figure 4.4: Density of probability distributions for each perturbed feature P'_f (coloured blue) overlapped with the BN reference probabilities P_{ref}. Each plot shows various distance measures between the two distributions.

where $F^\sharp$ is the set of considered latent features. The weighting Equation (4.1) acts as a normalisation function, *i.e.,* it ensures the sum of the feature components' weights equals one. The normalised importance weight for each feature is usually positively correlated with the respective probabilities distances.

Example 13. *Continuing Example 12, the computed sensitivity distances for each perturbed feature are summarised in the Table 4.2. Assuming the d_{corr} distance is chosen to determine feature importance, the Equation 4.1 produces the following scores: feature $(1, 1)$ is assigned the largest weight at $0.191\,915\,22$, followed by feature $(2, 1)$ at $0.182\,008\,56$. The remaining weights are as follows: $w_{f_{2,0}} = 0.17089575$, $w_{f_{3,1}} = 0.16851988$, $w_{f_{3,0}} = 0.14533679$, and lastly, $w_{f_{1,0}} = 0.1413238$ which consider a less important feature that has a miner impact on the network internal representation.*

The importance weight for an extracted latent feature of a DNN's layer reflects some

distance perturbed feature	d_{L_1}	d_{L_2}	d_{L_∞}	d_{JS}	d_{corr}	d_{cos}	d_{MSE}	d_{RMSE}	d_{MAE}	d_{AF}
(1, 0)	150	0.73	0.00956	0.224	0.142	0.114	0.000000879	0.000937	0.000249	0.278
(1, 1)	340	1.18	0.00989	0.353	0.448	0.361	0.00000232	0.00152	0.000567	0.735
(2, 0)	325	1.09	0.00946	0.365	0.332	0.267	0.00000198	0.00141	0.000541	0.625
(2, 1)	360	1.16	0.0103	0.393	0.395	0.323	0.00000224	0.00150	0.000600	0.710
(3, 0)	276	0.88	0.00889	0.258	0.170	0.137	0.00000129	0.00114	0.000460	0.408
(3, 1)	315	1.07	0.00960	0.324	0.318	0.264	0.00000192	0.00139	0.000525	0.608

Table 4.2: Example distance measures.

relevant amount of information/variance that the abstracted DNN uses at the considered layer. Although the current abstraction scheme does not relate latent features with the DNNs' decisions, which is explored in the next chapter, perturbing a specific part of the latent space and observing the implicit changes of the learning models' distribution contributes to understanding their internal decisions.

4.5 Experiments

This section presents the results of a set of experiments designed to assess the effectiveness of the BN sensitivity analysis method in examining the behaviour of the latent features under perturbation. First, the sensitivity distance was calculated, and weight was assigned in accordance with these calculations. Next, an empirical evaluation of the sensitivity of BN abstractions at detecting distribution shifts induced by adversarial examples was conducted. The section is divided into three Sub-Sections. Sub-Section 4.5.1 provides a description of the datasets and models utilised to conduct the experiments and the specified setup to construct the BNs. The evaluation of the sensitivity analysis given in Algorithm 1 was conducted in Sub-Section 4.5.2 to quantify the impact of perturbations in latent features on the BN distribution. In Sub-Section 4.5.3, the objective was to examine the BN sensitivity measure in the presence of adversarial distribution shifts instead of the designed random_shift. The outcomes from the adversarial shift detection experiments were presented and discussed in the results paragraph.

4.5.1 Datasets and Experimental Setup

Two trained Convolutional Neural Network (CNN) models were trained for the experiments: the first one targeted the MNIST classification problem (layers are listed in Table A.3

in Appendix A) with 99.38% validation accuracy, and the second model targeted the CIFAR-10 dataset (Table A.5) with 81.00% validation accuracy. The models are reasonably sized, with more than 15 layers including blocks of convolutional and max-pooling layers, followed by a series of dense layers. They have 312 202 and 890 410 trainable parameters, respectively.

The Bayesian Network abstraction scheme accepts a wide range of feature extraction techniques and discretisation strategies. To explore their impact on the presented approach, a wide set of the BN abstractions were considered. Two linear feature extraction techniques were examined: Principal Component Analysis (PCA) and Independent Component Analysis (ICA); and one non-linear technique: radial basis functions (RBF) kernel-PCA. The author decided to fix the number of extracted features at three features per layer; this choice of a relatively small number of hidden features enables the use of many intervals (5 or 10) for their discretisation while still obtaining reasonably-sized probability tables. Both uniform- and quantile-based discretisation strategies were applied, with or without the addition of two left- and right-most intervals that do not contain any element of the training sample. Finally, three hidden layers are considered to construct the BN abstractions: for the two models, the first two selected layers directly follow a block of convolutions, while the last is a dense ReLU layer that is situated few layers before the NN's output layer. Namely, 'max_pooling2d', 'max_pooling2d_1', and 'activation_4' (see Tables A.3 and A.5). The layers chosen criteria is based on a belief that the activation values at these layers capture relevant patterns w.r.t the NN decisions.

4.5.2 Sensitivity to Perturbation

The experiments reported in this section were designed to evaluate the proposed features sensitivity to a designed perturbation through the random_shift. The algorithm generates the R_X from training data $\mathbf{X}_{\text{train}}$ and calculate their probability distribution *w.r.t.* to the BN. It then iterates over all nine latent features for both models and shifts their intervals and re-calculate their probabilities under the BN. The plot in Figure 4.5 illustrates the probabilities distributions for the original P_{ref} probability vector and the probability vector for each perturbed feature obtained from the BN abstraction of the MNIST model.

Figures 4.6 and 4.7 quantify the difference between the original distribution represented by the BN and the resulting distribution after each perturbation for both the MNIST and CIFAR10 models, respectively. Various metrics of distance between the reference probabil-

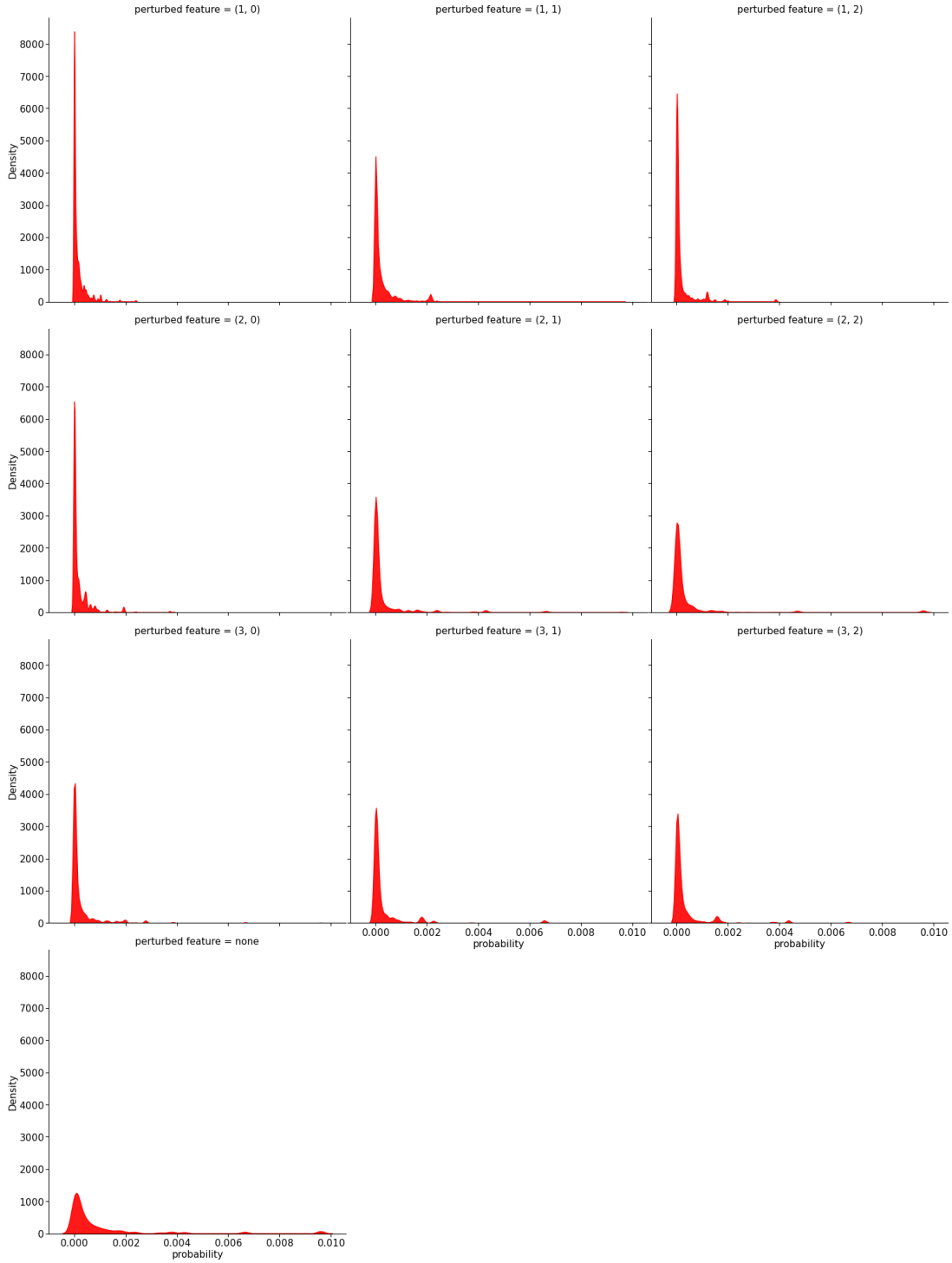


Figure 4.5: Probabilities distributions of nine perturbed features from the MNIST model. The probability in the last row is the clean P_{ref} probability.

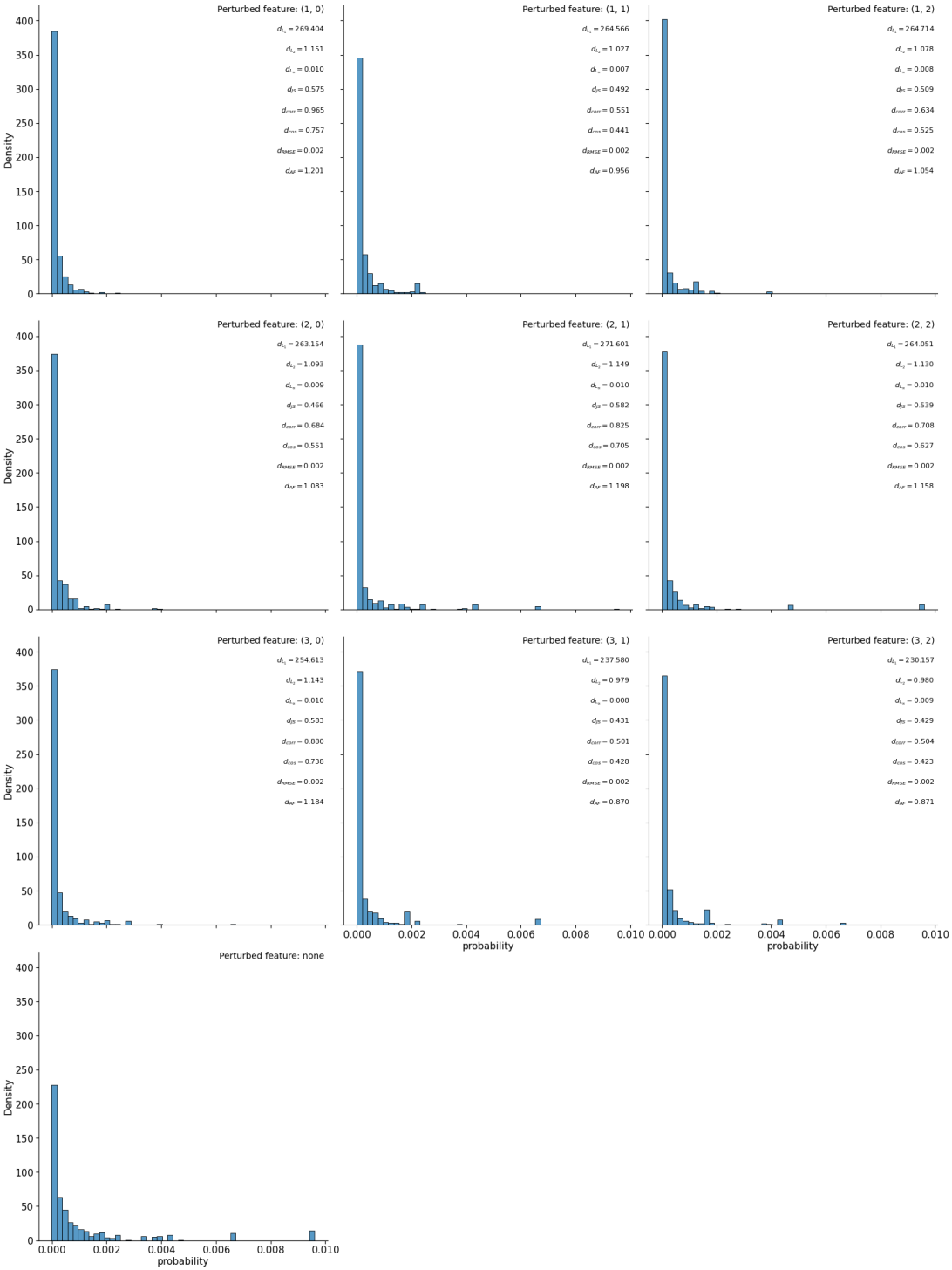


Figure 4.6: Distributions of probabilities for each perturbed feature obtained from the BN abstraction of the MNIST model. Each plot shows respective distance measures *w.r.t.* the probabilities obtained from the BN for the clean unperturbed features P_{ref}-shown in the last row.

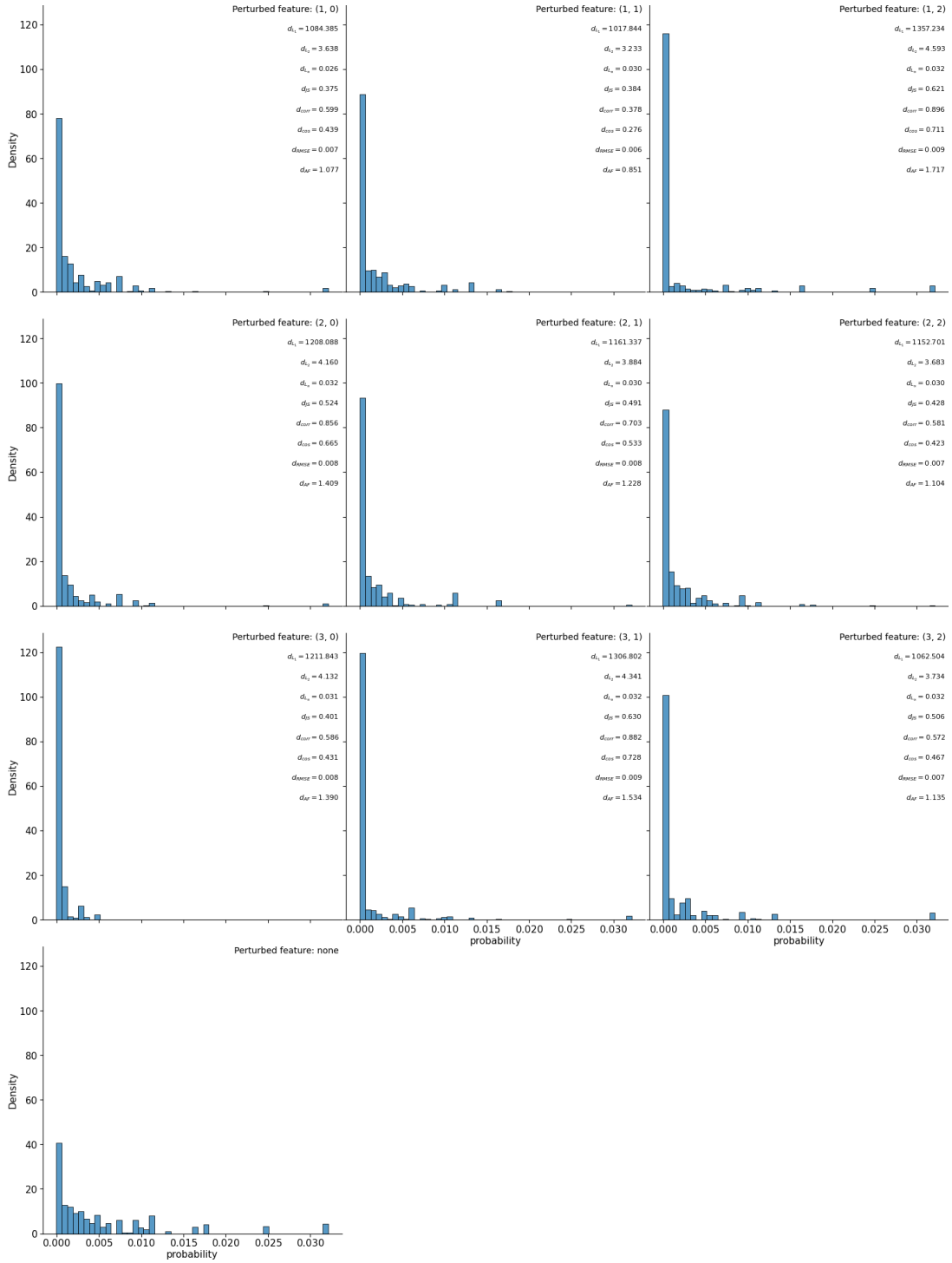


Figure 4.7: Distributions of probabilities for each perturbed feature obtained from the BN abstraction of the CIFAR10 model.

ities P_{ref} and the respective nine perturbed feature probabilities P'_f are annotated on plots. The computed difference between these two probability distributions illustrates the internal change of the BN distribution. Consequently, we can determine the safety violation risk by comparing an input probability belonging to the BN probability distribution.

Assigning features weights. The computed sensitivity distances for both models are normalised to produce the importance weights presented in Table 4.3. As can be seen, each distance metric produced different weights according to its definition. A specific method will be discussed in the next chapters to determine the best distance measure to be considered for assigning weights through the testing process.

distance: pert_feat	d_{L_1}	d_{L_2}	d_{L_∞}	d_{JS}	d_{corr}	d_{cos}	d_{MSE}	d_{RMSE}	d_{MAE}	d_{AF}
(1, 0)	0.116	0.118	0.119	0.125	0.154	0.146	0.125	0.118	0.116	0.125
(1, 1)	0.114	0.106	0.093	0.107	0.088	0.085	0.100	0.106	0.114	0.100
(1, 2)	0.114	0.111	0.105	0.111	0.101	0.101	0.110	0.111	0.114	0.110
(2, 0)	0.113	0.112	0.114	0.101	0.109	0.106	0.113	0.112	0.113	0.113
(2, 1)	0.117	0.118	0.119	0.126	0.132	0.136	0.125	0.118	0.117	0.125
(2, 2)	0.114	0.116	0.119	0.117	0.113	0.121	0.121	0.116	0.114	0.121
(3, 0)	0.110	0.117	0.119	0.127	0.141	0.142	0.124	0.117	0.110	0.124
(3, 1)	0.102	0.101	0.097	0.094	0.080	0.082	0.091	0.101	0.102	0.091
(3, 2)	0.099	0.101	0.115	0.093	0.081	0.081	0.091	0.101	0.099	0.091

distance: pert_feat	d_{L_1}	d_{L_2}	d_{L_∞}	d_{JS}	d_{corr}	d_{cos}	d_{MSE}	d_{RMSE}	d_{MAE}	d_{AF}
(1, 0)	0.103	0.103	0.095	0.086	0.099	0.094	0.094	0.103	0.103	0.094
(1, 1)	0.096	0.091	0.109	0.088	0.062	0.059	0.074	0.091	0.096	0.074
(1, 2)	0.128	0.130	0.115	0.142	0.148	0.152	0.150	0.130	0.128	0.150
(2, 0)	0.114	0.118	0.115	0.120	0.141	0.142	0.123	0.118	0.114	0.123
(2, 1)	0.110	0.110	0.109	0.113	0.116	0.114	0.107	0.110	0.110	0.107
(2, 2)	0.109	0.104	0.110	0.098	0.096	0.091	0.096	0.104	0.109	0.096
(3, 0)	0.115	0.117	0.113	0.092	0.097	0.092	0.121	0.117	0.115	0.121
(3, 1)	0.124	0.123	0.116	0.145	0.146	0.156	0.134	0.123	0.124	0.134
(3, 2)	0.101	0.105	0.117	0.116	0.095	0.100	0.099	0.105	0.101	0.099

Table 4.3: Calculated weights from the sensitivity distances using various measures, for the MNIST model on top and CIFAR-10 in the bottom. The headers show the considered distances: d_{L_1}, d_{L_2}, d_{L_∞}, d_{JS}, d_{corr}, d_{cos}, d_{MSE}, d_{RMSE}, d_{MAE}, and d_{AF}. The first column in each table indicates the perturbed feature that is under investigation.

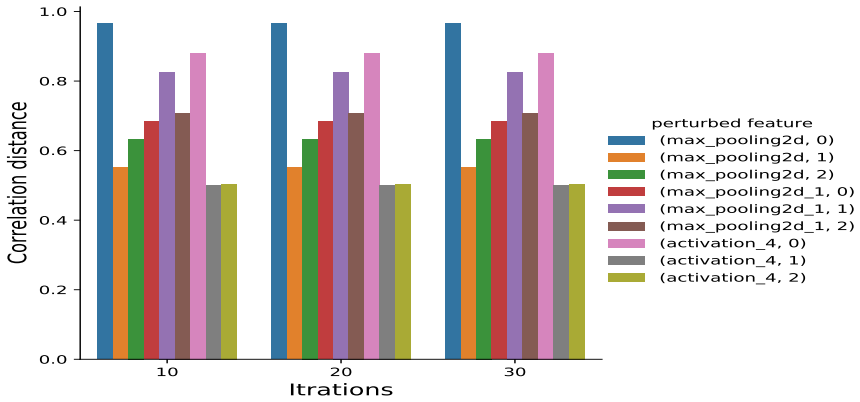


Figure 4.8: Correlation distance between P_{ref} and P'_f for 10, 20, and 30 iteration of perturbation. Hue indicates each perturbed feature f with specific colour.

Further details. In the process of calculating BN probabilities, we can grow the vector of these probabilities, *i.e.,* P_{ref} and P'_f, with duplicates whatever much we desire to obtain statistically significant results. This is implemented by tiling the P_{ref} vector i-th times, and perform random_shift on each feature i-th iterations, *i.e.,* Perturbing feature 0 of layer activation_4 (iteration 3/10). Figure 4.8 shows a selected distance (Correlation) calculated between P_{ref} and P'_f vectors of 10, 20, and 30 iteration of tiling and perturbation. The chart in Figure 4.8 exhibits consistency distance for all iterations, suggesting the iteration of 10 serves the purpose of making internal changes in the BN distribution. Furthermore, as a preliminary assessment, we have checked the distance between the first and second half of P' is close to 0. This specific examination is to ensure that the distance is meaningful and the 10 iterations was sufficient.

4.5.3 Sensitivity to Adversarial Distribution Shift

A set of experiments were carried out to assess whether the set of three features extracted for each considered hidden layer allowed for the capture of relevant properties of the learnt representations. In particular, to check whether the BN abstraction would detect the shift in the distribution of inputs that occurs when the NN is subject to adversarial examples. In other words, the experiments were aimed at discovering whether some distance measures indicated that the BN abstractions capture relevant latent features (and their dependencies) with sufficient precision to associate diverging probabilities between “legitimate” inputs

and adversarially perturbed ones. If such is the case, the results should conclude that the abstraction scheme and the associated BN are sufficiently precise to capture relevant dependencies in latent feature values that may not be matched (or matched too well, depending on the sign of the actual difference in probabilities) by some adversarial inputs.

To carry out these experiments, the following adversarial attacks were selected:

fgsm is the Fast Gradient Sign Method of Goodfellow et al. [35];

pgdlinf and pgdl2 are the Projected Gradient Descent approach of Madry et al. [65] with L_∞ and L_2 norm, respectively;

cwlinf and cwl2 are Carlini and Wagner [14]'s attack with L_∞ and L_2 norm, respectively, both targeting 0.1 confidence;

deepfool is the DeepFool attack by Moosavi-Dezfooli et al. [68].

Attacks involving the L_∞ norm target a maximum of $\varepsilon = 0.1$ perturbation in the input images, whereas pgdl2 targets a maximum perturbation $\varepsilon = 10$.

For each attack, an adversarial dataset X_{attack} was generated from the validation dataset X_{test} for both the MNIST and CIFAR10 models, where each dataset consisted of 10 000 inputs. Then, for each attack and BN abstraction $\mathcal{B}$ built and fitted using 20 000 elements drawn from the respective training datasets, a set of distances p were measured between the vectors of probabilities $\Pr(X_{test} \in \mathcal{B})$ and $\Pr(X_{\mathsf{attack}} \in \mathcal{B})$, denoted:
$d_p(\Pr(X_{test} \in \mathcal{B}), \Pr(X_{\mathsf{attack}} \in \mathcal{B}))$.

Results

Figure 4.9 shows the results for three selected distances L_2, cos, and AF. More detailed results are given in Appendix B. Each chart in the figure illustrates the calculated distances with four colours according to the discretisation method and the number of intervals in the vertical axis, using three sets of feature extraction techniques (pca, ica, and rbf.kpca) in the horizontal axis. The used distance metric and attack type are shown at each chart's top. First of all, it can be observed that some combinations of abstractions and distance measures exhibit notable differences between the validation dataset and the adversarial one for some attacks. For instance, every distance shown allows us to measure a shift in input distribution for every attack, except Carlini and Wagner [14]'s in some cases. Next, although the feature extraction technique does not have a noticeable impact on any

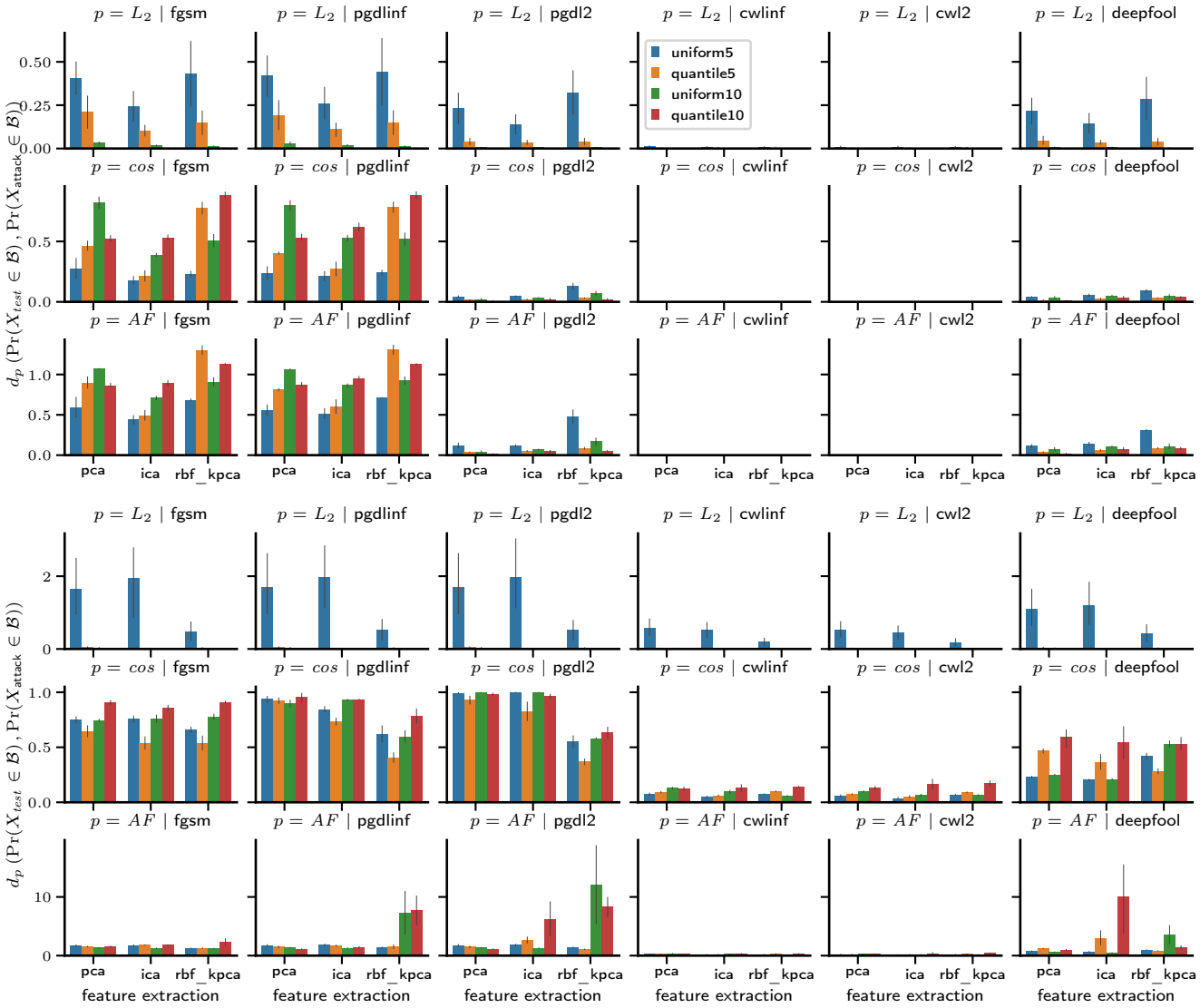


Figure 4.9: Selected distances (vertical axes) between probability vectors obtained for the validation dataset ($\Pr(X_{test} \in \mathcal{B})$) and probability vectors ($\Pr(X_{\text{attack}} \in \mathcal{B})$) obtained for datasets generated by selected adversarial attacks (shown on each column), for a range of BN abstractions $\mathcal{B}$. The top (resp. bottom) three rows show results for the MNIST (resp. CIFAR10) model. Hue indicates the discretisation strategy and the number of intervals. The grey vertical lines show confidence intervals.

measured distance, the discretisation strategy certainly plays a role in the ability of the BN to model each abstracted latent feature and their dependencies with sufficient precision.

For example, in the first row of the CIFAR-10 experiment (L_2 distance), the distribution shift is detected when using the uniform-based discretisation method with five intervals (distance with blue color).

Overall, the experimental results demonstrated that computing distances between two BN probability distributions, clean and perturbed by intervals-shift or adversarial attacks, can detect the distribution shift where it exists. It should be noted that, in the case of adversarial shift, this is measured based on *the latent features only*. Given this empirically confirmed property, BN-based computation of feature importance appears to be one tool, which adds to the growing set of useful techniques for the detection of important features as well as of adversarial examples. What is more, it adds a semantic twist to this analysis and allows for explaining in which way the changes in the features contribute to the distribution shift.

4.6 Discussions

A further discussion of a few aspects related to either the used method or the potential applications of the method is addressed in this section.

Hyper-parameters in BN Construction. The parametric nature of the scheme advanced by Berthier et al. [9] enables the exploration of a wide range of DNN abstractions. For instance, in the previous experiments, the sensitivity to adversarial distribution shift relied most on the *linear* dimensionality reduction techniques to extract latent features. Further experiments with more non-linear feature extraction techniques, like manifold learning [57], are desirable to assess the properties of extracted features in extended cases. The effect of more advanced discretisation strategies can also be explored, for instance by relying on kernel density estimations to partition each latent feature component into intervals that span across ranges of the real line that are either densely or non-densely exercised by the training sample.

Hyper-parameters in Weight Quantification. There are a number of building blocks in the weight quantification method (Algorithm 1), including: the perturbation made to generate new CPTs, the random shifting function, and the distance metrics for probabilities (P_{ref}) and (P'_f). This chapter has explored several different options of the use of distance

metrics for comparison. It would also be useful to study if and how the other hyper-parameters may affect the overall results.

Utility of Feature Weights. Quantifying the importance of the hidden features provides three advantages. **First,** visualising the most important features provides insight into the model's internal decisions by highlighting dominating regions in the feature space. **Second,** the importance measurement can be used to design high-level testing metrics that evaluate the robustness of the DNN. Some attempts have been made in Berthier et al. [9], where no feature weight is taken into consideration. **Third,** with *feature importance* as a defence, one can utilise the obtained importance in the training process and force the DNN to adjust its parameters according to the features that are most relevant for the prediction. Using extracted weights for training purposes is the most widely adopted direction. For example, Zhang et al. [102] propose a hierarchical feature alignment method that computes the difference between clean and adversarial feature representations and utilises it as a loss function when optimising network parameters, while Bai et al. [7] suggest that different channels of a DNN's intermediate layers contribute differently to a specific class prediction and propose a Channel-wise Activation Suppressing training technique that learns the channel importance, and leverages them to suppress the channel activation while training the network.

4.7 Conclusions

This chapter has advanced a novel technique that employs a BN abstraction to investigate how to measure the importance of high level features when they are used by the neural network to make classification decisions. The proposed algorithm estimates the importance of each feature by analysing the sensitivity of the abstraction to targeted perturbations. The derived weight values reflect the role of the corresponding feature in the underlying decision process. In addition, the sensitivity analysis method proves its ability to detect the distribution shifts before and after perturbation, which will open many doors for future exploration.

Chapter 5

Bayesian Network Prediction

5.1 Introduction

In the previous chapter, feature importance weights were introduced based on the DNN internal representations. However, as stated, the BN abstraction itself does not make any link with the actual DNN decisions. Since the proposed importance values from chapter 4 are defined according to how much each feature influences the classifier's decision, the justification there is about the final classification decision. Therefore, providing the abstracted BN model with the ability to perform prediction is a critical step in measuring the influence of the feature perturbation on the classification decisions based on the abstraction. This chapter is directed at providing an answer to the third subsidiary research question SRQ3, which were presented in Chapter 1:

- **SRQ3:** How to demonstrate the impact of the *feature importance* measure on the classification decisions based on the abstraction?

The work presented in this chapter reports on how the Bayesian Network model, presented in Chapter 3, can be transformed into a classifier that performs predictions via probabilistic inference. Achieving this allowed for investigating the degree to which the internal neural network representation contributed to the final prediction. Inference is the process of computing a probability value of selected nodes of a Bayesian network according to a given query that provides evidence of other variables' values. Bayesian networks do not directly support computing the probability of arbitrary pieces of evidence or unconsidered variables, *i.e.,* input's labels, when they are constructed. However, such probabilities can

be computed indirectly by adding an auxiliary node to the network [23]. In this work, the methodology followed the auxiliary node method that adds a node at the end the abstract BN, which will be a random discrete variable that takes its values in the set of labels.

The overall technique used to enable the BN to predict new input labels is illustrated in Figure 5.1. To construct the BN predictor, an auxiliary prediction node is added at the end of the BN and its conditional probability table (CPT) is calculated based on the BN's nodes distributions. A major question raised is: what should be the prediction node's predecessor nodes? Would connecting it with the previous layer's nodes only achieve a good prediction task and keep cheaper computational complexity? Or would it be better to ignore the computational side and make the extra node conditionally depend on every node of the original BN to acquire a higher precision?

After connecting the extra node in either way and generating its CPT using sample data from the training dataset, a new Bayesian network structure is created. To predict an input's label, it first transformed into observations suitable for the abstract BN to obtain a vector of evidence that refers to the BN's nodes values. Finally, the inference engine applies the Bayes principle to estimate the maximum posterior distributions of the output label. Further, to examine the robustness of the abstraction, the original DNN model is attacked to generate adversary perturbed input data from the test set. It is then converted into observations and passed into the Bayesian inference to evaluate the BN robustness accuracy.

To summarise, the main contributions of this chapter are:

- An advanced Bayesian network constructed from all, or chosen layers, of a neural network with the ability to perform the original classification task;
- A comparison of how a crafted perturbation on a test dataset affects the classification accuracy of a neural network on the one side, and its Bayesian network abstraction on the other side;
- An empirically evaluation showing that the BN predictor is a good approximator of the original DNN and exhibits a smaller gap between accuracy over clean data and accuracy over perturbed data.
- A mechanism to calculate feature weights based on the change in the Bayesian network predictions.

The remainder of the chapter is organized as follows. Section 5.2 reviews the methodological approaches proposed to extract simple deterministic models from neural networks to analyse their internal representation and understand their decisions. Section 5.3 provides

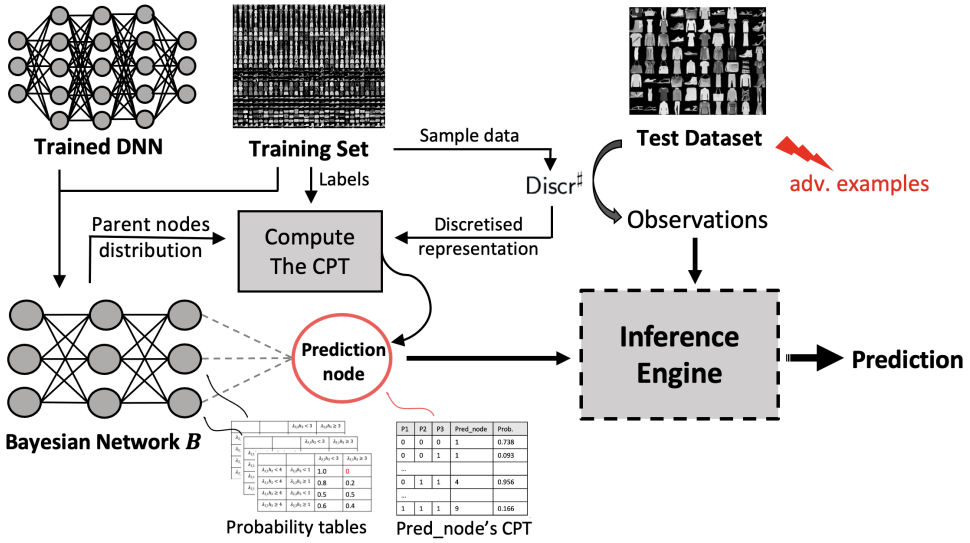


Figure 5.1: Schematic view of the proposed approach. Given a Bayesian Network $\mathcal{B}$ constructed from selected layers of a trained DNN model, a prediction node is added at the end of the $\mathcal{B}$ to allow it predicts an input's label. The prediction node probability is calculated using its parent nodes distribution and sample data transformed from the training dataset into a lower-dimensional, discretised representation through the discretisation function $\mathsf{Discr}^\sharp$ and coupled with their labels. Next, the test data is converted into observations by applying $\mathsf{Discr}^\sharp$ process, which returns a vector of elements referring to the BN's node values. The inference engine then takes these observations values of the $\mathcal{B}$'s nodes distribution and infer the probability of the output label. The adversarial attack indicates the abstracted BN's robustness evaluation step.

a brief preliminary explanation concerning the BN's conditional probability table. In Section 5.4, the method to construct a classifier based on the abstract BN is presented. The primary objective of this chapter, which is directed at extracting feature weights using the BN prediction, is described in Section 5.5. Section 5.6 evaluates the accuracy of the resulting BN approximator model and analyses its robustness using a test dataset with crafted perturbations. This section also presents the results from an experiment to calculate the DNN feature weights by observing a change in the BN prediction. The last two Sections of this chapter highlight significant findings, compare the BN predictor model to others, and then conclude the presented work.

5.2 Related Work

Since this chapter constructs a classifier based on an abstract Bayesian network form a neural network, the literature will emphasise the current neural network abstractions and approximation research. Many recent studies have focused on extracting simpler models of complex, high-dimensional deep neural networks, with the expectation that the abstractions are as close as possible to the original neural network while being interpretable and preservative of key properties. The existing related work is divided into two categories: (i) abstraction methods that generate a smaller model of a neural network with similar behaviour; and (ii) approximation techniques that approximate the neural network behaviour using transparent models to obtain an approximator with a prediction accuracy that is close to that of the original DNN.

5.2.1 DNN Abstractions

The methods that are used to construct abstraction models from neural networks can be categorised as Boolean, Causal, Clustering, Probabilistic, and Box abstractions. The categorisation is based on the algorithm used to abstract the internal representations of the network, or based on the abstraction structure used to store the network internal activation patterns.

Abstracting neural networks for verification purposes was first proposed by Pulina and Tacchella [76], who transform the networks into Boolean combinations of linear arithmetic constraints. Further, Boolean abstracting was recently applied by Cheng et al. [20] to store the neuron activation patterns in abstract form, and use it as a monitor. With Causal

abstraction theory [42, 31], neural representation values are aligned with variables in structural causal models providing causal explanations of neural network behaviours. Neural network abstractions through Clustering, as in DeepAbstract [5], is performed by merging neurons that behave similarly on some inputs to obtain an abstract network that simulates the behaviour of the original network. Dong et al. [26] developed an algorithm that combines probabilistic learning and abstraction through clustering to extract probabilistic finite-state automata from recurrent neural networks. Finally, Box abstraction is applied by Henzinger et al. [40] and Cheng [18] to compute bounds (boxes) over neuron valuations that have been seen during the training process, use them to build a run-time monitor.

The purpose of these abstraction approaches was either the verification of neural networks [76, 5], monitoring [20, 40], or abstract interpretation based methods [59, 42]. None of these abstraction models was employed to make a classification decision.

5.2.2 DNN Approximators

The transparent models used to approximate neural networks have the following characteristics. They have (i) simple and compact structure that is a human-level understandable, (ii) ability to break down a model into parts that can be explained separately, and (iii) a clear procedure to generate the output. These characteristics exist in models such as logistic regression, decision trees, and Bayesian networks that are inherently considered transparent.

The approaches adopted by Sato and Tsukimoto [82], Dancey et al. [22] involves extracting decision trees from neural networks. The trees approximate their predictive behavior and are utilised in explaining the NN's decisions. Other attempts are approximating the posterior distribution of neural networks to construct approximators that estimate the DNN's uncertainty. Ritter et al. [79] developed a Kronecker factored Laplace approximation model by optimising the posterior over the weights of a trained neural network to obtain uncertainty estimates. They further tested the robustness of their Kronecker factored prediction using untargeted and targeted adversarial attacks. Their experimental results suggested that their approximator model leads to better uncertainty estimates and is more robust to simple adversarial examples. Maddox et al. [64] proposed the SWA-Gaussian, that approximates posterior distribution over neural network weights to model Bayesian inference and estimate uncertainty.

The work in this chapter combines abstraction and approximation techniques which

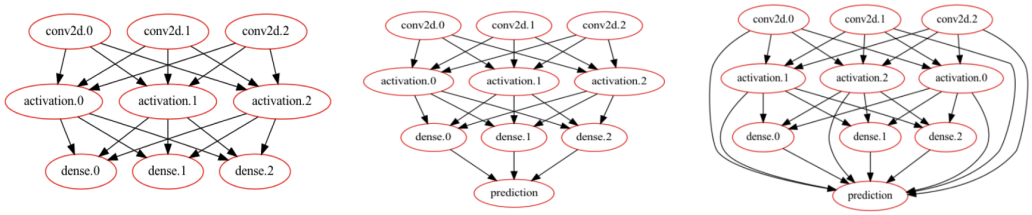
approximates an *abstracted* Bayesian network from a trained neural network to acquire a unique model. Since the presented BN is not designed to make classification decisions, author suggests to optimise the posterior of the training labels over the BN parameters to create an approximator, and test its prediction robustness versus the original DNN model. Observe that, all uncertain variables are modelled as probability distributions in Bayesian data analysis, and inference is carried out by constructing the posterior conditional probabilities for the desired variables given the observed data and prior assumptions. Therefore, the Bayesian theory is concerned with the parameter posterior instead of a point estimate that often leads to overconfident predictions as in DNNs. Thus, Bayesian inference is suspected to have a higher opportunity to predict robustly against imperceptible perturbations.

5.3 Preliminaries

The terminology of the Bayesian Network abstraction model was introduced in Chapter 3. As explained, the abstraction process results in a Directed Acyclic Graph (DAG), where each node contains an associated conditional probability table (CPT) or marginal probability table for input nodes. Since the aim of this chapter is to construct the prediction node's CPT, the preliminaries will emphasis on the BN conditional probability table. First, the main components of building the Bayesian network abstraction are reviewed. Followed with a simple illustrative example of the CPT.

Let $\mathcal{N}$ be a trained deep neural network with sequential layers $L = (l_1, \ldots, l_K)$, and let X be a training dataset. As an abstract model of $\mathcal{N}$ and X, a Bayesian Network (BN) is a directed acyclic graph $\mathcal{B}_{\mathcal{N},X} = (V, E, P)$, where V are nodes, E are edges that indicate dependencies between features in successive layers, and P maps each node in V to a probability table representing the conditional probability of the current feature over its parent features according to $\mathcal{N}$'s behaviors when it is subject to X.

Notice that in the description above, the focus is the input and hidden features of the DNN only, i.e. the output prediction layer is not considered; $\mathcal{B}_{\mathcal{N},X}$ is not designed to include output nodes that would allow classification. A more detailed description of how to create such a *prediction* node is given in the next Section. The BN conditional probability table is defined for each feature interval $f_{i,j}^{\sharp k} \in \mathbb{F}_{i,j}^{\sharp}$ for layer l_i, *w.r.t.* each combination of



(a) Berthier et al. BN. (b) BN classifier -Deepest nodes. (c) BN classifier -Fully connected.

Figure 5.2: Illustration of Bayesian network structures before (a) and after adding the prediction node (b & c). Two ways of connecting predictions to the rest of the hidden features (BN's nodes) are - (b) connecting deepest nodes only or (c) connecting all nodes.

act.0	act.1	act.2	dense.0	
0	0	0	0	0.30492
0	0	0	1	0.02006
0	0	0	2	0.67502
0	0	1	0	0.49985
...	...	...	...	...
1	2	1	1	0.77721
1	2	1	2	0.01373
...	...	...	...	...
2	2	2	1	0.34444
2	2	2	2	0.32777

Table 5.1: Example of the conditional probability table for the BN node "dense.0", which represents a conditional probability $P(\text{dense.0}|\text{act.0}, \text{act.1}, \text{act.2})$.

feature intervals $F^{\sharp}_{i-1}$ for layer l_{i-1}, as

$$\mathcal{CP}_i\left(f^{\sharp k}_{i,j} \,\middle|\, F^{\sharp}_{i-1}\right) \stackrel{\text{def}}{=} \Pr\left(x \rightsquigarrow f^{\sharp k}_{i,j} \,\middle|\, x \rightsquigarrow F^{\sharp}_{i-1}\right). \tag{5.1}$$

Intuitively, $\mathcal{CP}_i\left(f^{\sharp k}_{i,j} \,\middle|\, F^{\sharp}_{i-1}\right)$ gives the probability that any input $x \in X$ chosen uniformly exhibits feature interval $f^{\sharp k}_{i,j}$, knowing that it exhibits intervals $F^{\sharp}_{i-1}$ for all extracted features at layer l_{i-1}. To make the procedure clearer, the steps are sketched in the following example.

Example 14. *Figure 5.2-(a) is a Bayesian Network constructed from three selected layers of the CNN model $\mathcal{N}_{sm}$: conv2d, activation, and dense. Three features are extracted from each layer using a Principal Component Analysis. Then, each feature is discretised into*

three intervals that partition its value range; for extracted features, each interval is denoted with a distinct integer. Table 5.1 shows the conditional probability table for the node dense.0 *which represents a first extracted feature from the third DNN's layer. The table assigns a probability to each* dense.0 *feature interval w.r.t. each combination of the parent feature intervals from the previous layer (*activation.0*,* activation.1*, and* activation.2*). As an example, the first three rows of the table imply that when all* dense.0 *feature's parents exhibit intervals 0, the* dense.0 *will have a* 0.30 *probability of exhibiting interval 0,* 0.02 *probability of exhibiting interval 1, and* 0.67 *probability of exhibiting interval 2. In other words,*

$$P(dense.0 = 0 \mid act.0 = 0, act.1 = 0, act.2 = 0) = 0.30,$$
$$P(dense.0 = 1 \mid act.0 = 0, act.1 = 0, act.2 = 0) = 0.02, \text{ and}$$
$$P(dense.0 = 2 \mid act.0 = 0, act.1 = 0, act.2 = 0) = 0.67.$$

5.4 Probabilistic Inference using the BN

This section discusses how to solve a classification task on unseen inputs by using Bayesian inference on the BN abstraction model. As mentioned previously, some additional steps are needed to enable the BN to perform predictions. The two main steps are:

1. Transform input samples into a representation that is suitable for the BN;
2. Add an auxiliary *prediction* node at the end of the BN, and calculate its conditional probability table.

5.4.1 Abstracting the Training Data

The first step in inferring a label is to transform the input data that suits the DNN into a representation that allows to perform inference using the BN. For a sample of training input data $\mathbf{x}_{\text{train}} \in \mathbb{D}_X$, where $\mathbb{D}_X$ is the input space, its corresponding DNN activation $\hat{\mathbf{x}}_{\text{train}}$ is first calculated. Then, the feature extraction and discretisation function $\mathsf{Discr}^\sharp(\hat{\mathbf{x}}_{\text{train}})$ is used to obtain a lower-dimensional and discretised representation $\mathbf{r}_{\text{train}} \in \mathbb{F}^\sharp$, where $\mathbb{F}^\sharp$ is the discretised feature space. Here, components of the vector $\mathbf{r}$ are valuations for each node/variable in the BN.

Example 15. *Consider the BN in Figure 5.2-(a). Transforming an input* $\mathbf{x}$ *for the DNN into representation values* $\mathbf{r} = \mathsf{Discr}^\sharp(\mathbf{x})$ *may give, e.g.,* $\mathbf{r} = [1, 2, 1, 1, 1, 2, 0, 2, 1]$. *The nine*

values in the vector refer to the abstract intervals of highest probability for each one of the nine nodes of the BN, when the DNN is subject to $\mathbf{x}$. *For example, the first node* conv2d.0 *in Figure 5.2-(a) has evidence interval equal 1, and the second node* conv2d.1 *has evident evidence interval equal 2, and so on.*

5.4.2 Adding an Auxiliary Node

The previous step described how to adapt input data suitable for the DNN into data suitable to the BN. Such a transition is possible for all layers of the DNN, except for the prediction itself. Indeed, the general mechanism by which prediction is done in a DNN is to take the output of the last hidden layer l_K, make a linear transformation and compute a softmax activation. The output is then interpreted as a maximum probability of a particular class.

The way to approach prediction in BN scenario is to add an auxiliary sink node to the DAG, as illustrated in Figure 5.2. This additional discrete categorical prediction variable takes its values in the range of classes defined by the dataset used to train the original DNN. Assuming the CPT associated with this additional node has been calculated, the traditional inference procedure may be applied on the new BN to estimate the probably of the class associated with given abstract intervals. Similarly to the DNN case, the predicted class is then the class that is given maximal probability.

In this framework, the structure of the new BN differs according to how the new variable is connected to the original DAG. In the following, two options are presented: (1) Direct dependence to all features extracted from the deepest layer of the considered DNN; (2) Direct dependence to every feature extracted from the DNN. In the first case, the *prediction*'s parent nodes are defined as only the ones originating from the last hidden DNN layer (Figure 5.2-(b)). In the second case, the *prediction* node is connected to all other nodes (Figure 5.2-(c)). The latter choice is expected to improve the accuracy of the BN, at the cost of computational and memory overhead.

Generating the Conditional Probability Table. Let label the *prediction* node with Y and its parent nodes with $\pi_Y \in \mathbb{F}_K^\sharp$, where K-th layer is the deepest abstracted layer of the DNN $\mathcal{N}$. Given a BN $\mathcal{B}$, the node Y is placed as a sink and connected with its parent nodes π_Y. The probability distribution associated with Y is estimated through iterating over $\mathbf{x}_{\text{train}} = \{\mathbf{x}_{t1}, \ldots, \mathbf{x}_{tN}\}$, take true training labels $\mathbf{y}_{\text{train}} = \{y_{t1}, \ldots, y_{tN}\}$ and compute a CPT as:

$$\mathcal{CP}_Y\left(y \mid \mathbb{F}^\sharp_K\right) = \Pr\left(y \mid \mathbf{x} \rightsquigarrow \mathbb{F}^\sharp_K\right). \tag{5.2}$$

$$= \frac{\sum_{i=1}^{N} \mathbb{1}\left(y = y_{ti} \wedge \mathbf{x}_{ti} \rightsquigarrow \mathbb{F}^\sharp_K\right)}{\sum_{i=1}^{N} \mathbb{1}\left(\mathbf{x}_{ti} \rightsquigarrow \mathbb{F}^\sharp_K\right)}. \tag{5.3}$$

Here, y is the label of interest, $x \rightsquigarrow \mathbb{F}^\sharp_K$ implicitly calculates representation $\{\mathbf{r}_{t1}, \ldots \mathbf{r}_{tN}\} = \mathsf{Discr}^\sharp(\{x_{t1}, \ldots, x_{tN}\})$ and it denotes that the input x exercises/triggers the abstracted interval with discretised representation $\mathbf{r}_{\pi_Y}$. $\mathbb{1}$ gives 1 if the condition is satisfied and 0 otherwise.

The probability estimate given by the previous equation 5.3 is calculated for the full probability space (all possible values of y and $\mathbf{r}_{\pi_Y}$). For example, the CPT for the prediction node in Figure 5.2-(b) gives a probability for each label (e.g. $y \in [0, \ldots, 9]$ on MNIST), w.r.t. all parent combinations — in the example m^3 of them in total, where m is the number of intervals and 3 is the number of parents. The necessary condition for such a procedure to produce precise results is that the dataset is large enough. This assures that even low probability values are accurately estimated.

In the case where the node Y is connected to all of the previous nodes, one can simply substitute $\mathbb{F}^\sharp_K$ with $\mathbb{F}^\sharp_\mathcal{N}$ which includes all the discretised representations of the abstracted layers of $\mathcal{N}$. In a formal way:

$$\mathcal{CP}_Y\left(y \mid \mathbb{F}^\sharp_\mathcal{N}\right) = \Pr\left(y \mid \mathbf{x} \rightsquigarrow \mathbb{F}^\sharp_\mathcal{N}\right). \tag{5.4}$$

$$= \frac{\sum_{i=1}^{N} \mathbb{1}\left(y = y_{ti} \wedge \mathbf{x}_{ti} \rightsquigarrow \mathbb{F}^\sharp_\mathcal{N}\right)}{\sum_{i=1}^{N} \mathbb{1}\left(\mathbf{x}_{ti} \rightsquigarrow \mathbb{F}^\sharp_\mathcal{N}\right)}. \tag{5.5}$$

Reasoning with Bayesian networks. Once the BN is fully constructed, it represents a full distribution of the label and representation space: $P(y, \mathbf{r})$. With this distribution in hand, it is possible to reason with the BN. The following is a short summary of the key elements required to solve the inference in the BN classification problem.

- ***Prior*** - For general parameters $\boldsymbol{\theta}$, prior distribution $P(\boldsymbol{theta})$ defines the prior understanding/knowledge of the parameters. Very often, when there is no previous analyses to determine what the parameter values should be, one chooses a "flat prior".

For a continuous distribution, this is a constant value in some expected parameter range, for a discrete distribution it is the same constant for every class. In our case, $\boldsymbol{\theta}$ consists of all parameters of the CPTs in the Bayesian Network plus classification variable at the very end.

- ***Likelihood*** - For a general data space $\boldsymbol{d}$, likelihood is a distribution of the data given the parameters $P(\boldsymbol{d}|\boldsymbol{\theta})$. For a fixed data point, likelihood can be seen as a function in $\boldsymbol{\theta}$. In our case, two likelihood functions can be considered. Firstly, for images of a given class, we can say that there exists a distribution of them in image space $P(\boldsymbol{d}|y)$. Secondly, we can think of BN representation $\boldsymbol{r}$ and a likelihood function $P(\boldsymbol{r}|\boldsymbol{\theta})$, where now we have additional parameters of the BN in the likelihood.

- ***MLE*** - The Maximum Likelihood Estimator is a set of parameter values that maximise the likelihood function. For instance for some fixed data $\boldsymbol{d}$:

$$\hat{\boldsymbol{\theta}}_{MLE}(\boldsymbol{d}) = \text{argmax}_{\boldsymbol{\theta}}\, P(\boldsymbol{d}|\boldsymbol{\theta})\,. \tag{5.6}$$

- ***Posterior*** - Posterior distribution is the main result of Bayesian inference. Using inversion of conditional probabilities, one can write:

$$P(\boldsymbol{\theta}|\boldsymbol{d}) = \frac{P(\boldsymbol{d}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\boldsymbol{d})}\,, \tag{5.7}$$

  where $P(\boldsymbol{\theta}|\boldsymbol{d})$ is posterior $P(\boldsymbol{d}|\boldsymbol{\theta})$ likelihood, $P(\boldsymbol{\theta})$ prior and $P(\boldsymbol{d})$ evidence. Evidence is mostly considered as a normalisation constant.

- ***MAP*** - The Maximum Aposteriori Estimator is a set of parameter values that maximise the posterior distribution. Similarly as for the MLE, for some fixed data $\boldsymbol{d}$, The MAP is given by:

$$\hat{\boldsymbol{\theta}}_{MAP}(\boldsymbol{d}) = \text{argmax}_{\boldsymbol{\theta}}\, P(\boldsymbol{\theta}|\boldsymbol{d})\,. \tag{5.8}$$

The following section uses that introduced concepts for making a prediction with BN.

5.4.3 Performing Prediction

The way one can think of the setup in a predictive sense is as the following: the starting point is a set of images, which are samples from the likelihood function $P(\mathbf{d}|y)$, where $\mathbf{d}$

is the data space (i.e. each pixel is one dimension in $\mathbf{d}$) and y is an image class. The main goal of the machine learning is to invert this distribution in order to get a posterior $P(y|\mathbf{d})$. Then for every image in the test set, posterior distribution can be recovered and decided which class the image belongs to by computing the maximum aposteriori (MAP) class $\hat{y} = \operatorname{argmax} P(y|\mathbf{d})$.

In the BN prediction setup, one additional step is performed where the process starts with an image in data space $\mathbf{d} \in \mathbb{D}_X$ and compresses it into the BN representation $\mathbf{r} \in \mathbb{F}^\sharp$. Then the developed CPTs during the construction of the BN, can be used to obtain a posterior $P(y|\mathbf{r})$ and take a MAP estimate for the class. However, during fitting the BN with a prediction node distribution, the prediction node's CPT recorded its classification parameters, which were trained as a maximum likelihood estimator over a training set. Moreover, a prior of the classes $P(y)$ is inherently imprinted in the prediction of the BN from the training set. If all classes are expected to be equal (i.e. to have a "flat" prior), then the training set should be balanced between classes. To summarize, BN-base inference can be defined as following

Definition 5.4.1 (BN-based Inference)**.** *Given the BN representation vector* $\mathbf{r}$*, a prior over classes* y *as* $P(y)$*, and a likelihood function* $P(\mathbf{r}|y)$*, we can infer the posterior of our parameters as:*

$$P(y|\mathbf{r}) = \frac{P(\mathbf{r}|y) \cdot P(y)}{P(\mathbf{r})} = \frac{P(\mathbf{r}, y)}{P(\mathbf{r})}\,, \tag{5.9}$$

where distribution $P(\mathbf{r}, y)$ *is given by the BN appended with a prediction node, with other parameters of the BN CPTs fixed to their MLE estimates. Evidence* $P(\mathbf{r})$ *is not important for our case. Finally, calculating the MAP estimate gives us an inferred class estimation*

$$\hat{y} = \operatorname{argmax} P(y|\mathbf{r})\,. \tag{5.10}$$

It is necessary to be careful when constructing the full conditional distribution $P(y|\mathbf{r})$. In the case where the prediction node is connected just to the deepest nodes, then the total posterior will be given with multiplication of different CPT across the BN. This is illustrated in Figure 5.3, which shows the sink node Y and its parents. In this case, $\mathbf{r}_{\pi_Y} = (r_{\pi_{Y1}}, r_{\pi_{Y2}})$, while the full representation vector also includes representations from upper nodes, here marked with dots.

Example 16. *The BN in Figure 5.2-(b) is used to predict the MNIST test dataset output*

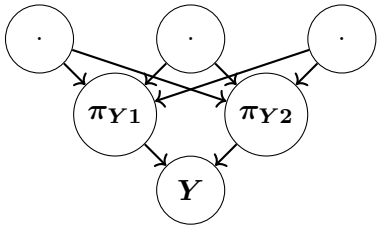


Figure 5.3: A simple illustration of the prediction node Y and its parents. Construction of the probability $P(y|\mathbf{r})$ includes not only representations from the parent nodes π_{Y1} and π_{Y2}, but all other nodes in the BN as well.

labels. The results of the first fifteen predicted images' digits are:

$$true\_labels: ([7, 2, 1, 0, 4, 1, 4, 9, 5, 9, 8, 0, 3, 2, 9])$$
$$pred\_labels: ([7, 2, 1, 6, 4, 1, 4, 9, 8, 9, 8, 0, 8, 2, 3])$$

A full classification diagnostic can be done through calculating the confusion matrix, which is a table used to evaluate the performance of a classification model by comparing the predicted outputs with the actual labels of a set of test data. The confusion matrix counts true predicted labels and the misclassified labels on a per-label basis. The plot in Figure 5.4 visualises the BN prediction confusion matrix.

5.5 Extracting Feature Weights using the BN Prediction

The fundamental purpose for using a BN abstraction model to predict a new input's label was to determine which hidden features influence the classification decision the most by generating feature weights. The method for extracting feature weights using the BN probabilistic inference is perturbing the value of each feature for every sample in the training data by applying the random shift presented in Sub-Section 4.3.2. And then calculating how many input had changed their predicted output after the perturbation. In formula, this amounts to:

$$W_{i,j} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\mathbf{x} \in \mathcal{D}_{\text{test}}} \mathbb{1}\left[P_{\text{BN}}(\mathbf{x}) \neq P_{\text{BN}}\left(\mathbf{x} \wedge \text{pert}(f_{i,j}^{\sharp})\right)\right], \tag{5.11}$$

where $W_{i,j}$ represents the importance weight of the feature $f_{i,j}^{\sharp}$, $P_{\text{BN}}(\mathbf{x})$ is the original

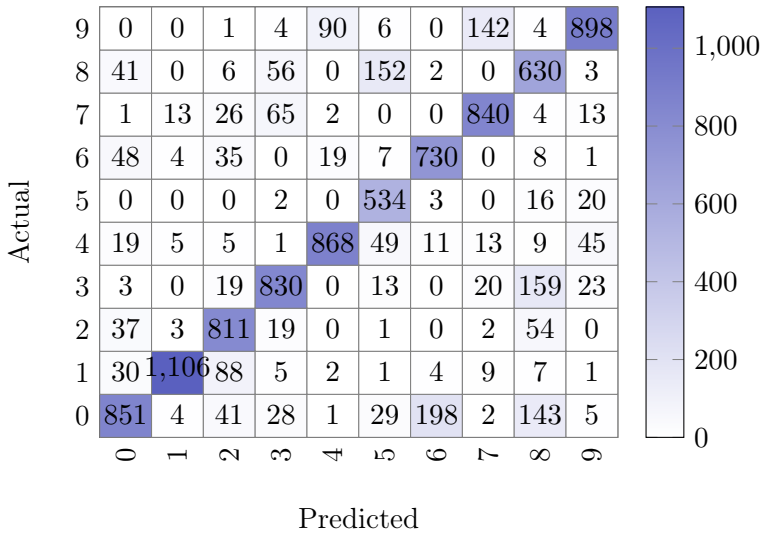


Figure 5.4: Confusion matrix for the BN actual vs. predicted values for a 10000 MNIST test data. The pairwise digits with the high confusion are shown in light blue.

prediction of the Bayesian network for input $\mathbf{x}$, $P_{\text{BN}}\left(\mathbf{x} \wedge \text{pert}(f^{\sharp}_{i,j})\right)$ is the BN prediction for input $\mathbf{x}$ after perturbing feature $f^{\sharp}_{i,j}$, $\mathcal{D}_{\text{test}}$ is the set of testing data, and function $\mathbb{1}$ returns 1 if the prediction changed or 0 otherwise. The previous equation represents unnormalised weights, which are further normalised, so that $\sum_{i,j} W_{i,j} = 1$.

In Equation 5.11, the assigned importance score for each feature is positively correlated with the respective number of smaples that changed their decision values. That means if a large number of the predictive samples have been misclassified, then the feature contributes much to the classification decision and will be deemed an important feature.

5.6 Experiments

In this section, the results of three experiments are presented: Experiment 1, Experiment 2, and Experiment 3. Experiments 1 and 2 are meant to test how well different built-in BN prediction models work on both clean and messed-up test data, and to see how well they compare to the performance of the original DNN models. Both approaches to connecting the prediction node to the BN are implemented and examined. Experiment 1 conducted in Sub-Section 5.6.1 is concerned with examining the first method of connecting the prediction node with the BN deepest nodes only. Experiment 2, which is presented in

Sub-Section 5.6.2, assessed the second method of connecting the prediction node with all nodes in the BN. In Experiment 3, shown in Sub-Section 5.6.3, the outcomes of the feature weighting mechanism were demonstrated. This mechanism utilised the Equation 5.11, which was formulated based on the variation in the BN prediction output. The BN classifier model was implemented in the DeepConcolic tool[1].

5.6.1 Experiment 1: Connecting Deepest Nodes Only

The purpose of Experiment 1 was to assess the performance of the BN prediction model in approximating DNN accuracy. The main challenge is to keep the approximator models simple and flexible enough to approximate the complex NN model accurately. Therefore, medium-size DNN models were trained to construct BN classifiers from as many layers as possible, so their respective accuracies could be compared. The CPT of the prediction node was also kept relatively small, as the prediction node was connected to the deepest BN nodes only (see Figure 5.2-(b)).

Datasets and Experimental Setup. To evaluate the approach of the BN prediction, we used two trained CNN models that target the MNIST and Fashion-MNIST classification problems with 97.78% and 89.03% test accuracy, respectively. The MNIST model comprises 8 layers and the F-MNIST model comprises 11 layers, both including convolutional, max-pooling and dense layers. A detailed description of the models are provided in Appendix A, see Tables A.2 and A.4.

To setup the Bayesian Network parameters, linear features were extracted using a Principal Component Analysis. The number of extracted features per layer ranged from 3 to 5. For the discretisation, the number of intervals was fixed to 5. Finally, three hidden layers were selected to construct the BN abstractions The considered hidden layers to be abstracted were the convolution ReLU, 2d max pooling, and dense ReLU situated at the second, third, and fifth layers of the MNIST model and the fifth, sixth, and ninth layers for the F-MNIST model.

Results and Discussion. Three distinct BNs were constructed from each CNN model, resulting in BN classifiers with 10, 13 and 16 nodes in total. Proceeding with the experiment, a 10,000 clean test data $\mathbf{d}_{\mathrm{t}} \in \mathbb{D}_{\mathrm{test}}$ was passed to the CNN models and their BNs

[1]The code is available at `https://github.com/AmanyAlshareef/DeepConcolic`.

	# Nodes per CNN layer	Raw data	Adversarial
CNN MNIST	–	97.78%	60.45%
Bayesian Networks	3	82.91%	56.70%
	4	91.66%	61.53%
	5	97.67%	63.83%
CNN F-MNIST	–	89.03%	45.13%
Bayesian Networks	3	71.49%	44.34%
	4	78.16%	46.65%
	5	87.98%	47.89%

Table 5.2: Prediction accuracy of the CNNs models and their abstracted BNs classifiers based on 10000 raw test data and 10000 adversarial samples from the MNIST and Fashion-MNIST datasets. The number of BN nodes is indicated per CNN layer.

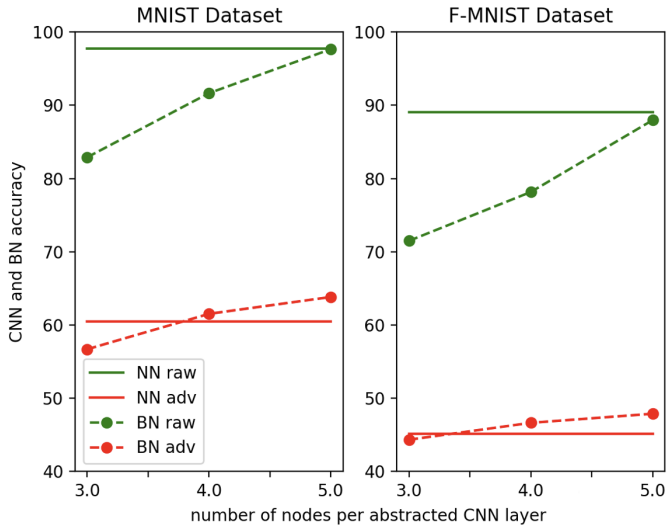


Figure 5.5: Evaluation results under the FGSM attack on the MNIST and Fashion-MNIST datasets. The chart shows that the CNN accuracy is constant, while the BN prediction accuracy is growing with the increased number of nodes. (Plots correspond to Table 5.2)

approximator to measure their accuracy on the test set. The obtained BN prediction accuracy almost reached the CNN accuracy at five extracted features from three CNN layers only. Figure 5.5 illustrates that the BN prediction accuracy grew when the number of nodes increased. The green dashed lines showed the BN prediction accuracy started at 82.91% for MNIST and at 71.49% for Fashion-MNIST with three nodes/features per CNN layer and increased to 97.67% and 87.98% with five nodes, respectively. With these outcomes, the abstracted BN was able to achieve almost the same accuracy as the original neural network.

To further assess the robustness of the DNN abestraction using the BN classifier, the Fast Gradient Sign Method with a maximum of $\varepsilon = 0.1$ perturbation in the input images was selected to craft adversarial samples from the test data $\mathbf{d}_{\mathrm{t}}$. A 10,000 perturbed input data was generated with $\mathbf{d}_{\mathsf{attack},t} = FGSM(\mathbf{d}_t)$ by attacking the CNN models, and then the corresponding projected and discretised activations $\mathbf{r}_{\mathsf{attack}}$ were fed to the BN. Table 5.2 summarises the accuracy results for both CNN and their approximator BN models in both clean and adversarial settings. The BN with five nodes' performance exceeded the CNN robustness accuracy. Although some of the information was lost in the dimensionality reduction and discretisation steps, these empirical results suggested that the abstraction scheme was more robust to attacks performed against the original CNN.

5.6.2 Experiment 2: Connecting All Nodes

Experiment 2 was directed at comparing the accuracy robustness for both a CNN and its abstract BN classifier when fully connecting the prediction node with all BN nodes and calculate its CPT accordingly (see Figure 5.2-(c)). This analysis required the CNN models to be complicated and the BN abstraction to be built from three CNN layers only, to investigate the extent to which the complex deep learning models could be expressed (abstracted) with a simpler model that provided a reasonably similar performance.

Datasets and Experimental Setup. To carry out the experiment, two standard vision benchmark datasets were employed: the MNIST and the CIFAR-10. The identical two trained CNN models from Chapter 4 experiment were used.

To create the BN abstraction, the linear feature extraction technique (PCA) was used along with the uniform discretisation strategy. For the selected CNN layers to be analysed, the same criteria from the previous chapter's experiment were followed. Three hidden

	Tot # of nodes	# Intervals	Clean data	Attack Type		
				FGSM	PGDl2	Deepfool
CNN MNIST	–	–	99.38%	92.67%	91.87%	89.39%
Bayesian Networks	6	3	74.78%	70.88%	72.17%	69.08%
		4	82.55%	71.86%	77.59%	77.86%
	9	3	84.45%	76.92%	78.96%	79.27%
		4	**93.08**%	**84.51**%	**87.32**%	**87.60**%
CNN CIFAR	–	–	81.04%	22.02%	19.8%	22.95%
Bayesian Networks	6	3	39.59%	23.87%	28.47%	29.14%
		4	49.16%	24.55%	30.04%	33.26%
	9	3	51.92%	27.10%	35.46%	38.22%
		4	**63.02**%	**30.50**%	**39.07**%	**40.25**%

Table 5.3: Comparison of prediction accuracies between CNN models (MNIST and CIFAR) and their abstracted BN classifiers based on 10,000 adversarial samples generated for three types of attacks. Significant results are highlighted in bold font.

layers to construct the BN abstractions were considered; for the two models, the first two selected layers directly follow a block of convolutions, while the last was a dense ReLU layer that is situated a few layers before the NN's output layer. In particular, layers named max_pooling2d, max_pooling2d_1, and activation_4 in Table A.3 and A.5.

Results and Discussion. The experiments were conducted by creating various BN combinations from two and three extracted features with three and four intervals for both MNIST and CIFAR-10 models. The prediction accuracy was measured on 10,000 clean test data for all models. In the following text, author will refer to the BN's number of nodes in total (not per CNN's layer) as the prediction node is connected to all of them. Table 5.3 summarised the obtained accuracy outcomes. The BN classifier for the MNIST dataset achieved the highest accuracy 93.08% at nine nodes and four intervals, and for the CIFAR-10, the accuracy was relatively lower at 63.02%. However, considering the number of the un-selected CNN layers (around 12 layers) and their non analysed representation, the performance of the BN with three DNN abstracted layers is deemed reasonable. Moreover, with the exponentially growing of the prediction node's CPT when considering all BN nodes as its predecessor, the largest BN the experiment reached is 9+1 nodes with 4 intervals each. Therefore, the computational cost for any larger BN is too expensive to be run by a CPU.

The robustness of these models was evaluated on more types of adversarial attacks: FGSM, PGD, and DeepFool. The prediction accuracy results are based on 10,000 perturbed

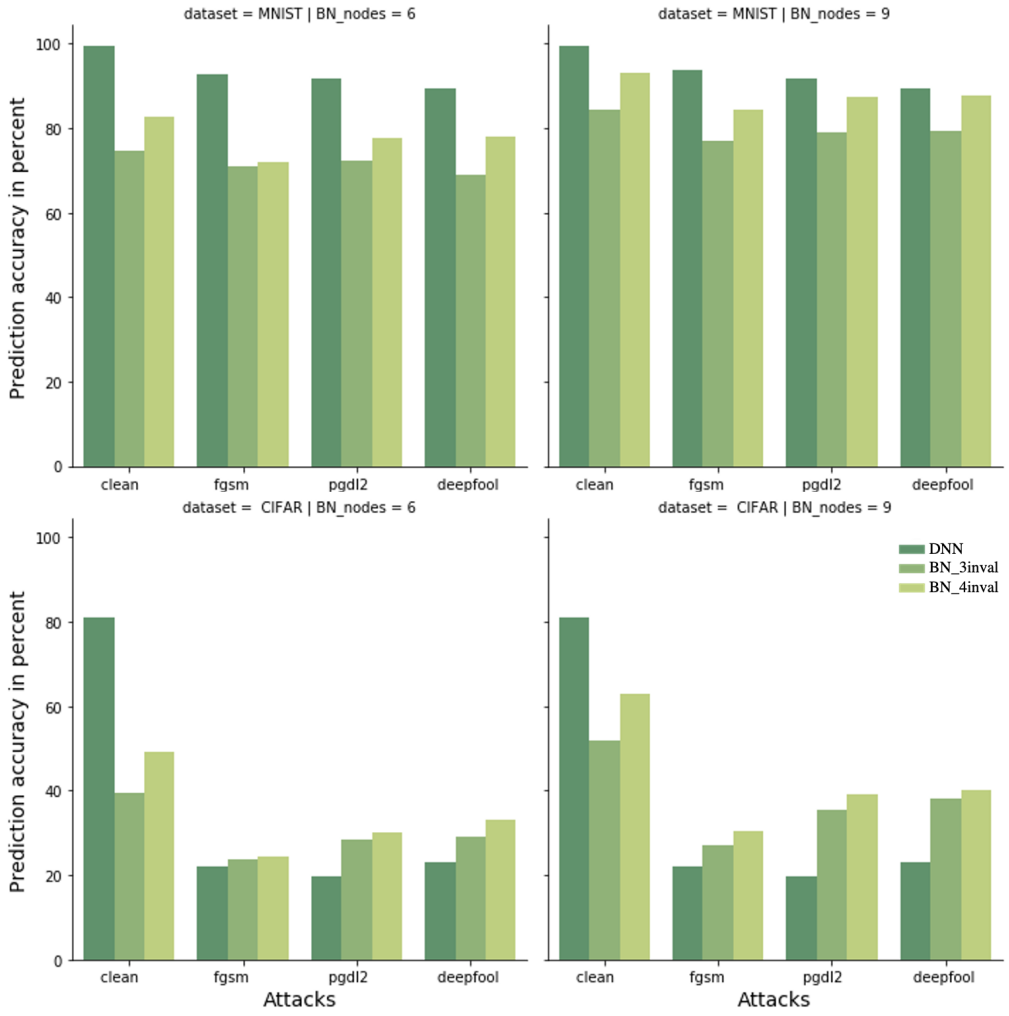


Figure 5.6: Prediction accuracy of the testing (clean) dataset $\mathbf{d}_t$ for neural network models shown with the dark green bar and their Bayesian networks with the different number of node's intervals shown with the rest of the shades of green. The charts also presented the accuracy percentage for adversarial dataset $\mathbf{d}_{\text{attack}}$ generated by CNN with selected attacks (FGSM, PGD, and DeepFool -shown on each column). Each chart clarified the total number of the BN's nodes at the top. Note that the CNN accuracies are similar across the horizontal charts, where the number of nodes only benefits the BN models. The top row shows results for the MNIST and the bottom for the CIFAR-10 model. Hue indicates the number of intervals used to discretise the BN's nodes.

samples generated for each attack type by attacking the original CNN model. Figure 5.6 demonstrates the drop in accuracy for both BN and CNN models. The performance of a very accurate MNIST CNN model was decreased from 99.38% to an average of 90.78% on three attacks - see the dark green bars, while the average dropping in the BN accuracy was 6.5% for the largest constructed BN with nine nodes (plus prediction node) and four intervals - see the lighter green on the top right chart. Moreover, for the CIFAR-10 model, the CNN's accuracy dramatically dropped from 81.04% to 20.00% on average. And even though the BN accuracy on clean images was 63.02% with nine BN nodes, its performance on adversarial attacks was 37.28% on average. Compared to the worst-case CNN prediction with pgdl2 at 19.80%, the BN achieved much better accuracy, reaching 39.07% - see the dark and lighter green on the bottom right chart. Therefore, the results concluded that even in a small BN case, the performance was highly robust to adversarial examples generated by CNN than the CNN itself.

5.6.3 Experiment 3: Feature Weights

The focus of Experiment 3 was centred on determining the impact of perturbing DNN latent features on the BN classification output. This was achieved by using the weighting Equation 5.11 that computed the number of outputs that changed their prediction after the perturbation. The process of calculating feature weights through the utilisation of the BN prediction was conducted on the same two CNN models as used for Experiment 2 above using fully connected BN classifiers. in this experiment, the applied perturbation was generated from the random shift (introduced in Sub-Section 4.3.2), where the concern here was concentrated on each individual feature. For each CNN model, the BN classifier was built using the three identical CNN layers. Two features were extracted from each layer and partitioned into five intervals, and the prediction node connected with all BN nodes. Using Equation 5.11, the resulting feature weights are shown in Table 5.4. Looking at the mnist weights, feature (activation_4, 0) was responsible for the largest number of misclassified samples that were correctly classified by the BN. Followed by feature (max_pooling2d, 0), which ranked as the second-most important feature for the mnist model. While the feature (activation_4, 1) was assigned the highest importance weight of 0.208465 for the cifar-10 model. Figure 5.7 shows the number of the changed decision labels for each feature after the perturbation.

The feature perturbation experiment asserted that each distinct feature had different

perturbed feature	mnist weight	cifar10 weight
(max_pooling2d, 0)	0.213669	0.173404
(max_pooling2d, 1)	0.067358	0.145740
(max_pooling2d_1, 0)	0.159750	0.135429
(max_pooling2d_1, 1)	0.123518	0.132581
(activation_4, 0)	0.246509	0.204380
(activation_4, 1)	0.189196	0.208465

Table 5.4: calculated weight for each BN's feature. First column indicates the perturbed feature name, the second and third column show each perturbed feature estimated weight for MNIST and CIFAR-10 models, respectively.

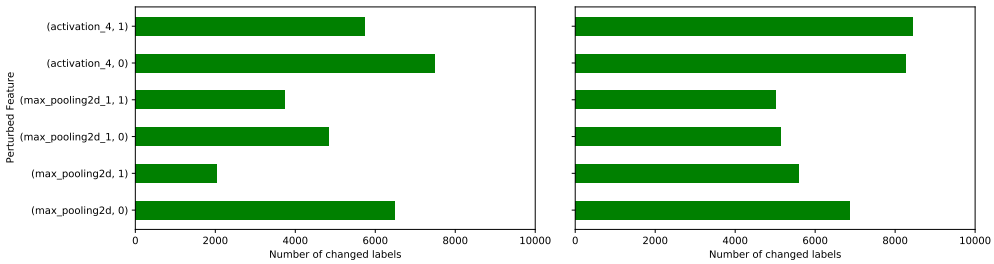


Figure 5.7: Number of the testing samples that are changed their classification labels after the perturbation for each considered feature.

impacts on the classification decision. This supported the definition of the obtained weights from the BN-based feature sensitivity analysis in Chapter 4. The usage of these obtained weights will be discussed in the next chapter.

Note that, when applying perturbation on the BN classifier scheme with the prediction node connected to the deepest nodes only, all nodes not connected with the prediction node did not change the classification decision when exposed to interval-shift. For instance, one example of the experiment output was: *Perturbing feature 0 of layer max_pooling2d_1 ...* 0 *input sample have changed their label, the BN prediction accuracy changed from 73.9% to 73.9%.* This is consistent with the pairwise comparison experiment conducted in Section 4.4, where features' dependence is determined with an edge between them.

5.7 Discussion

The experimental results presented in this chapter demonstrated that the abstracted BN performance in the adversarial setting is more robust than DNN models. It showed that including more nodes in the BN and more fine-grained discretising led to higher BN precision. The repeated experiments also suggested that BN approximates the DNN performance when they are comparable in size (means constructing the BN from most DNN layers), which ensures the BN prediction values robustness is not at the cost of some accuracy loss. On the other hand, scaling the Bayesian prediction model to be fully connected by making the probability tables for the prediction node conditionally dependent on all BN's node distribution seems computationally inefficient. Moreover, the author expects that by enlarging the Bayesian model complexity, one might lose robustness as the model will behave more and more like a DNN. Regardless, the findings emphasised connecting the new node with the deepest BN's nodes and identifying the right level of abstraction by encoding more layers and more components as needed without sacrificing computational complexity or test accuracy. Hence, an optimal complexity which balances performance and robustness can be guaranteed.

Comparing Abstractions The abstraction models discussed in Section 5.2 explored the internal structure of the "black box" learning models with different intentions. The closest method to the proposed approach is the DeepAbstract [5], where the proposed clustering-based model is used to verify the robustness of the original neural network. However, the abstraction itself does not make a prediction. They essentially compute an absolute error produced due to the abstraction and the perturbation of input and then a local robustness verification is performed by checking if an output neuron with a lower or upper bounded value exists. For the monitoring intention, the abstracted BN classifier model can be used as a safety monitor when the DNN is under attack since the BN abstraction is considered more robust and is *harder* to attack due to its discrete nature, where the discretisation function is not trivially differentiable.

Regarding the approximator models, they are designed to approximate an opaque model's accuracy and make prediction decisions. The KF Laplace approximation model in [79] is tested under the Fast Gradient Sign method attack on the MNIST dataset to asses its prediction robustness. The authors demonstrated that their approximator model happens to be highly robust to that sort of attack. However, their investigation of the robustness of

their approximator was a minor matter where the main concern was uncertainty estimates obtained from dropouts.

5.8 Conclusion

This chapter presented a method for constructing a classifier based on an abstract Bayesian network from a neural network to make predictions. The BN prediction with feature perturbation experiments demonstrated that each feature changed the classification decision by a different percent, which validates the feature importance assumption. In addition, comparing the accuracy of both models DNN and BN confirmed that the *abstracted Bayesian network classifier* is a good approximator of the DNN. Since the created classifiers are abstractions of the original neural network, the results enable the construction of meaningful abstractions with respect to the acceptable requirement on the accuracy and robustness.

Chapter 6

Weight-based Testing Metrics

6.1 Introduction

The high-level goals of software testing are to provide evidence that a system meets its requirements and proving that it is error-free. The fact that deep learning models are data-driven, not requirements-driven, makes defining their testing criteria challenging. Technically, the accuracy of the learning models is reported based on the test dataset. This standard metric for measuring the model's overall performance can not be sufficient or trustworthy in safety-related domains, where most testing scenarios are randomly chosen from the entire dataset. Besides, the provided test data may not have good coverage of the data distribution the model is trained on, and may not represent the data obtained in the real world. In that case, the quality of the test data is the essential factor influencing the acceptance of the accuracy metric of the evaluated model.

Furthermore, most of the current proposed DNN's testing techniques rely on neuron activation as a metric to measure the test data coverage, which corresponds to the code coverage of traditional systems. Such criterion does not prove a correlation to the system's decision logic. Moreover, these methods aim to transform the input data space to generate more test input and completely ignore the model-internal representations and their roles in the output decision. Knowing that real-world high-dimensional data usually lie on much lower-dimensional manifolds motivates investigating where the data is located and modeling it instead of narrowing in the input domain. There is little attempt to understand the machine learning's hidden representations and generate additional test cases based on that. The majority of feature relevance studies were presented for explainable machine learning

and interpretability.

In this chapter, the traditional binary coverage approach is transformed into a weighted probability problem that defines the coverage metric based on latent features importance. The Subsidiary Research Questions (SRQs) that this chapter seeks to address are:

1. **SRQ4:** Do existing testing metrics guarantee the coverage of a model's critical internal regions, as well as direct the test case generation algorithm to target the most relevant features?

2. **SRQ5:** Do the proposed coverage metrics deliver a reliable testing measurement in terms of reporting the coverage that prioritises the important internal representation of the model?

3. **SRQ6:** Does the generated test dataset from the feature weight directed concolic testing provide a *trustworthy* measure of a model's performance?

To retrieve the broad perspective and draw lines between preceding chapters, here is a synopsis of the main principles. The work explained in chapter 4 quantified the importance of latent features in neural networks. It defined the Feature Importance (FI) measure as the degree to which the internal representation contributes to the output prediction. The importance of a neural network's latent features is estimated by analysing the associated Bayesian network's sensitivity to distributional shifts. Various metrics were utilised to compute the difference between the original and perturbed BN probabilities. Each feature was assigned a weight value based on the measured sensitivity distance. To further justify the importance weights and demonstrate their roles in the classification decision, the BN model was enhanced to predict a new input's label. In Chapter 5, a methodology was introduced that extends the capabilities of the Bayesian Network by enabling it to function as a classifier, facilitating prediction through the use of probabilistic inference. The adopted approach was to add an auxiliary prediction node at the end of the Bayesian network and calculate its conditional probability table based on the BN nodes' distribution. With the BN's prediction ability, the described importance weights calculated based on the sensitivity distances between two BN's probability distribution showed their impact on the classification decision. The obtained weight scores provide insight into what region of the data set should be well represented in the test data set for a given model.

This chapter presents a testing approach that leverages the learned representations and feature weights to evaluate the coverage of a test dataset. The coverage criteria are based

on the hypothesis that the contributions of latent features to a model's classification task represent the most reasonable basis for informing the model's performance, where these are considered high-risk features that should be tested thoroughly. Examining the feature weights reveals the network's internal decision mechanism and how it processes the input data. This provides valuable information for identifying the responsible feature contributing to incorrect predictions and any biases or limitations in the network's representation. From that perspective, the weight-based testing criterion emphasises that maximum test coverage will be obtained from the presence of important features that have a dominating influence on other features and the output decision. Furthermore, the acquired weights will be used as guidance for the generation of test samples from the feature space. It is argued in this chapter that a semantic testing method, based on a model's behaviour, provides assurance evidence for the trustworthy and robust behaviour of the model. To summarise, the main contributions of this chapter are:

- A number of semantic testing metrics that measure a test dataset's coverage quality according to the calculated feature weights.

- A guided systematic approach to sample additional test cases targeting the most important features.

- An empirical study regarding the quality of the proposed weight-based coverage compared with the original BN-based coverage.

The structure of the remainder of this chapter is as follows. Section 6.2 presents the process for constructing the weighted feature model, where the three preceding chapters are linked together to build a BN abstracted model with associated weight for each node. In the following section, Section 6.3, a definition and detailed description of the proposed semantic testing metrics are presented. The test case generation algorithm using Concolic testing is detailed in Section 6.4. An evaluation of BN feature coverage and the proposed weight feature coverage metrics is presented in Section 6.5. Section 6.6 provides a discussion on how the proposed weight testing metrics can improve the overall model reliability and trustworthiness. In Section 6.7, a recent work introduced a related concept of importance based testing coverage for DNN is discussed and compared to the weight testing metrics. The last section, Section 6.8, concludes the work presented in this chapter.

6.2 The BN Weighted Feature Model

Preliminaries. The goal of the weighted features model is to construct an abstracted Bayesian network that includes the importance weight for each node of the BN.

As previously stated, the Bayesian Network $\mathcal{B}_{\mathcal{N},X} = (V, E, P)$ is an abstracted model from the DNN $\mathcal{N}$ and training dataset X. The V are nodes containing the extracted latent features from the $\mathcal{N}$, each feature is defined as a pair $f_{i,j}$. The E are directed edges indicating dependencies between features in successive layers, and P maps each node in V to a probability table representing the conditional probability of the current feature over its parent features w.r.t. X. The reason to construct such a layered structure of a Bayesian network is to simulate the structure of DNN, where features from layer l_i are influenced only by features in layer l_{i-1}. In other words, an assumption of connecting layer l_i directly, with some weights, to layer l_{i-1} only is taken into account (there are no skip connections, or similar). Therefore, to represent a DNN fully, it is enough to connect the BN nodes between features of the two consecutive DNN layers.

The feature sensitivity analysis process introduced in Chapter 4 (refer to Figure 1.1-step1 for an overview), was calculated based on the change in the BN's probability distribution as follows:

$$S_{i,j} = \frac{\delta(P_{ref}, P'_f)}{\sum_{f \in \mathbb{F}^\sharp} \delta(P_{ref}, P'_f)} \tag{6.1}$$

Where S_{ij} is the sensitivity weight of the feature $f^\sharp_{i,j}$, P_{ref} is the original (reference) probability distribution represented by the BN, P'_f is the probability after perturbing $f_{i,j}$, and δ is a function returning the distance between two probabilities distribution according to a given metric d_p. $\mathbb{F}^\sharp$ is the set of considered latent features.

The method for extracting features weights in the BN probabilistic inference, presented in Chapter 5 (refer to Figure 1.1-step2 for an overview), was computed based on the change in the BN prediction outputs as follows:

$$W_{i,j} = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{\mathbf{x} \in \mathcal{D}_{\text{test}}} \mathbb{1}\left[P_{\text{BN}}(\mathbf{x}) \neq P_{\text{BN}}\left(\mathbf{x} \wedge \text{pert}(f^\sharp_{i,j})\right)\right], \tag{6.2}$$

Where $W_{i,j}$ represents the importance weight of the feature $f^\sharp_{i,j}$, $P_{\text{BN}}(\mathbf{x})$ is the original prediction of the Bayesian network for input $\mathbf{x}$, $P_{\text{BN}}\left(\mathbf{x} \wedge \text{pert}(f^\sharp_{i,j})\right)$ is the BN prediction for input $\mathbf{x}$ after perturbing feature $f^\sharp_{i,j}$, $\mathcal{D}_{\text{test}}$ is the set of testing data, and function

$\mathbb{1}$ returns 1 if the prediction changed or 0 otherwise. The previous equation represents unnormalised weights, which is further normalised so that $\sum_{i,j} W_{i,j} = 1$.

The idea to combine the two weights is to use the BN prediction weight from Equation 6.2 to decide the best distance metric that produces a sensitivity weight in Equation 6.1 highly correlated with it. The reliance will be on the sensitivity weights, because the proposed testing metrics are essentially working with the feature space, so it is more reasonable to use the weights calculated based on the BN distribution rather than the BN prediction. Thus, the W_f is obtained, which is a vector of the correlated normalised weights for the extracted latent features from the DNN.

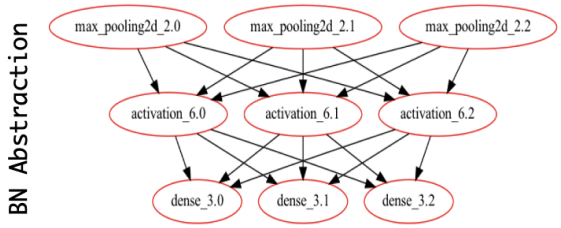
Definition 6.2.1 (Weighted Feature Model)**.** *A weighted feature model over a hidden DNN's features $\mathbb{F}^\sharp$ is a function $f{:}\,\mathbb{F}^\sharp_{i,j} \rightarrow \mathbb{R}^{\geq 0}$ that maps features into their importance values according to the firstly matched sensitivity weights.*

The function iterates over each considered distance metric, sorts its calculated distances, and then compares their rankings to the BN prediction weights rank. By computing the weighted average over features, we make an assumption that the features can be de-couple one from another. This assumption is possible since the BN-based weight feature coverage metric concentrates on each individual hidden feature interval in isolation.

Example 17. *Figure 6.1 shows a Bayesian Network constructed from three selected layers of a CNN: max_pooling2d_1, activation_6, and dense_3. The activation values are computed for each considered layer, then the dimensionality reduction is applied, and three feature components are produced that are discretised into five intervals. The figure illustrates sensitivity weights of the latent features obtained using various distance metrics. The features are sorted for each distance metric and their importance rank compared with the calculated weights from the BN prediction step. The comparison results demonstrate that the BN prediction weights correlated with d_L2, d_corr, d_RMSE, d_MSE, and d_AF distances shown with red rectangles. The example suggests that the third extracted feature from layer activation 6 has the highest importance score.*

6.3 Weight-based Semantic Testing

This section provides a detailed technical description of the proposed weight-based semantic testing metrics and algorithm. The section introduces three testing metrics. In



distances perturbed feature	d_L2	d_L∞	d_JS	d_Wass1	d_corr	d_cos	d_RMSE	d_MAE	d_AF
(max_pooling2d_2, 0)	0.074784	0.110878	0.087355	0.111071	0.063686	0.069357	0.111033	0.111073	0.056730
(max_pooling2d_2, 1)	0.117755	0.111138	0.109214	0.111083	0.112240	0.110499	0.111125	0.111114	0.120953
(max_pooling2d_2, 2)	0.092172	0.111144	0.099690	0.111079	0.082057	0.084531	0.111076	0.111092	0.077190
(activation_6, 0)	0.120717	0.111142	0.126169	0.111135	0.131510	0.127733	0.111130	0.111121	0.127265
(activation_6, 1)	0.115556	0.111138	0.125377	0.111122	0.109105	0.115235	0.111121	0.111121	0.116449
(activation_6, 2)	0.124626	0.111144	0.121799	0.111125	0.136390	0.137618	0.111137	0.111123	0.136042
(dense_3, 0)	0.120537	0.111135	0.114257	0.111129	0.130693	0.129624	0.111130	0.111120	0.126873
(dense_3, 1)	0.116058	0.111144	0.107665	0.111129	0.110986	0.106896	0.111122	0.111116	0.117464
(dense_3, 2)	0.117794	0.111137	0.108475	0.111127	0.123333	0.118508	0.111125	0.111118	0.121034

Compare

perturbed feature	weight
(activation_6, 2)	0.138963
(activation_6, 0)	0.129012
(dense_3, 0)	0.126149
(dense_3, 2)	0.125392
(max_pooling2d_2, 1)	0.111288
(dense_3, 1)	0.101901
(activation_6, 1)	0.092517
(max_pooling2d_2, 2)	0.088675
(max_pooling2d_2, 0)	0.086104

Figure 6.1: Example of a computed distances from the feature sensitivity weighting method shown in the first table annotated with the used distance metrics at the header. The second small table is the supportive normalised weights calculated via the BN prediction and sorted in descending order. The red rectangles indicate the correlation with the BN prediction weights. The diagram also shows the used structure of the Bayesian Network created from three selected CNN's layers and three extracted features from each layer.

Sub-Section 6.3.1, the Weight-based Feature Coverage metric is defined; an equation is formalised and an example is given. The second metric, named Weight-based Feature Dependence Coverage, is described in Sub-Section 6.3.2, with an example. Sub-Section 6.3.3 explains how the two previous metrics are combined to deliver the third testing metric called Generalised Weight Feature Coverage. The final subsection, Sub-Section 6.3.4, provides definitions of the coverage criteria for the suggested testing metrics.

6.3.1 Weighted Feature Coverage

The BN abstraction and its hidden feature weights are utilised to develop new coverage metrics that assess the quality of a test dataset in terms of reporting the coverage based on the non-uniform contribution theory. That is, the metrics focus on the semantic values of the neuron activation instead of the syntactic values of the adjusted weights, which is a very local and less decisive criterion. The weight-based metrics indicate that a more critical feature can take up more coverage share than a less important feature.

Definition 6.3.1 (Weight-based Feature Coverage)**.** *Given a trained DNN* $\mathcal{N}$*, the* weight-based feature coverage *of a* non-empty *set of inputs* $X \subset \mathbb{D}_X$ *is obtained via the BN*

abstraction $\mathcal{B}_{\mathcal{N},X}$ *as*

$$\text{WFCov}(\mathcal{B}_{\mathcal{N},\mathcal{X}}) \stackrel{def}{=} \sum_{(\!|f^{\sharp}_{i,j}|\!)\in V_{\mathcal{N},X}} w_{(\!|f^{\sharp}_{i,j}|\!)} \cdot \frac{\left|\left\{f^{\sharp k}_{i,j} \in \mathbb{F}^{\sharp}_{i,j} \mid \mathcal{P}_i\left(f^{\sharp k}_{i,j}\right) \geq \varepsilon\right\}\right|}{\left|\mathbb{F}^{\sharp}_{i,j}\right|}, \tag{6.3}$$

where $\sum_{(\!|f^{\sharp}_{i,j}|\!)\in V_{\mathcal{N},X}} w_{(\!|f^{\sharp}_{i,j}|\!)} = 1$.

The coverage metric in the equation above ranges over $[0, 1]$, and gives the weighted proportion of feature intervals that are adequately exercised by X. The $\text{WFCov}(\mathcal{B}_{\mathcal{N},\mathcal{X}})$ is similar to the basic feature coverage in Equation 3.3.1, where the factor $1/\left|V_{\mathcal{N},X}\right|$ that acts as an *equals* weight for all features is replaced with the computed importance weight. Note that, since the feature coverage metrics concentrate on each individual hidden feature interval, and the weights are computed per feature, the weights are implicitly divided equally between their intervals. The WFCov measure cannot be null if ε is sufficiently small, since the sum of all the entries of the probability tables is always one.

Example 18. *Consider the Bayesian Network shown in Figure 6.2. For layer* l_3*, we can compute the following marginals based on the given conditional probability table for the node* $f_{3,0}$ *as:* $\Pr(f_{3,0} < 3) \approx 0.453$*,* $\Pr(3 \leq f_{3,0} \leq 5) \approx 0.323$*,* $\Pr(f_{3,0} > 5) \approx 0.224$*. Then the sum of intervals' marginals is multiplied with the node weight which results in* $1 \times 0.173 = 0.173$*. Assuming similar non-negligible marginal probabilities for the nodes pertained to layers* l_1 *and* l_2*, then we obtain each node coverage* $3/3$ *and then multiply it with its per-node weight to obtain the coverage of the test set with the* $\text{WFCov}(\mathcal{B}_{\mathcal{N},\mathcal{X}}) = 1$*.*

6.3.2 Weighted Feature Dependence Coverage

The causal relationships exercised by a dataset X, that the BN's conditional probabilities define, are used to develop the following coverage metric:

Definition 6.3.2 (Weight-based Feature Dependence Coverage)**.** *Given a trained DNN* $\mathcal{N}$*, the* weight-based feature dependence coverage *of a* non-empty *set of inputs* $X \subset \mathbb{D}_X$ *is*

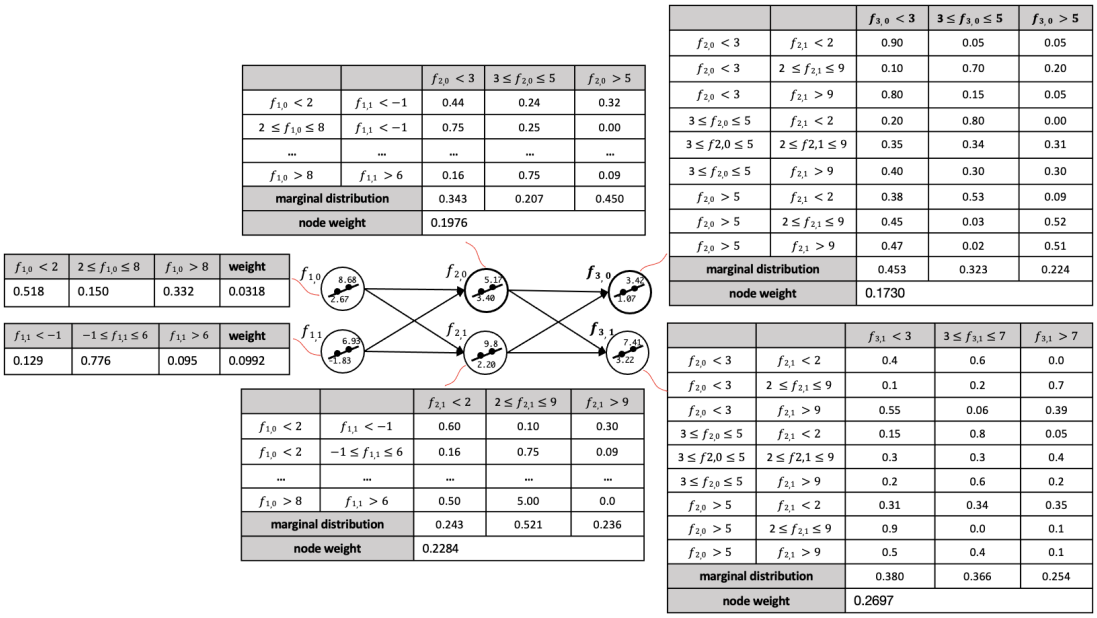


Figure 6.2: An abstracted Bayesian network from three DNN's selected hidden layers. Two features are extracted from each layer and discretised into three intervals. The features $f_{1,0}$ and $f_{1,1}$ have marginal tables. Features $f_{3,0}$ and $f_{3,1}$ are illustrated with a complete conditional probability table, while other CPTs have the same length (m^p number of intervals to the number of parents), but are shortened in the diagram. The weight column shows per-node probability.

obtained via the BN abstraction $\mathcal{B}_{\mathcal{N},X}$ *as*

$$\mathrm{WFdCov}\left(\mathcal{B}_{\mathcal{N},X}\right) = \sum_{\langle\!\langle f_{i,j}^{\sharp} \rangle\!\rangle \in V_{\mathcal{N},X}^{+}} w_{\langle\!\langle f_{i,j}^{\sharp} \rangle\!\rangle} \cdot \frac{\left| \begin{array}{c|c} (f_{i,j}^{\sharp k}, F_{i-1}^{\sharp}) \in & \mathcal{CP}_i\left(f_{i,j}^{\sharp k} | F_{i-1}^{\sharp}\right) \ge \varepsilon \\ \mathbb{F}_{i,j}^{\sharp} \times \mathbb{F}_{i-1}^{\sharp} & \vee \quad \mathcal{P}_i\left(f_{i,j}^{\sharp k}\right) < \varepsilon \end{array} \right|}{\left| \mathbb{F}_{i,j}^{\sharp} \times \mathbb{F}_{i-1}^{\sharp} \right|}. \tag{6.4}$$

where $\sum_{\langle\!\langle f_{i,j}^{\sharp} \rangle\!\rangle \in V_{\mathcal{N},X}^{+}} w_{\langle\!\langle f_{i,j}^{\sharp} \rangle\!\rangle} = 1$.

Here, $V_{\mathcal{N},X}^{+}$ represents a set of nodes excluding the first abstracted layer, for which a conditional probability table does not exist. The $\mathrm{WFdCov}\left(\mathcal{B}_{\mathcal{N},X}\right)$ gives the weighted percentage of assumed causal relationships between features of successive layers that are adequately exercised by X. Intuitively, the metric iterates over all nodes in $V_{\mathcal{N},X}^{+}$ and calculates a weighted coverage. For each node, it looks at its CPT which lives in the space

$\mathbb{F}_{i,j}^{\sharp} \times \mathbb{F}_{i-1}^{\sharp}$, and checks all the values larger than ε. In other words, $\mathcal{CP}_i(f_{i,j}^{\sharp k}|F_{i-1}^{\sharp}) \geq \varepsilon$. Further, for the metric to be independent from the previous feature coverage, it includes the conditional probability entries pertaining to hidden feature intervals that are not elicited by X, i.e., for which $\mathcal{P}_i(f_{i,j}^{\sharp k}) < \varepsilon$. Consequently, the impact of feature coverage is eliminated on feature-dependence coverage, so that the WFdCov $(\mathcal{B}_{\mathcal{N},X})$ only captures the causal relationships between hidden features that are not exercised by X,

Example 19. *Continuing Example 18, the weighted feature dependence coverage is now considered. The function iterates over the last* 4 *of the* 6 *nodes for which the CPT exists. For this example, let the calculated coverage for the node* $f_{3,0}$*,* $\varepsilon = 0.01$*. Taking a look into its CPT, there are* 26 *out of* 27 *items with probabilities larger than* 0.01*. Furthermore, all marginal probabilities are also larger than* 0.01*, which means that the coverage is* 26/27*. Now, we calculate how much the node amount to the total weighted feature dependence coverage. Because the weights in Figure 6.2 are normalised to sum to* 1 *for all nodes, they have to be firstly renormalised, so that only weights of nodes with CPT sum of* 1*. Thus, the normalisation constant is the sum of all nodes' weights except for the first layer's node weights. This amounts to* 0.8687*. Finally, the contribution of the node* $f_{3,0}$ *to the total coverage amounts to* $0.1730/0.8687 \cdot 26/27 = 0.1917$*. Similarly, for the node* $f_{3,1}$*, there are* 25 *out of* 27 *values with a probability larger than* 0.01*, which means the node will contribute to the total coverage as* $0.2697/0.8687 \cdot 25/27 = 0.2875$*. Summing up the contribution from all* 4 *nodes with CPT assuming there is one probability less than* 0.01 *for each* $f_{2,0}$ *and* $f_{2,1}$*, the final weighted feature dependence coverage amounts to* $0.2190+0.2532+0.1917+0.2875 = 0.9514$*.*

6.3.3 Generalised Weighted Feature Coverage

To deliver a consistent coverage measure that is based on every probability entry in the BN, the two feature metrics 6.3.1 and 6.3.2 can be combined to produce the generalised weight feature coverage. This generalised weighted feature metric gives a single, unified coverage. The capability of the BN abstraction can consider coverage that takes into account higher-order distributions as well. The following is a short description of the possibilities in this respect. In the simplest approach, one can consider two coverages decoupled from each other, and simply multiply them: WFCovTot = WFCov $\times$ WFdCov. This is in most

situations sufficient, however the other possibility is to average them on per-node level:

$$\mathrm{WFCovTot}\left(\mathcal{B}_{\mathcal{N},X}\right) = \sum_{(\!|f^\sharp_{i,j}|\!)\in V_{\mathcal{N},X}} w_{(\!|f^\sharp_{i,j}|\!)} \cdot \begin{cases} \mathrm{WFCov}_{(\!|f^\sharp_{i,j}|\!)} & \text{if } i = 1\,, \\ \frac{1}{2}\left(\mathrm{WFCov}_{(\!|f^\sharp_{i,j}|\!)} + \mathrm{WFdCov}_{(\!|f^\sharp_{i,j}|\!)}\right) & \text{otherwise}\,. \end{cases} \tag{6.5}$$

Here the $\mathrm{WFCov}_{(\!|f^\sharp_{i,j}|\!)}$ and $\mathrm{WFdCov}_{(\!|f^\sharp_{i,j}|\!)}$ are labelled with per-node coverages.

Further coverage metric. In the realm of probabilistic modelling, we can consider the coverage for the full probability distribution. Where the metric does not take into account only marginal and first-order conditional distributions, but also higher order correlations set by the BN. For example, we can calculate a joint probability as

$$P(f^\sharp_{1,1}, f^\sharp_{1,2}, \ldots f^\sharp_{1,|\lambda_1|}, \ldots f^\sharp_{i,j}, \ldots f^\sharp_{K,|\lambda_K|})\,, \tag{6.6}$$

iterate over all probability space and define the coverage as the fraction of them larger than some threshold ε. Since the per-node weights in such a case cannot be trivially introduced, this theoretical metric will be left for future work.

6.3.4 Coverage Criteria

The following test criteria are trivially derived from the coverage metrics given in Definition 6.3.1 and 6.3.2:

Definition 6.3.3 (Weight-based Feature Coverage Criterion)**.** *A non-empty set of inputs $X \subset \mathbb{D}_X$ satisfies the* Weight-based feature coverage criterion *that is obtained via the BN abstraction $\mathcal{B}_{\mathcal{N},X}$ iff* $\mathrm{WFCov}(\mathcal{B}_{\mathcal{N},\mathrm{X}}) = 1$.

Definition 6.3.4 (Weight-based Feature-dependence Coverage Criterion)**.** *A non-empty set of inputs $X \subset \mathbb{D}_X$ satisfies the* Weight-based feature-dependence coverage criterion *that is obtained via the BN abstraction $\mathcal{B}_{\mathcal{N},X}$ iff* $\mathrm{WFdCov}(\mathcal{B}_{\mathcal{N},\mathrm{X}}) = 1$.

6.4 Concolic Test Generation

The weight-based feature metrics are implemented on the DeepConcolic tool and the features weights are used as criteria to direct the concolic testing algorithm. A detailed description of the test generation procedure is provided in Algorithm 2.

For a given trained DNN $\mathcal{N}$ on a dataset X and the associated abstract Bayesian Network $\mathcal{B}_{\mathcal{N},X}$, the features weights W_f are calculated for all extracted features. We assume that suitable feature extraction and discretisation have been applied on a training sample X_{train} to obtain the structure of the $\mathcal{B}_{\mathcal{N},X}$. The test generation procedure starts by randomly sampling an initial seed set of test inputs X_0 from X_{test} data set that is correctly classified by $\mathcal{N}$, and initialising the probability tables in the BN to produce $\mathcal{B}_{\mathcal{N},X_0}$. Next, the algorithm identifies the test target intervals $Tar\_invals = \{f^{\sharp}_{i,j}\}$ through analysing the non-epsilon probabilities of the marginal or conditional probability tables in $\mathcal{B}_{\mathcal{N},X_0}$. The non-epsilon is the probabilities that are less than ε and not yet met by the current set of input test cases in X_0. Thus, the Tar_invals consist a set of hidden feature interval(s) that should be elicited by the test input to be generated.

The test case generation algorithm then iterates i times according to the following: First, identify the $t \in Tar\_invals$ with the highest importance weight in W_f, and select a test input $s \in X_0$ based on some heuristics, such as closeness to the targeted interval t. Then, constructing a Linear Programming (LP) problem based on t and solve the optimisation objective that seeks to minimise the distance between activation of input neurons and s. This problem is formulated as:

$$\text{Minimise: } \|(\hat{n}_{1,1}, \ldots, \hat{n}_{1,|l_1|}) - (s_{1,1}, \ldots, s_{1,|l_1|})\|_{\infty} \tag{6.7}$$

Where $n_{1,1}, \ldots, n_{1,|l_1|}$ is the set of all input neurons in $\mathcal{N}$.

After solving the LP problem and extracting the newly generated test input s' from values of input neurons: $s' = (n_{1,1}, \ldots, n_{1,|l_1|})$, the algorithm will check two properties of the new input s'. Does the s' pass the oracle, i.e., is it structurally close enough to s w.r.t the L_∞ norm? If yes, then, does the s' output the same classification label of s, in other words, is $f_{\mathcal{N}}(s') == f_{\mathcal{N}}(s)$? If yes, then, the s' is considered a valid input and added to the test input $X_0 = X_0 \cup \{s'\}$. Otherwise, the s' is considered *adversarial* for $\mathcal{N}$, as s' is both deemed close enough to s from which it is derived, and it is not assigned the same classification label as s by $\mathcal{N}$. Accordingly, the probabilities in $\mathcal{B}_{\mathcal{N},X_0}$ are updated to account for the new test s' and then recalculate the coverage. The test case generation continues if the test criteria obtained via the new $\mathcal{B}_{\mathcal{N},X_0}$ is not yet satisfied.

Note that, the new test s' may not actually improve reported coverage if it is just "closer" to the target interval than s but does not hit it. The expectation is that, s' will later be selected to generate a new input s'' according to the same process, and eventually

Algorithm 2 Test Dataset Generation

Input:
$\mathcal{N} \leftarrow$ DNN under test
$X \leftarrow$ data set
$\mathcal{B}_{\mathcal{N},X_{train}} \leftarrow$ abstract Bayesian network
$W_f \leftarrow$ features sensitivity weights
Output: test inputs X_0, coverage

1: $X_0 \leftarrow$ sampling initial seed test inputs from X_{test}
2: $\mathcal{B}_{\mathcal{N},X_0} \leftarrow$ initialising the BN probability tables with X_0
3: $Tar\_invals \leftarrow$ intervals with prob $\leq \varepsilon$
4: **for** $i = 1$ to max iterations **do**
5: $t \leftarrow Tar\_invals$ with highest weight in W_f
6: select a test input $s \in X_0$
7: construct an LP problem based on t
8: solve the optimisation objective: min $\|(\hat{n}_{1,1}, \ldots, \hat{n}_{1,|l_1|}) - (s_{1,1}, \ldots, s_{1,|l_1|})\|_\infty$
9: $s' = (n_{1,1}, \ldots, n_{1,|l_1|})$
10: **if** s' passes the oracle **then**
11: $s' \leftarrow$ newly generated test input
12: **if** $f_{\mathcal{N}}(s') = f_{\mathcal{N}}(s)$ **then**
13: $X_0 \leftarrow X_0 \cup \{s'\}$
14: update $\mathcal{B}_{\mathcal{N},X_0}$ probabilities
15: update coverage
16: **else**
17: $s' \leftarrow$ adversarial input
18: **end if**
19: **end if**
20: **end for**

the target interval might be reached. Further detail is provided for the main steps from the algorithm above, as follows:

- **Selection of a test input.** Finding a candidate test input s is implemented through searching for an input $s \in X_0$ whose feature value $\lambda_{i,j} \circ \hat{h}_i$ is close to the target interval boundaries $f_{i,j}^{[k}$ or $f_{i,j}^{k[}$. This assumption gives a simple heuristic for finding a good-enough candidate test input.

- **Construction of the LP problem and the set of constraints.** The problem that is solved at each symbolic analysis iteration of DeepConcolic is a Linear Programme (LP) as described in *Concolic testing* in Section 2.5.2. The construction of the LP

problem comprises a set of constraints to find a set of inputs s' that exhibit the same activation pattern behaviour as s, and an optimisation objective to find the optimal s', if one exists, using Chebyshev distance L_∞. The concolic test generation with weight guidance enhancement reuses the constraints suggested by Sun et al. [90] to construct a symbolic encoding of the DNN's layer behaviours and construct a set of constraints to encode ReLU operations.

- **Evaluation of the generated test cases.** To fulfil the BN-based criteria that is linearly encoding non-linear behaviours, the concolic algorithm uses a practical measure beside the test oracle that decides whether a test case passes or fails. The creation of this measurement is to counter a consequence of the loss of precision that is induced by dimensionality reduction. The decision mechanism to keep or reject any newly generated input is that, even if the resulting LP solution of a candidate test input s leads to a hidden feature valuation $\lambda_{i,j} \circ \hat{h}_i(x')$ that does not belong to $f_{i,j}^{\sharp k}$, due to the approximations induced by the feature mapping $\lambda_{i,j}$, the input s considers a legitimate new test cases. As stated before, even though such inputs do not improve coverage, they help populating the set of potential candidate tests inputs, which they indeed help further explorations of the input space.

6.5 Evaluation

This section reports on the experimental analysis conducted to evaluate the performance of the suggested coverage metrics and the usability of the weight in guiding the adapted concolic test case generation. The first set of analyses examined the quality of the existing BN-based feature metrics discussed in Section 2.5. Then, the efficiency of the developed features weights was tested and compared to the previous coverage results. The answers are provided for the first two investigated research questions and justified.

6.5.1 Datasets and Models

Two popular datasets, Fashion-MNIST [96] and CIFAR-10 [54], were chosen for the experiments. A medium-sized Fashion-MNIST model (named $\mathcal{N}_{fm}$ whose layers are listed in Table A.4) is trained with 89.03% validation accuracy. Three different layers with various functionality are chosen for the testing to fairly cover all types of the layers. The chosen layers are max_pooling2d_1, activation_2 and dense_1. For further exploration, the testing

criteria are also evaluated on a more complicated CNN model $\mathcal{N}_{ci}$ trained on the CIFAR-10 dataset with 81.00% validation accuracy, the model is illustrated in Table A.5. The selected layers to be covered with the experiments are: max_pooling2d_1, activation_4 and dense_1.

6.5.2 Experimental Setup

In the following experiments, the high-level criteria are used to investigate how a test dataset exercises a set of hidden features that has been learned from the training dataset and internally represented by any layer of the CNNs. Therefore, the reliance will be placed on the hidden features learned by the two trained CNN models (the Fashion-MNIST model and the CIFAR-10 model). The computation of the Bayesian Network abstraction relied on 20,000 classified training samples. Multiple strategies for linear dimensionality reduction and discretisation of each feature component were applied to construct various BN abstraction schemes. Two linear feature extraction techniques were selected: Principal Component Analysis (PCA) and Independent Component Analysis (ICA), with varying numbers of components to be extracted from the set of neuron activations at each chosen layer. Particularly, the extracted features range from two to five per layer. The Density-based (KDE) and uniform-based discretisation strategies were considered, with varying numbers of the uniform partitions bins that are: one, three, and five. The KDE density estimation was restricted to partitioning the features into three minimum and five maximum intervals. Finally, the extended Concolic testing tool was run on both DNN models with a maximum of 100 iterations per run. Each run was initialised with uniformly drawn test sets X_0 of 10 and 100 correctly classified inputs. The chosen epsilon value to compute the feature coverage was $\varepsilon = 10^{-8}$. Regarding the weight coverage experiments, the procedure for determining a distance metric to calculate the sensitivity weights through comparison with the BN prediction weights is executed once during the first iteration. Thereafter, the sensitivity analysis algorithm will rely on the same distance metric to compute the feature weights for the remaining iterations.

6.5.3 Experimental Results and Analysis

The experiment results obtained from the testing analysis conducted to evaluate the performance of the proposed weight metrics are presented. There were two evaluation objectives, as follows: The purpose of the first experiment was to evaluate the coverage quality of the

existing BN-based metrics so that a comparison with the proposed weight metrics could be made and improvements could be recognised. The second experiment aimed to determine the enhancements of the proposed weight feature coverage. In particular, demonstrating the efficiency of targeting the most important features throughout the process of generating new test cases.

SRQ4: Coverage Quality Analysis Using Existing Metrics. The aim of the SRQ4 experiment was to assess the extent to which the existing BN-based feature metrics discussed in Sub-Section 3.3.4 could improve the initial coverage with the test generation within a maximum 100 iterations. To analyse the testing outcomes, it was necessary to carefully select and decide how to split different categories. Since the objective of the experiment was to demonstrate the increase in coverage over the run time, the primary variables will be the initial coverage, the final coverage, and the time it takes to obtain the final coverage. So, there were two numerical parameters: run-time and coverage. Other parameters were categorical: initial or final; ICA or PCA; initial test sizes. Therefore, plotting the result in space of run-time vs. coverage, and having one error point representing each categorical class - initial PCA, initial ICA, final PCA and final ICA, would illustrate the desired intention. For each of those variables, the errors were calculated in the following way:

- For the run-time, standard normal error is expected, so the mean and one standard deviation are calculated. This amounts to 68% interval around the mean.
- For coverage, however, distribution is neither normal nor symmetric. Therefore, median and 68% interval around it is computed, equivalent to the previous case.

The plots in Figure 6.3 show the results of a standard test generation process, for two of our datasets. First and second rows show BFCov 3.11 and BFxCov 3.13 metrics respectively. Every column differs in the initial test size $X_0 \in \{10, 100\}$. Each individual plot shows initial and final coverage distributions (their medians and 68% regions), for PCA and ICA methods. The interpretation is that higher median line on coverage, better the median coverage; smaller the errors, and more precise is the metric. The smaller and longer median line represents the spread in the runtime.

The analysed outcomes illustrate that test generation process consistently enlarges the median of the coverage, which is expected. However, the spread of a distribution stays

similar, with a few exceptions. Larger number of runs could improve the precision of results, however, we believe the main reason for such a spread is that only runs with higher initial coverage managed to improve. The ones with low initial coverage were hard to improve and stayed the same. It can be observed that the constant 0.33333333 initial coverage that appeared frequently in many testing situations, did not increase in most cases (Note the minimum coverage -initial and final for all charts is 0.33), see the tables in Appendix C for the testing summary results.

Considering the initial test size, one can see that larger initial test size, *i.e.,* $X_0 = 100$ consistently results in larger (sometime comparable) coverage. A larger X_0 gives the synthesis algorithm more leeway to find candidates from which to derive new inputs that hit target intervals that are not exercised by any test in X_0. For the PCA and ICA, there's no apparent difference between two methods, one exception for the Fashion-MNIST, $X_0 = 100$, BFCov metric, where PCA results in much tighter distribution. As the same is not visible in the BFxCov metric, the significance of this result cannot be assessed. Both BFCov and BFxCov metric generally agree with the level of improvement during the testing. Considering runtime, the charts express that ICA method is slightly more expensive in all situations.

A further analysis of the obtained final coverage of all testing traces inspected via an inquiry about 1.00 achieved coverage revealed that the testing criteria satisfied only twice with the bfc criterion on the CIFAR-10 dataset. Both situations occurred with a 100 initial test size using the ICA with two and three extracted features per layer and the KDE discretisation method. This implied a total of 254 combination traces out of 256 (64 per testing criteria per CNN model) did not satisfy $\text{BFCov}(\mathcal{B}_{\mathcal{N},\text{X}}) = 1$ neither $\text{BFxCov}(\mathcal{B}_{\mathcal{N},\text{X}}) = 1$, after 100 iterations. Observing the final coverage in Figure 6.3, with red and yellow colours, the average median final coverage is around 0.87%, which mean there are 0.13% of the networks remain untested. What if the not covered features are the vital element of the neural network? There are neither guarantees nor any information about the untested elements. This issue will be evidenced in the following experiments. Thus, the experiment's outcomes indicated that the BN-based feature metrics **DO NOT** guarantee the coverage of a model's critical internal regions.

SRQ5: Weight Features Coverage using Proposed Metrics. This experiment was directed to assess whether the weight-based approach, which weights the features according to their importance scores when calculating the coverage, exhibited advantages in improving

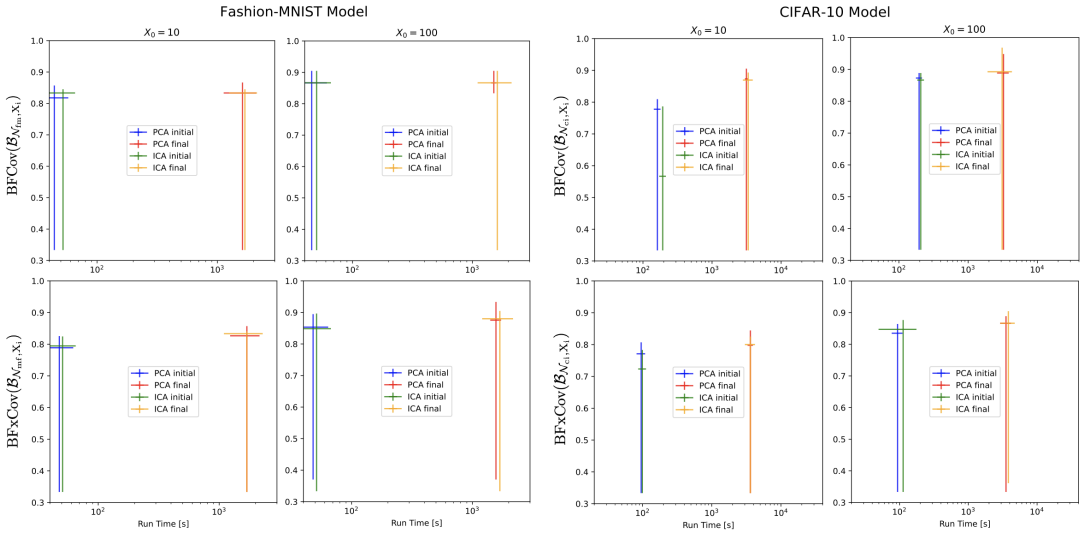


Figure 6.3: BN-based feature coverage plots show the overall distribution of initial and the respective final coverage of up to 100 iterations of Concolic test case generation. X-axis indicates the run time in seconds (initial and run time). The horizontal line on the coverage is the median.

the BN-based feature coverage. In particular, the study examined whether the weight-based feature metrics would achieve higher coverage with less run time than the original metrics.

Figure 6.4 shows the results of the weight coverage measures WFCov which was defined in Equation 6.3 and WFCovTot that was formalised in Equation 6.5, in equivalent arrangement as the previous Figure 6.3. Comparing the two figures, it can be seen that the lowest initial coverage in the majority of the plots are greater than the constant value of 0.33, which occurred often in the previous experiment. This expected raise of the initial coverage comes from the difference in the coverage definitions - one being weighted and the other not. This small growth in the initial coverage gives a greater opportunity for the coverage to be improved during the testing. An example from the weight coverage testing experiment (see Table C.8 in Appendix C) which gave 0.38950211 that increased to 0.72610162 final coverage with 51 new generated inputs. This is evident from the preceding finding, which reported that starting testing with a higher initial coverage has a better chance of increasing. Furthermore, the initial coverages in all plots, except for the CIFAR-10 with $X_0 = 100$, are consistent with initial coverages in Figure 6.3. This indicate that most of the features with higher weights are not covered yet.

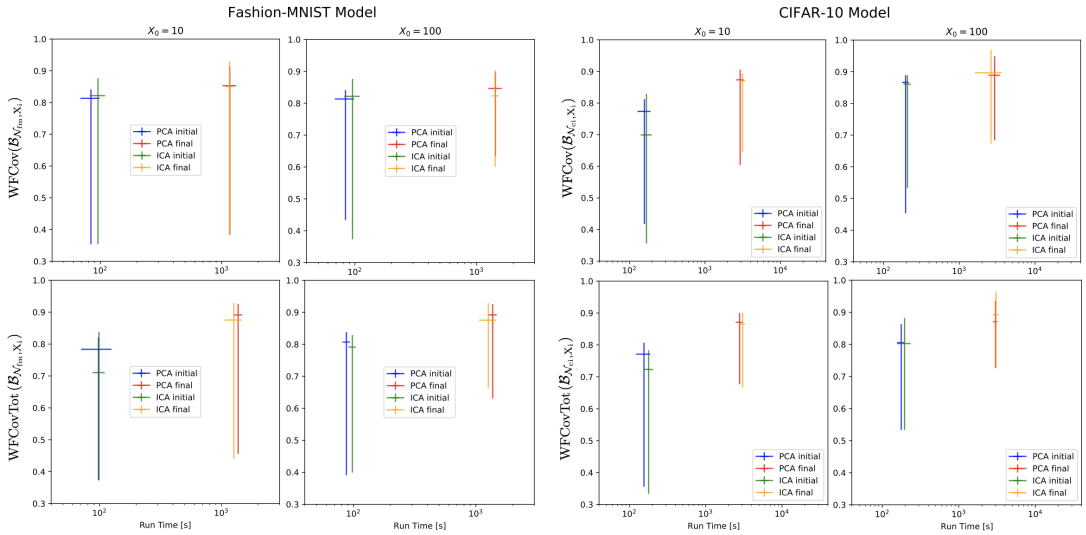


Figure 6.4: Weight-based feature coverage plots show the overall distribution of initial and the respective final coverage of up to 100 iterations of Concolic test case generation. X-axis indicates the run time in seconds (initial and run time).

Considering the final coverage, the charts show significant improvement in the coverage for the WFCovTot compared to BFxCov, for both datasets and PCA/ICA methods. The reason for this is that the generation process is led by the most important parts of the BN, which have larger weights. A notable observation is that the minimum final coverage increased considerably, which indicates the higher-importance intervals were covered with the new input first. This trend is the same for all coverages except for the F-MNIST model with $X_0 = 10$. Finally, considering runtime, weighted coverage takes more time for the initial computation, which spent additional time calculating feature weights. However, convergence is reached slightly faster compared to the non-weighted case. Improvement is of a few percent, thus not so significant. The results for WFCov with respect to BFCov give better improvements in coverage, for both datasets and methods. From the preceding outcomes, it can be concluded that the proposed weight coverage metrics **DO** provide a reliable testing measurement in terms of reporting the coverage that prioritises the important internal representation of the model.

6.5.4 Further Results

The above experiments clearly demonstrated the effectiveness of the weighted coverage compared with the basic coverage. Both metrics were able to generate new sets of inputs that achieved high coverage. However, the WFCov and WFCovTot enforced the higher coverage on the more relevant training dataset features. In addition, the weight feature metrics generate a comparable number of adversarial inputs. Further details are discussed below.

Quantitative improvement of coverage. In this part, a specific quantitative enhancement of weight-based feature coverage over BN-based feature coverage is presented. A detailed comparison of different testing scenarios is discussed, including the initial measured coverage, which interval is triggered, the coverage progress made through each passed test case and the total number of new generated test cases. These detailed testing cases focus on the Fashion-MNIST model with several BN specification combinations of the PCA feature extraction technique. These combinations are chosen by picking almost all available BN specifications; however, they are still not representative of the rest of the scenarios, see tables in Appendix C for the full test traces. The purpose of this illustration is to show the testing process in detail with specific increment numbers.

Table 6.1 demonstrates three testing scenarios: PCA-X10-N3-U5, PCA-X10-N4-KDE, and PCA-X100-N5-U5. As mentioned earlier, the PCA is used to extract the features in all situations. X10 and X100 indicate the initial test size $|X_0|$. N is the number of extracted features per CNN layer, in the selected specifications, it is three, four, and five. The last abbreviations represent the discretisation methods that are 5-bin-uniform intervals and KDE. In the first scenario (PCA-X10-N3-U5), the BFCov initial coverage was 0.8095 and then increased to 0.8254 by successfully generating a new input that hit the second interval of the first extracted feature from the max_pooling2d_1 layer. One more produced test input succeeded in improving the coverage to 0.8413 before the 100 iterations were over. The total number of generated tests is 26; one of them is adversarial input. On the other hand, the WFCov initial coverage was 0.8409 and increased through three newly generated tests that exhibit intervals from (max_pooling2d_1, 2) and (activation_2, 0) features. The coverage increased each time according to the interval weight into 0.8957, as illustrated in Table 6.2 that shows the calculated weights for all three scenarios for both WFCov and WFdCov metrics. The total number of generated tests is 30; three of them are adversarial

input.

Regarding the BFxCov and WFCovTot metrics, the scenarios only illustrate the calculation of the feature dependency coverage for the features of the activation_2 and dense_1 layers. For the features of the max_pooling2d_1 layer and the rest layers, the feature coverage will be calculated as above to obtain the total coverage as presented in the Appendix C's tables. The weight normalisation factor is adjusted accordingly in Table 6.2. The BFdCov initial coverage started at 0.8258 and increased to 0.8393 by targeting interval 3 of feature 0 in layer activation_2, subject to feature intervals (1, 1, 3) in layer max_pooling2d_1. It then reached the 0.8433 final coverage by generating a test input whose valuation value belongs to the second interval of feature 2 in layer activation_2, subject to feature intervals (3, 4, 4) in layer max_pooling2d_1. Moreover, although the WFdCov initial coverage, 0.7944, was lower than the BFdCov, it reached 0.8617 through two test cases as well. Thus, both metrics increased the coverage; however, the weight metric reported its coverage based on the importance feature scores.

In regard to the second BN specification instance (PCA-X10-N4-KDE), since the KDE computes feature intervals based on their associated density distribution, each feature is partitioned into a different number of intervals; see the example below. Here, we emphasise that the intervals are sorted in descending order in the test target Tar_invals set according to their feature weight. The rest of the two testing scenarios in the Table 6.1 are translated into the same first explained case.

Example 20. *The example shows part of the executed testing code output of the PCA-X10-N4-KDE specification:*
Abstracted layers: max_pooling2d_1, activation_2, dense_1
Computing Bayesian Network abstraction...
| Given 20000 classified training sample
| Extracting features for layer max_pooling2d_1... Extracted 4 features
| Discretising features for layer max_pooling2d_1...
| Discretisation of feature 0 involves 5 intervals
| Discretisation of feature 1 involves 4 intervals
| Discretisation of feature 2 involves 3 intervals
| Discretisation of feature 3 involves 3 intervals
| Extracting features for layer activation_2... Extracted 4 features
| Discretising features for layer activation_2...

| *Discretisation of feature 0 involves 5 intervals*
| *Discretisation of feature 1 involves 5 intervals*
| *Discretisation of feature 2 involves 3 intervals*
| *Discretisation of feature 3 involves 4 intervals*
| *Extracting features for layer dense_1... Extracted 4 features*
| *Discretising features for layer dense_1...*
| *Discretization of feature 0 involves 4 intervals*
| *Discretisation of feature 1 involves 3 intervals*
| *Discretisation of feature 2 involves 3 intervals*
| *Discretisation of feature 3 involves 3 intervals*

BN specification	criterion	init_cov	hit_interval	progs_cov	#gen_tests
X10-N3-U5	bfc	0.8095	l:max f:0 v:2	0.8254	25 + 1 adv.
			l:max f:1 v:3	0.8413	
	wfc	0.8409	l:max f:2 v:4	0.8646	27 + 3 adv.
			l:act f:0 v:0	0.88	
			l:act f:0 v:2	0.8957	
	bfdc	0.8258	l:act f:0 v:3 c:(1,1,3)	0.8393	45 + 0 adv.
			l:act f:2 v:2 c:(3,4,4)	0.8433	
	wfdc	0.7944	l:act f:0 v:5 c:(1,2,2)	0.8281	40 + 1 adv.
			l:act f:0 v:5 c:(3,2,3)	0.8617	
X10-N4-KDE	bfc	0.6639	l:dense f:3 v:0	0.7194	32 + 0 adv.
			l:act f:2 v:0	0.7719	
			l:act f:0 v:4	0.8369	
	wfc	0.7942	l:max f:0 v:5	0.8167	29 + 0 adv.
			l:dense f:0 v:1	0.9063	
	bfdc	0.8138	l:dense f:0 v:3 c:(2,1,1,1)	0.8203	33 + 0 adv.
	wfdc	0.8002	l:dense f:2 v:3 c:(1,3,1,1)	0.8292	32 + 0 adv.
X100-N5-U5	bfc	0.8952	l:max f:3 v:5	0.9048	17 + 0 adv.
	wfc	0.8804	l:max f:1 v:7	0.8923	24 + 0 adv.
			l:max f:2 v:2	0.9036	
			l:max f:2 v:6	0.9149	
			l:dense f:0 v:2	0.9260	
	bfdc	0.8947	l:act f:1 v:3 c:(1,1,2,2,2)	0.8953	12 + 0 adv.
	wfdc	0.8798	l:act f:2 v:4 c:(1,3,2,4,3)	0.8943	30 + 0 adv.

Table 6.1: Improvement of testing coverage (up to 100 iterations) for various BN specification scenarios (X10 and X100 indicate X_0' size, N3, N4 and N5 are the number of extracted features per DNN's layer, U5 is the uniform discretisation with five bins and KDE is the Kernel Density Estimation discretisation method. Four criteria are compared: bfc, bfdc, wfc, and wfdc for the Fashion-MNIST model. The hit_interval specifies exactly what interval is triggered and causes the coverage to increase; "l" donates the CNN's layer, "f" is the number of features, "v" is the interval, and "c" is the combination of feature intervals from the previous layer.

Features	BN specification			BN specification- wfdc		
	PCA-X10-N3-U5 d_{corr}	PCA-X10-N4-KDE d_{corr}	PCA-X100-N5-U5 d_{corr}	PCA-X10-N3-U5 d_{L_2}	PCA-X10-N4-KDE d_{L_2}	PCA-X100-N5-U5 d_{L_2}
(max_pooling2d_1, 0)	0.105467	0.112420	0.060254	0.112859	0.060395	0.056223
(max_pooling2d_1, 1)	0.101796	0.088920	0.083430	0.109274	0.084177	0.070053
(max_pooling2d_1, 2)	0.166113	0.096506	0.078934	0.111754	0.108896	0.073193
(max_pooling2d_1, 3)	—	0.078087	0.058197	—	0.088021	0.073698
(max_pooling2d_1, 4)	—	—	0.068820	—	—	0.076609
(activation_2, 0)	0.111107	0.079128	0.072504	0.120855	0.096951	0.062478
(activation_2, 1)	0.115871	0.032070	0.063710	0.119946	0.094191	0.069944
(activation_2, 2)	0.091184	0.081935	0.067060	0.101161	0.093914	0.081423
(activation_2, 3)	—	0.082476	0.050955	—	0.075684	0.052693
(activation_2, 4)	—	—	0.074110	—	—	0.063848
(dense_1, 0)	0.108244	0.089610	0.077759	0.115761	0.064766	0.062478
(dense_1, 1)	0.093658	0.085508	0.058747	0.101130	0.067194	0.049025
(dense_1, 2)	0.106560	0.085203	0.069006	0.107261	0.100158	0.072783
(dense_1, 3)	—	0.088138	0.047495	—	0.065654	0.052024
(dense_1, 4)	—	—	0.069020	—	—	0.072748

Table 6.2: Assigned feature weights for three testing scenarios: PCA-X10-N3-U5, PCA-X10-N4-KDE, and PCA-X100-N5-U5, calculated for WFCov and WFdCov metrics based on the Fashion-MNIST model. These abbreviations denote the BN specification: pca indicates the feature extraction method, X10 and X100 represent the size of the initial test sets $|X_0| \in \{10, 100\}$, N3, N4 and N5 are the number of extracted features per DNN's layer (three, four and five), and U5 and KDE are the discretisation strategies (uniform with five bins and Kernel Density Estimation).

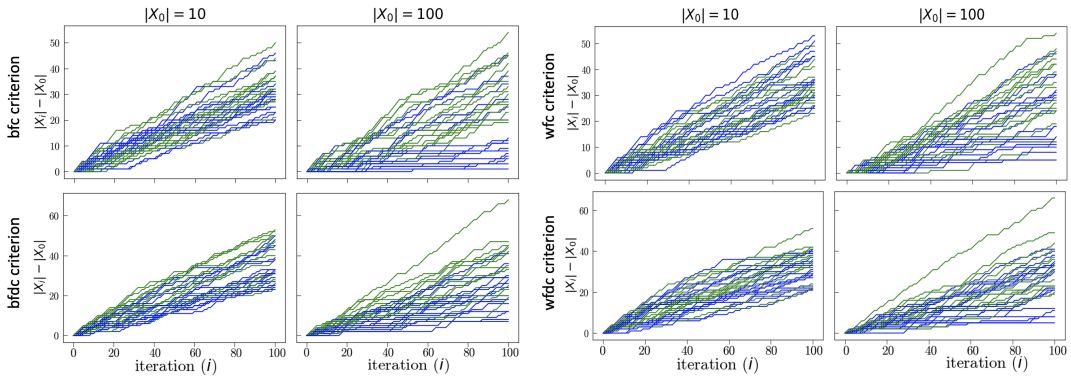


Figure 6.5: Summary of the new produced test inputs of up to 100 iterations of test case generation by DeepConcolic targeting BN-based coverage and weight-based coverage for the Fashion-MNIST model, for two sizes of initial test sets $|X_0| \in \{10, 100\}$. Each row specifies the used criterion: bfc, bfdc, wfc, and wfdc. Green and blue lines respectively indicate runs with ICA and PCA-based feature extractions.

Newly produced test inputs. The plot in Figure 6.5 shows the growth of the generated test set with respect to the testing iterations. Each of the four charts illustrates a specific criterion's (bfc, bfdc, wfc, and wfdc) ability to generate new sets of inputs. Overall, between 10% to 60% of iterations produce new test cases. Looking at the $X_0 = 10$ instance, there is a steady increase in the number of test samples, for both PCA and ICA. On the other hand, for the case of $X_0 = 100$, there is a significant fraction of realisations that do not change too much in the test suite size. The reason for this difference is in 10 being a very small sample, while 100 being much better. This is also evident from the previous section, with $X_0 = 100$ giving much larger initial coverage, as well as final coverage compared to $X_0 = 10$.

Generated adversarial examples. The directed Concolic test case generation with feature weights was able to generate adversarial samples. As stated earlier, the newly synthesised inputs were considered adversarial when they do not have the correct classification label. Figure 6.6 shows some adversarial examples that were found during the testing experiments running on $\mathcal{N}_{fm}$ and $\mathcal{N}_{ci}$ models and targeting the weight-based feature coverage and the weight-based feature dependence coverage criteria.

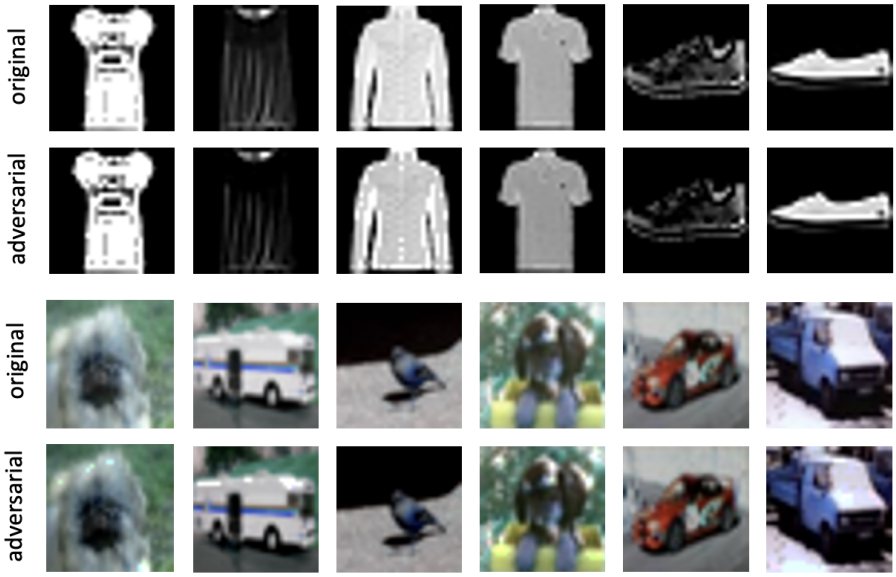


Figure 6.6: Some adversarial examples found by achieving WFCov and WFdCov testing criteria for the Fashion-MNIST model (above) and CIFAR-10 model (below).

6.6 *Trustworthy* Performance With the Weight Metrics

The reliability and trustworthiness of deep neural networks have become increasingly important as they are being used in safety-critical applications such as autonomous vehicles and medical diagnosis. One aspect of reporting the trustworthy performance of DNNs is to ensure they behave correctly by measuring how well they have been tested during the evaluation process. The below discussion is to answer the third subsidiary research question (SRQ6) considered in this chapter, which inquires whether the proposed weighted testing coverage provides trustworthiness in the learning systems' performance. Providing guarantees for the DNN's safety has been thoroughly investigated in recent years through the abstraction process. Gehr et al. [30] presented the AI^2 analyser that utilises abstract interpretation to verify the safety propriety of the CNN. In [76], The authors suggested employing abstractions of Boolean combinations of linear arithmetic constraints as a means for verifying the safety behaviour of learning models. On the other hand, the feature importance concept is used in explainable AI to determine the parts of the network that are responsible for the output decision. That helps in identifying potential vulnerabilities, improving overall transparency, and thus increasing trust in the model. By integrating these

two approaches, researchers can not only ensure the safety of DNNs but also gain a deeper understanding of how they work and make decisions.

The presented semantic metrics combined the two theories, Bayesian abstraction and identification of the DNN's high-level features importance, to provide a comprehensive evaluation of the model's performance. The weight-based testing metrics assign weights to different features of the internal representation based on their sensitivity importance. The weights are then used to prioritise the testing of those features, with the aim of enforcing higher coverage on the critical decision regions. This approach allows for a more efficient and effective testing process, as it focuses on the features that are most likely to impact the overall performance of the model. Thus, the presented methodology for selecting the test cases ensures the DNN has been properly tested on critical features and provides further evidence that the model is trustworthy.

From the preceding discussion, the answer to the SRQ6 is that the proposed methodology for selecting test cases, concentrating on features that have a significant impact on a model's performance, improves the reliability and trustworthiness of the DNN models.

6.7 Related Work

The majority of the existing research in the DNN area adapts testing methodologies that concern test adequacy criteria based on the neuron-level aspect, as discussed in Chapter 2. Regardless, a closely related work to the concept of importance-based semantic testing is the DeepImportance approach presented in [32]. This paper introduces a systematic testing approach that is developed based on the Importance-Driven (IDC) test adequacy criterion. Their approach is similar to the presented idea of analysing the model behaviour to identify the important high-level features first and then measuring the test set coverage based on that; however, their aim was to find *the important neurons* that are primary contributors in decision-making. Also, their method differs in the phrase of coverage adequacy, where the paper's goal is to test all generated combinations of important neuron clusters while ignoring the unimportant network's neurons. In comparison, the weight-based approach prioritises the higher-weight features but does not ignore the less important ones. The DeepImportance method employs the layer-wise relevance propagation technique from the explainable AI mentioned in Section 2.6 to develop a layer-wise functional understanding of the neuron's internal behaviour. The algorithm computes the relevance contribution scores for each neuron for all input samples and then produces clusters of the most important

neurons using iterative unsupervised learning. The authors then evaluated the semantic adequacy of a test set by targeting different combinations of important neurons' behaviours. The DeepImportance's test adequacy criterion is satisfied when all combinations of important neuron clusters are exercised.

6.8 Conclusion

This chapter has introduced a weight-based semantic testing approach that measures how well the DNN is tested by focusing on the importance features using its abstracted Bayesian Network. The conducted experiments empirically validate the applicability and effectiveness of the proposed weight metrics. The evaluation results demonstrated that the basic BN-based testing metrics do not guarantee coverage of a model's critical internal regions, where the test case generation algorithm was not designed to target the most relevant features. The developed weight-based feature coverage metrics achieved higher testing coverage than the original metrics, with an emphasis on covering the important learned representation, where the test generation algorithm is directed to synthesise new input targeting features with higher importance scores. This guarantees that the high-risk regions with the most influential features are tested thoroughly and the reported coverage was measured based on the important internal representation of the DNN model. This serves as a strong argument in favour of increasing the trustworthy performance of DNNs in safety-critical applications.

Chapter 7

Conclusion and Future Work

7.1 Introduction

This concluding chapter provides a summary of the work presented in this thesis, as well as the key findings and recommendations for future research. The chapter begins with a summary of the content that has been covered in this thesis in Section 7.2. Section 7.3 outlines the contributions that have been made and highlights the main findings of the conducted experiments in terms of the primary and subsidiary research questions to the thesis sought to address as presented in Chapter 1. Lastly, Section 7.4 wraps up the chapter by discussing possible directions for future study that may expand upon the work described in the thesis.

7.2 Summary of the Thesis

This thesis proposed weight-based semantic testing metrics that use developed feature importance weights to measure the coverage of a test set, and generate additional test cases guided by the priority weights to guarantee a more "trustworthy" measure of the model's performance. The thesis commenced with an introductory chapter, Chapter 1, that discussed the motivations behind the conducted studies. The motivation was bridging the gap of the usage of the testing approaches and the learned driven behaviour of neural networks. This is in line with the increased integration of machine learning into safety-critical applications. The chapter then presented the main research question to be investigated and the related subsidiary research questions. It proceeded with presenting the research

methodology to solve the current lack of testing strategies. The proposed solution was to utilise the semantic representation that is formed in the DNN through the training process to focus on in the testing phase. The contributions that have been made to achieve the desired goals are discussed at the end of the chapter.

Chapter 2 presented the concept of deep neural networks and their related challenges and concerns. It discussed the properties of DNNs that need to be tested to guarantee their reliability and safety. This chapter then provided a comprehensive review of the existing testing approaches concerning the DNNs. The literature review established that testing metrics and test input generation algorithms have made significant progress in improving the coverage of deep learning models, however, achieving guarantees for complete coverage of their behaviour and optimal targeting of critical features is still an active research area. The chapter closes with possible methods for exploring the internal logic of DNN models. In Chapter 3, Bayesian theory was explored. An overview of the idea of Bayesian Network (BN) was given, along with a discussion on how recent research has utilised Bayesian Networks to address common issues in DNNs. The chapter then presented the BN abstraction model that was used as a foundation model to construct the proposed weight metrics. The main phases of constructing a BN were discussed and a detailed explanation of how the DNN's layer features were extracted through the PCA and ICA algorithms. It also described the mathematical notations of constructing the BN and the implemented BN-based feature coverage metrics.

The following two chapters studied the significance and relevance of the DNN's latent representations to the model's behaviour and decision-making process. Chapter 4 explored the learned latent features of DNNs through the lens of the BN. It combined the sensitivity analysis and perturbation techniques to create a feature importance measure. This importance score expresses the causal relationship between the internal representations and the DNN behaviour, since more influential features have a stronger causal relationship to influence decision-making. In this chapter, the concept of *random_shift* was presented, and a proposed feature sensitivity analysis algorithm evaluated for detecting adversarial distribution shift. The analysis focus of this chapter relied on the BN probability distribution. Furthermore, in Chapter 5, a more concrete sense of the degree to which the hidden features contribute to the output decision was considered. The chapter presented a method for deriving an abstract BN model from a neural network to make a classification decision. The strategy involved adding an auxiliary prediction node to the BN and calculating its conditional probability table. The degree of change in the classification output for per-

turbed features was correlated with several distance metrics used to compute the difference between two probability distributions. This correlation was clarified by constructing a BN weighted feature model discussed in Chapter 6 that assigns importance scores to each node of the BN. Thus, the BN weighted feature model is constructed from a DNN to establish weight-based testing metrics.

Chapter 6 puts together the preceding analyses to design high-level testing criteria for neural networks, defined based on a BN abstraction. The first presented metric was the weight-based feature coverage that concentrates on each individual hidden feature interval and measures the weighted ratio of intervals that are adequately exercised by a given dataset. It simply checks the marginal probabilities for every interval in the BN's node that has a bigger probability than a predefined ε value. The second metric was the weight-based feature dependence coverage that concentrated on the causal relationships between hidden feature intervals in successive layers and measures the weighted ratio of intervals combinations that are exercised enough in the test dataset. This metric targets the conditional probabilities of the BN's node, which means the extracted features from the first DNN's selected layer are not taken into account. Therefore, a third metric was developed to consider the coverage of the first metric for the first layer's features and the average converge of both metrics for the rest of the features. Finally, the results from an extensive evaluation of the proposed testing metrics wad presented; evaluation that used two CNN models trained on two state-of-the-art classification data sets, Fashion-MNIST and CIFAR-10. The experimental results indicated that the weight metrics achieved higher coverage than the BN-based metrics by targeting the higher importance weight intervals in the test case generation process. To sum up, this study contributes to the development of effective techniques for testing and improving the trustworthiness and reliability of deep learning models.

7.3 Main Findings and Contributions

In this section, the main findings and contributions of the work presented in this thesis are discussed. The section begins by addressing each subsidiary research question separately and then answering the primary research question that this thesis seeks to resolve, as stated in Chapter 1, based on the given sub-answers. The following are the answers to the subsidiary research questions:

1. **SRQ1:** *Is a Bayesian abstraction of a neural network able to systematically analyse a DNN's interior decisions?*
   Bayesian theories are considered a standard principal for analysing deep learning models, as demonstrated in Chapter 3. In addition, as discussed in Chapter 2, learning important features either through propagation or perturbation is a main technique to explain the DNN's underlying behaviour and decision-making. Extracting a DNN's features, structuring their relationships in a Bayesian abstraction scheme, and then identifying their importance values serve to combine the best of decision analysis methods. The presented BN-based feature sensitivity analysis in Chapter 4 was able to give insight into the internal behaviour through the distribution change, which was able to detect an adversarial distribution shift. It was thus concluded that the chosen BN model was able to systematically analyse interior DNN's decisions.

2. **SRQ2:** *How can the importance of the latent features of a deep neural network be quantified using an abstracted Bayesian Network?*
   Bayesian sensitivity analysis offers valuable insights into model behaviour and the identification of influential factors as discussed in Chapter 4. Moreover, sensitivity to perturbations is a common method employed to explore the DNN's internal decision machinery. The presented BN-based feature sensitivity analysis algorithm provided a principled framework for quantifying the DNNs' latent features. The importance of the perturbed feature was estimated by comparing the BN's probability distributions before and after the applied perturbation. Therefore, the importance of the latent features was quantified through the abstracted Bayesian Network sensitivity distances.

3. **SRQ3:** *How to demonstrate the impact of the feature importance measure on the classification decisions based on the abstraction?*
   The impact of the feature perturbations on the classification decisions is assessed by enhancing the BN abstraction with prediction capability. Chapter 5 described how a BN classifier, based on a DNN abstraction, can be constructed. The interval shift procedure, presented in Section 4.3.2 could be then applied to each feature to observe the change in the classification label. This provided a quantitative assessment of the relevance of each feature to the classification decisions made by the BN. Afterward, the relationship between the feature importance scores and the number of misclassified input samples was analysed (Chapter 6).

4. **SRQ4:** *Do existing testing metrics guarantee the coverage of a model's critical internal regions, as well as direct the test case generation algorithm to target the most relevant features?*

   The existing DNN testing metrics, including structural coverage, do not provide an entire testing solution for the DNN models. In particular, they do not guarantee complete coverage of the semantically critical parts of the model or even prioritise the most relevant features. As discussed in Chapter 2, there are limitations associated with the structural coverage criteria where they do not explicitly address the high-dimensional and complex nature of DNNs and their underlying decision logic. These metrics may provide a certain amount of coverage information at a low level, such as neuron activation, but they may not effectively capture the actual contribution made by the internal representation of the deep learning model. Moreover, the BN-based testing metrics do not concentrate on the most critical features, which remain a weak point if they do not achieve 100% coverage, as shown by the experiments reported on Chapter 6.

5. **SRQ5:** *Do the proposed coverage metrics deliver a reliable testing measurement in terms of reporting the coverage that prioritises the important internal representation of the model?*

   The proposed weight-based testing metrics deliver a reliable testing measure in that they report the model coverage based on the important weights that reflect the core contributor features to the model decision. This in turn provides assurance evidence of its behaviour. The designed test generation algorithm targets the most relevant features to be covered first with the new test case. As it is difficult to reach the 100% coverage within a reasonable time, *i.e.,* 100 iterations, this prioritising of higher-weight features approach can guarantee that important regions of the model's behaviour are tested. The experiments reported on in Section 6.5 demonstrate how the weight coverage grows faster when a new test input is passed. This indicates that important features with higher weights are first selected to be tested. Thus, it is argued that the proposed coverage metrics do deliver a reliable testing measure.

6. **SRQ6:** *Does the generated test dataset from the feature weight directed concolic testing provide a* trustworthy *measure of a model's performance?*

   The assumption underpinning the weight-based semantic testing approach is that the latent feature space encodes internal and hidden representations learned by the DNN

model, and a good testing criterion should emphasise the important latent space to estimate the test set coverage. In real-world DNN testing scenarios, especially a complicated one, it is impossible to test all the deep learning model's behaviour combinations as there will be an extremely large number of configuration spaces. A coverage is driven by the fact that the critical behaviours of a learning system are identified through its latent features attitude; imposing a higher coverage on them would give safety assurance even if the final coverage is below 100%. Thus the weight testing approach makes them far more reliable and trustworthy for real-world applications than structural and unweighted testing methods.

Returning to the main research question of the thesis:

"Is it a better measure of trustworthiness to measure the coverage of the semantic aspect of deep neural networks and treat each internal component according to its contribution value to the decision when testing these learning models' performance than relying on traditional structural unweighted measures?"

Given the preceding discussion, it can be concluded that trustworthiness is enhanced if deep neural network coverage is reported based on its semantic level, which measures coverage with respect to the most influential latent representations. Thus, the semantic and weighted metrics increase reliability and confidence in the neural network's correct behaviours compared to the structure and unweighted metrics.

For further emphasis on the contributions made throughout this thesis, this section is concluded with a listing and an elaboration of the contributions:

1. The idea of importance weight within DNN testing metrics that integrate a BN abstraction scheme with the concept of feature importance from the explainable AI field. Therefore providing a solution to exploring the internal representation of a DNN and enforcing higher coverage on the features that are core contributors in the decision-making process.

2. A novel technique that utilises a BN model to estimate the importance of a neural network's latent features by analysing the BN's node sensitivity to distributional shifts. This method can then be used to compute the distance between the probability distributions represented by the abstracted BN and the distributions of inputs under

adversarial attacks, thus indicating the differences between the original and perturbed distributions.

3. An approximator classifier based on an abstract BN extracted from a neural network that was able to approximate the original DNN's prediction performance while outperforming it in an adversarial setting.

4. A method to calculate a DNN's latent feature weights based on the change in the BN predictions. The underlying assumption was that features that contribute the most to decision-making cause significant misclassification when they are perturbed.

5. A weighted feature model that assigns the BN feature nodes their importance values, according to the correlated sensitivity weights with a BN prediction weight.

6. Three semantic testing metrics based on BN abstraction and importance weights, namely *(i) Weight-based feature coverage*, *(ii) Weight-based feature dependence coverage*, and *(iii) Generalised weighted feature coverage*, and then enhancing the concolic test case generation to target the most influential features that have been identified as high-risk components that must be tested thoroughly.

7. An analysis of the weight metrics using two popular datasets: Fashion-MNIST and CIFAR-10. The analysis comprised a variety of BN specification scenarios, including different combinations of PCA, ICA, N2-N5, U1, U3, U5, KDE, and $|X_0| = \{10, 100\}$. The aim was to compare the coverage improvement of the weight-based metrics with respect to the BN-based metrics.

7.4 Limitations and Future Work

BN-based Feature Importance for DNN explanation. Bayesian networks can provide explanations for their predictions or inferences, which is important in many applications where the model's decision-making process needs to be understood or justified. A common method is tracing the probability calculations through the network to identify the factors that influenced the final prediction or inference. Using the proposed BN classifier and importance weights may offer an advanced method to interpret the DNN's decision-making process. It is certainly interesting for a future investigation to explore if the generated importance values can support the explanation of black-box learning models.

Markov blanket for the BN prediction. Further improvement can be implemented to the BN classifier model presented in Section 5.4. Not all information of the BN is needed to make a prediction. For instance, in the case where a prediction node is connected only to the last layer of the initially constructed BN (see Figure 5.3), the specification of the parent nodes will give full information about the output prediction $P(y|\mathbf{r})$. This concept is called a Markov blanket - which "covers" all other nodes as unnecessary.

A similar situation holds for the case where all of the nodes are connected to the last one. In this case, however, all of the nodes will play a role in the prediction of the final node, but only the final CPT will be necessary, while intermediate CPT distributions can be ignored. Both procedures can highly simplify the necessary BN and make inference significantly faster and more accurate. Implementation of them is left for future work.

Further Utility of the BN Classifier. The BN classifier model can be used as a safety monitor for DNNs since its discrete nature is *harder* to attack where the discretisation function is not trivially differentiable. This can be implemented by designing a classifier that combines the classification results from the original DNN and the BN. The combined classifier (safety monitor), outputs the decision label if they are agreed in both DNN and BN. This combination might be investigated and tested in future work.

Alternative Testing Algorithm. Berthier et al. [9] proposed test metrics over a BN by extending the MC/DC metrics proposed by Sun et al. [90]; and extending the Concolic test case generation method [89], based on symbolic computation [87], is extended to work with BNs. Therefore, weight-based metrics were also implemented in the concolic testing algorithm. However, there is still a wide range of test case generation algorithms, e.g. Fuzzing based, that can be explored for BNs, and future research can focus on comparing the effectiveness of different testing methods. Moreover, it will be useful to examine if the generated test cases can be more natural and diverse when compared with those generated directly on DNNs, as done in [43].

Importance Weight Computation. The presented method for computing the importance weight per feature was sufficient for the weight feature coverage approach, as illustrated from experiments. However, for the weight feature dependence coverage, it seems that computing the weight per interval would make the weight metric more reasonable since the metric seeks to capture the causal relationships between hidden features. In such

a case, the process of identifying the target interval can consider the weights of the feature intervals' combinations from the previous layer. Thus, if the importance weight was calculated at the interval level, the eliciting function from the Tar_invals set would rely on the feature weights as a prerequisite to giving priority to selecting their intervals. To decide which feature's interval to choose, the function sums up the weights of their combination intervals from the previous layer, and the higher sum is chosen. Thus, computing an importance value per feature's interval can be considered in future studies. Moreover, the perturbation of feature importance quantification in Algorithm 1 has relied primarily on the random shifting function presented in Subsection 4.3.2. Further perturbation methods can be explored in terms of perturbing the latent features.

Hyper-parameters in Overall Experiments The experiments conducted throughout the thesis have relied on a wide combination of parameters. Further investigation with advanced cases, for instance, using non-linear feature extraction techniques like Kernel-PCA [83] or manifold learning [57] can be explored in the future to assess the properties of extracted features in extended cases.

Real World Application. The material presented in this thesis has focused on creating a well-designed testing approach for the DNNs. Developing an assurance case study based on real-world scenarios, such as a self-driving car, can be considered in the context of future work.

Further Use of the Abstracted Bayesian Network. As suggested, a BN can be seen as an abstraction of the original DNN. It is therefore imperative to understand how this abstraction may further help in analysing or enhancing the original DNN. In addition to testing, it would also be interesting to explore if such abstraction may bring any benefit with respect to: verification [44], interpretation of DNN training [49], explainable AI [107], and safety case [103]. For example, scalability is the key obstacle of DNN verification due to its complexity [81]. Considering that an abstracted BN is significantly smaller than the original DNN, it will be interesting to understand if the BN can be used to alleviate the problem without losing the provable guarantee. A potential difficulty may be whether and how the verification result on the BN can be transferred to the DNN. Similar as the above discussion for testing and verification, the potential for the BN to be used as an intermediate step for the reliability assessment [104] and safety case [105] is worthy of exploration. This

may probably require a quantification of the error, or the loss of information, when using BN as an abstraction of the DNN.

Appendix A

Detailed Structures of the Trained Deep Neural Networks

This Appendix presents the structure of the trained deep neural network models used throughout the thesis experiments. Five models are considered: $\mathcal{N}_{sm}$, $\mathcal{N}_{sm-max}$, $\mathcal{N}_{mnist}$, $\mathcal{N}_{fm}$, and $\mathcal{N}_{ci}$ whose layers are listed in the four tables below. Layers are including blocks of convolutional and max-pooling layers, followed by a series of dense layers. The model incorporate ReLU activation functions except the activation output layer which involves a classical Softmax to output the predicted label. The first small MNIST model $\mathcal{N}_{sm}$ is often used to construct a BN utilised in the explanation examples.

Layer Name (Function)	Output Shape	#Parameters
conv2d (Conv2D)	(26, 26, 8)	80
activation (Activation)	(26, 26, 8)	0
flatten (Flatten)	(5408)	0
dense (Dense)	(42)	227178
activation_1 (Activation)	(42)	0
dense_1 (Dense)	(10)	430
activation_2 (Activation)	(10)	0

Table A.1: Structure of the small CNN model $\mathcal{N}_{sm}$ trained on the MNIST dataset with 98.00% test accuracy.

Layer Name (Function)	Output Shape	#Parameters
conv2d (Conv2D)	(26, 26, 8)	80
activation (Activation)	(26, 26, 8)	0
max_pooling2d (MaxPooling2)	(13, 13, 8)	0
flatten (Flatten)	(1352)	0
dense (Dense)	(42)	56826
activation_1 (Activation)	(42)	0
dense_1 (Dense)	(10)	430
activation_2 (Activation)	(10)	0

Table A.2: Structure of the small CNN model $\mathcal{N}_{sm-max}$ trained on the MNIST dataset with 97.78% test accuracy.

Layer Name (Function)	Output Shape	#Parameters
conv2d (Conv2D)	(26, 26, 32)	320
activation (Activation)	(26, 26, 32)	0
conv2d_1 (Conv2D)	(24, 24, 32)	9248
activation_1 (Activation)	(24, 24, 32)	0
max_pooling2d (MaxPooling2)	(12, 12, 32)	0
conv2d_2 (Conv2D)	(10, 10, 64)	18496
activation_2 (Activation)	(10, 10, 64)	0
conv2d_3 (Conv2D)	(8 , 8, 64)	36928
activation_3 (Activation)	(8 , 8, 64)	0
max_pooling2d_1 (MaxPooling2)	(4, 4, 64)	0
flatten (Flatten)	(1024)	0
dense (Dense)	(200)	205000
activation_4 (Activation)	(200)	0
dense_1 (Dense)	(200)	40200
activation_5 (Activation)	(200)	0
dense_2 (Dense)	(10)	2010
activation_6 (Activation)	(10)	0

Table A.3: Structure of the CNN model $\mathcal{N}_{mnist}$ trained on the MNIST dataset with 99.38% test accuracy.

Layer Name (Function)	Output Shape	#Parameters
conv2d (Conv2D)	(26, 26, 32)	320
activation (Activation)	(26, 26, 32)	0
max_pooling2d (MaxPooling2D)	(13, 13, 32)	0
conv2d_1 (Conv2D)	(9, 9, 64)	51264
activation_1 (Activation)	(9, 9, 64)	0
max_pooling2d_1 (MaxPooling2D)	(4, 4, 64)	0
flatten (Flatten)	(1024)	0
dense (Dense)	(100)	102500
activation_2 (Activation)	(100)	0
dense_1 (Dense)	(10)	1010
activation_3 (Activation)	(10)	0

Table A.4: Structure of the CNN model $\mathcal{N}_{fm}$ trained on the Fashion-MNIST dataset with 89.03% test accuracy.

Layer Name (Function)	Output Shape	#Parameters
conv2d (Conv2D)	(30, 30, 32)	896
activation (Activation)	(30, 30, 32)	0
conv2d_1 (Conv2D)	(28, 28, 32)	9248
activation_1 (Activation)	(28, 28, 32)	0
max_pooling2d (MaxPooling2)	(14, 14, 32)	0
conv2d_2 (Conv2D)	(12, 12, 64)	18496
activation_2 (Activation)	(12, 12, 64)	0
conv2d_3 (Conv2D)	(10, 10, 64)	36928
activation_3 (Activation)	(10, 10, 64)	0
max_pooling2d_1 (MaxPooling2	(5, 5, 64)	0
flatten (Flatten)	(1600)	0
dense (Dense)	(512)	819712
activation_4 (Activation)	(512)	0
dense_1 (Dense)	(10)	5130
activation_5 (Activation)	(10)	0

Table A.5: Structure of the CNN model $\mathcal{N}_{ci}$ trained on the CIFAR-10 dataset with 81.00% test accuracy.

Appendix B

Detailed Results for Sensitivity to Adversarial Shift Experiments

The plots in Figure 4.9 illustrated some statistics for a subset of the considered distances for comparing probability vectors. Figures B.1, B.2, and B.3 show the distances computed for the MNIST model, and Figures B.4, B.5, and B.6 show the results for the CIFAR10 model. In these plots, hue still indicates the discretisation strategy. However, it have discriminated between extended and non-extended strategies: the prefix '-X' denotes that latent features are discretised in such a way that left- and right-most intervals do not contain any (projected) training sample.

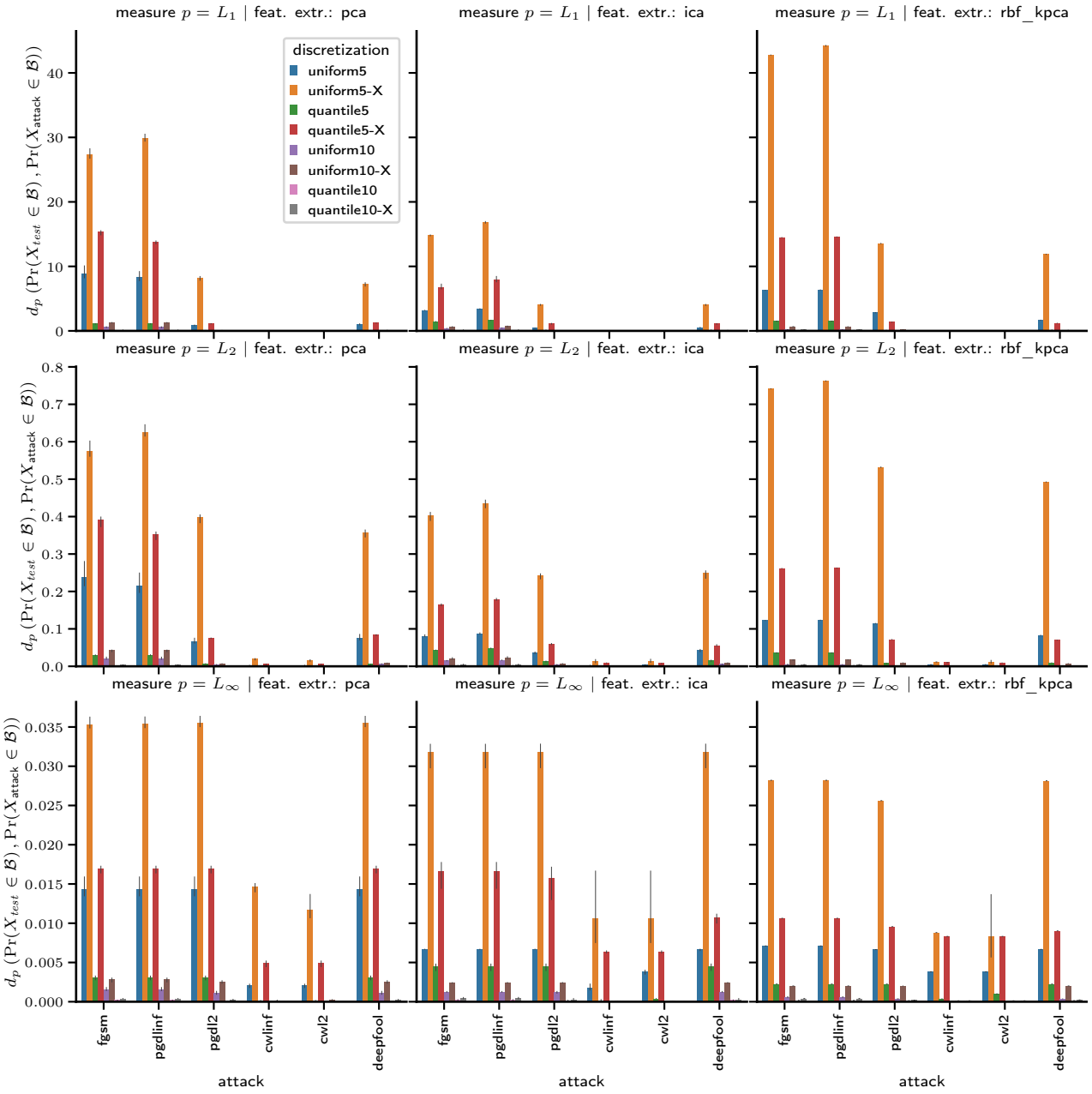


Figure B.1: Distances (vertical axes) between probability vectors obtained for the MNIST validation dataset ($\Pr(X_{test} \in \mathcal{B})$) and probability vectors ($\Pr(X_{\mathsf{attack}} \in \mathcal{B})$) obtained for datasets generated by selected adversarial attacks (attack, shown on the horizontal axes), for a range of BN abstractions $\mathcal{B}$. Every abstraction involves 3 layers for which 3 features have been extracted using PCA (left-hand side column), ICA (middle), or radial basis functions (RBF) kernel-PCA (right). Plotted data aggregates five independent runs, and shows confidence intervals.

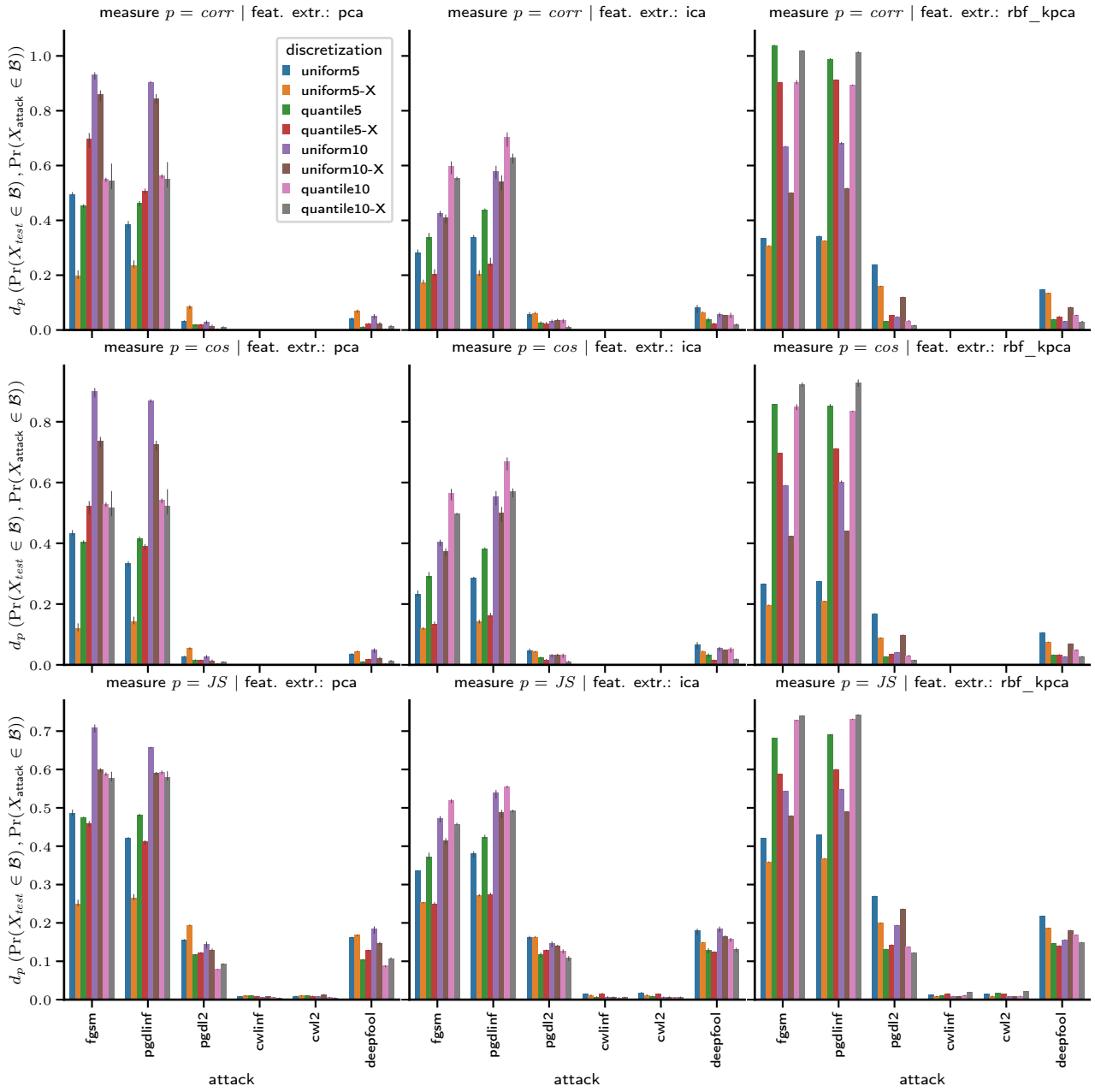


Figure B.2: See Figure B.1.

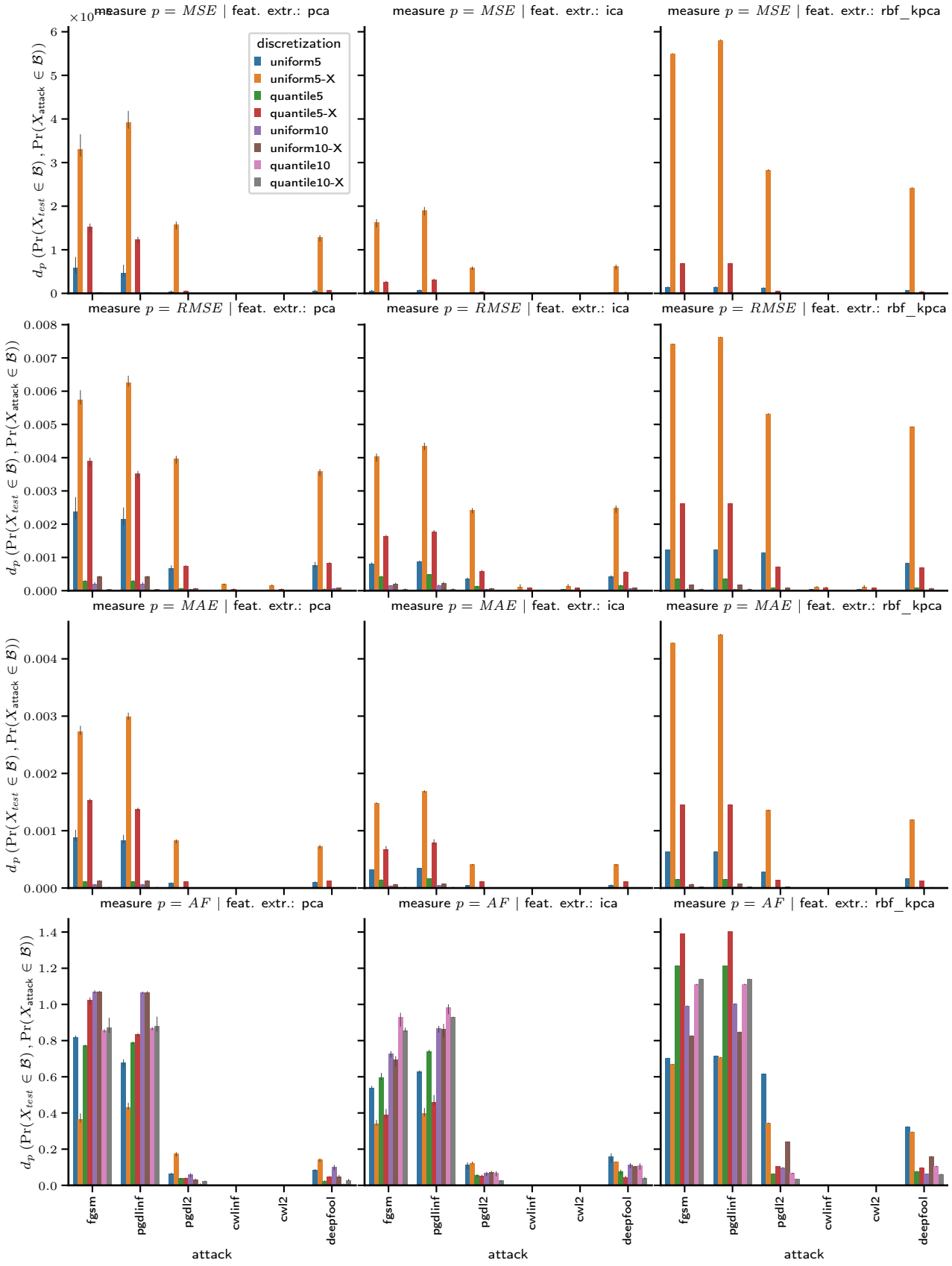


Figure B.3: See Figure B.1.

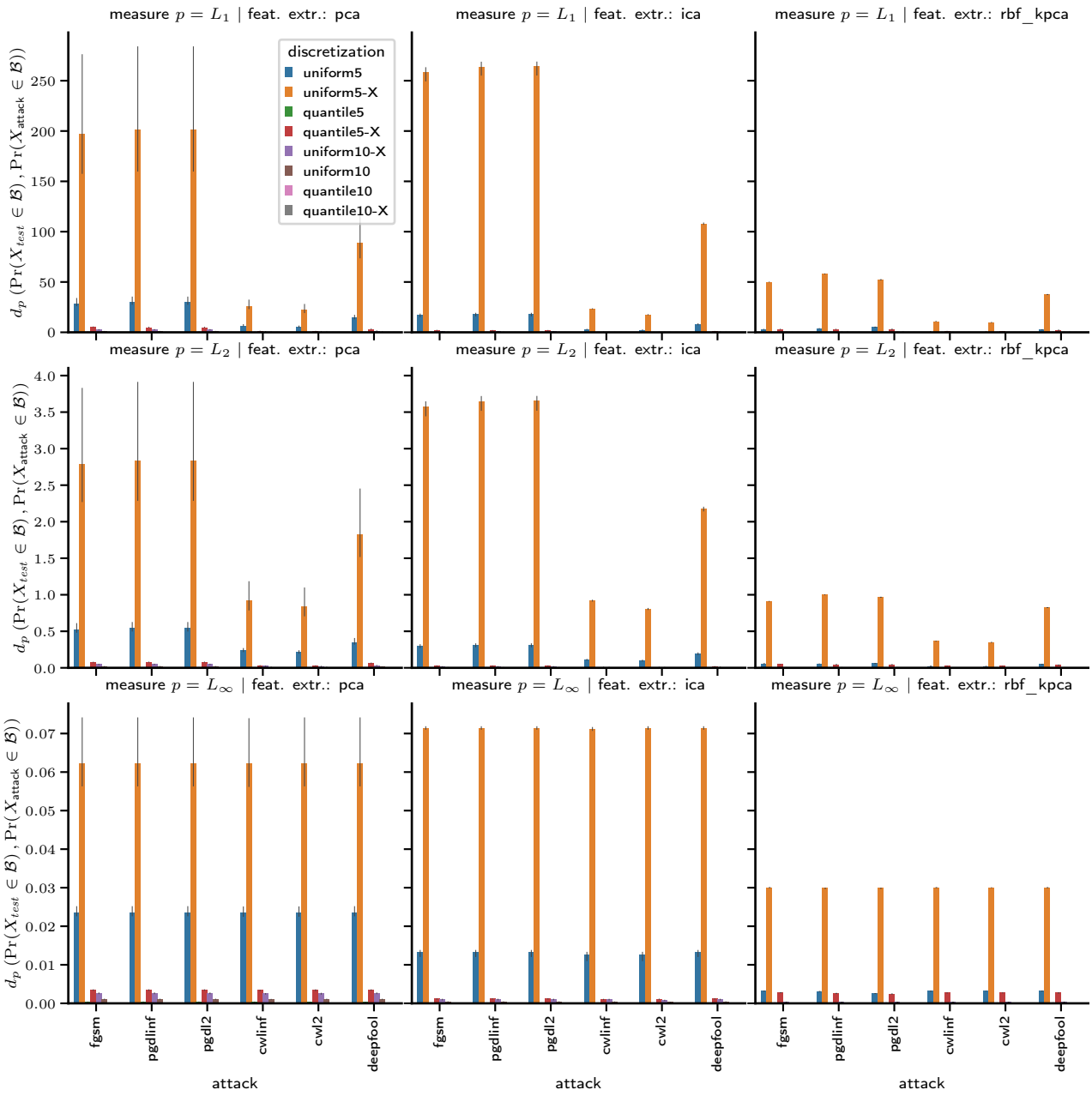


Figure B.4: Distances (vertical axes) between probabilities obtained for the CIFAR10 validation dataset and datasets generated by selected adversarial attacks (horizontal axes). See Figure B.1 for further details.

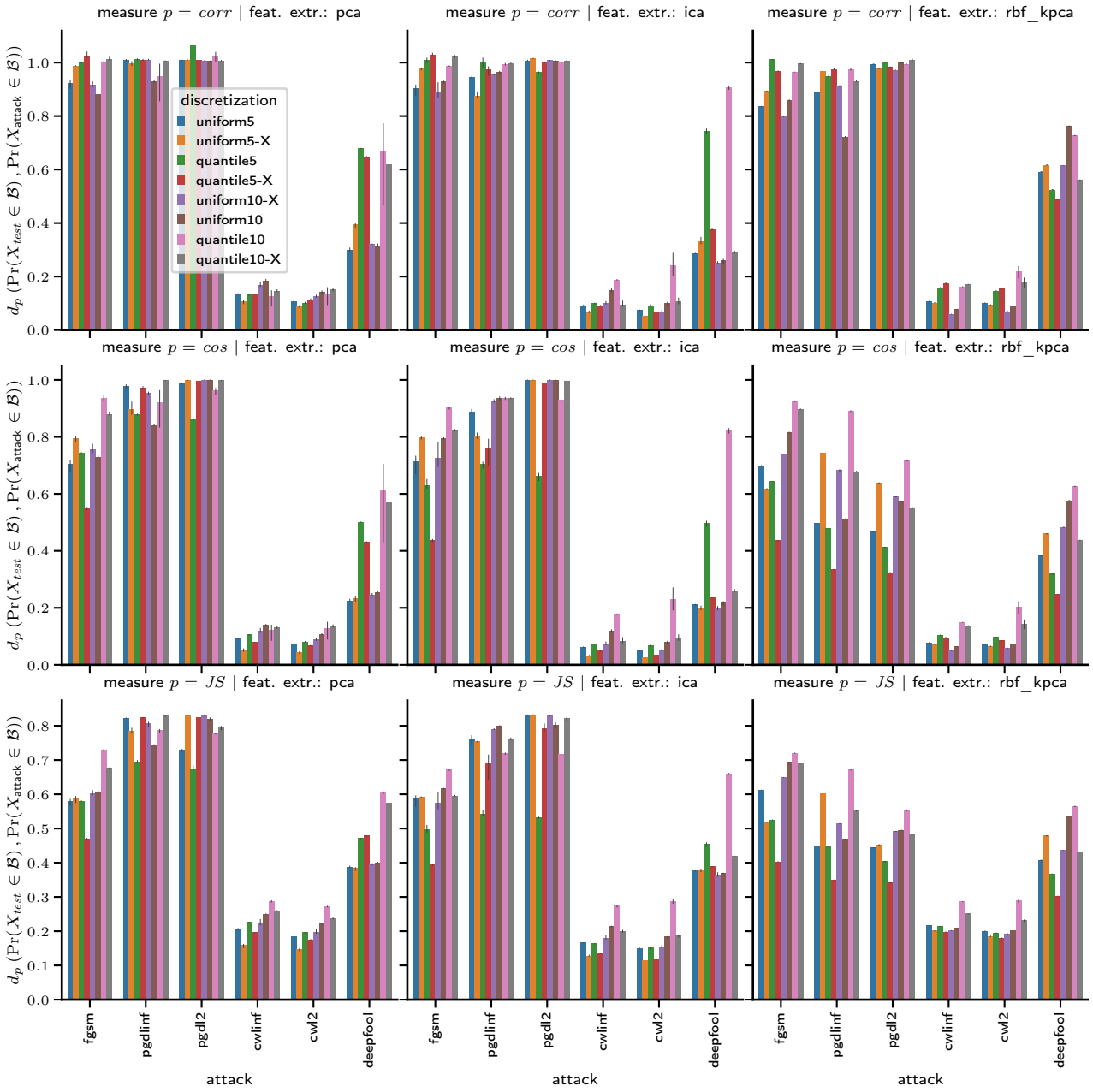


Figure B.5: See Figure B.4.

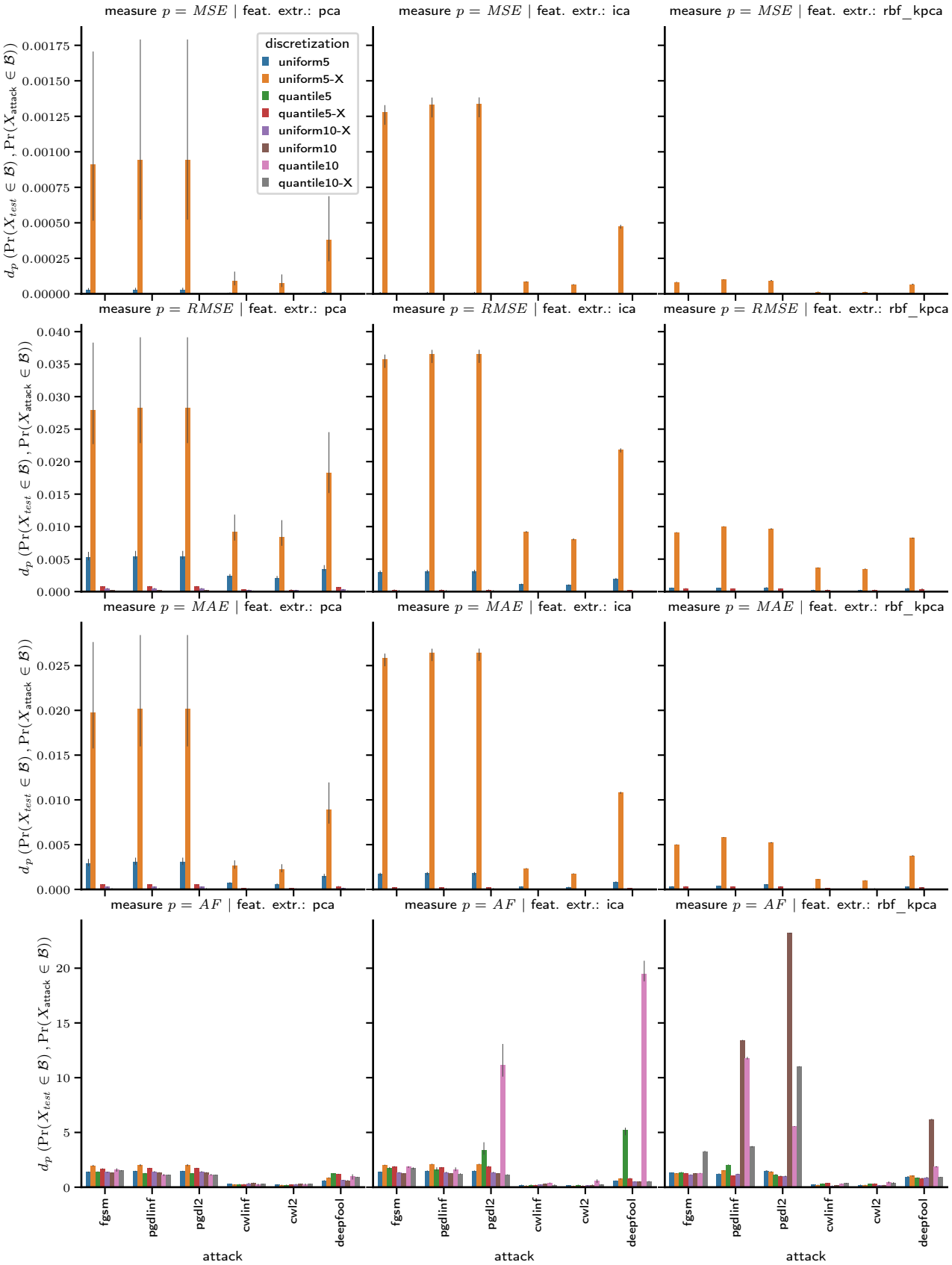


Figure B.6: See Figure B.4.

Appendix C

Detailed Report of the Test Case Generation Process

Sample run traces of the Concolic testing results from experiments in 6.5 for the Fashion-MNIST model.

crit	tech	n_feat	discr	n_bins	X_0	iters	init_time	run_time	init_cov	final_cov	n_tests	n_advs
bfc	pca	2	kde	0	10	100	42.9451	1421.8849	0.6889	0.8222	27	7
bfc	pca	2	uniform	1	10	100	35.1480	1437.1459	0.3333	0.3333	31	0
bfc	pca	2	uniform	3	10	100	37.2607	1459.0378	0.8667	0.8667	23	0
bfc	pca	2	uniform	5	10	100	35.4834	1460.0572	0.8571	0.8810	28	0
bfc	pca	3	kde	0	10	100	53.6450	1414.9115	0.6222	0.8074	23	0
bfc	pca	3	uniform	1	10	100	35.3554	1447.3732	0.3333	0.3333	46	0
bfc	pca	3	uniform	3	10	100	34.4621	1448.5533	0.8667	0.8667	25	0
bfc	pca	3	uniform	5	10	100	35.8400	1443.3160	0.8095	0.8413	26	1
bfc	pca	4	kde	0	10	100	54.6701	1560.0553	0.6639	0.8370	32	0
bfc	pca	4	uniform	1	10	100	42.2838	1887.5911	0.3333	0.3333	37	0
bfc	pca	4	uniform	3	10	100	36.0109	1435.6444	0.8333	0.8333	29	0
bfc	pca	4	uniform	5	10	100	39.4033	1447.1066	0.8333	0.8452	20	0
bfc	pca	5	kde	0	10	100	60.2695	1474.1696	0.8178	0.8178	30	0
bfc	pca	5	uniform	1	10	100	36.6983	1475.1917	0.3333	0.3333	35	1
bfc	pca	5	uniform	3	10	100	41.9029	1681.0971	0.8400	0.8400	31	0
bfc	pca	5	uniform	5	10	100	85.1116	3391.7663	0.8286	0.8476	43	0
bfc	ica	2	kde	0	10	100	52.2359	1485.5166	0.6556	0.8222	28	0
bfc	ica	2	uniform	1	10	100	41.0524	1474.3457	0.3333	0.3333	32	0
bfc	ica	2	uniform	3	10	100	40.5774	1458.5520	0.8333	0.8333	29	0
bfc	ica	2	uniform	5	10	100	40.5958	1489.8313	0.8333	0.8333	45	1
bfc	ica	3	kde	0	10	100	56.5854	1436.1970	0.8056	0.8056	50	0
bfc	ica	3	uniform	1	10	100	56.5722	1743.5538	0.3333	0.3333	32	0
bfc	ica	3	uniform	3	10	100	58.2087	1958.5377	0.8667	0.8667	21	0
bfc	ica	3	uniform	5	10	100	38.8829	1524.1769	0.8413	0.8413	28	0
bfc	ica	4	kde	0	10	100	61.4400	1479.6287	0.8333	0.8333	27	0
bfc	ica	4	uniform	1	10	100	40.5249	1671.6204	0.3333	0.3333	39	0
bfc	ica	4	uniform	3	10	100	57.8474	1899.1035	0.8333	0.8333	39	0
bfc	ica	4	uniform	5	10	100	43.4968	1566.5456	0.8452	0.8452	38	1
bfc	ica	5	kde	0	10	100	66.2342	1554.2865	0.8333	0.8333	35	0
bfc	ica	5	uniform	1	10	100	42.3174	1474.9113	0.3333	0.3333	39	2
bfc	ica	5	uniform	3	10	100	46.9393	1655.6087	0.8533	0.8533	32	0
bfc	ica	5	uniform	5	10	100	90.3024	3243.4972	0.8381	0.8381	41	9

Table C.1: Testing coverage of the Fashion-MNIST model for the **bfc criterion** conducted with $|X_0| = 10$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5, init_tests: 10 100.

crit	**tech**	**n_feat**	**discr**	**n_bins**	X_0	**iters**	**init_time**	**run_time**	**init_cov**	**final_cov**	**n_tests**	**n_advs**
bfc	pca	2	kde	0	100	100	46.7662	1446.6068	0.8778	0.8778	3	0
bfc	pca	2	uniform	1	100	100	35.3350	1517.5484	0.3333	0.8333	12	1
bfc	pca	2	uniform	3	100	100	35.2596	1434.7129	0.8667	0.8667	45	0
bfc	pca	2	uniform	5	100	100	38.3315	1428.6853	0.9048	0.9048	9	0
bfc	pca	3	kde	0	100	100	53.6741	1471.3466	0.9185	0.9185	8	0
bfc	pca	3	uniform	1	100	100	37.2173	1535.4706	0.3333	0.8519	21	1
bfc	pca	3	uniform	3	100	100	36.4960	1462.0410	0.8667	0.8667	13	0
bfc	pca	3	uniform	5	100	100	36.8323	1483.4661	0.9048	0.9048	6	0
bfc	pca	4	kde	0	100	100	57.5008	1464.0187	0.9111	0.9111	1	0
bfc	pca	4	uniform	1	100	100	35.9408	1582.6597	0.3333	0.3333	34	0
bfc	pca	4	uniform	3	100	100	37.9154	1479.3794	0.8667	0.8667	12	0
bfc	pca	4	uniform	5	100	100	41.3874	1595.8074	0.8810	0.8810	27	2
bfc	pca	5	kde	0	100	100	63.9144	1517.8523	0.8400	0.8400	28	0
bfc	pca	5	uniform	1	100	100	36.8912	1553.1489	0.3333	0.8000	37	0
bfc	pca	5	uniform	3	100	100	44.2986	1477.7786	0.8667	0.8667	5	0
bfc	pca	5	uniform	5	100	100	93.5451	1732.5798	0.8952	0.9048	7	0
bfc	ica	2	kde	0	100	100	51.1289	1400.9177	0.6972	0.8556	30	0
bfc	ica	2	uniform	1	100	100	40.0092	1407.5578	0.3333	0.3333	45	0
bfc	ica	2	uniform	3	100	100	39.4629	1424.6443	0.8667	0.8667	24	1
bfc	ica	2	uniform	5	100	100	41.1890	1422.8675	0.9048	0.9048	26	0
bfc	ica	3	kde	0	100	100	54.2018	1435.5329	0.8333	0.8333	20	0
bfc	ica	3	uniform	1	100	100	42.3377	1497.5697	0.3333	0.3333	33	0
bfc	ica	3	uniform	3	100	100	39.4257	1488.7183	0.8667	0.8667	19	0
bfc	ica	3	uniform	5	100	100	43.0618	1491.6955	0.9048	0.9048	26	0
bfc	ica	4	kde	0	100	100	64.9012	1466.4144	0.9097	0.9097	42	0
bfc	ica	4	uniform	1	100	100	41.8670	1478.8399	0.3333	0.3333	27	0
bfc	ica	4	uniform	3	100	100	41.3283	1510.3829	0.8667	0.8833	45	0
bfc	ica	4	uniform	5	100	100	46.4161	1607.6889	0.9048	0.9048	32	1
bfc	ica	5	kde	0	100	100	69.8413	1530.5924	0.8889	0.9000	35	0
bfc	ica	5	uniform	1	100	100	41.3580	1518.5604	0.3333	0.3333	58	4
bfc	ica	5	uniform	3	100	100	48.7790	1689.1757	0.8667	0.8667	39	0
bfc	ica	5	uniform	5	100	100	99.1542	3505.7239	0.8952	0.9048	46	0

Table C.2: Testing coverage of the Fashion-MNIST model for the **bfc criterion** conducted with $|X_0| = 100$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5.

crit	tech	n_feat	discr	n_bins	X_0	iters	init_time	run_time	init_cov	final_cov	n_tests	n_advs
bfdc	pca	2	kde	0	10	100	47.6531	1449.4794	0.7212	0.8778	48	0
bfdc	pca	2	uniform	1	10	100	36.6878	1418.9644	0.3333	0.3333	24	0
bfdc	pca	2	uniform	3	10	100	37.2643	1461.6637	0.7367	0.8667	28	1
bfdc	pca	2	uniform	5	10	100	37.8916	1495.3261	0.7276	0.8571	31	0
bfdc	pca	3	kde	0	10	100	53.5896	1495.8031	0.6563	0.8074	55	5
bfdc	pca	3	uniform	1	10	100	37.1217	1827.8893	0.3333	0.3333	27	1
bfdc	pca	3	uniform	3	10	100	58.8029	2315.7853	0.7889	0.8222	27	2
bfdc	pca	3	uniform	5	10	100	37.3438	1539.8429	0.8167	0.8355	45	0
bfdc	pca	4	kde	0	10	100	59.1374	1476.9927	0.8202	0.8393	33	0
bfdc	pca	4	uniform	1	10	100	37.5844	1502.7604	0.3333	0.3333	39	0
bfdc	pca	4	uniform	3	10	100	39.4870	1558.8701	0.8265	0.8333	38	0
bfdc	pca	4	uniform	5	10	100	40.8073	1593.5740	0.8308	0.8333	35	2
bfdc	pca	5	kde	0	10	100	70.2425	1524.0801	0.8130	0.8178	23	0
bfdc	pca	5	uniform	1	10	100	38.8336	1503.7606	0.3333	0.3333	33	0
bfdc	pca	5	uniform	3	10	100	45.6559	1706.4096	0.8248	0.8267	29	0
bfdc	pca	5	uniform	5	10	100	86.5898	3215.6987	0.7997	0.8095	27	0
bfdc	ica	2	kde	0	10	100	55.3190	1520.1307	0.4159	0.8222	38	0
bfdc	ica	2	uniform	1	10	100	41.2882	1488.9691	0.3333	0.3333	47	0
bfdc	ica	2	uniform	3	10	100	41.8589	1544.9271	0.7233	0.8333	31	0
bfdc	ica	2	uniform	5	10	100	40.7891	1461.2797	0.7349	0.8333	27	0
bfdc	ica	3	kde	0	10	100	57.4653	1462.7800	0.7914	0.8333	38	0
bfdc	ica	3	uniform	1	10	100	40.1330	1516.2741	0.3333	0.3333	53	0
bfdc	ica	3	uniform	3	10	100	42.2171	1514.0768	0.8129	0.8444	23	0
bfdc	ica	3	uniform	5	10	100	42.9990	1509.4027	0.8246	0.8413	53	3
bfdc	ica	4	kde	0	10	100	64.2046	1520.2412	0.7947	0.8056	40	0
bfdc	ica	4	uniform	1	10	100	43.2260	1559.5174	0.3333	0.3333	45	0
bfdc	ica	4	uniform	3	10	100	42.5949	1560.0383	0.8243	0.8333	30	2
bfdc	ica	4	uniform	5	10	100	46.1146	1626.4510	0.8189	0.8333	31	0
bfdc	ica	5	kde	0	10	100	70.2647	1560.1238	0.7973	0.8000	32	0
bfdc	ica	5	uniform	1	10	100	43.4275	1531.6755	0.3333	0.3333	52	0
bfdc	ica	5	uniform	3	10	100	49.2259	1772.8203	0.8381	0.8400	43	0
bfdc	ica	5	uniform	5	10	100	92.7534	3912.7070	0.8472	0.8476	51	1

Table C.3: Testing coverage of the Fashion-MNIST model for the **bfdc criterion** conducted with $|X_0| = 10$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5. Note that the coverage measure calculated here is the one defined in Eq. 3.13

crit	**tech**	**n_feat**	**discr**	**n_bins**	X_0	**iters**	**init_time**	**run_time**	**init_cov**	**final_cov**	**n_tests**	**n_advs**
bfdc	pca	2	kde	0	100	100	51.5480	1519.9664	0.7933	0.9333	19	0
bfdc	pca	2	uniform	1	100	100	35.6062	1513.4178	0.3889	0.3889	30	0
bfdc	pca	2	uniform	3	100	100	39.0140	1450.7343	0.7055	0.8667	12	0
bfdc	pca	2	uniform	5	100	100	38.3916	1493.9801	0.7326	0.9048	29	0
bfdc	pca	3	kde	0	100	100	54.4793	1524.6849	0.9247	0.9778	28	0
bfdc	pca	3	uniform	1	100	100	38.6926	1512.6250	0.3704	0.3704	41	0
bfdc	pca	3	uniform	3	100	100	37.9216	1505.4127	0.8177	0.8667	8	0
bfdc	pca	3	uniform	5	100	100	38.5802	1514.9469	0.8742	0.9048	22	2
bfdc	pca	4	kde	0	100	100	60.4336	1521.8947	0.9111	0.9556	16	0
bfdc	pca	4	uniform	1	100	100	43.7112	1571.6940	0.3333	0.3333	38	2
bfdc	pca	4	uniform	3	100	100	39.6744	1516.9576	0.8534	0.8667	9	2
bfdc	pca	4	uniform	5	100	100	42.6184	1583.5776	0.8880	0.8929	24	0
bfdc	pca	5	kde	0	100	100	65.6434	1537.0091	0.8549	0.8756	21	1
bfdc	pca	5	uniform	1	100	100	39.0387	1538.0626	0.3556	0.3556	26	0
bfdc	pca	5	uniform	3	100	100	45.4213	1631.9692	0.8635	0.8667	18	0
bfdc	pca	5	uniform	5	100	100	98.2440	2155.3066	0.8999	0.9023	12	0
bfdc	ica	2	kde	0	100	100	52.8003	1554.4054	0.6337	0.8222	19	0
bfdc	ica	2	uniform	1	100	100	40.1206	1513.7268	0.3333	0.3333	42	0
bfdc	ica	2	uniform	3	100	100	42.0035	1490.0996	0.7401	0.8667	25	0
bfdc	ica	2	0uniform	5	100	100	40.4906	1482.1536	0.7412	0.9048	34	0
bfdc	ica	3	kde	0	100	100	58.9102	1486.1128	0.8486	0.8889	23	0
bfdc	ica	3	uniform	1	100	100	42.5440	1523.4632	0.3333	0.3333	45	0
bfdc	ica	3	uniform	3	100	100	40.9721	1835.4897	0.8262	0.8667	21	1
bfdc	ica	3	uniform	5	100	100	43.0884	1554.6687	0.8748	0.9048	34	0
bfdc	ica	4	kde	0	100	100	64.2071	1539.2974	0.8652	0.8819	36	1
bfdc	ica	4	uniform	1	100	100	41.9546	1548.7468	0.3333	0.3333	48	3
bfdc	ica	4	uniform	3	100	100	44.4082	1506.0188	0.8522	0.8667	29	0
bfdc	ica	4	uniform	5	100	100	48.4263	1620.7155	0.8994	0.9048	34	1
bfdc	ica	5	kde	0	100	100	70.6241	1658.0050	0.8966	0.9000	48	1
bfdc	ica	5	uniform	1	100	100	41.7154	1504.0362	0.3333	0.3333	69	1
bfdc	ica	5	uniform	3	100	100	48.3277	1736.8599	0.8773	0.8800	44	0
bfdc	ica	5	uniform	5	100	100	101.1713	3461.1105	0.9042	0.9048	42	0

Table C.4: Testing coverage of the Fashion-MNIST model for the **bfdc criterion** conducted with $|X_0| = 100$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5. Note that the coverage measure calculated here is the one defined in Eq. 3.13

crit	tech	n_feat	discr	n_bins	X_0	iters	init_time	run_time	init_cov	final_cov	n_tests	n_advs
wfc	pca	2	kde	0	10	100	71.6723	1022.1930	0.8155	0.8664	45	0
wfc	pca	2	uniform	1	10	100	65.7569	1310.6860	0.3333	0.3333	26	0
wfc	pca	2	uniform	3	10	100	60.7207	1264.1704	0.6283	0.8162	37	2
wfc	pca	2	uniform	5	10	100	73.9942	1265.7471	0.8412	0.8964	26	0
wfc	pca	3	kde	0	10	100	56.2461	1416.1383	0.7951	0.8551	36	0
wfc	pca	3	uniform	1	10	100	74.3095	1402.7631	0.3333	0.3333	32	0
wfc	pca	3	uniform	3	10	100	82.0052	1424.3509	0.8261	0.8761	36	0
wfc	pca	3	uniform	5	10	100	91.7244	1441.0070	0.8410	0.8957	30	3
wfc	pca	4	kde	0	10	100	94.6159	1400.6081	0.7942	0.9063	29	0
wfc	pca	4	uniform	1	10	100	81.1268	1520.1569	0.3361	0.5006	30	0
wfc	pca	4	uniform	3	10	100	103.3624	1513.8222	0.7912	0.9042	52	1
wfc	pca	4	uniform	5	10	100	93.6512	1525.7589	0.8133	0.8249	35	5
wfc	pca	5	kde	0	10	100	100.7937	1413.8517	0.8465	0.8465	48	1
wfc	pca	5	uniform	1	10	100	89.4304	1541.0355	0.5633	0.6733	29	0
wfc	pca	5	uniform	3	10	100	97.6989	1524.6447	0.8235	0.8235	45	0
wfc	pca	5	uniform	5	10	100	101.0189	1882.1538	0.8206	0.8419	53	0
wfc	ica	2	kde	0	10	100	86.6773	1313.5587	0.6447	0.8208	33	0
wfc	ica	2	uniform	1	10	100	79.2832	1402.8811	0.3333	0.3333	30	0
wfc	ica	2	uniform	3	10	100	75.1059	1412.3651	0.7739	0.7739	44	0
wfc	ica	2	uniform	5	10	100	78.5539	1457.9834	0.8232	0.8232	23	0
wfc	ica	3	kde	0	10	100	95.1058	1415.2290	0.6135	0.8115	35	0
wfc	ica	3	uniform	1	10	100	91.1368	1431.4768	0.3733	0.6006	41	0
wfc	ica	3	uniform	3	10	100	81.9869	1507.4976	0.8985	0.9185	28	0
wfc	ica	3	uniform	5	10	100	83.6709	1501.1748	0.8771	0.8771	26	0
wfc	ica	4	kde	0	10	100	97.6331	1431.1476	0.3780	0.7905	31	0
wfc	ica	4	uniform	1	10	100	96.1033	1400.0087	0.3543	0.6331	34	0
wfc	ica	4	uniform	3	10	100	109.5934	1474.2480	0.8763	0.8763	27	0
wfc	ica	4	uniform	5	10	100	102.6772	1560.9641	0.8342	0.8342	49	0
wfc	ica	5	kde	0	10	100	117.9092	1403.2288	0.5842	0.8062	45	0
wfc	ica	5	uniform	1	10	100	111.0023	1478.4274	0.3533	0.3667	31	0
wfc	ica	5	uniform	3	10	100	113.3586	1173.4413	0.8217	0.8717	37	0
wfc	ica	5	uniform	5	10	100	108.4664	1352.8543	0.8541	0.9041	32	0

Table C.5: Testing coverage of the Fashion-MNIST model for the **wfc criterion** conducted with $|X_0| = 10$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5.

crit	**tech**	**n_feat**	**discr**	**n_bins**	X_0	**iters**	**init_time**	**run_time**	**init_cov**	**final_cov**	**n_tests**	**n_advs**
wfc	pca	2	kde	0	100	100	89.5749	1479.1873	0.8620	0.8620	13	2
wfc	pca	2	uniform	1	100	100	80.4553	1502.1501	0.3431	0.4237	46	0
wfc	pca	2	uniform	3	100	100	96.4745	1162.0738	0.8550	0.8550	19	1
wfc	pca	2	uniform	5	100	100	80.4457	1337.2828	0.9080	0.9397	38	0
wfc	pca	3	kde	0	100	100	93.8532	1411.3970	0.8380	0.9280	30	1
wfc	pca	3	uniform	1	100	100	94.1073	1414.1066	0.3733	0.3733	31	1
wfc	pca	3	uniform	3	100	100	91.2665	1476.9056	0.8547	0.8667	24	0
wfc	pca	3	uniform	5	100	100	89.7237	1530.5902	0.9046	0.9046	14	0
wfc	pca	4	kde	0	100	100	96.3938	1474.7704	0.8746	0.8746	31	0
wfc	pca	4	uniform	1	100	100	100.3782	1403.3856	0.3333	0.3333	38	0
wfc	pca	4	uniform	3	100	100	99.9520	1399.0190	0.8629	0.8629	12	0
wfc	pca	4	uniform	5	100	100	98.4018	1327.8989	0.8955	0.8955	5	0
wfc	pca	5	kde	0	100	100	105.4235	1139.1993	0.9052	0.9052	9	1
wfc	pca	5	uniform	1	100	100	106.7322	1420.4718	0.4633	0.6016	30	5
wfc	pca	5	uniform	3	100	100	99.2986	1477.7786	0.8667	0.8867	7	0
wfc	pca	5	uniform	5	100	100	97.5451	1532.5798	0.8804	0.9260	24	0
wfc	ica	2	kde	0	100	100	81.1289	1300.9177	0.6972	0.8900	30	0
wfc	ica	2	uniform	1	100	100	97.0092	1307.5578	0.3333	0.3333	45	0
wfc	ica	2	uniform	3	100	100	91.4629	1324.6443	0.8667	0.8667	29	1
wfc	ica	2	uniform	5	100	100	78.1890	1332.8675	0.9048	0.9348	25	0
wfc	ica	3	kde	0	100	100	99.2018	1406.5329	0.8333	0.8333	20	0
wfc	ica	3	uniform	1	100	100	97.3377	1357.5697	0.3957	0.7706	43	0
wfc	ica	3	uniform	3	100	100	98.4257	1388.7183	0.8667	0.9248	17	0
wfc	ica	3	uniform	5	100	100	96.0618	1488.6955	0.9048	0.9048	26	0
wfc	ica	4	kde	0	100	100	98.9012	1405.4144	0.9097	0.9197	42	0
wfc	ica	4	uniform	1	100	100	99.8670	1378.8399	0.3295	0.5132	27	3
wfc	ica	4	uniform	3	100	100	101.3283	1450.3829	0.8667	0.8914	22	0
wfc	ica	4	uniform	5	100	100	102.4161	1407.6889	0.9048	0.9048	65	0
wfc	ica	5	kde	0	100	100	100.8413	1330.5924	0.8889	0.9000	35	0
wfc	ica	5	uniform	1	100	100	104.3580	1418.5604	0.3332	0.3946	34	2
wfc	ica	5	uniform	3	100	100	99.7790	1529.1757	0.8667	0.8821	31	0
wfc	ica	5	uniform	5	100	100	101.1542	2005.7239	0.8952	0.9048	48	1

Table C.6: Testing coverage of the Fashion-MNIST model for the **wfc criterion** conducted with $|X_0| = 100$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5.

crit	tech	n_feat	discr	n_bins	X_0	iters	init_time	run_time	init_cov	final_cov	n_tests	n_advs
wfdc	pca	2	kde	0	10	100	91.6185	1405.2280	0.7088	0.8669	22	0
wfdc	pca	2	uniform	1	10	100	86.4695	1384.5034	0.3333	0.3333	38	15
wfdc	pca	2	uniform	3	10	100	76.5983	1400.2942	0.7177	0.9162	28	0
wfdc	pca	2	uniform	5	10	100	96.1450	1375.8432	0.7228	0.7952	33	6
wfdc	pca	3	kde	0	10	100	89.6857	1421.9675	0.7736	0.9253	31	0
wfdc	pca	3	uniform	1	10	100	99.2423	1430.5642	0.4333	0.6306	42	0
wfdc	pca	3	uniform	3	10	100	84.7547	1408.7683	0.7970	0.8736	25	0
wfdc	pca	3	uniform	5	10	100	80.5137	1412.7958	0.8256	0.8713	41	1
wfdc	pca	4	kde	0	10	100	75.8682	1292.8947	0.7936	0.9142	32	0
wfdc	pca	4	uniform	1	10	100	88.0218	1406.9535	0.7426	0.9050	27	0
wfdc	pca	4	uniform	3	10	100	95.1104	1440.7451	0.8538	0.8829	21	2
wfdc	pca	4	uniform	5	10	100	121.9802	1505.5630	0.8113	0.9287	21	0
wfdc	pca	5	kde	0	10	100	101.0295	1413.8691	0.7917	0.8977	21	0
wfdc	pca	5	uniform	1	10	100	92.4942	1422.8142	0.3875	0.4137	37	0
wfdc	pca	5	uniform	3	10	100	98.2813	1101.0252	0.8220	0.9238	41	0
wfdc	pca	5	uniform	5	10	100	96.2866	1133.0642	0.7917	0.8920	39	0
wfdc	ica	2	kde	0	10	100	97.8444	1192.9527	0.5945	0.8708	39	0
wfdc	ica	2	uniform	1	10	100	87.4824	1138.6497	0.3633	0.3633	39	0
wfdc	ica	2	uniform	3	10	100	89.9641	1121.0669	0.6897	0.9104	32	0
wfdc	ica	2	uniform	5	10	100	80.1775	1175.8846	0.7105	0.8612	32	0
wfdc	ica	3	kde	0	10	100	87.1131	1386.3698	0.3725	0.8415	39	1
wfdc	ica	3	uniform	1	10	100	83.0666	1017.6566	0.5632	0.6633	42	1
wfdc	ica	3	uniform	3	10	100	97.7907	1132.7667	0.8190	0.8862	35	0
wfdc	ica	3	uniform	5	10	100	101.0065	1114.1877	0.8609	0.9694	52	1
wfdc	ica	4	kde	0	10	100	107.6131	1534.2199	0.3915	0.7905	42	0
wfdc	ica	4	uniform	1	10	100	97.4008	1502.2667	0.7099	0.9146	36	0
wfdc	ica	4	uniform	3	10	100	99.9568	1606.8881	0.8793	0.8860	26	2
wfdc	ica	4	uniform	5	10	100	83.8189	1092.7179	0.8381	0.8754	34	0
wfdc	ica	5	kde	0	10	100	100.3748	1522.0686	0.5644	0.9655	38	0
wfdc	ica	5	uniform	1	10	100	94.4318	1470.3590	0.3333	0.3333	48	6
wfdc	ica	5	uniform	3	10	100	114.8881	1027.4694	0.8199	0.8674	32	3
wfdc	ica	5	uniform	5	10	100	107.0260	1172.6079	0.8288	0.9292	22	0

Table C.7: Testing coverage of the Fashion-MNIST model for the **wfdc criterion** conducted with $|X_0| = 10$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5. Note that the coverage measure calculated here is the one defined in Eq. 6.5

crit	**tech**	**n_feat**	**discr**	**n_bins**	X_0	**iters**	**init_time**	**run_time**	**init_cov**	**final_cov**	**n_tests**	**n_advs**
wfdc	pca	2	kde	0	100	100	100.5799	1287.3795	0.8546	0.8620	35	0
wfdc	pca	2	uniform	1	100	100	88.6599	1048.6327	0.4834	0.5648	40	0
wfdc	pca	2	uniform	3	100	100	96.2278	1130.3268	0.8178	0.8750	22	1
wfdc	pca	2	uniform	5	100	100	98.8459	1106.9342	0.7572	0.9380	6	2
wfdc	pca	3	kde	0	100	100	92.0546	1421.9111	0.8841	0.9597	22	1
wfdc	pca	3	uniform	1	100	100	77.4343	1408.5405	0.8633	0.9047	32	1
wfdc	pca	3	uniform	3	100	100	97.0313	1350.2098	0.8033	0.8547	33	0
wfdc	pca	3	uniform	5	100	100	84.7304	1108.4397	0.8743	0.9146	23	1
wfdc	pca	4	kde	0	100	100	94.9381	1100.4921	0.9121	0.9628	34	0
wfdc	pca	4	uniform	1	100	100	90.7781	1472.9081	0.3333	0.3333	38	0
wfdc	pca	4	uniform	3	100	100	107.8601	1505.0698	0.8490	0.8929	8	0
wfdc	pca	4	uniform	5	100	100	100.9209	1531.0027	0.8905	0.9872	12	0
wfdc	pca	5	kde	0	100	100	109.7964	1510.2603	0.9220	0.9779	9	1
wfdc	pca	5	uniform	1	100	100	97.8863	1102.3763	0.3931	0.5633	41	0
wfdc	pca	5	uniform	3	100	100	102.8431	1113.3482	0.8522	0.9450	11	0
wfdc	pca	5	uniform	5	100	100	99.2440	1655.3066	0.8747	0.9052	30	0
wfdc	ica	2	kde	0	100	100	90.8003	1354.4054	0.6337	0.9222	19	0
wfdc	ica	2	uniform	1	100	100	90.1206	1413.7268	0.5333	0.8767	42	4
wfdc	ica	2	uniform	3	100	100	92.0035	1470.0996	0.7401	0.8667	25	0
wfdc	ica	2	uniform	5	100	100	88.4906	1482.1536	0.7412	0.9048	34	0
wfdc	ica	3	kde	0	100	100	98.9102	1486.1128	0.8486	0.8889	23	0
wfdc	ica	3	uniform	1	100	100	95.5440	1523.4632	0.3333	0.3333	45	0
wfdc	ica	3	uniform	3	100	100	104.9721	1435.4897	0.8262	0.8667	21	1
wfdc	ica	3	uniform	5	100	100	83.0884	1554.6687	0.8748	0.9648	34	0
wfdc	ica	4	kde	0	100	100	114.2071	1519.2974	0.8652	0.8819	36	1
wfdc	ica	4	uniform	1	100	100	91.9546	1130.7468	0.3333	0.3333	48	3
wfdc	ica	4	uniform	3	100	100	94.4082	1106.0188	0.8722	0.9667	29	0
wfdc	ica	4	uniform	5	100	100	98.4263	1120.7155	0.8994	0.9048	34	1
wfdc	ica	5	kde	0	100	100	103.6241	1458.0050	0.8966	0.9001	48	1
wfdc	ica	5	uniform	1	100	100	98.7154	1504.0362	0.3895	0.7261	51	5
wfdc	ica	5	uniform	3	100	100	109.3277	1616.8599	0.8773	0.9700	44	0
wfdc	ica	5	uniform	5	100	100	111.1713	1261.1105	0.9042	0.9248	42	0

Table C.8: Testing coverage of the Fashion-MNIST model for the **wfdc criterion** conducted with $|X_0| = 100$ initial tests. Header possible values are: tech: pca ica, num_feats: 2 3 4 5, discr: KDE U1 U3 U5. Note that the coverage measure calculated here is the one defined in Eq. 6.5

Bibliography

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology. Technical Report 610.12, IEEE, 1990.

[2] Amany Alshareef, Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Quantifying the importance of latent features in neural networks. In *CEUR Workshop Proceedings*, volume 3087, 2022.

[3] Amany Alshareef, Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Robust bayesian abstraction of neural networks. In *proceedings of the ICMLC 2023*, 2023.

[4] Amany Alshareef, Nicolas Berthier, Sven Schewe, and Xiaowei Huang. Weight-based semantic testing approach for deep neural networks. In *The IJCAI Workshop on Artificial Intelligence Safety 2023*, volume 3505, 2023.

[5] Pranav Ashok, Vahid Hashemi, Jan Křetínskỳ, and Stefanie Mohr. Deepabstract: neural network abstraction for accelerating verification. In *International Symposium on Automated Technology for Verification and Analysis*, pages 92–107. Springer, 2020.

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

[7] Yang Bai, Yuyuan Zeng, Yong Jiang, Shu-Tao Xia, Xingjun Ma, and Yisen Wang. Improving adversarial robustness via channel-wise activation suppressing. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=zQTezqCCtNx`.

[8] Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

[9] Nicolas Berthier, Amany Alshareef, James Sharp, Sven Schewe, and Xiaowei Huang. Abstraction and symbolic execution of deep neural networks with bayesian approximation of hidden features. *arXiv preprint arXiv:2103.03704*, 2021.

[10] Francesco Bodria, Fosca Giannotti, Riccardo Guidotti, Francesca Naretto, Dino Pedreschi, and Salvatore Rinzivillo. Benchmarking and survey of explanation methods for black box models. *arXiv preprint arXiv:2102.13076*, 2021.

[11] Emanuele Borgonovo and Elmar Plischke. Sensitivity analysis: A review of recent advances. *European Journal of Operational Research*, 248(3):869–887, 2016.

[12] Simon Burton, Lydia Gauerhof, and Christian Heinzemann. Making the case for safety of machine learning in highly automated driving. In *Computer Safety, Reliability, and Security: SAFECOMP 2017 Workshops, ASSURE, DECSoS, SASSUR, TELERISE, and TIPS, Trento, Italy, September 12, 2017, Proceedings 36*, pages 5–16. Springer, 2017.

[13] Igor Buzhinsky, Arseny Nerinovsky, and Stavros Tripakis. Metrics and methods for robustness evaluation of neural networks with generative models. *Machine Learning*, pages 1–36, 2021.

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE Computer Society, 2017.

[15] Enrique Castillo and Uffe Kjærulff. Sensitivity analysis in gaussian bayesian networks using a symbolic-numerical technique. *Reliability Engineering & System Safety*, 79 (2):139–148, 2003.

[16] Enrique Castillo, José Manuel Gutiérrez, and Ali S Hadi. Sensitivity analysis in discrete bayesian networks. *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, 27(4):412–423, 1997.

[17] Hei Chan and Adnan Darwiche. Sensitivity analysis in bayesian networks: From single to multiple parameters. *arXiv preprint arXiv:1207.4124*, 2012.

[18] Chih-Hong Cheng. Provably-robust runtime monitoring of neuron activation patterns. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1310–1313. IEEE, 2021.

[19] Chih-Hong Cheng, Chung-Hao Huang, and Hirotoshi Yasuoka. Quantitative projection coverage for testing ml-enabled autonomous systems. In *Automated Technology for Verification and Analysis: 16th International Symposium, ATVA 2018, Los Angeles, CA, USA, October 7-10, 2018, Proceedings 16*, pages 126–142. Springer, 2018.

[20] Chih-Hong Cheng, Georg Nührenberg, and Hirotoshi Yasuoka. Runtime monitoring neuron activation patterns. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 300–303. IEEE, 2019.

[21] Federica Cugnata, Ron S Kenett, and Silvia Salini. Bayesian networks in survey data: Robustness and sensitivity issues. *Journal of Quality Technology*, 48(3):253–264, 2016.

[22] Darren Dancey, David A McLean, and Zuhair A Bandar. Decision tree extraction from trained neural networks. American Association for Artificial Intelligence, 2004.

[23] Adnan Darwiche. *Modeling and reasoning with Bayesian networks*. Cambridge university press, 2009.

[24] Erik Daxberger, Eric Nalisnick, James U Allingham, Javier Antorán, and José Miguel Hernández-Lobato. Bayesian deep learning via subnetwork inference. In *International Conference on Machine Learning*, pages 2510–2521. PMLR, 2021.

[25] Terrance DeVries and Graham W Taylor. Learning confidence for out-of-distribution detection in neural networks. *arXiv preprint arXiv:1802.04865*, 2018.

[26] Guoliang Dong, Jingyi Wang, Jun Sun, Yang Zhang, Xinyu Wang, Ting Dai, Jin Song Dong, and Xingen Wang. Towards interpreting recurrent neural networks through probabilistic abstraction. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 499–510. IEEE, 2020.

[27] Yizhen Dong, Peixin Zhang, Jingyi Wang, Shuang Liu, Jun Sun, Jianye Hao, Xinyu Wang, Li Wang, Jin Song Dong, and Dai Ting. There is limited correlation between coverage and robustness for deep neural networks. *arXiv preprint arXiv:1911.05904*, 2019.

[28] Ruth Fong, Mandela Patrick, and Andrea Vedaldi. Understanding deep networks via extremal perturbations and smooth masks. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 2950–2958, 2019.

[29] Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, pages 1050–1059. PMLR, 2016.

[30] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2018.

[31] Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. *Advances in Neural Information Processing Systems*, 34: 9574–9586, 2021.

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

[33] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.

[34] Miguel A Gómez-Villegas, Paloma Maín, and Rosario Susi. Sensitivity analysis in gaussian bayesian networks using a divergence measure. *Communications in Statistics—Theory and Methods*, 36(3):523–539, 2007.

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

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

[37] Riccardo Guidotti, Anna Monreale, Stan Matwin, and Dino Pedreschi. Black box explanation by learning image exemplars in the latent feature space. In *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2019, Würzburg, Germany, September 16–20, 2019, Proceedings, Part I*, pages 189–205. Springer, 2020.

[38] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. Dlfuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 739–743, 2018.

[39] Abdullah Hamdi and Bernard Ghanem. Towards analyzing semantic robustness of deep neural networks. In *European Conference on Computer Vision*, pages 22–38. Springer, 2020.

[40] Thomas A Henzinger, Anna Lukina, and Christian Schilling. Outside the box: Abstraction-based monitoring of neural networks. *arXiv preprint arXiv:1911.09032*, 2019.

[41] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[42] Yaojie Hu and Jin Tian. Neuron dependency graphs: A causal abstraction of neural networks. In *International Conference on Machine Learning*, pages 9020–9040. PMLR, 2022.

[43] Wei Huang, Youcheng Sun, Xingyu Zhao, James Sharp, Wenjie Ruan, Jie Meng, and Xiaowei Huang. Coverage-guided testing for recurrent neural networks. *IEEE Transactions on Reliability*, pages 1–16, 2021. doi: 10.1109/TR.2021.3080664.

[44] Xiaowei Huang, Marta Kwiatkowska, Sen Wang, and Min Wu. Safety verification of deep neural networks. In *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*, pages 3–29. Springer, 2017.

[45] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review*, 37, 2020. ISSN 1574-0137.

[46] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. A survey of safety and trustworthiness of deep neural networks. *Computer Science Survey*, 2020.

[47] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[48] Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Logan Engstrom, Brandon Tran, and Aleksander Madry. Adversarial examples are not bugs, they are features. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/e2c420d928d4bf8ce0ff2ec19b371514-Paper.pdf`.

[49] Gaojie Jin, Xinping Yi, Liang Zhang, Lijun Zhang, Sven Schewe, and Xiaowei Huang. How does weight correlation affect generalisation ability of deep neural networks? *Advances in Neural Information Processing Systems*, 33:21346–21356, 2020.

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

[51] Matthew Kirk. *Thoughtful machine learning: A test-driven approach.* " O'Reilly Media, Inc.", 2014.

[52] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Being bayesian, even just a bit, fixes overconfidence in relu networks. In *International conference on machine learning*, pages 5436–5446. PMLR, 2020.

[53] Agustinus Kristiadi, Matthias Hein, and Philipp Hennig. Learnable uncertainty under laplace approximations. In *Uncertainty in Artificial Intelligence*, pages 344–353. PMLR, 2021.

[54] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. Cifar-10 (canadian institute for advanced research). URL `http://www.cs.toronto.edu/~kriz/cifar.html`.

[55] Alex Labach, Hojjat Salehinejad, and Shahrokh Valaee. Survey of dropout methods for deep neural networks. *arXiv preprint arXiv:1904.13310*, 2019.

[56] Carlos Lassance, Vincent Gripon, and Antonio Ortega. Representing deep neural networks latent space geometries with graphs. *Algorithms*, 14(2):39, 2021.

[57] JA Lee. A global geometric framework for non-linear dimensionality reduction. In *Proceedings of the 8th European symposium on artificial neural networks, 2000*, volume 1, pages 13–20, 2000.

[58] Kimin Lee, Kibok Lee, Honglak Lee, and Jinwoo Shin. A simple unified framework for detecting out-of-distribution samples and adversarial attacks. *Advances in neural information processing systems*, 31, 2018.

[59] Jianlin Li, Jiangchao Liu, Pengfei Yang, Liqian Chen, Xiaowei Huang, and Lijun Zhang. Analyzing deep neural networks with symbolic propagation: Towards higher precision and faster verification. In *International static analysis symposium*, pages 296–319. Springer, 2019.

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

[61] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30, 2017.

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

[63] Divyam Madaan, Jinwoo Shin, and Sung Ju Hwang. Adversarial neural pruning with latent vulnerability suppression. In *International Conference on Machine Learning*, pages 6575–6585. PMLR, 2020.

[64] Wesley J Maddox, Pavel Izmailov, Timur Garipov, Dmitry P Vetrov, and Andrew Gordon Wilson. A simple baseline for bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[65] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks, 2017.

[66] Tanwi Mallick, Prasanna Balaprakash, and Jane Macfarlane. Deep-ensemble-based uncertainty quantification in spatiotemporal graph neural networks for traffic forecasting. *arXiv preprint arXiv:2204.01618*, 2022.

[67] Tim Miller. Explanation in artificial intelligence: Insights from the social sciences. *Artificial intelligence*, 267:1–38, 2019.

[68] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[69] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[70] Hasna Njah, Salma Jamoussi, and Walid Mahdi. Interpretable bayesian network abstraction for dimension reduction. *Neural Computing and Applications*, pages 1–19, 2022.

[71] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*, pages 4901–4911. PMLR, 2019.

[72] Kazuki Osawa, Siddharth Swaroop, Mohammad Emtiyaz E Khan, Anirudh Jain, Runa Eschenhagen, Richard E Turner, and Rio Yokota. Practical deep learning with bayesian principles. *Advances in neural information processing systems*, 32, 2019.

[73] 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.

[74] Vitali Petsiuk, Abir Das, and Kate Saenko. Rise: Randomized input sampling for explanation of black-box models. *arXiv preprint arXiv:1806.07421*, 2018.

[75] Konstantin Posch, Jan Steinbrener, and Jürgen Pilz. Variational inference to measure model uncertainty in deep neural networks. *arXiv preprint arXiv:1902.10189*, 2019.

[76] Luca Pulina and Armando Tacchella. An abstraction-refinement approach to verification of artificial neural networks. In *International Conference on Computer Aided Verification*, pages 243–257. Springer, 2010.

[77] Arvind Ramanathan, Laura L Pullum, Faraz Hussain, Dwaipayan Chakrabarty, and Sumit Kumar Jha. Integrating symbolic and statistical methods for testing intelligent systems: Applications to machine learning and computer vision. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 786–791. IEEE, 2016.

[78] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[79] Hippolyt Ritter, Aleksandar Botev, and David Barber. A scalable laplace approximation for neural networks. In *6th International Conference on Learning Representations, ICLR 2018-Conference Track Proceedings*, volume 6. International Conference on Representation Learning, 2018.

[80] Raanan Y Rohekar, Shami Nisimov, Yaniv Gurwicz, Guy Koren, and Gal Novik. Constructing deep neural networks by bayesian network structure learning. *Advances in Neural Information Processing Systems*, 31, 2018.

[81] Wenjie Ruan, Xiaowei Huang, and Marta Kwiatkowska. Reachability analysis of deep neural networks with provable guarantees. *arXiv preprint arXiv:1805.02242*, 2018.

[82] Makoto Sato and Hiroshi Tsukimoto. Rule extraction from neural networks via decision tree induction. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, volume 3, pages 1870–1875. IEEE, 2001.

[83] Bernhard Schölkopf, Alexander Smola, and Klaus-Robert Müller. Kernel principal component analysis. In *International conference on artificial neural networks*, pages 583–588. Springer, 1997.

[84] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[85] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *International conference on machine learning*, pages 3145–3153. PMLR, 2017.

[86] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[87] Youcheng Sun, Xiaowei Huang, and Daniel Kroening. Testing deep neural networks. *CoRR*, abs/1803.04792, 2018. URL `http://arxiv.org/abs/1803.04792`.

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

[89] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. Concolic testing for deep neural networks. In *ASE*, page 109–119, 2018.

[90] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. Structural test coverage criteria for deep neural networks. *ACM Trans. Embed. Comput. Syst.*, 18(5s), October 2019. ISSN 1539-9087. doi: 10.1145/3358233. URL `https://doi.org/10.1145/3358233`.

[91] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *International conference on machine learning*, pages 3319–3328. PMLR, 2017.

[92] 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.

[93] Muhammad Usman, Youcheng Sun, Divya Gopinath, Rishi Dange, Luca Manolache, and Corina S Păsăreanu. An overview of structural coverage metrics for testing neural networks. *International Journal on Software Tools for Technology Transfer*, pages 1–13, 2022.

[94] Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao, Cho-Jui Hsieh, and Luca Daniel. Evaluating the robustness of neural networks: An extreme value theory approach. *arXiv preprint arXiv:1801.10578*, 2018.

[95] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems: 24th International Conference, TACAS 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, Part I 24*, pages 408–426. Springer, 2018.

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

[97] Qiuling Xu, Guanhong Tao, Siyuan Cheng, and Xiangyu Zhang. Towards feature space adversarial attack by style perturbation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10523–10531, 2021.

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

[99] Dong Yu, Li Deng, Dong Yu, and Li Deng. Feature representation learning in deep neural networks. *Automatic Speech Recognition: A Deep Learning Approach*, pages 157–175, 2015.

[100] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*, pages 818–833. Springer, 2014.

[101] Cheng Zhang, Kun Zhang, and Yingzhen Li. A causal view on robustness of neural networks. *Advances in Neural Information Processing Systems*, 33:289–301, 2020.

[102] Xiaoqin Zhang, Jinxin Wang, Tao Wang, Runhua Jiang, Jiawei Xu, and Li Zhao. Robust feature learning for adversarial defense via hierarchical feature alignment. *Information Sciences*, 560:256–270, 2021.

[103] Xingyu Zhao, Alec Banks, James Sharp, Valentin Robu, David Flynn, Michael Fisher, and Xiaowei Huang. A safety framework for critical systems utilising deep neural net-

works. In *Computer Safety, Reliability, and Security: 39th International Conference, SAFECOMP 2020, Lisbon, Portugal, September 16–18, 2020, Proceedings 39*, pages 244–259. Springer, 2020.

[104] Xingyu Zhao, Wei Huang, Alec Banks, Victoria Cox, David Flynn, Sven Schewe, and Xiaowei Huang. Assessing the reliability of deep learning classifiers through robustness evaluation and operational profiles. *arXiv preprint arXiv:2106.01258*, 2021.

[105] Xingyu Zhao, Wei Huang, Vibhav Bharti, Yi Dong, Victoria Cox, Alec Banks, Sen Wang, Sven Schewe, and Xiaowei Huang. Reliability assessment and safety arguments for machine learning components in assuring learning-enabled autonomous systems, 2021.

[106] Xingyu Zhao, Wei Huang, Xiaowei Huang, Valentin Robu, and David Flynn. Baylime: Bayesian local interpretable model-agnostic explanations. In *Uncertainty in artificial intelligence*, pages 887–896. PMLR, 2021.

[107] Xingyu Zhao, Xiaowei Huang, Valentin Robu, and David Flynn. BayLIME: Bayesian Local Interpretable Model-Agnostic Explanations. In *UAI*, 2021.

[108] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2921–2929, 2016.