Distribution-Aware Testing of Neural Networks Using Deep Generative Models

---

A Dissertation

Presented to

the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment

of the requirements for the Degree

Doctor of Philosophy (Computer Engineering)

by

Swaroopa Dola

August 2023

Acknowledgements

I would like to express my deepest gratitude to my advisors, Dr. Mary Lou Soffa and Dr. Matthew B. Dwyer. Your guidance and feedback helped me grow as a researcher and become confident in my work. I'm grateful for the patience you have shown throughout my dissertation.

My sincere thanks to the dissertation committee members, Dr. Sebastian Elbaum, Dr. Yangfeng Ji, and Dr. Tom Fletcher, who have provided their valuable time and feedback for this dissertation. I am also thankful to Dr. Andrew Grimshaw, who guided me during the initial years of my Ph.D. I would like to acknowledge the members of the LESS Lab at the University of Virginia for the feedback they provided on my research and presentations.

I am grateful to my parents and my brother who have encouraged me at every stage of my life. Special thanks to all my friends for their wonderful support throughout my grad school. For my husband Praveen, thank you for always believing in me and motivating me to pursue my goals. I could not have undertaken this journey without your support! Finally, for my son Rishi, you always made my worries disappear with your beautiful smile and amazing conversations.

This work was supported, in part, by the National Science Foundation under awards 2019239 and 212982.

Abstract

The reliability of software with a Deep Neural Network (DNN) as a component is urgently important today given the increasing number of critical applications deployed with DNNs. The need for reliability raises a need for rigorous testing of the safety and trustworthiness of these systems. In the last few years, there have been a number of research efforts focused on testing DNNs. However, the existing methods have several limitations. The methods that test DNNs with pixel-level manipulations on seed inputs do not consider the validity of the inputs which results in invalid inputs wasting testing time and inflating coverage values. The methods that use feature-level manipulations avoid invalid inputs. However, they either lack a mechanism to automatically learn the features which makes them costly and infeasible for complex datasets or they do not systematically test DNNs with feature-diverse and rare inputs because of which the methods are inadequate to ensure the reliability of DNNs. This thesis advances the state of DNN testing by proposing a DNN testing framework, DisTest, that addresses all these limitations. A black-box test adequacy criterion, IDC, and a black-box test generation algorithm, CIT4DNN, are developed over the framework for measuring the feature diversity of test inputs and generating diverse and rare test inputs respectively. Experimental studies demonstrate that the test coverage measured by IDC reflects the feature interactions present in a test set and is more sensitive to feature diversity than the baseline test coverage metrics. CIT4DNN generates test inputs with 100% test adequacy and is cost-effective when compared to the baseline test generation algorithms.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Deep Neural Networks (DNN) are increasingly being deployed in mission and safety critical systems, e.g., [8, 100, 120, 82]. Similar to traditional *programmed* software components, these *learned* DNN components require significant testing to ensure that they are reliable and thus fit for deployment.

A DNN learns to approximate an unknown target function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ over a data distribution $\mathcal{X}$, by training on the observed data that is representative of $\mathcal{X}$. The DNN is expected to make accurate predictions on valid inputs, the inputs that are drawn from $\mathcal{X}$. The input domain of a DNN $\mathcal{I} = \mathbb{R}^n$ is often high dimensional; for example, the CIFAR10 dataset has 3072 dimensions [70]. However, the realistic or valid data is distributed on a manifold of low-dimension, d, within the high dimensional input domain, $d \ll n$ [37]. This concept is demonstrated in Figure 1.1 using the Swiss roll dataset [14]. The Swiss roll dataset is a 3-D dataset. However, the valid data can be represented in a 2-D space by unrolling the dataset. Inputs sampled from the 3-D space might not lie on the data manifold resulting in invalid inputs; the points marked in grey on the plot. As a result, testing methods that consider the whole input domain $\mathcal{I}$ risk overestimating the valid inputs, which can be inefficient and misleading.

Many DNN testing methods do not check the validity of the inputs used in testing [99, 129, 47, 125, 95, 144]. These methods test DNNs with both valid and invalid inputs. Figure 1.2 depicts valid test inputs and selected invalid test inputs from two DNN test generation techniques [99, 125]. Not distinguishing between valid and invalid input can be problematic for DNN testing and presents two challenges. The first challenge, *(C1)*, is that the *testing techniques that generate invalid inputs increase cost with little value added for testing the functional logic of DNNs.* When a test case fails,

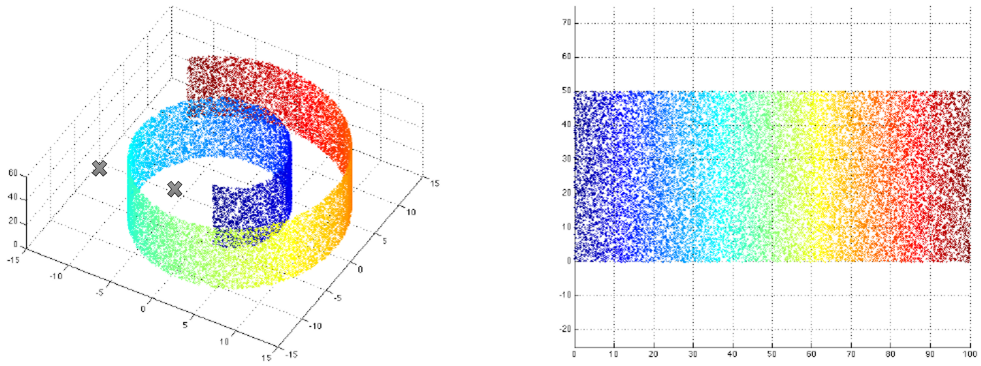


Figure 1.1: The Swiss roll dataset [14]. The left plot shows that dataset that is 3-D. The right plot shows the data unrolled in a 2-D space. The points marked in grey on the left plot are invalid inputs as they do not lie on the data manifold.

developer time is required to triage the failure. A high rate of invalid inputs runs the risk that developers will avoid using these techniques, thereby negating their purported value. Whereas for traditional software, the coverage produced by invalid inputs is confined to the input validation and error logic, for DNNs an analogous separation of coverage is not guaranteed. This results in the second challenge, *(C2) – invalid tests can artificially boost coverage thereby providing a false notion of adequacy to the developers.* These challenges highlight the importance of including input data distribution in the testing process for DNNs.

DNN testing methods that apply pixel-level manipulations on the seed images to generate test inputs, e.g., [99, 129, 47, 135, 145] do not vary the features of $\mathcal{X}$ [155, 35]. Features are the underlying factors of variation of $\mathcal{X}$, combinations of which give rise to the diversity in the data. For example, for an input domain including handwritten zeros, stroke thickness, and slant are two possible features. A combination of these two features results in samples with varying stroke thickness and slant as shown in Figure 1.3. Methods that do not exercise the feature space systematically present a challenge to DNN testing, *(C3) – the methods that do not adequately explore the feature space risk missing inputs on which the DNN has not learned to make accurate predictions* [59]. DeepHyperion and other research that aims to generate feature-diverse inputs rely on manual expertise to construct a model of the input features [155, 40, 20]. *Manually identifying the features of the input data is costly and challenging for complex datasets*, and this research refers to this challenge as *(C4)*. For example, ImageNet [33] dataset has 1000 categories and manually identifying the features and constructing a model to vary the features is not practical.

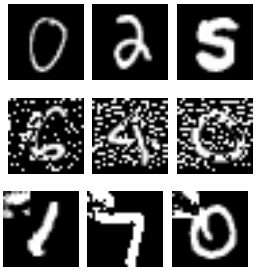


Figure 1.2: Valid tests vs Invalid tests. Top Row: Valid tests from MNIST training dataset. Middle Row: Invalid tests from DeepConcolic. Bottom Row: Invalid tests from DeepXplore

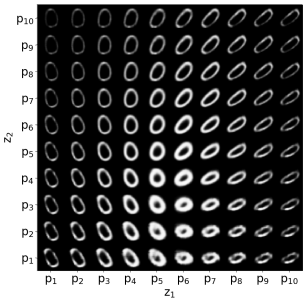


Figure 1.3: Diversity of digit 0 of MNIST with respect to variation in the slant and stroke thickness features.

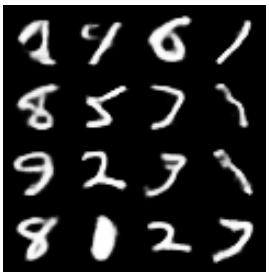


Figure 1.4: Rare inputs sampled from the MNIST data distribution.

Hence research methods should develop automatic strategies to test DNNs with diverse inputs and formulate test adequacy over the feature interactions to systematically cover the feature space.

Feature combinations occur in real-world data with varying probabilities and inputs with low probability of occurrence are referred to as rare inputs. For example, Figure 1.4 shows rare input sampled from the MNIST data distribution. While existing methods might handle rare inputs, they do not systematically explore the target density of rare inputs and do not achieve feature interaction test adequacy [61, 155]. This results in the challenge, *(C5) – Failing to test the system behavior using the systematic exploration of the rare inputs can make deep learning systems unreliable*. For example, in a fatal Tesla crash, the Autopilot was not able to detect the white side of a tractor-trailer against a brightly lit sky and a GM autonomous vehicle crashed into a bus in this rare circumstance [1, 41]. There is a need for testing methods to include systematic coverage of the target density of the rare inputs in the feature space to ensure the reliability of DNNs.

This research proposes distribution-aware DNN testing to address challenges *(C1)-(C5)* faced by the existing DNN testing approaches. First, the research demonstrates the limitations of the existing methods with respect to invalid inputs and shows how including the data distribution in the test generation can address these limitations. Then a framework, *DisTest* is proposed that includes a model of the data distribution and contains building blocks for adequately testing DNNs with valid, diverse, and rare inputs. Next, this research using the components of DisTest proposes a black-box test adequacy metric, *IDC*, and a black-box test generation algorithm, *CIT4DNN*.

Finally, empirical evaluation is used to demonstrate the effectiveness of IDC and CIT4DNN.

Developing DisTest has several challenges. The framework requires 1) a representation of the feature space of the training data that can be formulated without manual expertise, 2) a mechanism to systematically explore the feature space in order to adequately test with diverse inputs, and 3) a method to use low-probability feature combinations for testing with rare inputs. DisTest meets these challenges by using a generative model to automatically learn a low-dimensional representation of the features of the training data, called the *latent space*. DisTest contains combinatorial interaction testing (CIT) tools for systematically exploring the latent space to test DNNs with feature-diverse inputs [28]. Rare inputs are spread out on the low-density regions of the latent space. DisTest provides a representation of *target density* on the latent space to support testing with rare inputs.

IDC applies CIT on a target density of the partitioned latent space to measure test coverage proportional to the t-way feature combinations present in a test set. While IDC measures test coverage, CIT4DNN applies constrained CIT on the partitioned latent space to generate test inputs that cover the t-way feature combinations of a target density resulting in test diversity and rare inputs.

Thesis Statement: *A DNN testing framework equipped with a model of the input data distribution can be developed to address the challenges related to the validity, diversity, and rareness of the inputs used in DNN testing.*

The primary contributions of this work are:

1. demonstration of limitations of existing DNN testing methods in their treatment of invalid inputs using empirical studies,

2. a technique to guide gradient-based testing algorithm to include valid inputs in testing,

3. DisTest, a framework for incorporating input data distribution in the testing process and using it as a domain for testing DNNs,

4. IDC, a black-box test adequacy criterion for measuring feature interaction adequacy of DNNs, and experimental evaluation that demonstrates the sensitivity of IDC to feature diversity in test sets,

5. CIT4DNN, a black-box test generation method for generating diverse, and rare test inputs

that systematically cover a target density of the input distribution, and experimental evaluation that demonstrates the effectiveness of CIT4DNN.

The remainder of this dissertation is organized as follows. Chapter 2 describes the concepts discussed in this work including deep neural networks, generative models, and combinatorial interaction testing. Chapter 3 demonstrates the limitations of the existing DNN testing methods, and Chapter 4 presents the framework, DisTest, for testing DNNs using the input data distribution. Chapter 5 and Chapter 6 discuss the two applications developed over DisTest, IDC and CIT4DNN, and experimental evaluation demonstrating their effectiveness. Chapter 7 presents the related literature and Chapter 8 concludes the work and provides possible future research directions.

Chapter 2

Background

This chapter presents the concepts, notations, and combinatorial interaction testing metrics used in this research.

2.1 Deep Neural Networks

A neural network, $\mathcal{N} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, is trained to make predictions using a corpus of training data, $(\boldsymbol{x}, \boldsymbol{y}) \in T$, where training inputs, $\boldsymbol{x} \in \mathbb{R}^m$, are associated with training labels, $\boldsymbol{y} \in \mathbb{R}^n$, that define the desired definition of $\mathcal{N}$ [43]. Networks are a directed graph of *neurons* with trainable parameters associated with the incoming edges to a neuron and a non-linear activation function applied on the outgoing edge of the neuron.

The training process repeatedly selects a training instance, $t \in T$, and uses training loss, $||\mathcal{N}(\pi_1(t)) - \pi_2(t)||$, to modify parameters, where $\pi_i(\cdot)$ returns the ith element of a tuple. Training continues either for a fixed duration or until a specific criterion on the history of the training loss is met. A trained network is evaluated to determine its capability to generalize to the unknown distribution from which the training input data are drawn, $\pi_1(T) \sim \mathcal{X}$, by computing *test accuracy* of the network over a set of labeled instances that is disjoint from T.

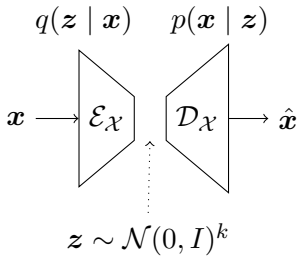


Figure 2.1: Variational Auto-encoder

2.2 Learned Feature Representations

Any cut through the graph comprising a trained $\mathcal{N}$ defines a function from $\mathbb{R}^m$ to some $\mathbb{R}^c$, where c is the number of neurons incident to the cut-set. Mapping inputs to $\mathbb{R}^c$ through such a function yields a low-dimensional representation of an input. A network has the capacity to learn many such representation functions and the process of training adjusts the parameters to produce representations from which accurate prediction of training labels is possible, thereby minimizing training loss. The learned representations of training inputs, $\pi_1(T)$, are determined in large part by the training labels, $\pi_2(T)$. For example, input data from a forward facing automobile camera might be used to train models that predict the probability of collision, the throttle and steering angle to stay within a lane, or the presence of a pedestrian by a roadside. Training such networks on identical inputs will result in different learned representations, since the features of those inputs that are relevant for each of these prediction problems vary.

Researchers have explored a range of techniques for learning representations of input data that are unbiased by training labels. Autoencoders are defined as the composition of an encoder, $\mathcal{E} : \mathbb{R}^m \to \mathbb{R}^k$, and a decoder, $\mathcal{D} : \mathbb{R}^k \to \mathbb{R}^m$, networks that are trained to minimize reconstruction loss, $||\boldsymbol{x} - \mathcal{D} \circ \mathcal{E}(\boldsymbol{x})||$ [43]. A trained encoder defines a k-dimensional representation function and its associated decoder approximates the inverse of that function. For an accurately trained autoencoder, the learned latent k-dimensional representation must do an adequate job of representing the features present in the training inputs. If it did not, then recovery accuracy would be low.

A variational autoencoder (VAE) resembles an autoencoder in that it composes an encoder network, $\mathcal{E}_{\mathcal{X}} : \mathbb{R}^m \to \mathbb{R}^k$, with a decoder trained to minimize recovery error, $||\boldsymbol{x} - \mathcal{D}_{\mathcal{X}} \circ \mathcal{E}_{\mathcal{X}}(\boldsymbol{x})||$ [68]. Despite this similarity, VAEs are designed to perform variational inference which demands a struc-

ture on the learned latent space. In particular, each of the k latent dimensions consists of a pair, (μ, σ), which define the parameters of a Normal distribution. As depicted in Figure 2.1, the encoder learns to map inputs to the parameters of a k-dimensional multivariate Normal posterior distribution, $q(\boldsymbol{z} \mid \boldsymbol{x})$. The decoder learns the dual problem of mapping points in the latent distribution, $\mathcal{N}(0,1)^k$, to the parameters of an input distribution, $p(\boldsymbol{x}|\boldsymbol{z})$, from which recovered inputs, $\hat{x}$, can be sampled. The VAE is trained by maximizing the likelihood of the input data $p(\boldsymbol{x}|\boldsymbol{z})$ and minimizing the Kullback-Leibler (KL) divergence between the encoder distribution and the prior distribution, $KL(q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$; the latter is commonly assumed to be a standard Normal distribution. The KL divergence metric measures how different two distributions are, so minimizing this metric would result in the encoder distribution closely following the prior distribution. The VAE training objective is:

$$\frac{1}{N}\sum_{i=1}^{N}[\mathbb{E}_{q(z|x^{(i)})}[\log p(x^{(i)}|z)] - KL(q(z|x^{(i)})||p(z))] \tag{2.1}$$

Like any neural network, a VAE only learns to approximate a target function. Training may yield a VAE that imperfectly recovers inputs or that imperfectly matches the assumed prior. Recent work has investigated a number of architectural and training strategies that improve VAEs in terms of their accuracy – as judged by the sharpness of images generated using them, the lack of mode collapse, and their ability to match a prior. There are several variations of VAEs proposed in literature including β-VAE, FactorVAE, Total Correlation VAE (β-TCVAE), Two-Stage VAE, and σ-VAE with different objectives including learning disentangled representations, reducing the distance between the posterior and the prior distributions, and increasing the quality of the reconstructions [55, 64, 21, 32, 110].

2.2.1 Metrics

There is strong empirical evidence that generative models, such as VAEs, can learn rich and comprehensive representations of complex input spaces. For example, researchers have explored the capability of such models to learn the features present in human faces and to be able to combine those features to produce realistic images not in the training data [62]. Measuring input realism is challenging, but a number of metrics have been developed for images, such as Fréchet-Inception

Distance (FID) [54] and Inception Score (IS) [17].

The Fréchet-Inception Distance is the Fréchet distance of two distributions taken from a lower-dimensional "embedding" space [38]; the embedding used is Inception v3 [54]. FID has established itself as the de-facto standard metric for both quality and diversity despite having a significant number of known shortcomings including high bias and sensitivity to test set sizes as well as varying greatly with different underlying implementations and image preparation methods [10].

A critical issue with FID and related single-valued metrics is their inability to differentiate a lack of diversity from other failure modes. A recent line of work introduces a 2-dimensional metric, *Density and Coverage* [90]. Similar to FID, Density and Coverage use an embedding space, but instead of measuring the distribution distance, they measure how often generated points (in the embedding space) occur near real points, where "near" means within the manifold created by taking the k-nearest-neighbor ball of each point in the original dataset. Intuitively, Coverage is the percentage of real points that have at least one generated point near them. Density follows as the rate at which generated points are near-real points.

2.3 Out-of-Distribution Input Detection

Test inputs that do not belong to the training data distribution are referred to as out-of-distribution (OOD) inputs. OOD detection is an active research area in Machine Learning [143, 105, 113, 79, 53, 3]. There are classifier-based and generative model based approaches. The classifier-based approaches require labeled data where as generative model based approaches can train in an unsupervised manner and hence avoid the labeling cost. The classifier-based approaches, such as Baseline and ODIN, use softmax scores of the classifiers for classifying OOD inputs [52, 79]. The generative model-based approaches, such as Likelihood Regret and Input-Complexity, use the likelihood of the test inputs estimated by the generative models for classifying OOD inputs [143, 113].

2.4 Combinatorial Interaction Testing

Thorough black-box testing of software with a high-dimensional input space is challenging. Applying methods such as category-partitioning [96] to construct a finite partition for each input helps

to a degree, but the combinatorics generally preclude complete coverage of input-partition combinations. For example, for a system with 10 variables and 5 partitions for each variable, a total of $5^{10} = 9765625$ test cases are required to cover the full combinatorial space. Combinatorial Interaction Testing (CIT) focuses testing on a combinatorial sub-space within which it can guarantee systematic coverage [28, 30, 71]. CIT methods have been used to both generate test suites that achieve a desired strength and to measure the strength of a given test to define coverage criteria [28, 72].

CIT when used in test generation, produces test suites that systematically cover a partitioned input space up to a user-specified arity, t, which is referred to as *combinatorial strength* [28]. Even with coarse-granularity partitions, these methods have been shown effective in exposing faults arising from the interactions between different inputs taking on different values [131]. A CIT test set is formulated as a covering array where each row corresponds to a test, each column corresponds to a *factor*, and the values in a cell are the *levels* of the factors. A row constitutes a test description defining a possible set of values, defined by the partitions, for each input. A t-way covering array guarantees that all t-tuples of factor-level combinations are present in some row of the array and thereby assures that any interaction among inputs up to t has a chance to be exposed by tests generated from the CA. For example, 1-way coverage means that within some row, each factor takes on each of its possible levels; this implies that the number of rows in the array is at least as large as the number of levels. Consider a testing software that uses two inputs $z_1, z_2 \in [-6, 6]$ that abstracts the handwritten digits as shown in Figure 1.3. The figure shows each input partitioned into 10 intervals $\{p_1, p_2, .., p_{10}\}$. A 1-way CA for this configuration results in 10 test descriptions, $\{(p_i, p_j) : i = j \ \ \& \ \ i, j \in [1, 10]\}$, that cover all the diagonal inputs whereas a 2-way CA for this results in 100 test descriptions, $\{(p_i, p_j) : i, j \in [1, 10]\}$, that cover all the inputs shown in the Figure 1.3.

The bulk of research over the past decades has focused on how to efficiently generate CIT test sets, for different values of t, given a factor-level model of a system. Researchers have also explored how the principles of CIT can be applied to define test coverage metrics. Maximoff et al proposed combinatorial interaction test coverage metrics for measuring the test coverage of existing test suites, including *total t-way coverage* which is also referred to as *Total variable-value configuration coverage* [89, 72]. The *total t-way coverage* metric is used in this research.

Definition 1 (Total variable-value configuration coverage [72])**.** *For a given combination of t variables, total variable-value configuration coverage is the proportion of variable-value configurations that are covered by at least one test case in a test set. This measure may also be referred to as total t-way coverage.*

As described above, CIT assumes that there are no constraints among factor-level pairs. In many systems, however, constraints arise naturally, e.g., that a car cannot be in gear unless the key is in the ignition. This presents challenges both to CIT-based test generation and coverage metrics. Researchers have developed methods for generating constrained CIT test suites that efficiently incorporate constraints expressed as propositional logic formulae by exploiting data computed by modern SAT solvers. Cohen et al. defined a general framework that permits propositional constraints over input-partition combinations to be incorporated into greedy CIT test input generation methods [29]. The resulting constrained CIT (CCIT) approach efficiently generates a *constrained covering array*, which is a CA whose rows are consistent with the constraints.

Chapter 3

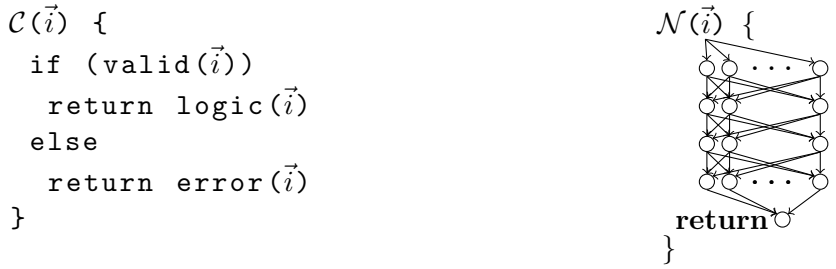
DNN Testing Using Input Data Distribution

This chapter [1] focuses on the challenges that DNN generalization presents to testing, and in particular, how current DNN testing techniques treat valid and invalid inputs. To understand these challenges, consider the implementation of a traditional software component $\mathcal{C}$, which is developed to meet a specification $\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m \cup e$, where e denotes the error behavior intended for *invalid inputs*. In this setting, the input domain $\mathbb{R}^n$ is partitioned into valid inputs, V, and invalid inputs, $\overline{V} = \mathbb{R}^n - V$, which should yield e.

The testing of $\mathcal{C}$ selects a test set $T \subset \mathbb{R}^n$ and assesses whether $\forall t \in T : \mathcal{C}(t) = \mathcal{S}(t)$. As sketched in Figure 3.1a, typically $\mathcal{C}$ is comprised of *input validation*, which determines if an input value lies in $\overline{V}$ and then executes either *functional logic* which realizes the behavior of $\mathcal{S}$ on V or *error processing* for invalid data. Developers have come to rely on several intuitions about such software. First, input validation logic is distinct from functional logic, demanding testing approaches that exploit its properties [50, 77, 80, 128] to effectively support it [118, 154, 42]. Second, *test suites that achieve higher coverage are better* in that they exercise more of the validation, functional, and error logic.

Now, consider a DNN, $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, which is trained to accurately approximate the specifica-

[1] This chapter is adapted from *Dola, Swaroopa, Matthew B. Dwyer, and Mary Lou Soffa. "Distribution-aware testing of neural networks using generative models." 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE). IEEE, 2021.*



(a) Code (b) DNN

Figure 3.1: Structure of code and DNN components $\mathcal{C}$ and $\mathcal{N}$.

tion, $\mathcal{S}$, possibly unavailable . As sketched in Figure 3.1b, $\mathcal{N}$ is comprised of layers of neurons that are cross-coupled by connections labeled with learned parameters. When the learned parameters for $\mathcal{N}$ are such that $Pr(\mathcal{N}(i) = \mathcal{S}(i) \mid i \in V) \geq 1 - \epsilon$, for a desired error ϵ, the network is expected to generalize to the valid input distribution, V. Even if $\mathcal{N}$ were trained to detect invalid data and respond appropriately, its structure does not force a distinction between input validation, functional logic, or error processing. In practice, this distinction is uncommon and in this case, $\mathcal{N}$ does not even have an analog for e in its output domain. Because of the lack of this distinction, whether an input lies in V or $\overline{V}$, the computation performed by $\mathcal{N}$ overlaps to a large degree, e.g., common sets of neurons are activated. As a result, Invalid inputs can inflate coverage thereby providing a false notion of test adequacy to the developers. Hence, *DNN test suites that achieve higher coverage are not necessarily better!* Also, when the testing uses a high number of invalid inputs it increases the testing cost without adding much value and wastes developer time spent in debugging the failures.

This research studies the effects of DNN test generation techniques not distinguishing between valid and invalid data and characterizes the potential impact of the issues identified above. The approach leverages a growing body of research from the Machine Learning (ML) community that learns models of the training distribution, V, from which the training data is drawn [3, 147, 151, 19]. While there are many such models, this work employs the *variational auto-encoder* (VAE), leaving the study of alternative models to future work.

Leveraging VAE models, this research studies the techniques representative of the current state of DNN testing research to make two important observations. First, the existing DNN testing techniques, such as DeepXplore [99], DLFuzz [47], and DeepConcolic [125], produce large numbers

of test cases with invalid inputs, which increases test cost without a clear benefit as claimed by the challenge *(C1)*. Second, the existing DNN test coverage metrics, e.g., [99, 84], are unable to distinguish valid and invalid test cases, which risks biasing test suites toward including more invalid inputs in pursuit of higher coverage as claimed by the challenge *(C2)*.

Building on these observations, this work proposes a novel algorithm that combines a VAE model with existing test generation techniques to produce test cases with only valid inputs. More specifically, the algorithm formulates the joint optimization of the probability density of valid inputs and the objective of existing DNN test generation techniques, and uses gradient ascent to generate valid tests. An experimental analysis on the datasets used in the DNN testing literature [75, 93] shows the cost-effectiveness of the proposed approach.

This research makes the following contributions:

1. identification of limitations in existing DNN test generation and coverage criteria in their treatment of invalid input data,

2. development of a technique for incorporating an explicit model of the valid input space of a DNN into test generation to address those limitations, and

3. experimental evaluation that demonstrates the extent of the limitations and the effectiveness of our technique in mitigating them.

3.1 Analysis of Existing DNN Test Generation Techniques

The methodology for analyzing test inputs generated by existing test generation techniques is depicted in Fig. 3.2. DNN(s) under test and the deep generative model are trained on the same dataset. Test inputs generated by existing DNN test generation techniques for the DNN(s) under test are passed as inputs to the deep generative model which estimates their densities. These densities are used by the decision logic to classify inputs as valid or invalid.

For the experiments, a VAE is used for expressing the deep generative model logic, and in particular, the model proposed by An and Cho [3] where the decoder of a VAE outputs distribution parameters for the samples generated by the encoder. The probability of generating the original test input from a latent variable is calculated using these distribution parameters. This probability

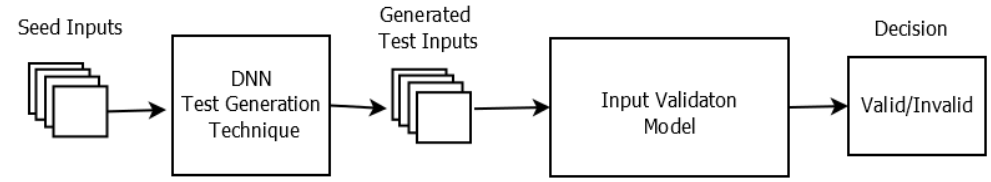


Figure 3.2: Technique for identifying invalid test inputs

is referred to as reconstruction probability. Valid inputs have a higher reconstruction probability when compared to invalid inputs.

For a dataset under test, referred to as the valid dataset, the method requires identifying another dataset that has a different distribution. The inputs from this dataset are considered as invalid inputs. The invalid dataset is selected in such a way that the invalid and valid datasets should model disjoint data categories. After identifying an invalid dataset, the reconstruction probability threshold is computed for identifying invalid inputs. Reconstruction probabilities are calculated for inputs from both valid and invalid datasets. A range of thresholds are generated from the combined reconstruction probability values of valid and invalid inputs. Then the F-measure is computed, which is a measure of a test's accuracy, for these threshold values. The F-measure is the harmonic mean of precision and recall. A good F-measure balances precision and recall and results in a fewer number of both false positives and false negatives which means fewer valid inputs are falsely classified as invalid and fewer invalid inputs are falsely classified as valid. The threshold value with the highest F-measure is selected for the experiments. When classifying test inputs generated by DNN test generation techniques, test inputs with reconstruction probability less than the selected threshold are classified as invalid by the VAE classifier.

The experiment measures the percentage of invalid inputs generated by multiple test generation techniques and the coverage of both valid and invalid tests. The results of the experiments are used to answer the research questions related to the limitations of existing techniques presented in Section 3.3.

3.2 Distribution-Aware Test Generation

This section describes the proposed technique to generate valid test inputs, and the workflow is described in Fig. 3.3. The approach leverages the existing gradient ascent based test generation technique's objective formulation. The objective of existing test generation techniques is modeled to increase test coverage and produce inputs that cause the model to make incorrect predictions. This objective is augmented with probability density estimated by a generative model. Gradient ascent is used to solve the joint optimization. Maximizing the joint optimization will result in inputs that follow the distribution of the training data of the DNN under test along with satisfying the objective of the baseline testing technique.

A detailed description of the test generation algorithm using a VAE as the generative model is provided in Algorithm 1. The decoder of the VAE outputs the distribution parameters ($\mu_{\hat{\mathbf{x}}}$, $\sigma_{\hat{\mathbf{x}}}$) for the samples generated by the encoder as per the OOD detection algorithm proposed in [3]. The algorithm requires a DNN under test, an objective function of a baseline gradient ascent based test generation technique $obj1$, a probabilistic encoder and decoder as inputs and produces both a test suite of valid inputs and their test coverage as output. For every input of the seed set, the probabilistic encoder generates parameters in latent space as shown in line 4 of the Algorithm 1. In lines 5-7, a sample from the latent space is used by the decoder to calculate the reconstruction probability of the input. The objective is modeled as a weighted sum of $obj1$ and reconstruction probability in line 8. Lines 9-11 show the gradient ascent. The gradient is calculated for the objective and at this stage, domain constraints, if any, are applied to the gradient and a new test input is generated. In lines 12-13, the generated test is tested for validity. If this test input causes the model to mispredict and has a reconstruction probability higher than the threshold, then on lines 14-15 the coverage is updated and input is added to the generated test suite. The procedure continues until all seeds are processed. This technique is evaluated using DeepXplore as a baseline test generation technique in Section 3.3.

3.3 Evaluation

The design and evaluation of experiments for studying existing techniques and demonstrating effectiveness of this research are described in this section. Experiments are designed to answer the

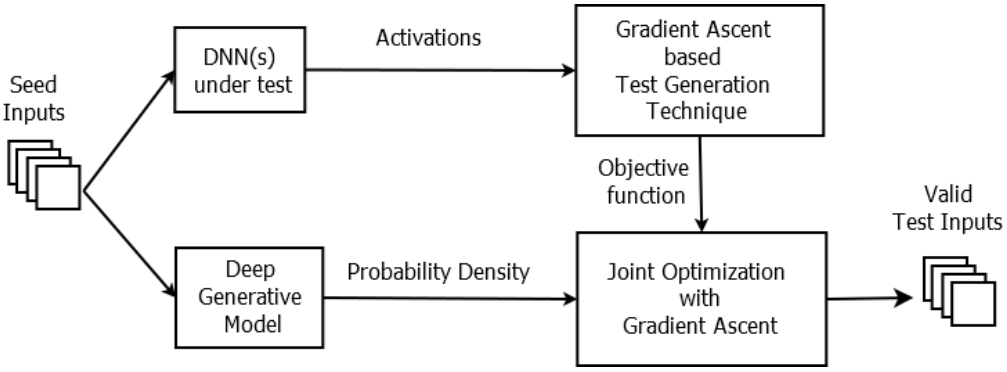


Figure 3.3: Technique for generating valid test inputs

Algorithm 1 Valid test input generation using VAE

Input:
X ← Seed inputs
DNN ← DNN under test
obj1 ← Objective function of test generation technique
s ← Step size for gradient ascent
max_iterations ← maximum iterations for gradient ascent
f_θ, g_ϕ ← Trained probabilistic encoder and decoder
λ ← hyperparameter for balancing two goals
α ← Reconstruction probability threshold
Output: Set of test inputs, coverage

```
gen_test = {}
for x in X do
    for i=1 max_iterations do
        μ_z, σ_z = f_θ(z|x)
        draw sample from z ~ N(μ_z, σ_z)
        μ_x̂, σ_x̂ = g_φ(x|z)
        obj2 = p_θ(x|μ_x̂, σ_x̂)
        obj = obj1 + λ × obj2
        gradient = ∂obj/∂x
        gradient = Constraints(gradient)
        x = x + s × gradient
        p = Reconstruction_Probability(x, f_θ, g_φ)
        if Counter_Example(DNN, x)  p ≥ α then
            gen_test.add(x)
            update coverage
            break
        end if
    end for
end for
```

following research questions:

RQ1: Do existing test generation techniques produce invalid inputs?

RQ2: Existing test generation techniques are guided by test coverage criteria. How do invalid inputs affect test coverage metrics?

RQ3: VAE based input validation can be incorporated into test generation techniques. How effective is this technique in generating valid inputs and what is the overhead?

RQ4: Is the determination of invalid inputs sensitive to the generative model used?

3.3.1 Evaluation Setup

All experiments are conducted on servers with one Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz processor with 32 cores, 62GB of memory, and an NVIDIA TITAN Xp GPU. The software that supports our evaluation as well as all of the data described below is available at `https://github.com/swa112003/DistributionAwareDNNTesting`.

Test Generation Frameworks

Three state-of-the-art test generation techniques: DeepXplore [99], DLFuzz [47], and DeepConcolic [125] are studied to demonstrate the limitations of existing techniques in terms of generating valid test inputs and satisfying test coverage criteria. The choice of these frameworks is guided by the categorization of test input generation techniques presented in a recent survey [152] and the availability of open-source code. The survey categorizes test generation frameworks into three algorithmic families; one technique from each family is selected for the experiments. DeepXplore is selected from domain-specific test input synthesis, DLFuzz from fuzz and search-based test input generation and DeepConcolic from symbolic execution based test input generation categories.

Test Coverage Criteria

DeepXplore and DLFuzz use neuron coverage as the test adequacy criteria whereas DeepConcolic can be used with neuron coverage, neuron boundary coverage, and MC/DC coverage criteria for DNNs [99, 84, 127]. Neuron coverage is used as the test adequacy criteria for generating tests using

Dataset	Name	Architecture		Accuracy
		Source	#l:#n:#p	
MNIST	MNI-1	LeNet-1 [76]	3:52:7206	98.66%
	MNI-2	LeNet-4 [76]	4:148:69362	99.03%
	MNI-3	LeNet-5 [76]	5:268:107786	99.08%
	MNI-4	Custom [125]	7:1300:312202	99.03%
SVHN	SVH-1	ALL-CNN-A [122]	7:2248:1.2M	96%
	SVH-2	ALL-CNN-B [122]	9:2824:1.3M	95.67%
	SVH-3	ALL-CNN-C [122]	9:2824:1.3M	95.98%
	SVH-4	VGG19 [116]	19:28884:38M	94.69%

Table 3.1: Models used in our studies with number of layers (#l), neurons (#n), parameters (#p), and test accuracy; "M" denotes millions of parameters.

all three frameworks. Resulting test inputs from test generation are analyzed using neuron coverage and extended neuron coverage metrics, i.e., k-multisection neuron coverage, neuron boundary coverage, and strong neuron activation coverage. The remaining coverage criteria discussed in these works [84, 127] are left for future study.

Datasets and DNN Models

Two popular datasets MNIST [75] and SVHN [93] are used for the experiments. Generative models can assign higher densities to datasets whose distributions are different from their training datasets in some cases[92]. For example, a VAE trained on CIFAR10 [70] can assign higher densities to inputs from the SVHN dataset. When such a model is used for invalid input identification, it might result in high densities being assigned to invalid inputs which will result in false negatives. Also selecting the threshold density for deciding invalid inputs becomes challenging in such scenarios. This problem is actively being addressed by the ML research community[104]. Generative models trained on MNIST and SVHN do not have this issue [92], so these two datasets are selected for our research.

MNIST is a collection of grayscale images of handwritten digits with 60000 training images and 10000 test images. All three frameworks used in the experiments support test generation for the MNIST dataset. Similar to DeepXplore, the experiments use LeNet-1, LeNet-4 and LeNet-5 networks from LeNet family [76] and a custom architecture used in the DeepConcolic work [125] for MNIST classification. All four models are convolutional networks with max-pooling layers and

Dataset	**MNIST**	**SVHN**
Valid	MNIST Test	SVHN Test
Invalid	FashionMNIST Test	CIFAR10 Test
F-measure	0.99	0.94
False Positives	0.3%	2.4%
False Negatives	1.42%	6.19%

Table 3.2: F-measure and percentage of false positives and false negatives for VAE-based input validation model

the number of layers ranging from 3 to 7.

SVHN contains color images of digits in natural scenes and the dataset has 73257 training images and 26032 test images. All three frameworks are extended with the SVHN dataset. SVHN classification models are trained with the ALL-CNN-A, ALL-CNN-B and ALL-CNN-C network architectures proposed in [122] and VGG19 [116] for the experiments. These models are convolutional networks with dropout and either global average pooling or max-pooling layers and the number of layers range from 7 to 19. The models are summarized in Table 3.1 where measures of their architecture and test accuracy are reported.

VAE Models

For MNIST, a VAE is trained that outputs distribution parameters using the model architecture described in [3]. The FashionMNIST dataset [141], is similar to MNIST and contains 28x28 greyscale images. However, the distribution is different from that of MNIST as FashionMNIST contains clothing images. FashionMNIST is used as the invalid input space for calculating the reconstruction probability threshold. Since the VAE is not trained on FashionMNIST distribution and FashionMNIST clothing inputs are semantically unrelated to MNIST digit inputs, the VAE should output lower reconstruction probabilities for test inputs from the FashionMNIST dataset.

Experiments are conducted with different variations of the generator architecture used in [109] for selecting a VAE network for the SVHN dataset. For each of the variants, the encoder is created by transposing the generator network as suggested in [109]. The network that achieved the highest F-measure for identifying invalid inputs is selected for our experiments. CIFAR10 [70] is used as the invalid input dataset for calculating the reconstruction probability threshold of VAE trained on SVHN. F-measure values and percentage of false positives for MNIST and SVHN test datasets

DNN	Testing Technique	Valid (%)	Invalid (%)	Total (%)
MNI-1	DeepXplore	38.5	**55.8**	55.8
	DLFuzz	50	**50**	50
	DeepConcolic	-	**55.8**	55.8
MNI-2	DeepXplore	65.5	**75**	75
	DLFuzz	71.6	70.9	71.6
	DeepConcolic	-	**58.1**	58.1
MNI-3	DeepXplore	70.9	**79.1**	79.1
	DLFuzz	78	76.9	78
	DeepConcolic	-	**64.6**	64.6
MNI-4	DeepXplore	66.6	**73.1**	73.3
	DLFuzz	71.7	48.0	71.7
	DeepConcolic	-	**63.1**	63.1

Table 3.3: Neuron Coverage of test inputs generated by DeepXplore, DLFuzz, and DeepConcolic for MNIST classifiers

are given in Table 3.2.

3.3.2 Results and Research Questions

This section presents the results of the experiments used to answer the research questions.

RQ1. Do existing test generation techniques produce invalid inputs?

Test inputs are generated for MNIST and SVHN classifiers using the DeepXplore, DLFuzz, and DeepConcolic techniques. The DeepXplore framework supports three types of input transformations: lighting, occlusion, and blackout. Tests are generated using all three transformations to answer RQ1.

Five hundred seed inputs are randomly sampled from each MNIST and SVHN test dataset for DeepXplore and DLFuzz. DeepXplore and DLFuzz use gradient ascent for test generation, and the hyperparameters reported in their respective works [99, 47] are used for the study. Similarly, a neuron coverage threshold of 0.25 is selected as it is commonly used in DeepXplore and DLFuzz experiments in their original work. The DeepConcolic tool uses a single seed input for test generation for neuron coverage, and a timeout of 12 hours is used for test generation in the primary work [125]. The same strategy is used along with the global optimization approach for generating tests. Generated tests are classified as valid or invalid by using the reconstruction probability metric of VAE. The top row of Fig. 3.4 shows the percentage of invalid test inputs generated by these

DNN	Testing Technique	Valid (%)	Invalid (%)	Total (%)
SVH-1	DeepXplore	44.4	**44.4**	44.4
	DLFuzz	44.8	44.4	44.8
	DeepConcolic	-	**44.2**	44.2
SVH-2	DeepXplore	45.5	**45.5**	45.5
	DLFuzz	45.6	45.5	45.6
	DeepConcolic	-	**45.5**	45.5
SVH-3	DeepXplore	45.4	**45.4**	45.4
	DLFuzz	45.7	45.4	45.8
	DeepConcolic	-	**45.2**	45.2
SVH-4	DeepXplore	74.2	74	74.8
	DLFuzz	75.9	73.3	75.9
	DeepConcolic	-	**72.3**	72.3

Table 3.4: Neuron Coverage of test inputs generated by DeepXplore, DLFuzz and DeepConcolic for SVHN classifiers

frameworks for MNIST and SVHN DNN models.

The percentage of tests generated by DeepXplore varies depending on the constraint used. For all four MNIST classifiers, occlusion constraint produced a high percentage of invalid test inputs i.e., greater than 90% while blackout constraint generated less than 1% invalid inputs. The lighting constraint generated 94% and 63% invalid inputs for models MNI-1 and MNI-3 and less than 1% for the other two. DLFuzz generated invalid inputs in the range 36% to 46% for MNI-1, MNI-2, and MNI-3 classifiers while less than 1% for MNI-4.

For SVHN classifiers, the occlusion and blackout constraints generated a higher number of invalid tests when compared to lighting constraints on average. DLFuzz generated invalid inputs are in the range 9% to 20% for SVHN classifiers. All the test inputs generated by the DeepConcolic framework for both MNIST and SVHN classifiers are classified as invalid by the VAE model.

Result for RQ1: All three testing techniques studied produced significant numbers of invalid tests; 42% on average and ranging from 73-100% in the worst-case.

RQ2. Existing test generation techniques are guided by test coverage criteria. How do invalid inputs effect test coverage metrics?

For this research question, neuron coverage(NC), multi-granularity coverage criteria i.e., k-multisection neuron coverage (KMNC), neuron boundary coverage (NBC) and strong neuron ac-

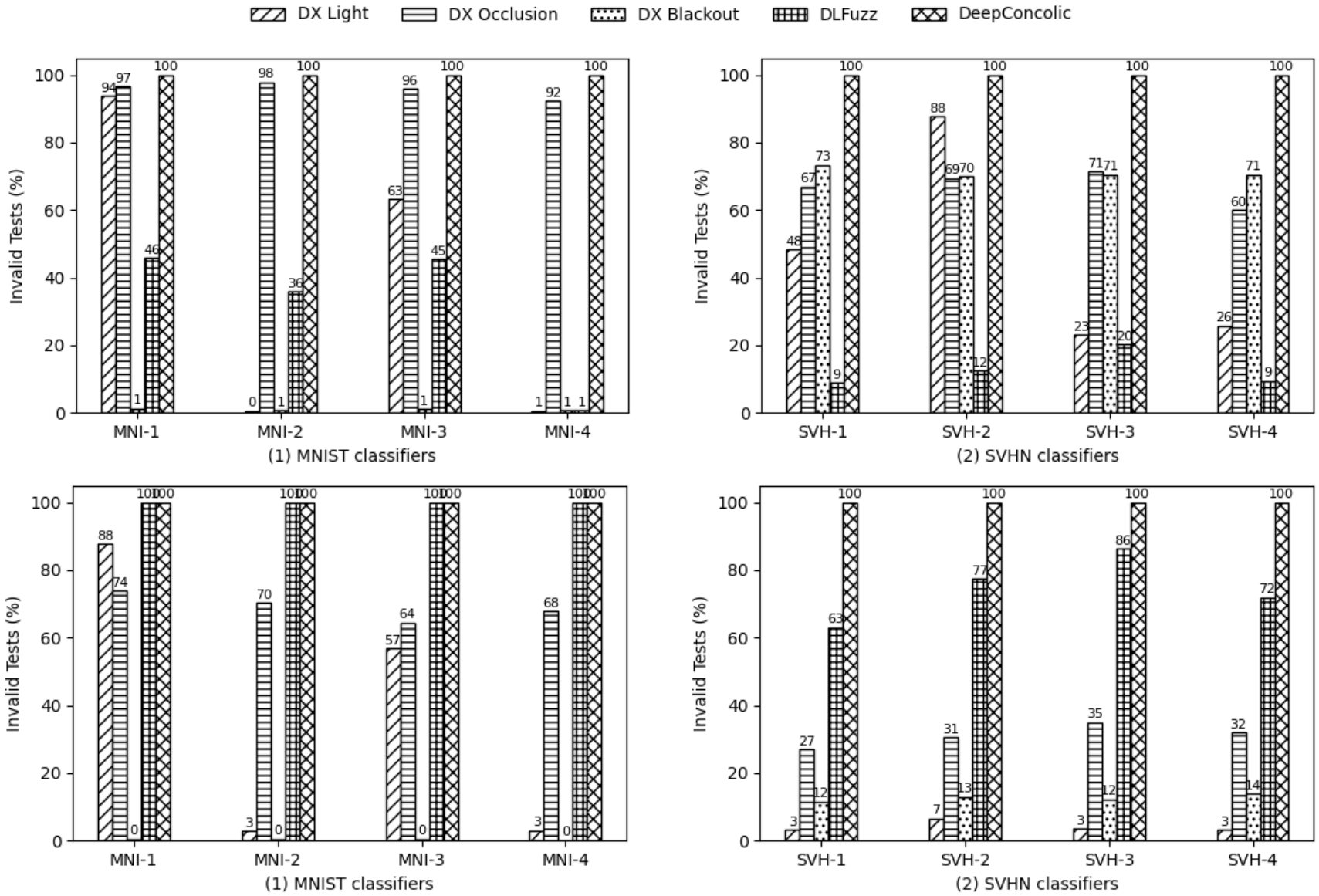


Figure 3.4: Percentage of invalid test inputs identified by VAE (top pair) and PixelCNN++ (bottom pair) input validation techniques.

tivation coverage (SNAC) of both valid and invalid tests generated by the three frameworks are measured. The k-value of 100 is used for measuring KMNC coverage. The cumulative neuron coverage is also measured for the valid and invalid test inputs. Results are presented in Tables 3.3 and 3.4 for neuron coverage metric and Tables 3.5 and 3.6 have data for multi-granularity coverage criteria.

Across 8 DNNs, 3 test generation techniques, and 4 coverage criteria, 72% of the time invalid tests achieved coverage greater than or equal to that achieved by valid tests. The entries in Tables 3.3, 3.4, 3.5 and 3.6 corresponding to this insight are highlighted in bold. 25% of the time invalid tests outperform valid for coverage, and 25% of the time invalid coverage boosts overall coverage by more then 10%.

Result for RQ2: Invalid inputs yield high coverage for a variety of coverage criterion when compared to valid inputs and they frequently increase coverage beyond that which would be achieved with valid inputs alone.

RQ3. VAE based input validation can be incorporated into the test generation techniques. How effective is this technique in generating valid inputs and what is the overhead?

To answer this question, test inputs are generated by using VAE based input validation along with a gradient ascent based test generation technique as described in Algorithm 1. DeepXplore is selected as the baseline test generation technique and density estimated by VAE is incorporated as a goal into its objective to formulate a joint optimization resulting in DeepXplore+VAE instanciation of the proposed algorithm. Result of a joint optimization is sensitive to the weights of different goals used in the objective function. To address this, the weights of the goals of the baseline's objective are fixed and a sweep over a range of density weights is performed to find the best configuration. Gradient ascent is used to generate test inputs for MNIST and SVHN models. 200 seed inputs are randomly identified from each of the two datasets and used the same seed set and gradient ascent parameters, i.e., step size and maximum iterations for DeepXplore and DeepXplore+VAE. The experiments are repeated three times and average results are presented in this section.

The number of valid tests generated along with their neuron coverage for DeepXplore+VAE and DeepXplore are measured to demonstrate the effectiveness of DeepXplore+VAE. The validity

of the inputs is measured with respect to the OOD detection algorithm used, i.e., the VAE in this case. DeepXplore+VAE generates only valid test inputs. Since DeepXplore generates both valid and invalid test inputs, the input validation module is added to DeepXplore to capture only the valid test inputs.

Neuron coverage achieved by DeepXplore and DeepXplore+VAE are presented in Figures 3.5 and 3.6 for MNIST and SVHN classifiers respectively. The plots show the coverage over a range of 200 seed inputs. DeepXplore+VAE achieved neuron coverage greater than or equivalent to that of the DeepXplore baseline for all 8 DNN models. For the scenarios where DeepXplore is able to achieve neuron coverage comparable to DeepXplore+VAE, the latter outperformed DeepXplore in terms of the number of valid inputs generated. Fig. 3.7 contains a comparison of the number of valid inputs generated by DeepXplore and DeepXplore+VAE for MNIST and SVHN classifiers. The total valid inputs generated by DeepXplore+VAE for the MNIST models are 5.6 times the valid inputs generated by DeepXplore. For the SVHN dataset, DeepXplore+VAE generated 1.6 times more valid inputs when compared to DeepXplore. Hence, having VAE in the test objective guides gradient ascent effectively in searching for valid inputs.

Table 3.7 shows the performance data of DeepXplore+VAE and DeepXplore algorithms for 200 seed inputs. Every iteration of these algorithms has two components, 1) gradient ascent, and 2) input validation. For each seed input, gradient ascent is performed until it finds a valid test input or for a maximum of 30 iterations whichever happens first. Input validation is performed only when the differential oracle fails the generated test input in that iteration. In all the cases, DeepXplore+VAE ran for fewer iterations and input validations when compared to DeepXplore. For the scenarios where the difference between DeepXplore+VAE and DeepXplore's number of iterations and input validations is high, DeepXplore+VAE is faster because DeepXplore is spending more time on generating invalid inputs which are then rejected by the input validation module. When this difference is small, DeepXplore has better overall run-time, but DeepXplore+VAE generates more valid inputs and has a lower cost per valid input when compared to DeepXplore. Due to DeepXplore+VAE's improved effectiveness in generating valid tests, it improves on DeepXplore's "time to produce a valid test" reducing it from 4.7 to 1.7 minutes, on average measured across three runs.

DNN	Testing Technique	Coverage	Valid (%)	Invalid (%)	Total (%)
MNI-1	DeepXplore	KMNC	11.3	**58**	58.8
		NBC	-	**1.9**	1.9
		SNAC	-	**1.9**	1.9
	DLFuzz	KMNC	49.2	45.3	56.4
		NBC	-	-	-
		SNAC	-	-	-
	DeepConcolic	KMNC	-	**8.2**	8.2
		NBC	-	-	-
		SNAC	-	-	-
MNI-2	DeepXplore	KMNC	7.1	**62**	62.4
		NBC	-	**3.4**	3.4
		SNAC	-	**6.8**	6.8
	DLFuzz	KMNC	49.7	40.2	54.2
		NBC	-	-	-
		SNAC	-	-	-
	DeepConcolic	KMNC	-	**11.2**	11.2
		NBC	-	**2.4**	2.4
		SNAC	-	**3.4**	3.4
MNI-3	DeepXplore	KMNC	12.5	**59.2**	59.9
		NBC	0.2	**3.4**	3.5
		SNAC	0.4	**6.7**	7.1
	DLFuzz	KMNC	45.8	41.6	52.2
		NBC	-	0.2	0.2
		SNAC	-	-	-
	DeepConcolic	KMNC	-	**14.8**	14.8
		NBC	-	**1.1**	1.1
		SNAC	-	**2.2**	2.2
MNI-4	DeepXplore	KMNC	18.7	**56.6**	57.5
		NBC	-	**1.8**	1.8
		SNAC	-	**2.4**	2.4
	DLFuzz	KMNC	47.5	1.6	47.5
		NBC	0.4	-	0.4
		SNAC	0.5	-	0.5
	DeepConcolic	KMNC	-	**27.9**	27.9
		NBC	-	**3.4**	3.4
		SNAC	-	**4.3**	4.3

Table 3.5: Multi-granularity neuron coverage of test inputs generated by DeepXplore, DLFuzz and DeepConcolic for MNIST classifiers

DNN	Testing Technique	Coverage	Valid (%)	Invalid (%)	Total (%)
SVH-1	DeepXplore	KMNC	30.7	**42.4**	46.8
		NBC	1.6	**5.2**	6.6
		SNAC	0.7	**9.1**	9.3
	DLFuzz	KMNC	55	38.8	57.9
		NBC	2.1	0.4	2.3
		SNAC	3.3	0.7	3.6
	DeepConcolic	KMNC	-	**17**	17
		NBC	-	**1**	1
		SNAC	-	**2**	2
SVH-2	DeepXplore	KMNC	35.2	**45.4**	50.1
		NBC	2.3	0.7	2.6
		SNAC	1.3	1.2	1.9
	DLFuzz	KMNC	58	43.5	61.3
		NBC	0.6	**1.5**	1.8
		SNAC	1	**1.5**	2
	DeepConcolic	KMNC	-	**22.5**	22.5
		NBC	-	**0.5**	0.5
		SNAC	-	**0.8**	0.8
SVH-3	DeepXplore	KMNC	32.2	**44.9**	48.9
		NBC	2.4	0.7	2.8
		SNAC	1.4	1.2	2.2
	DLFuzz	KMNC	52.9	51	60.2
		NBC	0.5	**1.6**	1.9
		SNAC	0.8	**2.4**	2.8
	DeepConcolic	KMNC	-	**20.2**	20.2
		NBC	-	**5.4**	5.4
		SNAC	-	**4.5**	4.5
SVH-4	DeepXplore	KMNC	24.7	**40.4**	41.8
		NBC	1	**4**	4.1
		SNAC	1.8	**5.5**	5.7
	DLFuzz	KMNC	41.4	29.5	43.4
		NBC	1.7	1	1.8
		SNAC	2.7	1.8	2.9
	DeepConcolic	KMNC	-	**15.2**	15.2
		NBC	-	**1.1**	1.1
		SNAC	-	**2.0**	2.0

Table 3.6: Multi-granularity neuron coverage of test inputs generated by DeepXplore, DLFuzz and DeepConcolic for SVHN classifiers

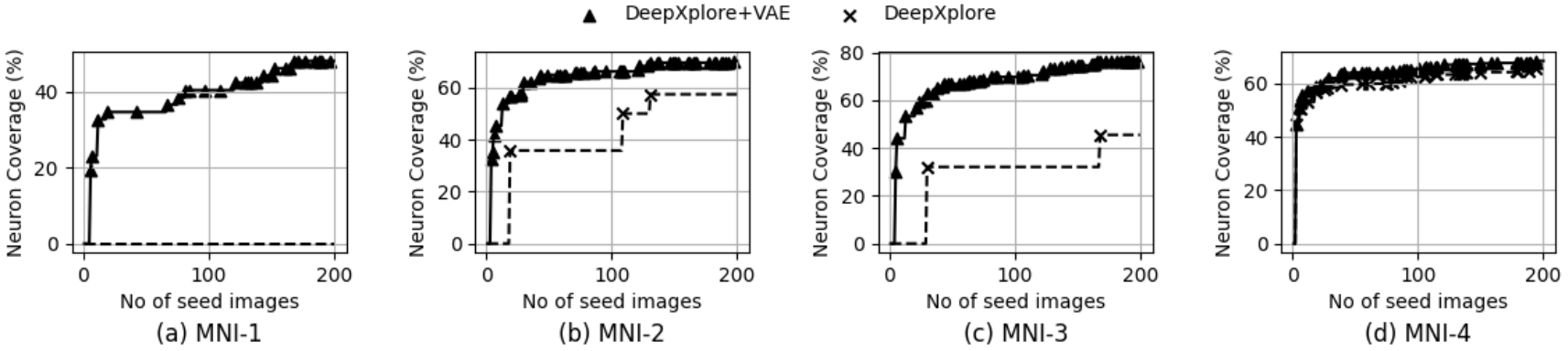


Figure 3.5: Neuron Coverage of valid inputs generated by DeepXplore and DeepXplore extended with VAE for MNIST models

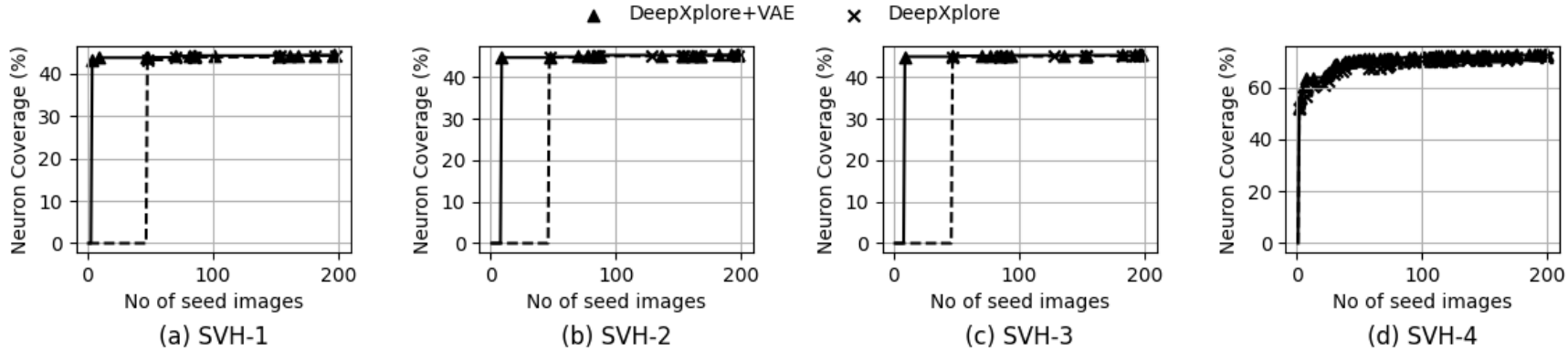


Figure 3.6: Neuron Coverage of valid inputs generated by DeepXplore and DeepXplore extended with VAE for SVHN models

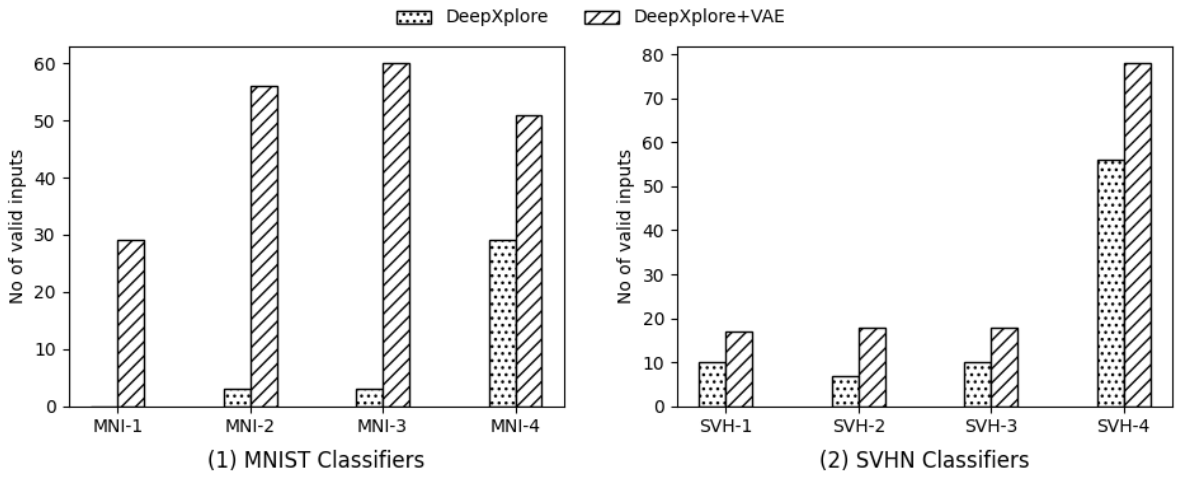


Figure 3.7: Number of valid inputs generated by DeepXplore and DeepXplore extended with VAE for MNIST and SVHN models

DNN	DeepXplore+VAE				DeepXplore				Additional Iterations	Additional Input validations
	Run-time in mins	Valid Inputs	Iterations	Input validations	Run-time in mins	Valid inputs	Iterations	Input validations		
MNI-1	96.74	29	5413	882	103.82	1	5972	1832	-559	-950
MNI-2	73.5	54	4910	413	103	3	5913	1812	-1003	-1399
MNI-3	60.39	56	4863	200	96.66	3	5917	1587	-1054	-1387
MNI-4	54.97	52	4736	52	46.57	29	5199	375	-463	-323
SVH-1	97.12	17	5637	27	64.7	12	5737	47	-100	-20
SVH-2	97.96	20	5578	28	66.83	9	5798	60	-220	-32
SVH-3	90.34	21	5539	29	69.83	11	5703	80	-164	-51
SVH-4	143.81	83	4126	219	130.57	53	4547	446	-421	-227

Table 3.7: Run-time analysis of test generation algorithms of DeepXplore+VAE and DeepXplore for MNIST and SVHN classifiers. The last two columns Additional Iterations and Additional Input Validations refer to the metrics measured for (DeepXplore+VAE - DeepXplore).

Result for RQ3: Incorporating a VAE into test generation eliminates the generation of invalid test inputs, significantly increases the generation of valid inputs, reduces the time to generate valid tests, and increases coverage achieved on generated valid tests.

RQ4. Is the determination of invalid inputs sensitive to the generative model used?

To answer RQ4, a PixelCNN++ based input validation is used. PixelCNN++ is an autoregressive deep generative model [111]. PixelCNN++ models are trained for MNIST and SVHN datasets. For each dataset, a threshold for identifying invalid inputs is calculated by using an invalid dataset and F-measure analysis similar to VAE based detection technique described in Section 3.1. The

Dataset	**MNIST**	**SVHN**
Valid	MNIST Test	SVHN Test
Invalid	FashionMNIST Test	CIFAR10 Test
F-measure	0.99	0.92
False Positives	0.14%	2%
False Negatives	0.56%	10.66%

Table 3.8: F-measure and percentage of false positives and false negatives for PixelCNN++ based input validation model

F-measure, precision, and recall of the selected thresholds for both datasets are presented in Table 3.8.

The percentage of test inputs generated by DeepXplore, DLFuzz and DeepConcolic for the MNIST and SVHN classification models that are classified as invalid by PixelCNN++ based input classifier are presented on the bottom row of Fig. 3.4. PixelCNN++ for the MNIST models, classified a high percentage of test inputs generated by DeepXplore's light and occlusion constraints as invalid and classified all test inputs as valid for blackout constraint. For the SVHN classifiers, occlusion and blackout constraints result in a higher number of invalid inputs when compared to the light constraint.

The PixelCNN++ classified all test inputs generated by DLFuzz as invalid for MNIST models and more than 60% of test inputs as invalid for SVHN models. All inputs generated by DeepConcolic are identified as invalid for both models.

The results follow the same trend as observed by VAE based classifier. However, the percentage of test inputs classified as invalid by PixelCNN++ is less when compared to that of VAE for DeepXplore generated tests. For DLFuzz, the PixelCNN++ approach resulted in more invalid tests when compared to the VAE based classifier. Both the VAE and PixelCNN++ based techniques classified all test inputs generated by DeepConcolic as invalid.

Result for RQ4: Test generators are judged to produce invalid tests with different OOD techniques, but the number of invalid tests is sensitive to the deep generative model architecture used.

3.4 Threats to Validity

The study is designed to provide a degree of generalizability by spanning all of the algorithmic families of DNN test generation approaches, as well as 2 datasets, 8 models, 4 coverage criteria, and 2 approaches to out-of-distribution detection. Moreover, the datasets and models chosen are those that have been used in prior research – which was both a convenience choice and a means of promoting comparison among methods, e.g., against baselines. Despite these measures, the research findings may be dependent on these choices.

Further study, especially with additional OOD techniques, beyond VAE and PixelCNN++, is warranted to understand the generalizability of our findings as relates to the rate at which invalid inputs are generated and the degree of coverage achieved by those inputs. The research carried out on adapting test generation with OOD is more limited using a single model, a VAE, and a single test generation approach, DeepXplore which is a representative of the class of optimization-based test generation approaches. It is not a simple matter to extend this study to other families of test generation methods, but that will be necessary to understand the extent to which the benefit of integrating OOD methods with DNN test generation techniques broadly generalizes.

All of the experiments are run multiple times and cross-checked with the prior work, e.g., that the same level of coverage is achieved for baseline techniques as was reported in prior work. These measures were taken to assure the quality of the data reported here and the code is made available on github for transparency and replicability.

Chapter 4

DisTest: A Framework For Distribution-Aware Testing

This chapter presents DisTest – a framework for testing DNNs with valid, diverse, and rare inputs. DisTest uses an abstraction of the features of the data distribution as a domain for testing DNNs. The main challenge in formulating the feature abstraction of the data distribution is the lack of formal input specifications for DNNs. DisTest addresses this lack by exploiting the fact that the training data for a DNN implicitly defines a space of features that can play the role of the unavailable input specification.

To illustrate, Figure 4.1 depicts how varying different features for the dSprites dataset [88] gives rise to different images in the dataset. dSprites inputs are 64 by 64 black and white images, so the input domain of a model processing these inputs has 4096 dimensions. The plot shows a small neighborhood in the 6-dimensional latent space learned by training a variational autoencoder (VAE) [21] on the dSprites training dataset. The latent space is a lower dimensional space inferred from the input data distribution. The center column in the plot repeats the seed image of the neighborhood – an axis-aligned ellipse centered in the frame. A row illustrates the variation in a single latent dimension by decrementing to the left and incrementing to the right the latent vector computed by the VAE for the seed image. Visually these dimensions appear to capture five features that correspond; for the first 5 rows, to horizontal coordinate, vertical coordinate, scale, rotation, and shape variation. The bottom row shows a learned latent dimension that does not vary with

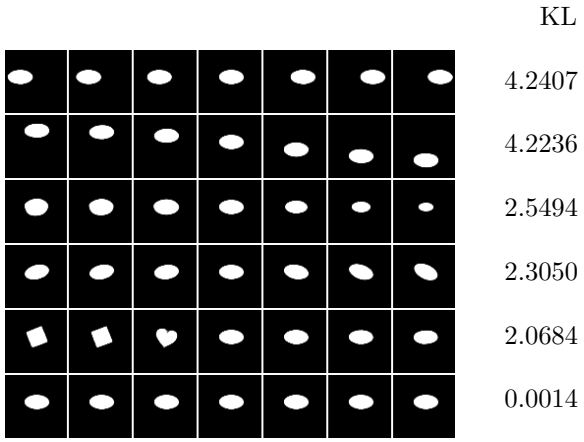


Figure 4.1: Feature variation in the latent space.

the input and because it encodes no feature of input data, it can be ignored when considering feature coverage. This is referred to as a *noise* dimension because it is almost identical to the assumed Gaussian prior used by the VAE [13]. The near-zero Kullback-Leibler (KL)-divergence metric (0.0014) measures the similarity between the empirical distribution of values in the dimension across the training set and a standard Gaussian; all other KL values are at three orders-of-magnitude larger.

While helpful in illustrating the concept of learned features representations, the dSprites dataset is specifically designed to generate inputs in $\mathbb{R}^{4096}$ from a set of 5 orthogonal features – the same features illustrated in Figure 4.1. In general, however, the set of features defining a dataset will not be known. Consider a DNN, $\mathcal{N} : \mathbb{R}^m \to \mathbb{R}^n$, trained to make predictions using a corpus of training data, $(\boldsymbol{x}, \boldsymbol{y}) \in T$ where $\boldsymbol{x} \in \mathbb{R}^m$ and $\boldsymbol{y} \in \mathbb{R}^n$. The first element of a training instance, $\boldsymbol{x}$, is an input vector that represents a sample drawn from the unknown input distribution, $\boldsymbol{x} \in \mathcal{X}$, over which the DNN should generalize. The manifold hypothesis [37] states that real-world data is concentrated on a low-dimensional surface, $\mathcal{X}$, embedded in the input domain, $\mathbb{R}^m$, where $\dim \mathcal{X} \ll m$.

Leveraging properties of $\mathcal{X}$ seems promising for defining a testing domain for several reasons. First, due to measure concentration as $m - \dim \mathcal{X}$ grows, $\frac{|\mathcal{X}|}{|\mathbb{R}^m|}$ quickly approaches zero. Thus *testing that focuses on $\mathcal{X}$ has the potential to be more tractable* than testing over the full input domain. Second, inputs in $\mathbb{R}^m$ that are not in $\mathcal{X}$ do not reflect the domain of definition of a DNN trained on $\mathcal{X}$ and *test generation on $\mathcal{X}$ does not result in invalid inputs* which addresses the challenge *(C1)*

and *coverage that focuses on $\mathcal{X}$ will not be artificially inflated* by such inputs thereby addressing the challenge *(C2)*. Third, the dimensions of $\mathcal{X}$ can be understood as representing combinations of features [13]. Thus *testing methods that exercise the span of the dimensions of $\mathcal{X}$ result in testing DNNs with feature-diverse inputs* to which DNN behavior may be sensitive and this addresses the challenge *(C3)*. Unfortunately, a precise definition of $\mathcal{X}$ is not available as described by the challenge *(C4)*. The DisTest framework addresses this challenge, *(C4)*, by using generative models that learn a low-dimensional representation comprising the features of $\mathcal{X}$ in the *latent space* through training on the inputs from the training data, $\boldsymbol{x} \in T$ [4, 69].

Machine Learning research has shown that deep generative models such as variational autoencoders (VAE) and generative adversarial networks (GAN) [68, 44] are effective in learning the latent space of real-world datasets. The dimensions of the latent space represent features, and combinations of these features represent diverse inputs of the data distribution. This research uses a VAE to instantiate the generative model of DisTest, leaving the exploration of other generative models to future work.

A VAE includes an encoder, latent space, and a decoder which is the generator network as described in Chapter 2. The latent space learned by a VAE has similar data samples placed close to each other. The latent space of a VAE: (a) has a well-defined mathematical structure that is amenable to algorithmic processing; (b) is dense in the sense that movements within the mathematical structure correspond to movements within $\mathcal{X}$; (c) is low-dimensional and in some cases can have a lower dimension than $\mathcal{X}$; (d) accurately reflects T; and (e) reflects feature diversity present in $\mathcal{X}$ that is not present in any instance of T. Also, the latent space being a probability distribution has a low-density region containing rare inputs. As a result, *testing methods that target the low-density region of the latent space, test DNNs with rare inputs* and this addresses the challenge *(C5)*. Hence, DisTest addresses all the challenges *(C1)* to *(C5)* with the existing methods.

4.1 The DisTest Framework

Figure 4.2 shows the DisTest framework that uses a VAE to learn a latent space representation of the data distribution, $\mathcal{X}$, and uses the learned latent space as a test domain. This research

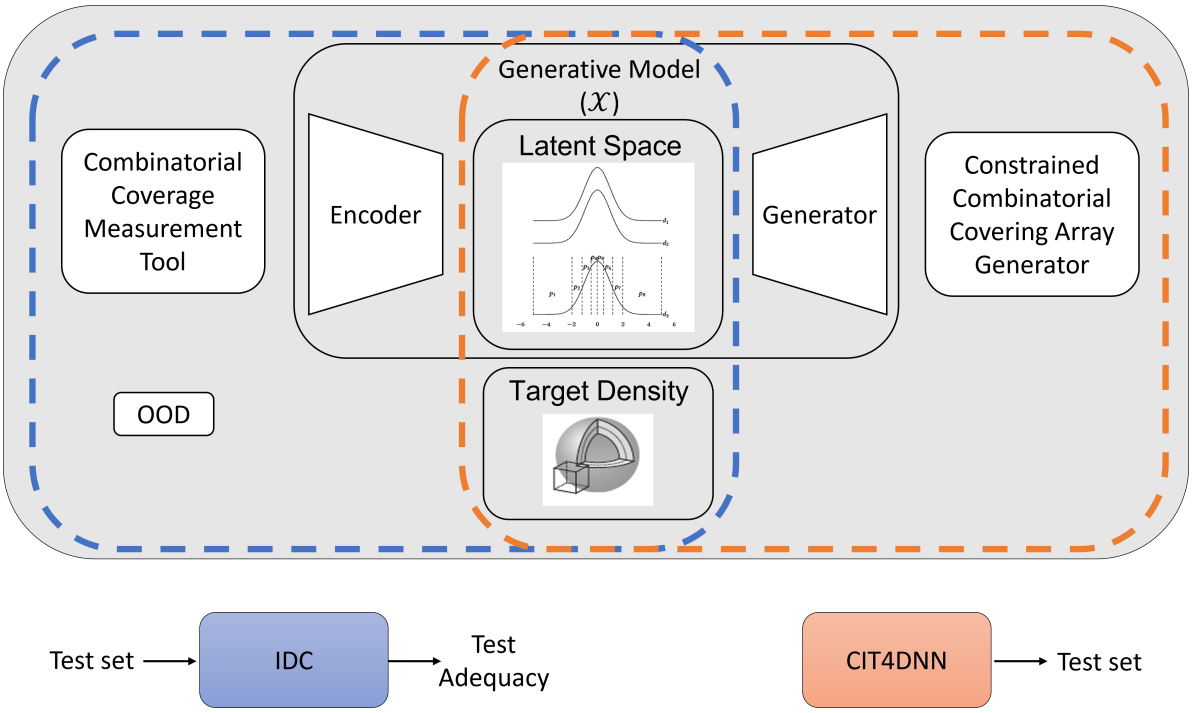


Figure 4.2: The DisTest Framework. The modules included in the dashed blue box are used for developing IDC whereas the modules included in the dashed orange box are used for developing CIT4DNN.

developed two applications over DisTest, Input Distribution Coverage (IDC) which is a black-box test adequacy criterion, and CIT4DNN, a black-box test generation algorithm. Both IDC and CIT4DNN use the latent space as a domain for testing and apply combinatorial interaction testing on a target density of the latent space to measure feature interaction test adequacy and generate diverse and rare inputs that systematically cover the latent space respectively.

DisTest comprises a VAE with an encoder, latent space and decoder, target density, combinatorial coverage measurement tool, constrained combinatorial covering array generator, and out-of-distribution detector (OOD). All the modules of DisTest are described in Chapter 2. This section describes the latent partitioning strategy that supports the systematic exploration of the latent space, the representation of target density in order to support testing on a specific region of the latent space, and the parameters of the DisTest framework.

4.1.1 Latent Partitions

The features encoded in VAE latent dimensions need not correspond with a human-understandable meaning in order to use the latent space as a test domain. Any input in the training data set always has a representation in the latent space as a combination of the values of the latent dimensions. Hence the full product of the latent dimensions captures the feature diversity of the training data set. Since a VAE learns to generalize the training distribution, in turn, the interactions among latent dimensions can capture the feature diversity of the data distribution.

Coordinates within the latent space are continuous, but DisTest discretizes the space by partitioning each dimension in order to produce a finite combinatorial space to perform testing. Given the approximate nature of a trained VAE, one cannot expect that direct mapping from input features to ranges of values within latent dimensions exists. It is possible that multiple features are combined into the values of a single latent dimension, or that a single feature is spread across the values of multiple latent dimensions. While some VAE architectures can disentangle or decorrelate latent dimensions, they are unable to produce a well-defined mapping from features to regions of the latent space that DisTest can manipulate to partition the latent space.

DisTest sidesteps this challenge by using an efficient systematic partitioning of latent dimension value ranges that does not attempt to interpret how features are mapped to such values. By spanning the value ranges across all dimensions, such a partitioning scheme ensures that regardless of how a feature is encoded in the latent space there is a partition element that will include it. To facilitate the efficient calculation of the feasible coverage domain DisTest requires that dimension partitions be symmetric around 0.

Definition 2 (Symmetric Interval Partition)**.** *A symmetric interval partition of the range of values of a latent dimension is comprised of a set of intervals, $\mathcal{P}$, that form a partition*

$$[-\infty, \infty) = \bigcup_{p \in \mathcal{P}} p \qquad \forall p \in \mathcal{P} : \forall p' \in \mathcal{P} - \{p\} : p \cap p' = \emptyset$$

that are open on the right, closed on the left, and divided evenly among positive and negative values

$$[l, u) \in \mathcal{P} \implies [-u, -l) \in \mathcal{P}$$

and do not span the origin, $[0, u) \in \mathcal{P}$.

The partition $\mathcal{P}$ of DisTest denotes an interval partition for a latent vector of dimensionality k. It defines an associated partition function, $P : \mathbb{R} \to \mathcal{P}$, that maps latent space coordinates to intervals in the partition set. This function can be lifted to vectors component-wise; $P((l_1, \ldots, l_k)) = (p_1, \ldots, p_k)$ such that $l_i \in [l_{pi}, u_{pi})$. Such a k-dimensional partition forms a hyper-rectangle in the latent space of the VAE.

The VAE is assumed to have a standard Normal $\mathcal{N}(0, 1)^k$ prior which makes it easy to partition the latent space with equal density divisions. An important required property for test domain to be accurate is that the aggregate posterior of the VAE matches the standard Normal prior assumption. Distance metrics such as Maximum Mean Discrepancy(MMD) [45] can be used to select VAEs whose latent distribution closely follows $\mathcal{N}(0, 1)^k$.

4.1.2 Target Density

The architecture and training process of a VAE imposes a mathematical structure on the latent space that DisTest can leverage. First, by assuming a spherical standard Normal prior the geometry of the coverage domain is determined by the Gaussian Annulus theorem [6]. The theorem observes that the density of a high-dimensional Gaussian is concentrated in a shell, of increasing radius and decreasing thickness, as the number of latent dimensions grows. A well-known property is used for computing the boundaries of the shell: that the distribution of the square of the distance from the origin of samples from multivariate standard Normal distribution follows a Chi-squared distribution [73]. Second, the spherical nature of the prior distribution means that all dimensions are assumed to follow the same univariate standard Normal distribution which makes different strategies for partitioning the latent space possible. Moreover, the symmetries of this prior can be leveraged to reduce the cost of computing the test coverage and generating the test inputs as demonstrated in Section 5.3 and Section 6.5 respectively. Finally, it has been shown that VAEs learn a dense latent representation where linear movement within that space corresponds to movement along the manifold in the input space [114]. The density of the representation allows for a simple interval-based partitioning scheme to be applied. Any symmetric partitioning scheme suffices for DisTest, but one efficient and general partition simply equally divides the space based on probability

density within each dimension. Such a partitioning scheme is defined for a single dimension and then applied to all dimensions based on the assumption that the prior is spherical.

Definition 3 (Equal Density Partition)**.** *A symmetric interval partition, $\mathcal{P}$, has equal density if*

$$\forall p, p' \in \mathcal{P} : Pr(p) = Pr(p')$$

and can be defined such that:

$$\mathcal{P} = \{[Q_{\mathcal{N}}(\frac{j-1}{|\mathcal{P}|}), Q_{\mathcal{N}}(\frac{j}{|\mathcal{P}|})) \mid j \in [1, |\mathcal{P}|]\}$$

where $Q_{\mathcal{N}}$ is the quantile function for $\mathcal{N}(0,1)$.

Such a partitioning gives rise to a combinatorial space of $|\mathcal{P}|^k$ partitions each with approximately equal density. DisTest operates on a subset of partitions that comprise a target density within which either adequacy measures are computed or test inputs are generated. DisTest partitions latent dimensions uniformly due to the assumption about the spherical symmetry of the assumed standard Normal prior distribution. This assumption also leads to the fact that the radii of points in that distribution follow a Chi distribution; this is nothing more than the distance formula of k-coordinates each distributed Normally. The target density parameter of DisTest establishes a threshold on the density of this distribution which gives rise to a k-dimensional shell.

Definition 4 (Target Density and Shell)**.** *For a multivariate Normal distribution, $\mathcal{N}(0,1)^k$, a target density, d, defines a unique thinnest k-dimensional shell with inner, r_{in}, and outer, r_{out}, radii:*

$$\begin{aligned} \min \quad & r_{out} - r_{in} \\ \text{s.t.} \quad & Pr(\{x \mid r_{in} \leq ||x|| \leq r_{out}\}) = d \end{aligned}$$

The radii of the thinnest target density shell can be calculated from the interval function of the Chi distribution with k degrees of freedom [6], i.e., $(r_{in}, r_{out}) = Chi.interval(d, k)$. As k grows so will r_{in}, since little probability mass is located near zero, which gives rise to the shell-like structure. This geometry of the target density implies that some of the latent partition combinations may be infeasible, which makes it challenging for IDC and CIT4DNN to restrict their attention to feasible

t-way feature combinations. This research develops methods using an SMT solver to overcome this challenge as described in Chapter 5 and Chapter 6.

4.1.3 Parameters of DisTest

DisTest comprises a number of parameters, the dimensionality, partition granularity, and target density of the latent space, the hyper-parameters of the OOD and the generative models, and the strength of CIT. The selection criterion of these parameters is described in Chapter 5 and Section 5.3 and Section 6.5 include experiments that study the effect of these paramters on IDC and CIT4DNN respectively.

4.2 Applications

This section discusses how the latent partitions, target density, and other DisTest modules are used for developing IDC and CIT4DNN.

4.2.1 IDC

IDC is designed to measure test adequacy with respect to the feature interactions present in a test set. Figure 4.2 shows the components of DisTest used by IDC in a dashed blue box. The dashed blue box in the figure includes the OOD, encoder, latent space, and combinatorial coverage measurement tool. Since the test set might include invalid inputs, IDC first filters them out using the OOD module. IDC then uses the encoder to convert test inputs into their latent encodings and using the combinatorial coverage measurement tool [72] calculates test coverage as the percentage of the t-way combinations of a target density of the partitioned latent space covered by the test set. The resulting test adequacy measured by IDC reflects the feature diversity of test inputs. However, to calculate the test coverage, IDC requires the number of feasible t-way combinations in a target density. This research develops Alg. 2, which uses the radii of the target density shell to formulate constraints representing the geometry of the target density and uses an SMT solver to calculate the number of feasible partition combinations within the target density. Chapter 5 describes the research objectives, methodology, and parameters of IDC and demonstrates the effectiveness of IDC using experimental evaluation.

4.2.2 CIT4DNN

CIT4DNN is developed to generate rare and diverse test inputs that systematically cover a target density of the latent space. Figure 4.2 shows the components of DisTest used by CIT4DNN in a dashed orange box. The dashed orange box in the figure includes the latent space, generator, and constrained combinatorial covering array generator. CIT4DNN uses the partitioned latent space and radii of the target density shell to formulate constraints on the required target density. These constraints are used by the constrained combinatorial covering array generator of to generate test descriptions that cover the t-way combinations of the target density of the partitioned latent space. This research developed the constrained combinatorial covering array generator module by extending an existing combinatorial covering array generator [29] with an SMT solver as described in Alg. 3. CIT4DNN uses the generator module to convert the test descriptions into test inputs. The resulting test inputs generated by CIT4DNN satisfy the IDC test adequacy criterion. Chapter 6 describes the research objectives, test generation algorithm, and parameters of CIT4DNN followed by the experimental evaluation.

Chapter 5

IDC: Feature Interaction Test Adequacy

This chapter [1] presents *input distribution coverage* (IDC) – a novel black-box test adequacy criterion for DNNs developed over the DisTest framework. IDC uses the OOD, encoder, latent space, and combinatorial coverage measurement tool components of the DisTest framework. IDC defines the adequacy criterion over the domain of valid inputs and measures the test coverage as a function of the feature combinations present in the test inputs. IDC relies on using the latent space of a Variational Autoencoder (VAE) as a test coverage domain, but leveraging a VAE's latent space is only possible for inputs that lie on the input data distribution. To address this challenge, the IDC instantiates the out-of-distribution (OOD) component of DisTest with a state-of-the-art method to filter any test inputs prior to employing a VAE.

Figure 5.1 depicts the high-level elements of IDC. IDC records coverage information only after filtering OOD inputs; in Figure 5.1 c such inputs are depicted as being filtered. IDC then abstracts feasible test inputs to feasible feature vectors that capture how an input varies relative to a set of generative features that describe the DNN input space. As described in Section 5.2, feasibility is judged relative to a model of the input distribution – more specifically a test is feasible if its probability within the distribution exceeds a user-defined threshold. Finally, IDC applies existing

[1] This chapter is adapted from *Dola, Swaroopa, Matthew B. Dwyer, and Mary Lou Soffa. "Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing." ACM Transactions on Software Engineering and Methodology 32.3 (2023): 1-48.*

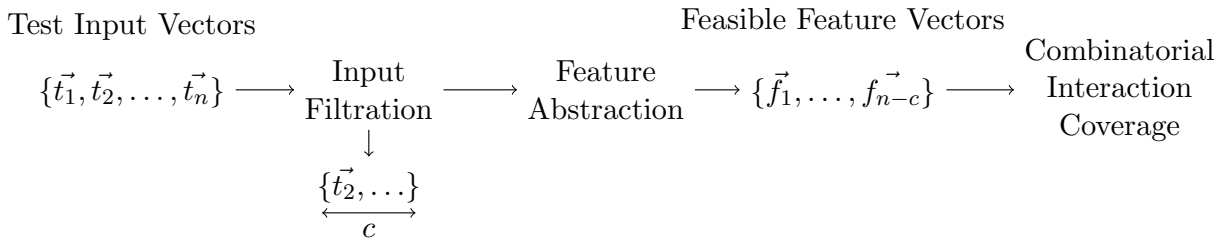


Figure 5.1: Conceptual IDC workflow. Tests are filtered to focus coverage on feasible inputs. Feasibility is judged relative to a model of the input distribution. Combinatorial interaction coverage is computed in terms of abstracted input feature vectors.

combinatorial interaction coverage measures, e.g., [72], to compute coverage measures over the set of abstracted feature vectors.

As described in Section 5.1, IDC is instantiated by selecting OOD and VAE components and then configuring a number of parameters adapted from combinatorial interaction testing (CIT) [28, 29] to control the granularity and interaction strength of the test adequacy measure computed. This permits developers to customize IDC to define stringent adequacy measures for critical systems and more relaxed measures for less consequential systems.

The primary contribution of this work lies in:

- defining and implementing IDC over the DisTest framework which can be instantiated with state-of-the-art OOD detectors and VAEs and configured by parameters controlling the strength of testing, to produce test adequacy measures that are well-suited to the needs of a DNN tester;

- demonstrating that IDC is effective in measuring the feature interactions present in a test suite and is robust to variation in its components by demonstrating its performance across a range of state-of-the-art VAEs [32, 64, 55];

- demonstrating a positive correlation between the test coverage measured by IDC and the number of fault-revealing test inputs present in the test suites;

- demonstrating that IDC is sensitive to test inputs generated using diverse test generation techniques [99, 47, 129] across three data sets [75, 139, 70]; furthermore, demonstrating that IDC is more sensitive to variation in tests than state-of-the-art white-box DNN coverage criteria [99, 84, 65];

- demonstrating that IDC can complement DNN mutation testing techniques by including substantially greater feature diversity in the testing process; and
- demonstrating that IDC can compute test adequacy efficiently for a variety of instantiations of the components used.

These findings demonstrate that IDC overcomes the limitations of prior DNN coverage approaches by avoiding infeasible inputs in testing and enabling the calculation of test adequacy metrics that capture the diversity present in a DNN's input space.

5.1 Overview

IDC measures test adequacy by applying combinatorial interaction testing (CIT) techniques to compute coverage measures that capture the feature interactions across a DNN's input space. As depicted in Figure 5.2, IDC exploits state-of-the-art ML models, such as OOD detectors and VAEs, which are trained on the same data as the DNN under test, but learn distinct classifications and representations of that data. Importantly, these models are only concerned with the training data set which makes them appropriate for use in IDC's black-box test adequacy measures. The remainder of this section elaborates on how the elements in Figure Figure 5.2 combine to realize the conceptual IDC workflow.

5.1.1 Input Filtration

There are two distinct goals when filtering test inputs to compute the IDC metric. First, to avoid the problems of prior work [34, 5] when tests lie off of the data distribution, their coverage should never be counted. Second, the input distribution over which a model aims to generalize is impractical to cover exhaustively, so IDC permits test coverage to be computed over a subset of that space defined by a *target density*. This enables a developer of a non-critical system to configure a relaxed target density, say 0.95, to define full coverage for IDC. In contrast, a critical system developer might specify a greater density, say 0.99999, which demands substantially more thorough testing.

To support the first of these goals, IDC uses an *OOD Filter* (❶) incorporating state-of-the-art out-of-distribution detection approaches, e.g., [143, 53, 105], to filter inputs that are judged not to

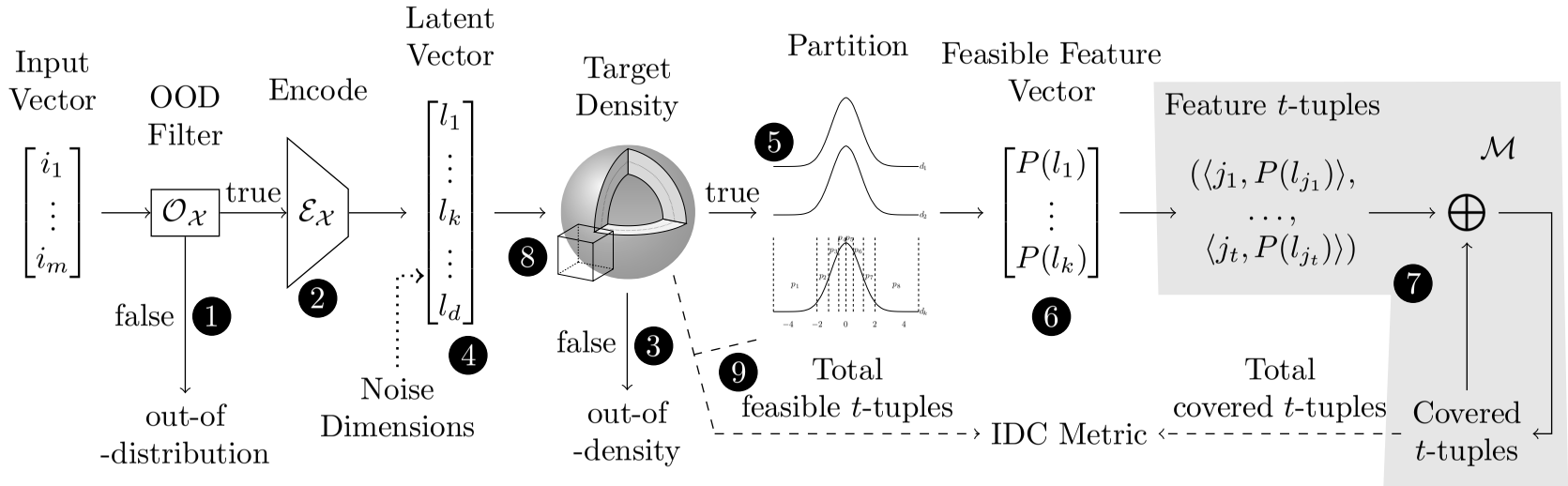


Figure 5.2: Detailed IDC workflow for a single test input. Inputs are filtered using an OOD detector and relative to a chosen target density of the latent distribution. Unfiltered inputs are referred to as feasible tests and are abstracted by encoding them to a latent vector and partitioning non-noise dimensions of those vectors. Combinatorial coverage is then computed over the total set of feasible feature vectors. The total feasible coverage domain is computed based on the partition and target density. Their ratio defines the IDC metric.

be in $\mathcal{X}$. Thus, unlike prior work [34, 5], IDC *will not accumulate coverage for out-of-distribution inputs*.

The data over which a model aims to generalize, $\mathcal{X}$, lies on a surface in the input space whose exact definition is unknown, making calculations relative to it impossible. IDC addresses this issue by mapping inputs in $\mathcal{X}$ to a low-dimensional space that reflects the variation in $\mathcal{X}$, is dense in the sense that movement within the space corresponds to movement along the surface $\mathcal{X}$, and exhibits the structure of a spherical multivariate Normal distribution. IDC leverages a trained VAE for this purpose and incorporates its encoder layer, $\mathcal{E}_{\mathcal{X}}$, to *Encode* (❷) latent vectors for test inputs.

Within this latent space, data are concentrated in a shell centered around the origin, which permits the target density to be expressed as the shell's inner and outer radii. Latent vectors whose distance lies either within the inner or beyond the outer radii are considered infeasible and are filtered out (❸). The feasibility of a test input relative to the input distribution is a special case of Definition 9 given in Section 5.2. As discussed below, these radii also play an important role in defining the total feasible coverage domain for IDC.

5.1.2 Feature Abstraction

IDC uses the learned representation of the features of $\mathcal{X}$ that result from training a VAE. In some contexts, as illustrated in Figure 4.1, the dimensions of the VAE's latent space correspond to features that can be easily recognized, but this is not always the case. For example, the VAE may learn relevant features in only a subset of the dimensions of the latent space. The *Noise Dimensions*, $[l_{k+1}, ..., l_d]$ (❹), are identified by near zero average KL-divergence values of training inputs within these dimensions. The noise dimensions are ignored in IDC so that the latent vectors more succinctly represent feature variation in the test inputs.

The latent dimensions are continuous, but intervals of values within those dimensions can correspond to recognizable features, e.g., the fifth row of Figure 4.1 shows intervals corresponding to diamonds, hearts, and ellipses. Determining the most appropriate intervals for a given problem is challenging given the complexity of a VAE's learned latent space. IDC takes a pragmatic approach by defining a *Partition* (❺) that divides each latent dimension into a set of disjoint intervals. A partition function, $P(\cdot)$, can be applied across dimensions to produce *Feasible Feature Vectors* (❻). Elements of the partition define the abstract features present in a test input and permit the

mapping of latent vectors to feature vectors.

5.1.3 Computing Interaction Coverage

IDC can be configured to use any number of combinatorial interaction coverage measures [72], $\mathcal{M}$, applied to the set of abstracted feature vectors. The grey background in Figure 5.2 depicts the inner workings of the total t-way coverage measure, which accumulates the t-way feature combinations, or t-tuples, present in each test's feature vector (❼). A tuple represents the co-occurrence of t features in the input and is encoded as:

$$(\langle j_1, P(l_{j_1})\rangle, .., \langle j_t, P(l_{j_t})\rangle)$$

where j_1 is the index of the first dimension included in the tuple and j_t is the index of the last dimension in the tuple and $P(l_{j_i})$ is the j_i^{th} element of the feasible feature vector. The accumulated set of tuples across a test suite defines the *Total covered t-tuples*.

To assess whether the covered t-tuples are adequate, IDC computes the size of the feasible coverage domain, e.g., the number of t-tuples that are consistent with the feature model. Determining the number of feasible tuples is, however, non-trivial due to the distribution of the latent space. A feature vector defines a hypercube in the latent space whose sides correspond to the intervals associated with the features in the vector; depicted as the cube in Figure 5.2 (❽). If the intersection of such a hypercube and the target density shell is empty, then IDC considers the feature combination associated with the hypercube to be infeasible. Enumerating the set of feature combinations to determine those that are feasible is intractable, but exploiting the spherical symmetry of the latent space and information from the partition permits efficient calculation of the number of *Total feasible t-tuples* – as depicted by dashed arrows in the figure (❾)

5.2 Approach

IDC applies combinatorial interaction coverage measures on feature abstractions of feasible test inputs to measure DNN test adequacy. As depicted in Figure 5.2, IDC is capable of generating test adequacy measures varying in their rigor. Instantiating IDC for a target dataset ($\mathcal{X}$) requires

defining: an out-of-distribution detector ($\mathcal{O}_\mathcal{X}$), the encoder of a VAE ($\mathcal{E}_\mathcal{X}$), a combinatorial testing metric ($\mathcal{M}$) along with a combinatorial strength (t), a density measure (d) that defines the portion of the $\mathcal{X}$ over which coverage is measured, and a partition of the latent space of the VAE ($\mathcal{P}$). A combination of these defines an IDC metric.

Definition 5 (Input Distribution Coverage)**.** IDC$(\mathcal{O}_\mathcal{X}, \mathcal{E}_\mathcal{X}, \mathcal{M}, d, \mathcal{P}, t)$ *defines a t-strength variant of a test adequacy metric $\mathcal{M}$ that is normalized to target density d of the $\mathcal{P}$ partitioned latent space generated by $\mathcal{E}_\mathcal{X}$ and is applicable to in-distribution inputs as defined by $\mathcal{O}_\mathcal{X}$.*

As demonstrated in Section 5.3, while the choice of these components impacts the sensitivity of IDC to feature variation in tests, IDC is broadly applicable and effective in measuring the feature diversity in tests.

The remainder of this section describes the component, $\mathcal{E}_\mathcal{X}$, and parameters and defines how their instantiations are used to calculate IDC test coverage metrics.

5.2.1 From Inputs to Latent Representations

IDC$(\mathcal{O}_\mathcal{X}, \mathcal{E}_\mathcal{X}, \mathcal{M}, d, \mathcal{P}, t)$ converts test input vectors to vectors of abstract features present in inputs by leveraging learned representations of the input distribution $\mathcal{X}$. These representations result from training a network to encode samples from $\mathcal{X} \subset \mathbb{R}^m$ into a low-dimensional latent space, $\mathcal{E}_\mathcal{X} : \mathbb{R}^m \to \mathbb{R}^d$, where $d \ll m$ as shown in Figure 5.2. For IDC the latent space must reflect the feature variation within $\mathcal{X}$, be dense, and be well-structured so that it is amenable to efficient partitioning and volume-related computations.

Trained encoder networks from a variety of VAE architectures, e.g., [55, 64, 21, 32], give rise to latent representations that meet these requirements, and these architectures are designed with different objectives in mind. For example, the Total Correlation VAE aims to learn non-noise dimensions that are weakly correlated, whereas the two-Stage VAE aims to more accurately match the spherical Normal prior. Selecting the most appropriate VAE architecture and latent dimension for a given data set can be challenging, but Section 5.3 shows that the sensitivity of IDC metrics to feature variation in test inputs is largely insensitive to that choice.

A VAE encoder maps an input to the parameters of a multivariate Gaussian distribution, $[(\mu_1, \sigma_1), \ldots (\mu_d, \sigma_d)]$. In principle, any consistent choice of values from the distribution across

dimensions would suffice. IDC uses the mean values, μ_i, to represent the input in the latent space since they are the most likely values.

Definition 6 (Latent Vector)**.** *Given a trained VAE encoder, $\mathcal{E}_{\mathcal{X}} : \mathbb{R}^m \to \mathbb{R}^d \times \mathbb{R}^d$, the latent vector representation for an input, $\boldsymbol{x} \sim \mathcal{X} \subset \mathbb{R}^m$, is the d-dimensional vector, $\pi_1(\mathcal{E}_{\mathcal{X}}(\boldsymbol{x}))$, encoding the means of the per-dimension latent distributions.*

Ideally, one would train a VAE with orthogonal latent dimensions that exactly correspond to the features describing a dataset. The resulting latent representation would be noise-free, perfectly spherical, and define a minimal space over which feature coverage can be evaluated. In general, however, the number of features that describe real-world datasets is not known, which makes determining the dimensionality for the latent space difficult. Using a VAE whose latent dimension is larger than the number of generative factors for the data set may cause some of the dimensions to reflect just the Gaussian noise of the assumed prior [13, 81, 32]. Such *noise dimension* do not vary with the input and, consequently, they add no value in determining input coverage.

To illustrate the challenge of noise dimensions, Figure 5.3 shows the marginal empirical distributions of the inputs from the Fashion-MNIST test set for latent dimensions 1, 2, and 8 of a Two-Stage VAE. One can observe that the inputs are spread rather evenly within dimension 1, but they are quite concentrated in the other dimensions – with 2 having more spread than 8. The VAE has learned that it does not require the expressive power of dimension 8 to capture the dataset and consequently it maps all inputs to coordinates very near 0 in that dimension. From a coverage perspective most of the partitions of dimension 8, i.e., 1-9 and 12-20, are empty and this means that combinations with those partitions will never be observed and, thus, such dimensions can be eliminated from our coverage calculation.

In IDC noise dimensions are identified by computing their KL-divergence relative to a standard Normal prior across a range of samples from $\mathcal{X}$ [13]. Low KL-divergence values indicate that little information is carried in those dimensions [13] since they match the standard Normal prior. Thought of another way, since just the mean values are used as latent coordinates, low KL-divergence means that the variance in the mean values is also very low.

Definition 7 (Noise Dimensions)**.** *A dimension, $i \in [1, d]$, of the latent space of a trained VAE encoder, $\mathcal{E}_{\mathcal{X}} : \mathbb{R}^m \to \mathbb{R}^d$, is considered noise if $KL(\mathcal{E}_{\mathcal{X}}(T)[i], \mathcal{N}(0, 1)) \approx 0$.*

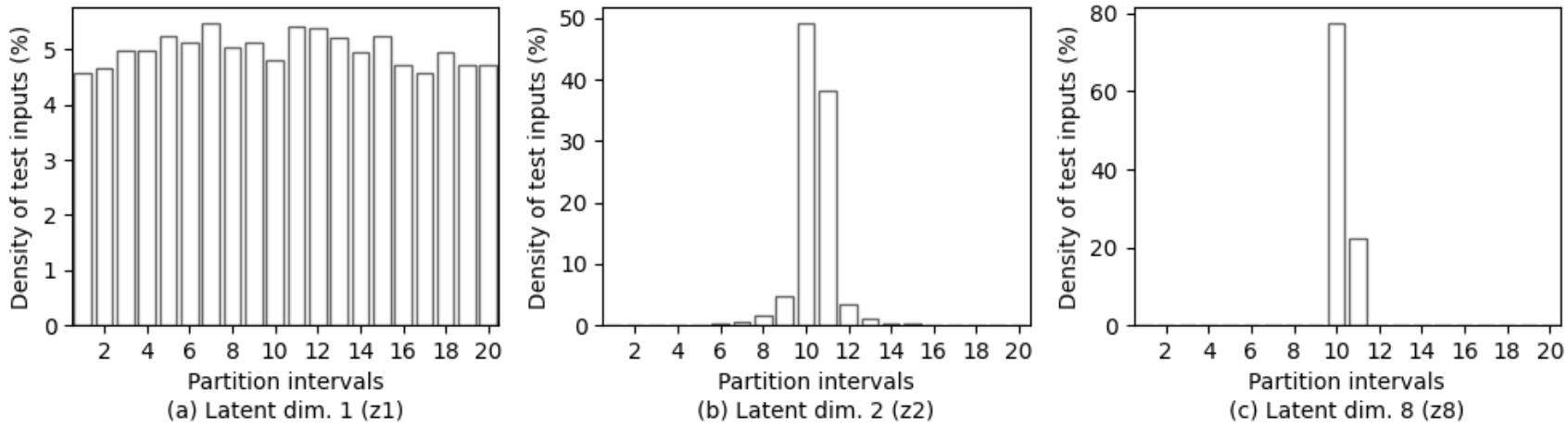


Figure 5.3: The marginal empirical distributions of test inputs for latent dimensions 1, 2 and 8 of a Two-Stage VAE trained on the Fashion-MNIST dataset. The latent dimensions are partitioned into 20 intervals.

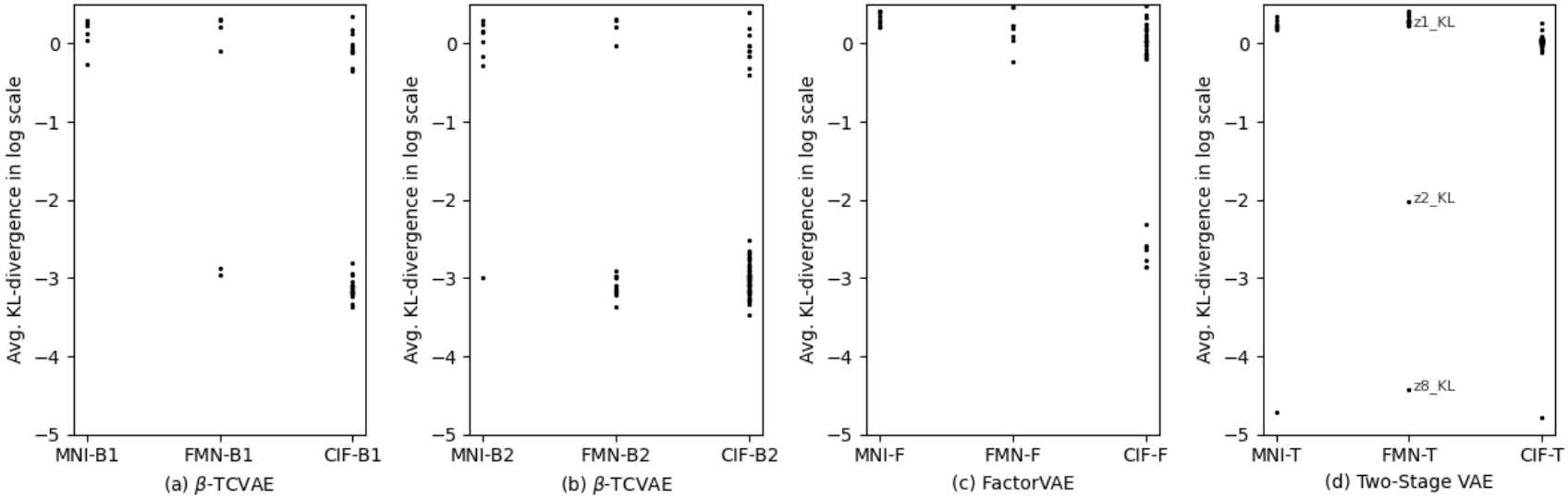


Figure 5.4: Average KL-divergence of the latent dimensions of MNIST, Fashion-MNIST, and CIFAR10 VAEs trained using β-TCVAE, FactorVAE, and Two-Stage VAE configurations described in Table 5.2 plotted in logarithmic scale. Each marker in the scatter plot is the KL-divergence of a latent dimension. z1_KL, z2_KL, and z8_KL in the Two-Stage VAE plot are the KL-divergence values of the latent dimensions 1, 2, and 8 of FMN-T VAE respectively.

Determining an appropriate threshold for judging KL-divergence to be near-zero is dependent on both the data set and $\mathcal{E}_{\mathcal{X}}$. To illustrate how the threshold is calculated, the average KL-divergence for a variety of VAEs and datasets are plotted in Figure 5.4 on a logarithmic scale; details of the VAEs are not important here and are provided in Section 5.3. By analyzing the gaps in KL-values and the marginal distribution of test inputs across dimensions (as in Figure 5.3), A threshold of 10^{-2} was empirically determined to be effective in distinguishing noise and non-noise dimensions for three datasets and four VAEs per dataset. For different datasets and VAEs a similar threshold analysis should be performed.

The KL values for dimensions 1, 2, and 8 as z1_KL, z2_KL, and z8_KL are shown for the Two-Stage VAE for Fashion MNIST (FMN-T in Figure 5.4) respectively. The chosen threshold setting means that the dimension 2 is a non-noise dimension and it would be used to compute IDC coverage. A negative consequence, as mentioned above, is that this will make certain combinations of dimension partitions, i.e., those involving partitions 1-4 and 16-20 of dimension 2, impossible to cover so it will not be possible to achieve 100% IDC.

The prevalence of noise dimensions in a VAE's latent space varies with the architecture and training objective of the VAE. As shown in Section 5.3 this can vary significantly, since some VAEs tend to concentrate information in latent dimensions whereas others spread information across available dimensions. Removing noise dimensions helps IDC focus on the feature variation present in inputs and yield a metric whose feasible domain can be computed cost-effectively.

The research findings in Section 5.3 suggest that IDC can be instantiated with any state-of-the-art VAE, like the β-TCVAE, FactorVAE, and Two-Stage VAE described in Chapter2. All provide an adequate model of the data distribution to compute coverage information that can distinguish the features present in test inputs produced by a variety of different test generation methods. Further work to minimize the number and correlation of VAE dimensions may lead to further improvements in VAEs and this, in turn, could standardize the choice of a VAE to use in IDC for a given dataset.

5.2.2 Feasible Feature Vectors

Figure 5.5 provides a detailed view of the Partition and the target density shell in Figure 5.2. As illustrated on the left side of Figure 5.5, the three dimensions of the sphere are partitioned, within

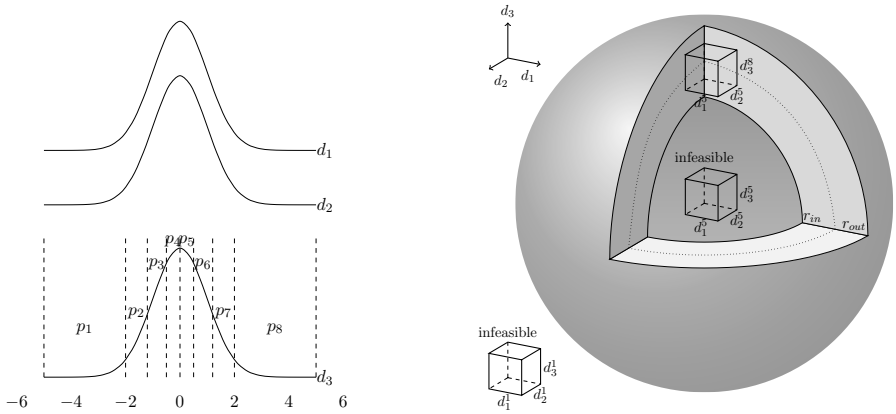


Figure 5.5: Normally distributed dimensions with equal density partitions (left), Multi-variate spherical Gaussian with hyper-rectangle partitions (right)

the range $[-6, 6]$, into regions of equal probability density based on the assumed Normal prior using Definition 3. Due to the spherical prior assumption, all the latent dimensions have the same partitioning.

Such a partitioning gives rise to a combinatorial space of $|\mathcal{P}|^k$ partitions each with approximately equal density. Figure 5.5 depicts such partitions as rectangles whose sides are labeled d_i^j, where i is the index of the dimension and j the index of the partition. As discussed below, IDC operates on a subset of partitions that comprise a target density within which adequacy measures are computed.

In Figure 5.2, a latent vector, l, is checked to determine whether it lies within the target density, $||l|| \in [r_{in}, r_{out}]$, and is ignored by IDC in test adequacy calculations if it does not. The determination of target density only considers non-noise dimensions. Including noise dimensions would increase k while adding only near-zero mean value coordinates along those dimensions and this runs the risk that $||l||$ falls short of r_{in}.

Feasible latent vectors are partitioned, as described above, to form the feature vectors over which coverage is computed. A feature vector represents a set of latent vectors and if any of those latent vectors is feasible, then so too is the feature vector.

Definition 8 (Feasible Feature Vector)**.** *A k-dimensional interval partition, $p \in \mathcal{P}^k$, is feasible relative to a target density, $[r_{in}, r_{out}]$, if $\exists c \in p : ||c|| \in [r_{in}, r_{out}]$.*

While all feature vectors computed from feasible latent vectors are, by definition, themselves feasible, some combinations of partitions may be infeasible. Determining the size of the set of feasible partition combinations is necessary to normalize IDC adequacy metrics.

5.2.3 Feasible Partition Combinations

The full combinatorial space of feasible feature vectors is intractable as a coverage domain so IDC($\mathcal{O}_\mathcal{X}, \mathcal{E}_\mathcal{X}, \mathcal{M}, d, \mathcal{P}, t$) allows coverage to be computed over combinatorial sub-spaces determined by the strength parameter, t. A t-way feature combination defines the feature values for t out of the d dimensions in a feature vector; recall that features correspond to elements of the partition $\mathcal{P}$. The target density constraints mean that some combinations may be infeasible, so IDC restricts its attention to feasible t-way feature combinations.

Definition 9 (Feasible t-way Feature Combination)**.** *For k-dimensional feature vectors partitioned with $\mathcal{P}$, a t-way feature combination, $\langle p_1, \ldots, p_t \rangle$, where $p_i \in \mathcal{P}$, is feasible relative to a target density, $[r_{in}, r_{out}]$, if*

$$\exists c \in \langle p_1, \ldots, p_t, P_{t+1}, \ldots, P_k \rangle : ||c|| \in [r_{in}, r_{out}]$$

where $P_i = \bigcup_{p \in \mathcal{P}} p$.

In essence, this definition asks whether the t-way combination can be extended with $k - t$ partitions to form a feasible feature vector and if so, then the combination is judged feasible.

IDC uses the total number of feasible t-way combinations to normalize combinatorial feature coverage measures as shown in Figure 5.2. This allows IDC to report coverage measures as a percentage of the feasible coverage domain. First, a simpler unconstrained problem is considered to explain how the number of feasible combinations is computed. For a latent space of k dimensions each divided into $|\mathcal{P}|$ partitions, the size of the set of feasible and infeasible t-way combinations is $\binom{k}{t} \cdot |\mathcal{P}|^t$. One could potentially enumerate the $|\mathcal{P}|^t$ sized space and check feasibility of the combinations, but for large $\mathcal{P}$ and t it is too costly. Algorithm 2 presents a more efficient calculation of the number of feasible t-way combinations that exploits the symmetry of the spherical standard Normal distribution.

Since all dimensions range over the same set of values determined by the target density, and are partitioned equivalently, any subset of the t dimensions gives rise to the same feasible space as any other. This enables the calculation of the feasible partition combinations for a single representative subset of dimensions of size t and then multiplying that result by the number of such subsets, i.e., $\binom{k}{t}$, as shown on line 2 in the Algorithm.

Algorithm 2 Calculate the number of feasible t-way combinations

function FEASIBLECOMBINATIONS($t, k, \mathcal{P}, [r_{in}, r_{out}]$)
  return $\binom{k}{t} \cdot$ FTCCOUNT($0, [], \{\}, t, \mathcal{P}, [r_{in}, r_{out}]$)
end function
function FTCCOUNT($depth, path, visited, t, \mathcal{P}, [r_{in}, r_{out}]$)
  $c \leftarrow 0$ ▷ Count rooted at this path
  if $d < t$ **then** ▷ Extend path
    for $p \in \mathcal{P}$ **do**
      $lc \leftarrow 0$ ▷ Local count for path extension
      $bag = path \cup p$ ▷ Order-independent path extension
      if $bag \notin visited$ **then**
        $newpath \leftarrow path + p$ ▷ Append partition to path
        if CHECKCONSTRAINT($newpath, k, \bigcup_{p \in \mathcal{P}} p, [r_{in}, r_{out}]$) **then**
          $lc \leftarrow$ FTCCOUNT($depth + 1, newpath, visited, t, \mathcal{P}$)
        end if
        $visited \leftarrow visited \cup (bag, lc)$ ▷ Record bag with local count
      else
        $lc \leftarrow visited[bag]$ ▷ Retrieve local count for bag
        $c \leftarrow c + lc$
      end if
    end for
  else
    $c \leftarrow 1$ ▷ Feasibility check already performed
  end if
  return c
end function
function CHECKCONSTRAINT($\{[l_1, u_1], \ldots, [l_n, u_n]\}, k, [l_{\mathcal{P}}, u_{\mathcal{P}}], [r_{in}, r_{out}]$)
  $vs \leftarrow fresh(k)$ ▷ A fresh variable for each dimension
  $\phi \leftarrow true$ ▷ Initialize constraint encoding
  for $i \in [1, n]$ **do**
    $\phi \leftarrow \phi \wedge (vs[i] \geq ((u_i > 0)?(l_i)^2 : (u_i)^2))$ ▷ Explicit squared lower bound
    $\phi \leftarrow \phi \wedge (vs[i] < ((u_i > 0)?(u_i)^2 : (l_i)^2))$ ▷ Explicit squared upper bound
  end for
  for $i \in [n + 1, k]$ **do**
    $\phi \leftarrow \phi \wedge (vs[i] \geq 0 \wedge (vs[i] \leq (u_{\mathcal{P}})^2)$ ▷ Maximal squared partition constraints
  end for
  $\phi \leftarrow \phi \wedge (\sum_{i \in [1,k]} vs[i]) \geq (r_{in})^2$ ▷ Squared inner radius constraint
  $\phi \leftarrow \phi \wedge (\sum_{i \in [1,k]} vs[i]) \leq (r_{out})^2$ ▷ Squared outer radius constraint
  return ISSAT(ϕ)
end function

The function FTCCOUNT in the Algorithm, calculates the number of feasible partition combinations by performing a depth-first search (DFS) that enumerates the $|\mathcal{P}|^t$ partition combinations testing if each is feasible according to Definition 9. The DFS considers an ordering of t dimensions along a path in the DFS tree, but as observed above the dimensions are symmetric. For example for two dimensions, i and j, if a combination $(i,p),\ldots,(j,p')$ is feasible then so is $(i,p'),\ldots,(j,p)$. To exploit this *dimension-partition symmetry*, the algorithm uses an order-independent representation of the tree path as a bag of partitions – line 9. For paths that have an unexplored bag representation, the Algorithm extends the path – line 11 – checks that the extended path is feasible – line 12 – and if so recursively counts the combinations rooted at that extended path – line 13 – and records that count along with the bag – line 15. The recorded bag-count pairs are used subsequently to avoid revisiting a DFS sub-tree – line 10 – and to determine the count for such sub-trees – line 17.

The function CHECKCONSTRAINT determines whether the set of interval partitions described by a DFS path is feasible relative to the inner and outer radii of the target density shell. This can be formulated as a satisfiability query over per-dimension coordinate variables ranging over the values defined by each interval. A symbolic L2 norm query is non-linear, as it involves the square root of the sum of the squares of the coordinate variables, which can lead to poor performance by modern SMT solvers. To avoid this complexity, the constraint is formulated over *squared-coordinate* variables – line 27. For each explicit constraint in the path – lines 29-31 – the constraints use the square of the lower and upper bounds of the interval partition; negative intervals require transposing lower and upper bounds. For any remaining dimensions – lines 33-34 – the constraints use the minimal and maximal values for the set of all partitions, $\mathcal{P}$. With squared-coordinates, there is no need to incorporate the square-root function since the inner and outer radii of the target density shell can be directly squared – lines 36 and 37, respectively.

The full space of t-way combinations is $|\mathcal{P}|^t$, which is the number of dimension-partition permutations with replacement. Exploiting symmetries and avoiding the re-exploration of paths in FTCCOUNT reduces the problem to computing the number of dimension-partition *combinations* with replacement, which is $\frac{(|\mathcal{P}|+t-1)!}{t!(|\mathcal{P}|-1)!}$. This computation constitutes an upper-bound on the complexity of FEASIBLECOMBINATIONS since (a) for even very large path lengths, values of k, and $\mathcal{P}$ the linear constraint formulation and use of incremental SMT solving results in essentially constant time performance for CHECKCONSTRAINT, and (b) the detection of infeasible combinations only

further reduces exploration of DFS paths. Section 5.3 studies problems with $|\mathcal{P}| = 10$ and $t = 3$ and the use of Algorithm 2 reduces the number of explored paths, and SMT queries, from 1000 to 220.

5.2.4 Refining IDC Metrics

IDC is sensitive to three parameters. First, the partition granularity, $|\mathcal{P}|$, can be varied to yield a coarse or fine partition of the latent space; Figure 5.5 shows three dimensions partitioned 8 ways. Second, the target density, d, of the latent space can be varied, which makes the shell enclosing the high probability region of the multivariate standard Normal distribution thinner or thicker. Third, requirements on the combinations of dimension-partition pairs, t, can be varied. In Figure 5.5 there are 8^3 possible 3-way combinations. The rectangle in the shell above the origin with sides $\langle d_1^5, d_2^5, d_3^8 \rangle$ is a combination with d_1 and d_2 having a value in p_5 and d_3 having a value in p_8. Any input mapped by the $\mathcal{E}_{\mathcal{X}}$ to a latent vector in this rectangle will cover a single 3-way combination, 3 2-way combinations, and 3 1-way combinations. Varying the strength of combinatorial coverage in IDC provides a means of controlling coverage granularity. In Section 5.3 these parameters are fixed to study IDC using a relatively fine coverage metric that spans the vast majority of the latent space. Future work will explore how best to tune these parameters based on the needs of software testers.

5.3 Evaluation

The cost-effectiveness of IDC and how it compares to alternative approaches to assessing DNN testing techniques is explored through a set of experimental studies focused on the following questions:

RQ1 How well does IDC reflect the feature interactions present in a test suite?

RQ2 Does IDC correlate with the fault detection capabilities of test suites?

RQ3 How costly is it to compute IDC?

RQ4 How does IDC vary with changes to the underlying VAE?

RQ5 How does IDC compare with existing coverage metrics in measuring coverage of in-distribution test inputs generated using diverse test generation techniques?

RQ6 How does IDC compare with DNN mutation testing with respect to feature diversity in a test set?

5.3.1 Dataset Selection and IDC Instantiation

To evaluate IDC, IDC must be instantiated and applied to compute the test adequacy of DNN test suites. As described in Section 5.2 there are 6 parameters to choose for instantiating IDC and several of those parameters depend on the choice of dataset. The rationale for selecting datasets to be used across the evaluation is described first. The choice of datasets was influenced by the availability of other artifacts that are used to define experimental treatments or to serve as baselines in studies designed to answer the RQs.

Dataset and Artifact Selection

Answering RQ1 requires a dataset for which the ground truth set of features is known. This led to the choice of using the dSprites [88] dataset which is comprised of 64 by 64 black and white images of a small set of white shapes placed in various positions on a black background. This synthetic dataset was designed for the evaluation of unsupervised techniques that attempt to disentangle – or decorrelate – dimensions in the learned representation. The developers of dSprites provide a key that allows mapping from 5-dimensional orthogonal feature space to images in the dataset. As explained below, this permits the definition of experimental treatments for answering RQ1.

The application of IDC does not require knowing the ground truth set of features, and this results in more freedom in choosing datasets for answering questions RQ2 through RQ6. To control costs in answering RQ2 through RQ6, a set of artifacts are developed that could support studies across these questions. Since the set of features will vary with the dataset, multiple datasets are selected for the experiments in order to explore IDC's ability to handle that diversity. The breadth of the RQs also requires a variety of other artifacts available: multiple DNNs for the datasets that can serve as test subjects (RQ2,RQ5), multiple VAE models for the datasets (RQ2,RQ3,RQ4), neural network test coverage metrics that are applicable to DNNs for the datasets (RQ5), neural network test generation techniques that are applicable to the datasets (RQ5), and neural network mutation techniques that are applicable to the datasets (RQ6).

Collectively these requirements led to the selection of MNIST [75], FashionMNIST [141], and

CIFAR10 [70] for the experiments. MNIST is a collection of 28 by 28 gray-scale images of hand-written digits with 60000 training images and 10000 test images . FashionMNIST is a collection of 28 by 28 gray-scale images of articles of clothing with 60000 training and 10000 test images. CIFAR10 is a collection of 32 by 32 color images of a variety of animals and vehicles with 50000 training and 10000 test images. While there is no known ground truth set of features for these datasets, a study of them showed that their feature sets are quite distinct, e.g., the thinness and tilt of a digit in MNIST, the arms and neckline of a blouse in FashionMNIST, and the legs of an animal or the wings of a plane in CIFAR10. This made them a good choice to assess the breadth of applicability of IDC.

In addition, these datasets are widely studied in machine learning and DNN testing literature so there are a wealth of existing artifacts that could be reuse from prior work. For each of the datasets, standard DNN models with different architectures are selected from the literature as described in Table 6.1. LeNet-1, LeNet-4 and LeNet-5 [76] models are used for the MNIST dataset. Three of the benchmark models available in the Fashion-MNIST github repository [139] are used for the Fashion-MNIST dataset, and All-CNN-A, All-CNN-B, and All-CNN-C models [122] are used for the CIFAR10 dataset. All the models used in the studies are convolutional neural networks with convolutional, max-pooling, dropout, and dense layers. The available implementations of existing DNN test generation techniques – DeepXplore [99], DeepTest [129], and DLFuzz [47] – are applicable to these datasets and models as are the implementations of white-box coverage frameworks – Neuron Coverage [99], extended Neuron Coverage metrics [84], and Surprise Adequacy [65]. Internal threats to validity are reduced by selecting only those test generation techniques for which implementations are available. For comparison with mutation approaches, DeepCrime [58] is selected since it has been shown to be superior to prior work, e.g., DeepMutation [85], and has artifacts that support its application to the selected datasets. As discussed in RQ6, its application to CIFAR10 is found to be cost-prohibitive so DeepCrime is considered with only MNIST and FashionMNIST.

RQ2 also uses ImageNet dataset [33] for demonstrating the applicability of IDC to complex datasets. ImageNet is a large scale dataset containing images from 1000 object categories, with a training dataset of size 1281167 and test dataset of size 50000. Three DNN pretrained models available with the Keras framework [24], VGG16 [117], VGG19 [117] and ResNet50 [51] are used as the DNN models under test, IMG-M1, IMG-M2 and IMG-M3, for the ImageNet experiment. The

Dataset	Name	Architecture	#Parameters	Test Accuracy
MNIST [75]	MNI-M1	LeNet-1 [76]	7206	97.88%
	MNI-M2	LeNet-4 [76]	69362	98.52%
	MNI-M3	LeNet-5 [76]	107786	98.53%
Fashion [141]	FMN-M1	Custom [102]	1.6M	93.58%
	FMN-M2	Custom [140]	3.3M	92.26%
	FMN-M3	Custom [39]	211690	92.57%
CIFAR10 [70]	CIF-M1	ALL-CNN-A [122]	1.2M	82.67%
	CIF-M2	ALL-CNN-B [122]	1.3M	83.84%
	CIF-M3	ALL-CNN-C [122]	2.9M	81.45%
ImageNet [33]	IMG-M1	VGG16 [117]	138.4M	71.3%
	IMG-M2	VGG19 [117]	143.7M	71.3%
	IMG-M3	ResNet50 [51]	25.6M	74.9%

Table 5.1: Models used in our studies with number of parameters (#p), and test accuracy; "M" denotes millions of parameters. **Name**s are shorthands used in subsequent discussion.

models are described in Table 6.1.

IDC Instantiation

Once a dataset has been selected, Definition 5 requires the choice of six parameters to instantiate IDC: an OOD input detection technique ($\mathcal{O}$), a VAE encoder ($\mathcal{E}$), a combinatorial coverage metric ($\mathcal{M}$), the target density (d), the latent partition ($\mathcal{P}$), and the combinatorial strength (t). The space of choices for instantiating IDC is large, so several pragmatic choices are considered to reduce the size of that space in answering the RQs.

Choosing $\mathcal{M}$, d, and t To explore the use of IDC to gain insight into how thoroughly a test suite reflects the features present in a dataset, IDC is configured such that its coverage metric would not easily saturate. In the extreme selecting t=1 and $|\mathcal{P}| = 2$ would yield a trivial coverage space which would have saturated on even the smallest sample of test inputs. In the combinatorial coverage literature, e.g., [72], the standard metric for assessing feature interactions is total 2-way interaction coverage. Exploring the use of 2-way coverage in preliminary studies determined that it also led to relatively early saturation. This led to the selection of $\mathcal{M}$ as total t-way interaction coverage with $t = 3$ since it yielded an IDC instantiation that did not easily saturate.

IDC can be focused on different regions of the latent space by defining the target density.

Dataset	Name	VAE		Latent Size	Non-Noise Latent Size
		Objective	Architecture		
MNIST	MNI-B1	β-TCVAE	Burgess et al [13]	6	6
	MNI-B2	β-TCVAE	Burgess et al [13]	8	7
	MNI-F	FactorVAE	Burgess et al [13]	8	8
	MNI-T	Two-Stage VAE	InfoGAN [22]	10*	9
Fashion	FMN-B1	β-TCVAE	Burgess et al [13]	6	4
	FMN-B2	β-TCVAE	Burgess et al [13]	16	4
	FMN-F	FactorVAE	Burgess et al [13]	8	8
	FMN-T	Two-Stage VAE	InfoGAN [22]	16*	15
CIFAR10	CIF-B1	β-TCVAE	Burgess et al [13]	32	11
	CIF-B2	β-TCVAE	Burgess et al [13]	64	11
	CIF-F	FactorVAE	Burgess et al [13]	32	25
	CIF-T	Two-Stage VAE	InfoGAN [22]	64*	63
dSprites	DSP-B1	β-TCVAE	Burgess et al [13]	5	5
	DSP-B2	β-TCVAE	Burgess et al [13]	10*	5
	DSP-F	FactorVAE	Burgess et al [13]	5	5
ImageNet	IMG-F	FactorVAE	Burgess et al [13]	64	33

Table 5.2: VAE models used in IDC evaluation detailing the training objective, encoder/decoder architecture, latent dimension size, and the number of non-noise latent dimensions. Models marked with an asterisk use latent sizes from prior work with the best performance and serve as an upper bound for exploring alternative latent sizes.

Choosing a low value for d yields a small space to be covered, whereas choosing a value close to 1 forces IDC to consider a much greater portion of the latent space – extending out on the tails of the distribution. Consistent with the choice for t that selects a challenging coverage goal and, after some preliminary experimentation, d is selected as 0.9999. Fixing the choices for $\mathcal{M}$, t, and d reduces the breadth of this study, but they represent a point in the space of IDC instantiations that corresponds to a rigorous coverage metric of the kind that might be applied for testing critical systems. Experimentation with more rigorous instantiations that increase t and d is left for future work.

Choosing $\mathcal{O}$ The design of our experiments for RQ1, RQ2, RQ4, and RQ6 makes use of the test datasets for dSprites, MNIST, FashionMNIST, and CIFAR10. These are assumed to be on the data distribution so an OOD filter is not required for those experiments. RQ3 and RQ5 use the most accurate OOD algorithm available – the state-of-the-art Likelihood Regret [143] (LR)

score based technique. Similar to the OOD detector threshold computation in Section 3.3 one must define a set of inputs that are considered to be out-of-distribution and then use that set to compute an LR score threshold. The OOD inputs for the datasets used by LR are used in this work. For MNIST, a randomly sampled set of inputs, 1000 each from Fashion-MNIST [141], CIFAR10 [70], SVHN [94], KMNIST [26], notMNIST [12], Noise and Constant are used as OOD inputs. Noise images are randomly sampled images with pixels in the range [0, 255], and Constant images are images with randomly sampled constant value in the range [0, 255] across each of the channels [143]. For Fashion-MNIST, Fashion-MNIST is replaced with MNIST for defining OOD inputs. For CIFAR10, MNIST, Fashion-MNIST, SVHN, CelebA [**celeba**], Noise and Constant are used as OOD inputs. For each dataset, the F-score is calculated to determine the threshold. This involves sampling from the test dataset and OOD inputs and computing their LR scores and then sweeping through a range of threshold values to determine the threshold with the highest F-score.

RQ3 also uses two alternative OOD techniques, Input-Complexity (IC) [113] and ODIN [79], since these established the state-of-the-art prior to the development of the LR technique.

Choosing $\mathcal{E}$ The central role that the latent space plays in IDC motivated the exploration of a variety of different VAE models in answering RQ1 through RQ4; RQ5 and RQ6 uses the most effective VAE per dataset as determined in the earlier studies. To explore how varying the VAE impacts IDC, three state-of-the-art families of VAE models – β-TCVAE [21], FactorVAE [64] and Two-Stage VAE [32] – are selected to develop the models summarized in Table 5.2. As explained in Chapter 2, a VAE is comprised of the encoder and decoder architectures, the size of the latent space, and the training objective. The best-performing VAE published in the literature (marked with an asterisk in Table 5.2) are used as a starting point, e.g., a 16 dimensional latent space for the Two-Stage VAE yielded the lowest FID score for Fashion MNIST [32]. Then the objective function, architecture, and latent size are varied by incorporating elements from prior work. The goal is to use a less sophisticated objective, a simpler architecture, or a smaller latent size in order to reduce the ability of the VAE to accurately model the data distribution. In this way, the research could explore how IDC operates with a state-of-the-art VAE and with VAEs that fall short of the state-of-the-art.

Section 5.2 describes the identification of noise dimensions in the latent space by calculating the

KL-divergence between each dimension, as expressed by the mean and variance output by the VAE encoder, and a standard Normal distribution across the test datasets of MNIST, Fashion-MNIST, and CIFAR10. Dimensions whose KL value is greater than or equal to 10^{-2} are considered non-noise dimensions. Table 5.2 reports the number of non-noise dimensions computed for each of the VAE configurations.

As with any neural network, training a VAE is subject to stochastic variation. To explore the impact of such variation, RQ1 trains each VAE in Table 5.2 ten times and provide a statistical characterization of the findings. For the remaining RQs, a VAE in trained only a single time, since the results from RQ1 suggest that it is not a significant factor influencing IDC performance.

Choosing $\mathcal{P}$ The studies use the equal density partitioning scheme from Definition 3 since it seeks to evenly balance the distribution across bins. Consistent with the choice for t and d, $|\mathcal{P}|$ was selected so that the IDC metric does not prematurely saturate. This is especially relevant for RQ5 where the experimental design seeks to determine whether coverage metrics, including IDC, are sensitive to incremental increases in coverage produced by different test generation approaches. Toward this end a preliminary study was conducted that varied the partition size between 5 and 25 and evaluated the total 3-way coverage computed using all of the VAEs in Table 5.2 for the MNIST, Fashion-MNIST, and CIFAR10 test sets. Based on the results, shown in Figure 5.6, a value of 20 is selected for $|\mathcal{P}|$ since it guaranteed that regardless of the VAE or dataset the test set would not achieve more than 80% coverage. This leaves a substantial portion of the total 3-way coverage space uncovered and this space is used as the domain for RQ5. While this choice is based on the experiment design for RQ5, the same partition size is used for RQ2, RQ3, RQ4, and RQ6 leaving the exploration of alternative partitioning schemes and sizes to future work.

5.3.2 Experimental Studies and Results

For all of the RQs, except RQ3, the cost to execute IDC is not relevant, so to facilitate experimentation a large and diverse set of machines are used to execute the IDC algorithms. For RQ3 that studies the cost of IDC, experiments are performed on a server with an Intel(R) Xeon(R) Bronze 3104 CPU, 1.70GHz, with 125GB of memory, and an NVIDIA GeForce GTX 1080 Ti GPU with 11G of VRAM.

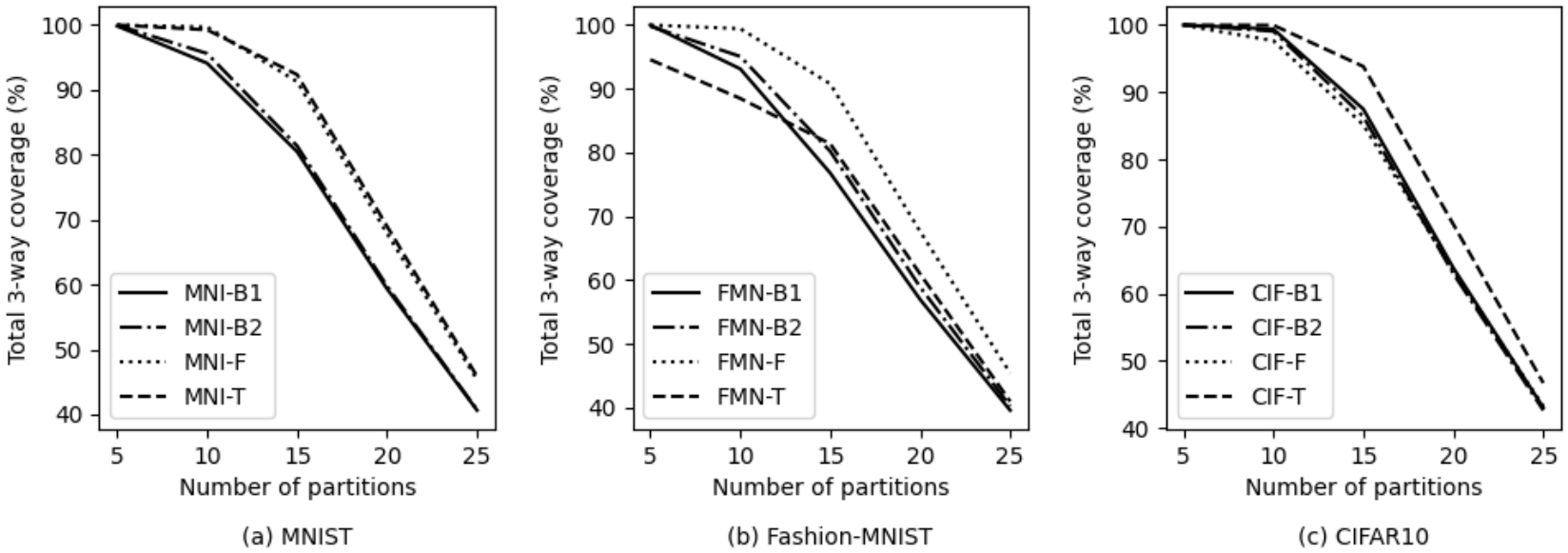


Figure 5.6: Plots showing the variation in the total 3-way coverage with respect to the number of partitions of IDC.

RQ1: How well does IDC reflect the feature interactions present in a test suite?

To answer this question, the study uses an experiment that allows the manipulation of the diversity of features present in a test suite and controls feature diversity while manipulating test suite size.

Manipulating feature diversity requires a dataset for which the ground truth set of features is known. Rather than generate a synthetic dataset, the study uses the dSprites dataset [88] which has been used in numerous studies of generative models in the ML community [13, 64, 55]. The feature model of dSprites defines 5 orthogonal features: shape, scale, orientation, X-position, and Y-position. The shape feature is defined as the set of squares, ellipses, and hearts. The other features range over continuous values that are partitioned into disjoint intervals: scale has 6 intervals of scaling factors between 0.5 and 1; orientation has 40 intervals of degrees of rotation between 0 and 360 degrees; and X/Y-positions have 32 intervals of displacement along the axes between 0 and 1. In total there are $3 * 6 * 40 * 32 * 32 = 737280$ combinations across the dSprites feature model. For each combination, sampling from the feature intervals gives a specific shape, scale, orientation, and X/Y-position which is used to generate a 64 by 64 black and white image.

For any choice of values in the feature model, e.g., shape is heart, orientation is between 9 and 18 degrees, one can select the subset of the test or training data corresponding to those values. This allows the construction of sets of inputs that vary with the dSprites ground truth features.

The experimental treatments involve: (1) varying the feature diversity of test inputs by including K tests with previously unseen features, and (2) holding the feature diversity of test inputs fixed and increasing test size by $9K$. For treatment 1, K is determined per feature by separating the test set into subsets corresponding to specific feature values. For shape K is $73728/3 = 24576$, since there are 3 shape values. For scale and X/Y position K is 12288 and 2304, respectively and for orientation, K is approximately 1843. For treatment 2, the same separation approach is applied to the training set to generate subsets with specific feature values. By design, treatment 2 adds substantially more tests than treatment 1 in order to provide more convincing evidence that the size of the test suite is not contributing to the measured feature interactions.

The study measures total 3-way coverage of a VAE model using a 10-way partitioning of latent dimensions. For such a model, there are a total of $\binom{5}{3}10^3 = 10000$ 3-way combinations that can be observed.

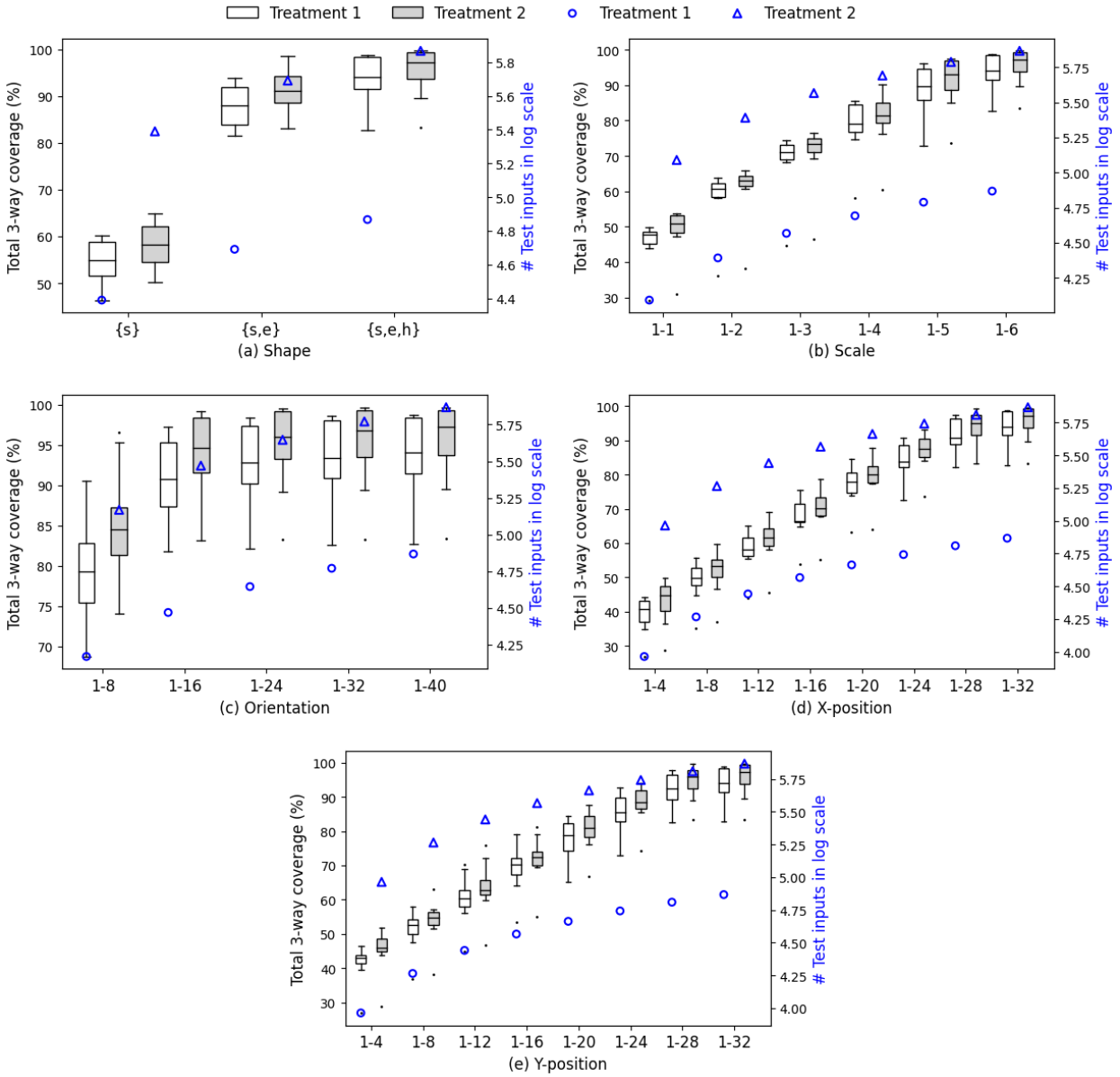


Figure 5.7: Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the VAE configuration, DSP-B1.

For each feature, the study successively applied the two treatments to subsets of values, i.e., sets of shapes or intervals, and record the total 3-way coverage. The study also recorded the sizes of the sets resulting from these treatments for additional context.

Three VAE architectures, a β-TCVAE with a latent size of 5 (DSP-B1), another β-TCVAE with a latent size of 10 (DSP-B2) and a FactorVAE with a latent size of 5 (DSP-F) are used for this study, and the configurations are described in Table 5.2. Each VAE architecture is trained 10 times to account for the stochastic variation in the training process. The study uses a 90%/10% training/test split which results in test sets with 73728 images.

Fig. 5.7 plots the total 3-way coverage IDC recorded when using a DSP-B1 VAE. The plots for the total 3-way coverage of DSP-B2 and DSP-F VAEs, which follow a similar trend, are included in Appendix A. Along the x-axis, the figure shows a series of pairs of box plots which show the median – centerline of the box – and the first and third quartiles. The white box shows the coverage measured – along the left y-axis – when selecting the test subset corresponding to the feature values in the x-axis label. For example, in Fig. 5.7(a) the label "{s,e}" means shapes may be either a square or an ellipse and in Fig. 5.7(c) the "1-16" label means the orientation may be in any of the first 16 intervals which correspond to a rotation of $[0, 144]$ degrees. The gray box shows the coverage measured when selecting the subset of the full dataset – training and test data – corresponding to the feature values. The plots include a blue circle – below the white box – and a blue triangle – above the gray box – that indicate the test set size in log-scale along the right y-axis.

For a given white box, the results for treatment 1 are the next white box to the right and the results for treatment 2 are the next gray box to the right. For example, in Fig. 5.7(b) consider the white box corresponding to "1-2" with a median total 3-way coverage value of 60.64%. Treatment 1 adds 12278 tests with scale feature value "3" to compute the white box corresponding to "1-3" with a median of 70.93%. Treatment 2 adds 221,212 tests with scale feature values "1-2" to compute the gray box for "1-2" with a median of 63.16%.

Since movement along the x-axis in these plots corresponds to cumulative growth in the feature value sets it is natural that the coverage metric will saturate – though it does so at different rates for different features. Prior to saturation, however, 585 out of 750 treatments show that adding a number of diverse tests increases total 3-way coverage more than adding approximately 9 times

the number of non-diverse tests. Variation in VAE training appears to not impact this trend as the quartiles resulting from treatment 1 and 2 minimally overlap in the regions of the plots prior to saturation.

The study measured the statistical significance of the hypothesis that IDC is significantly more sensitive to increases in test diversity in comparison to increases in test size using the Wilcoxon signed-rank test which is a non-parametric test to compare two related samples. The p-value of the tests for DSP-B1, DSP-B2, and DSP-F are 1.81e-33, 3.99e-17, and 3.69e-34 respectively which indicate a strong confidence in the test hypothesis. Next, the study measured the p-values for each feature across the three VAE configurations. The test hypothesis is valid for shape, scale, X-position, and Y-position features with p-values 7.53e-9, 6.06e-25, 3.01e-30, and 9.27e-28 respectively while the test failed for the orientation feature with a p-value of 0.99.

The high p-value for orientation motivated the exploration of the dSprites data further. While the dSprites feature model defines the 5 features as orthogonal, the generation of input images introduces a correlation between the shape and orientation features. This is due to the rotational symmetry of squares and ellipses. The images are only able to reflect 90 degrees of variation for squares and 180 degrees of variation for ellipses. The collapsing of the feature space to the image space for shape and orientation relationships can be observed in the data. Consider the plot for the orientation feature in Fig. 5.7(c). In the feature space a square rotated by 0, 90, 180, or 270 degrees all map to the same image. The VAEs feature space is based on the features present in this image space which only reflects distinctions within some 90 degree interval. Thus, the manipulation of the orientation feature moving along the x-axis in this plot results in all 3-way combinations involving squares to have been observed by the time interval 10 is considered – in the bar labeled "1-16". The interaction between orientation and the ellipse also contributes to this early saturation. The same trend can be observed in the shape feature data in Fig. 5.7(a). This research conjectures that this correlation of features in the image space led to the study's inability to show that orientation feature diversity is more significant than test size in increasing IDC coverage.

Result for RQ1: IDC is effective in measuring the feature interactions present in a test suite. The t-way combinations of the partitioned latent space of the VAE reflect feature interactions present in the test suite.

RQ2: Does IDC correlate with the fault detection capabilities of test suites?

To address this question, this study performs an experiment to demonstrate that test suites with higher total 3-way coverage uncover more faults when compared to test suites with lower total 3-way coverage. IDC uses the input data distribution as a domain for measuring test coverage. The total 3-way coverage measured using IDC is effective in identifying faults when faults are spread across the data distribution, in which case a higher test coverage measured by IDC corresponds to higher fault detection capability. The fault detection capability of IDC is demonstrated with an experimental study that explores the number of fault-revealing test inputs contained in test suites with monotonically increasing test coverage values. The study uses three DNN models each for the MNIST, Fashion-MNIST, and CIFAR10 datasets (see Table 6.1) to identify the number of fault-revealing test inputs present in the test suites used in this study.

For this experiment, a sequence of test suites with increasing total 3-way coverage are generated. The sequence of tests is constructed incrementally and at each step a new set of tests are added that cover previously uncovered partitions of a latent dimension. Since covering more partitions along a latent space corresponds to more feature interactions, this method generates test suites with monotonically increasing total 3-way coverage. Test suites with increasing total 3-way coverage are created for each of the VAE architectures shown in Table 5.2 for the three datasets. The methodology for creating test suites for the MNI-B1 VAE configuration is explained in detail. The same methodology is followed for creating test suites for the other VAE configurations. MNI-B1 VAE has 6 non-noise latent dimensions, $\{z1, z2, z3, z4, z5, z6\}$, and each latent dimension is partitioned into 20 intervals. Test suites $TS_i : i \in [1, 20]$ are created for MNI-B1 for each of the 6 latent dimensions. TS_1 for $z1$ contains all the test inputs from the MNIST test dataset that are in the first interval of the partitioned $z1$. $TS_i : i \in [2, 20]$ of $z1$ contains all the test inputs from TS_{i-1} augmented with the test inputs in the partition i of $z1$. The number of fault-revealing test inputs and the total 3-way coverage are measured for all the test suites. The total 3-way coverage of the test suites varies with the underlying VAE configuration used by IDC, and it remains the same for different DNN models under test as IDC is a black-box method.

The total 3-way coverage of the test suites created for the MNIST dataset and the number of fault-revealing test inputs measured using DNN models, MNI-M1, MNI-M2, and MNI-M3 are

shown in Figure 5.8 and Figure 5.9. Data for Fashion-MNIST and CIFAR10 datasets show the same trends and are provided in Figure A.3 and Figure A.4 in Appendix A. The curve showing the total 3-way coverage corresponding to the test suites TS_1-TS_{20} is plotted in the top row of Figure 5.8. The curve is monotonically increasing which confirms that the test generation approach produced a sequence of tests with increasing total 3-way coverage. There is a sequence of 20 tests constructed for each latent dimension of the VAE. While all of these test sequences exhibit the monotonic coverage increase, there is some fluctuation based on how the test data populate each dimension partition.

To explain in more detail, consider the vertical dotted lines in the plots toward the upper left corner of Figure 5.8. The intersection point of the vertical dotted line and the curve of $z1$ in the leftmost plot of the top row is the total 3-way coverage of the test inputs of TS_{10} created for the first latent dimension of MNI-B1 VAE; the intersections with each of the other dimension curves show coverage for the other per-dimension test suites. The intersection points of the vertical dotted line with the curves in the plots of the bottom row indicate the number of test inputs of TS_{10} created for each latent dimension of the MNI-B1 VAE that are mispredicted by the DNN MNI-M1.

The bottom row of Figure 5.8 and the two rows in Figure 5.9 show the number of mispredicted labels by test inputs corresponding to the sequence of test suites along the x-axis for a different DNN under test with varying VAE across the columns. Note that the number of non-noise dimensions varies with the VAE.

It can be observed from the plots in Figure 5.8 and Figure 5.9 that the number of fault-revealing test inputs follows a monotonically increasing relationship with the test suites on the x-axis which implies that the fault-revealing test inputs are distributed across the latent space. This is due to the design of the per-dimension test suites which control the partitions involved in a test – and thereby the interactions present in the test suite. To emphasize the point, a plateau in these plots would indicate that the partitions covered and interactions appearing in a test suite are increased without detecting any new faults, but this is not observed in the data.

There are 60 pairs of trend lines across this experiment – each dimension's total 3-way coverage curve compared with each dimension's fault-detection curve per DNN. The study measures the Spearman's rank correlation coefficient across these pairs to determine the relationship between the total 3-way coverage of a test and its fault-detection effectiveness. The correlation coefficient

values range from 0.99 to 1 across all the models and VAE configurations, which implies a positive correlation between the total 3-way coverage and the number of faults detected by a test suite. Hence the test suites with higher total 3-way coverage uncover more faults when compared to the test suites with lower total 3-way coverage.

The plots for the FMN-T VAE configuration shown in Figure A.3 in Appendix A contains a latent dimension whose curves for the total 3-way coverage and the number of fault-revealing test inputs follow the shape of a step function. This latent dimension has a KL-divergence value of 10^{-2} which is the threshold point described in Section 5.3.1. This latent dimension has very few test inputs in the first eight intervals because of which the number of fault-revealing test inputs and the total 3-way coverage follow the same trend.

A case study is also conducted to demonstrate that IDC is effective for complex datasets by repeating the experiment for ImageNet [33]. A FactorVAE configuration, IMG-F in Table 6.1, is trained with 64 latent dimensions using the ImageNet dataset. Three DNN models trained on the ImageNet dataset, IMG-M1, IMG-M2 and IMG-M3, (see Table 6.1) are used in the experiment. While the DNN models are trained using 224x224 size inputs, the study downsampled the ImageNet dataset size to 32x32 using the procedure described in a research work [25] in order to speed up the VAE training.

Results of the experiment with the ImageNet dataset are presented in Figure 5.10. The plots show the total 3-way coverage and the number of test inputs mispredicted corresponding to the test suites TS_1-TS_{20}. The total 3-way coverage and the number of fault-revealing test inputs have a positive correlation with Spearman's rank correlation coefficient 1. The results show that IDC can be applied to complex datasets, such as ImageNet.

Result for RQ2: IDC has a positive correlation with the number of faults present in the test suites across all the DNN configurations studied.

RQ3: How costly is it to compute IDC?

To compute cost the study measures the runtime of the major components of IDC. While there are many components to IDC, the preliminary experiments determined that the bulk of the cost lies in performing OOD filtering and the VAE encoding, each of which is performed per-test. Computing the total number of feasible partition combinations using Algorithm 2 and the combinatorial cov-

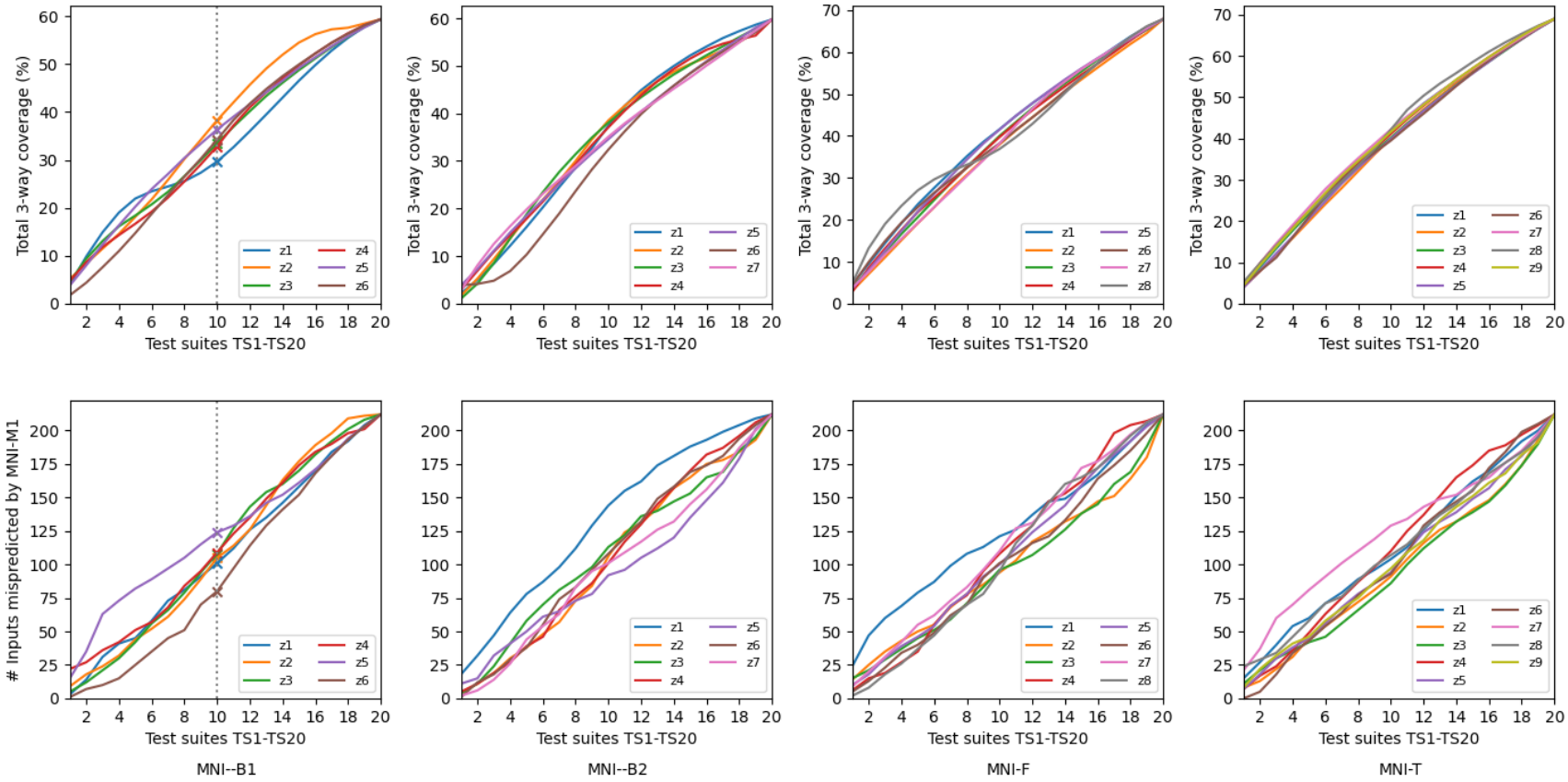


Figure 5.8: The total 3-way coverage (top row) and the number of fault-revealing test inputs (bottom row) of the test suites created using MNIST dataset for the latent dimensions (z) of VAE configurations MNI-B1, MNI-B2, MNI-F and MNI-T from Table 5.2. DNN model MNI-M1 from Table 6.1 is used for calculating the number of fault-revealing test inputs in the test suites for the bottom row plots.

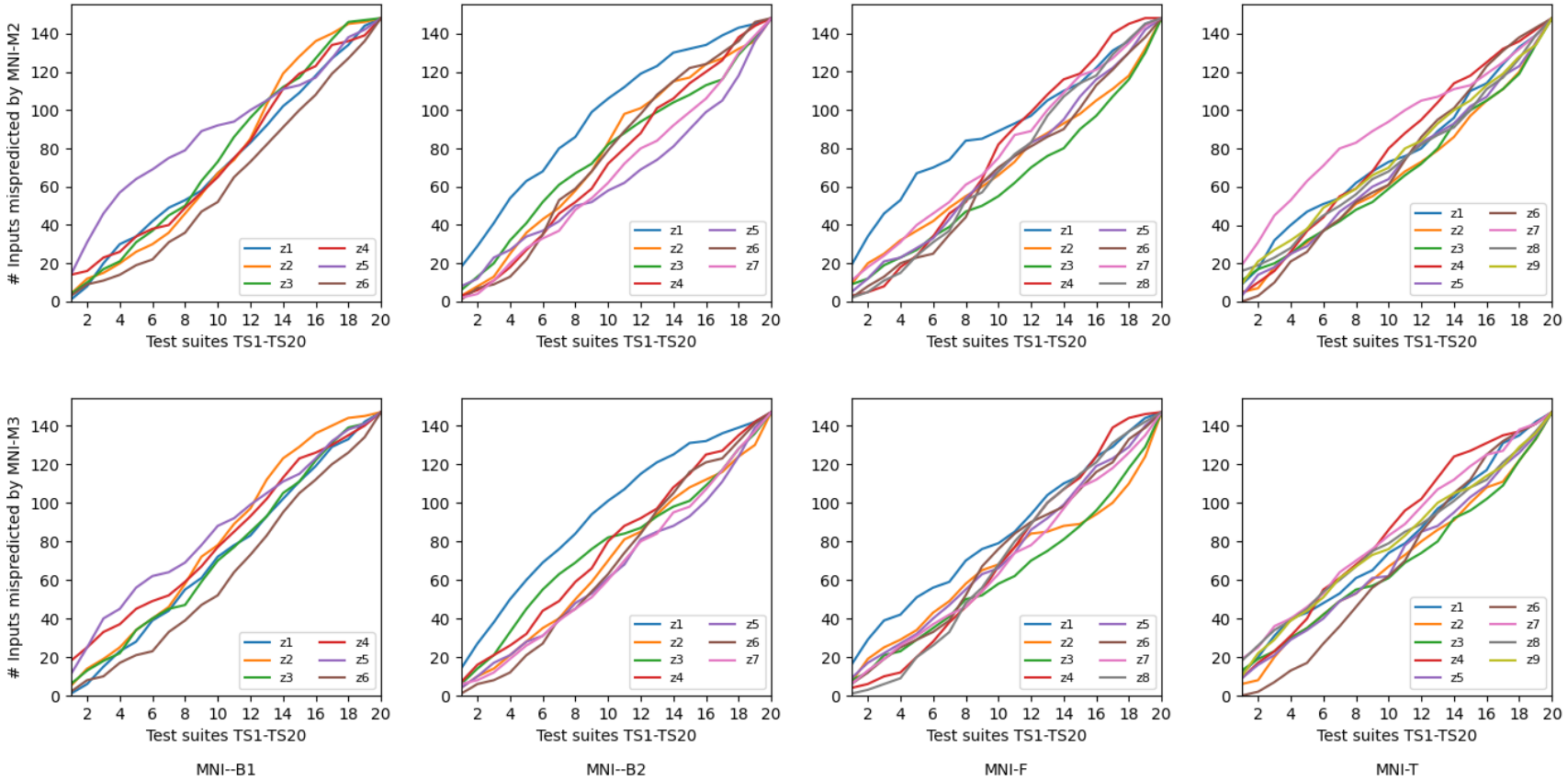


Figure 5.9: The number of fault-revealing test inputs of the test suites created using MNIST dataset for the latent dimensions (z) of VAE configurations MNI-B1, MNI-B2, MNI-F and MNI-T from Table 5.2. DNN models MNI-M2, and MNI-M3 from Table 6.1 are used for calculating the number of fault-revealing test inputs in the test suites for the top and bottom row plots respectively.

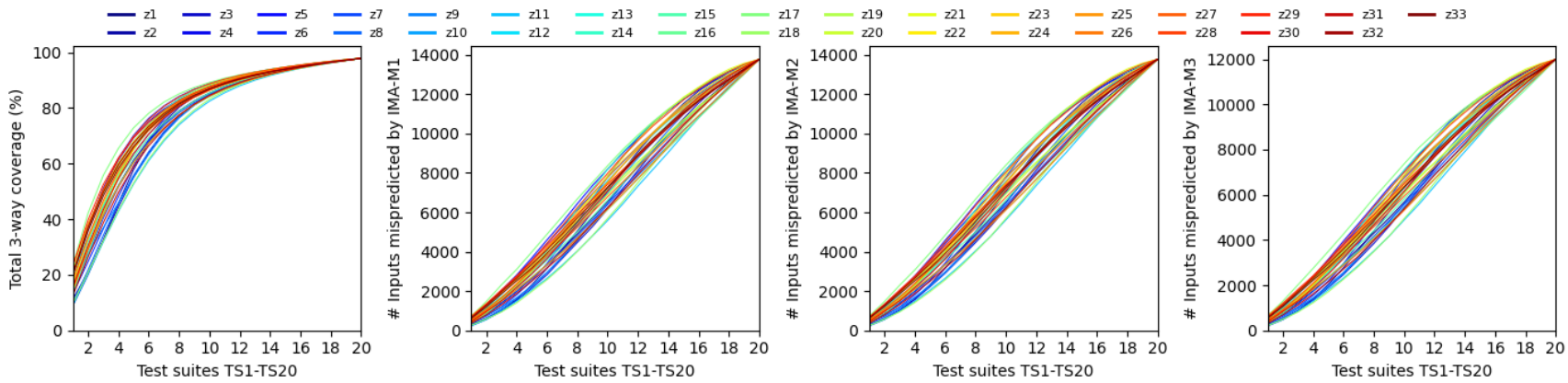


Figure 5.10: The total 3-way coverage (leftmost) and the number of fault-revealing test inputs (three rightmost) of the test suites created using ImageNet dataset for the latent dimensions (z) of a FactorVAE configuration. DNN models IMG-M1, IMG-M2 and IMG-M3 from Table 6.1 are used for calculating the number of fault-revealing test inputs in the test suites. The legend for these plots, shown spanning the page above them, defines the colors for each of the 33 latent dimensions - z1 through z33.

erage metric, each of which is computed once per test suite, also take non-trivial time. The other components, e.g., determining noise dimensions, computing the partitions, and filtering based on target density as depicted in Figure 5.2, have runtimes that are orders of magnitude less than any of the above mentioned components, so these are not considered in this study.

To evaluate the time performance of IDC, the study measured the runtime to compute the total 3-way coverage for different VAE configurations, datasets, and OOD Filters. The study uses β-TCVAE and Two-Stage VAE configurations from Table 5.2 for MNIST, Fashion-MNIST and CIFAR10 datasets. Table 5.3 provides a breakdown of the costs of each component of IDC using the LR OOD Filter, in seconds, for test sets of size 10000. The runtimes of all the components are the average values across 10 repetitions of the experiment.

The LR based OOD Filter is the dominant contributor to IDC runtime across all datasets and VAEs as it requires training the encoder of its VAE. It takes more than 99% of the total runtime of the IDC instantiation across our experiments. The β-TCVAE encoders, MNI-B1, FMN-B1, and CIF-B1 are substantially faster than the Two-Stage VAE encoders MNI-T, FMN-T, and CIF-T, partially due to the fact that the latter requires two encoders to be evaluated, but a significant factor could be that the β-TCVAE configurations are implemented in Pytorch [97] whereas Two-Stage VAE configurations are implemented in Tensorflow [2]; the latter exhibits non-trivial startup time. The cost of computing the number of feasible partition combinations and the overhead of the Combinatorial Coverage Measurement (CCM) tool is dependent on the number of non-noise latent dimensions. In this evaluation, the number of non-noise dimensions ranged from 4 to 63 – see Table 5.2. Only for the CIF-T VAE, with 63 latent non-noise dimensions, did the cost of the CCM tool become non-negligible, while the optimizations in Algorithm 2 managed cost even in that case.

This study demonstrates that the cost of IDC is strongly dependent on the cost of filtering OOD inputs. There are several alternative OOD filtering techniques that could be used to instantiate IDC that vary in runtime and in their accuracy in detecting OOD inputs [34, 5]. The study compares LR with two OOD techniques, Input-Complexity (IC) based OOD detection [113], and ODIN [79] by generating 2000 CIFAR10 test inputs using DeepTest and DeepXplore in order to produce tests that are both in and out of the distribution. Table 5.4 shows that LR and ODIN are more selective than IC filtering, about 8% and 11% more tests respectively. The data also show that IC is about 6

Dataset	VAE	OOD Filter	Encode	Feasible Partition Combinations Algorithm	Combinatorial Coverage Measurement Tool
MNIST	MNI-B1	8216.39	1.64	0.90	2.52
	MNI-T		21.33	0.97	2.67
Fashion	FMN-B1	8421.52	1.64	0.85	2.54
	FMN-T		22.54	1.03	3.16
CIFAR10	CIF-B1	10968.44	1.87	0.97	3.11
	CIF-T		28.46	1.53	30.01

Table 5.3: Average runtime in seconds across 10 repetitions running IDC using different datasets and VAEs with an LR OOD filter and using total 3-way combination coverage for 10000 test inputs.

OOD Method	#In-distribution Inputs	Avg. Time per Input
Likelihood-Regret [143]	1686	1.17
Input Complexity [113]	1845	0.19
ODIN [79]	1628	0.09

Table 5.4: The number of in-distribution inputs and the average time taken in seconds by the OOD method for classifying a single test input are shown in the table. The dataset consists of 2000 test inputs generated for the CIFAR10 dataset.

times faster than LR, whereas ODIN is approximately 13 times faster than LR. Instantiating IDC with ODIN for CIFAR10's test dataset using the CIF-T VAE reduces the total runtime by more than an order of magnitude from 11047 to 893 seconds.

The paper reporting on LR OOD [143] shows that across a broader range of datasets it is generally more accurate than ODIN – albeit at an increased cost. For a given application of IDC, it is advisable to evaluate the accuracy of OOD alternatives on the target dataset. This will allow one to control costs while achieving good accuracy in rejecting OOD inputs.

Result for RQ3: Aside from the OOD Filter step the cost of IDC is negligible. The cost of OOD Filtering dominates the cost of IDC, but it can be reduced substantially by choosing algorithms that are well-suited to the target dataset.

RQ4: How does IDC vary with changes to the underlying VAE?

The VAE plays a critical role in IDC as it defines the latent representation over which coverage is computed. As described in Section 5.2 different VAEs can be used in IDC and they may vary in the number of non-noise dimensions and the degree to which they match the latent prior. These variations can influence IDC and coverage.

Dataset	VAE	MMD Non-noise+Noise	MMD Non-noise
MNIST	MNI-B1	0.0804	0.0804
	MNI-B2	0.1430	0.0927
	MNI-F	0.0460	0.0460
	MNI-T	0.0784	0.0392
Fashion	FMN-B1	0.2370	0.0728
	FMN-B2	0.4720	0.0672
	FMN-F	0.0590	0.0590
	FMN-T	0.0583	0.0213
CIFAR10	CIF-B1	0.3530	0.1261
	CIF-B2	0.4521	0.1364
	CIF-F	0.1200	0.0870
	CIF-T	0.0162	0.0158

Table 5.5: MMD values calculated using all the latent dimensions of VAEs and only the non-noise dimensions of VAEs.

A study is designed with treatments varying different parameters of the VAE to measure the variation in total 3-way coverage computed by IDC with each VAE for randomly chosen sequences of test suites. The study uses the VAEs in Table 5.2 for MNIST, Fashion-MNIST, and CIFAR10 datasets which vary with total latent dimension, encoder architectures, and training objective. The study uses a partial factorial design for each dataset that includes (1) two β-TCVAE configurations with different latent dimension while keeping architecture and objective same, (2) β-TCVAE and FactorVAE that have same encoder architecture and latent dimension but different training objective functions, and (3) Two-stage VAEs that have different encoder architectures, latent dimension, and training objective functions. The study uses the latent dimension of the best performing VAE for each dataset as a starting point and reduced that dimension for the other treatments. The number of non-noise dimensions and the extent to which the learned latent distribution matches the standard Normal prior are determined at VAE training time by a number of different factors and while they cannot be directly manipulated, additional data is reported to provide greater context for interpreting the coverage growth data.

Ten test suites, $TS_i : i \in [1, 10]$ are created by randomly sampling from the test datasets. TS_1 is created first by randomly selecting 1000 inputs from the test dataset; $TS_i : i \in [2, 10]$, is created by augmenting TS_{i-1} with 1000 random samples from the testset. The study measures

the cumulative total 3-way coverage across this sequence of tests. Figure 5.11 plots the mean total 3-way coverage computed using IDC with increasing test size for each VAE across 20 repetitions of the experiment. The x-axis increases the size of the test suite in steps of 1000 and the y-axis shows the corresponding increase in the total 3-way coverage. While all of the plots show a similar shape, i.e., increasing monotonically with the rate of increase slowing with test suite size, there are notable differences.

To study the relationship between the 3-way IDC coverage and how well the VAE models learn a latent representation that matches the standard Normal prior, the Maximum Mean Discrepancy (MMD) [45] values of the VAE models are computed and presented in Table 5.5. The MMD values are computed for the latent space using all the latent dimensions and the non-noise dimensions of the VAEs. A comparison between MMD values of latent space with and without noise dimensions indicates that the distribution of the latent space is closer to the standard Normal distribution when the noise dimensions are filtered out. This is accounted for by IDC as it uses only non-noise dimensions for measuring test coverage.

To better characterize the variation in coverage across our experiment, the study calculates the area under the curve (AUC) of the total 3-way coverage versus the 10 test suites for all 20 repetitions of the experiment across the three datasets and four VAEs. Figure 5.12 shows the median, first and third quartiles of the AUC values in the form of the box and whisker plots. From the plots, it is evident that the Two-stage VAE configurations MNI-T, FMN-T, and CIF-T show the highest percentage coverage values across all of the experiments.

The study results imply that the best-performing VAEs have the largest number of non-noise dimensions and the lowest non-noise MMD – see Table 5.5. The low MMD values mean that the assumed standard Normal prior is a closer match to the learned latent space. These results are promising as it appears that IDC does not suffer when a larger latent dimension is used, as long as MMD is low. The total 3-way coverage of β-TCVAE configurations {MNI,FMN,CIF}-B1 and {MNI,FMN,CIF}-B2 are among the lowest across all of the experiments. These models have the highest non-noise MMD and the smallest non-noise latent dimension. The FactorVAE configurations MNI-F, FMN-F, and CIF-F fall in between these extremes both in terms of latent dimension and MMD and in terms of the measured IDC metric. The p-values are measured using the Mann-Whitney U Test, which is a non-parametric test, to demonstrate the confidence in assessing

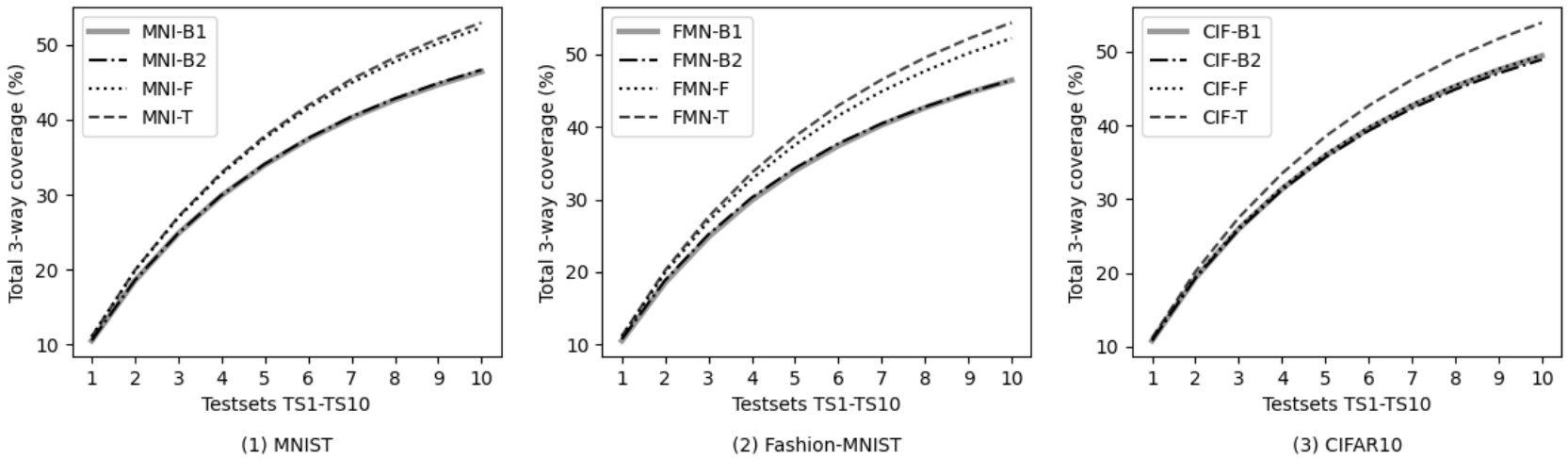


Figure 5.11: Total 3-way coverage (%) computed using IDC for different VAE configurations using randomly sampled test suites from MNIST, Fashion-MNIST, and CIFAR10 test datasets averaged across 20 repetitions of the experiment.

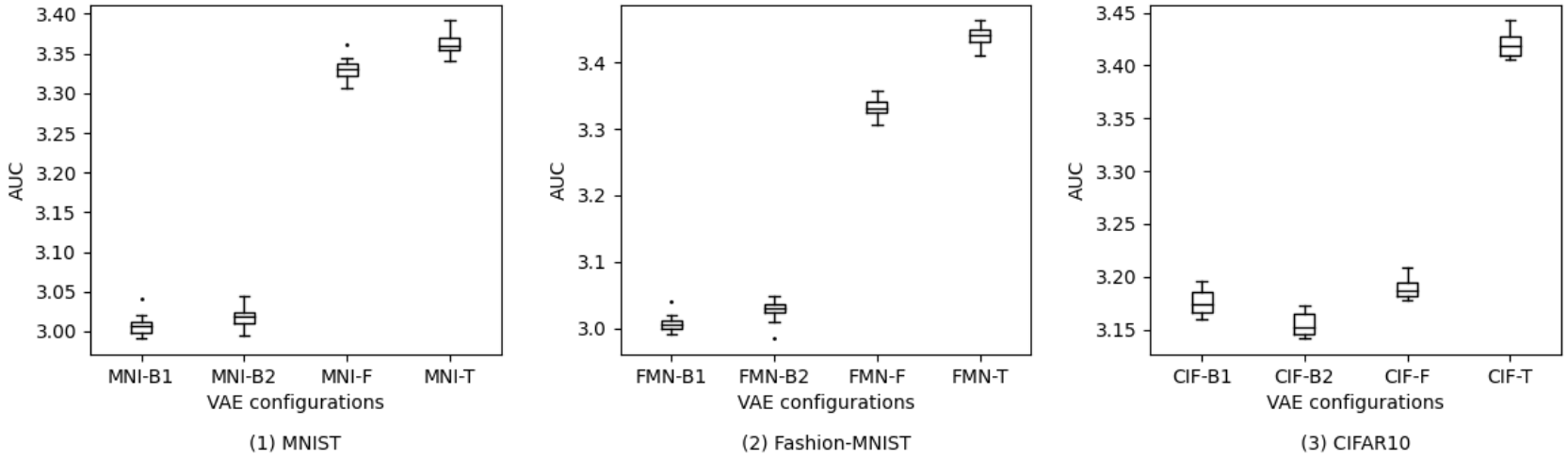


Figure 5.12: AUC of curves in Fig. 5.11 computed for different VAE configurations for MNIST, Fashion-MNIST, and CIFAR10 test datasets.

Test	MNIST	Fashion	CIFAR10
FactorVAE > β-TCVAE1	7.2e-12	7.2e-12	2.4e-4
FactorVAE > β-TCVAE2	7.2e-12	7.2e-12	7.2e-12
Two-stageVAE > FactorVAE	3.7e-09	7.2e-12	7.2e-12

Table 5.6: p-value calculated for different tests using Mann–Whitney U test for the AUC of the curves shown in the Figure 5.12. In the table, β-TCVAE1 represents {MNI,FMN,CIF}-B1, β-TCVAE2 represents {MNI,FMN,CIF}-B2, FactorVAE represents {MNI,FMN,CIF}-F and Two-stage VAE represents {MNI,FMN,CIF}-T configurations.

whether the total 3-way coverage measured by the Two-stage VAE configurations is greater than that of the FactorVAE configurations, and the FactorVAE configurations have more total 3-way coverage than β-TCVAE configurations. The results are presented in Table 5.6, and they show a strong confidence in the research findings.

The study data clearly show that IDC is sensitive to variations in VAE configuration. They also suggest that parameters of the trained VAEs, such as the number of non-noise dimensions and MMD can be an effective means of selecting the best VAE to use in IDC for a given dataset.

Result for RQ4: While IDC is effective in measuring coverage across a range of VAE configuration settings, using models with lower MMD yields more sensitive IDC metric when compared to models with higher MMD values.

RQ5: How does IDC compare with existing coverage metrics in measuring coverage of in-distribution test inputs generated using diverse test generation techniques?

To answer RQ5, an experiment is used to explore the extent to which test coverage computed by IDC and existing white-box test coverage metrics are sensitive to different types of test inputs. This experiment compares the percentage of coverage using the total 3-way coverage metric for different test generation techniques. The experiment uses test inputs that are generated by a collection of different state-of-the-art DNN test generation techniques, namely DeepXplore [99], DeepTest [129], and DLFuzz [47]. These three techniques were selected because they use very different algorithmic approaches to generate tests and, therefore, are likely to produce different kinds of test inputs. DeepXplore generates tests using gradient ascent and three input transformations: brightness, occlusion, and blackout. DeepTest generates tests by applying seven image transforma-

tions on the inputs: translation, rotation, scale, shear, brightness, contrast, and blur. DeepXplore uses a cross-referencing test oracle, whereas DeepTest uses a metamorphic test oracle. DLFuzz uses fuzzing for generating test inputs, and it uses a constraint-based test oracle. As has been demonstrated in prior work, these techniques generate test inputs that are both in-distribution and out-of-distribution [34]. OOD Filter is used to filter out the OOD test inputs generated by the three test generation techniques.

A total of nine test suites were constructed for each of the MNIST, Fashion-MNIST, and CIFAR10 datasets by starting with the original test dataset FTS, and incrementally adding 1000 test inputs generated using each of the test generation techniques. For example, the first test suite, named DX, is constructed by adding the test inputs generated by DeepXplore to the FTS. The remaining test sets are built cumulatively by adding tests generated by different test generators to DX in this order: DeepTest translation (DT-Trans), DeepTest scale (DT-Scale), DeepTest shear (DT-Shear), DeepTest rotation (DT-rotation), DeepTest contrast (DT-Contrast), DeepTest brightness (DT-Bright), DeepTest blur (DT-Blur), and DLFuzz.

Total 3-way coverage metric measured using IDC is compared with six white-box DNN test coverage metrics: neuron coverage(NC) [99], k-multisection neuron coverage(KMNC) [84], neuron boundary coverage(NBC) [84], strong neuron activation coverage(SNAC) [84], likelihood-based surprise coverage(LSC) [65], and distance based surprise coverage(DSC) [65]. A neuron activation threshold of 0.25 is used for measuring NC and k value of 100 is used for calculating KMNC. The number of buckets for measuring LSC and DSC is 1000, the same value used in the prior works. As RQ4 demonstrated that test adequacy measured using IDC is sensitive to the VAE configuration used, IDC is configured using the MNI-B1, FMN-B1, and CIF-B1 VAEs, from Table 5.2, since these lead to the least sensitive instantiations of IDC as shown in Figure 5.11; using any other VAE would yield greater increases in test coverage for this study.

Figures 5.13, 5.14 and 5.15 show the study results for MNIST, Fashion-MNIST and CIFAR10 datasets respectively. A plot of each figure corresponds to different comparison groups of coverage measures. The IDC data are repeated across the plots in each figure to promote comparison; they are rescaled for each plot. The x-axis of each plot lists the name of the cumulative test set and the y-axis shows the increase of each metric as a percentage. Data points are interpreted as the new coverage achieved by adding 1000 tests generated by a new algorithmic technique; each x-tick adds

a new test generation algorithm. DeepTest brightness results are omitted in the plots of MNIST and Fashion-MNIST in the Figure because the vast majority of these test inputs are classified as OOD; for CIFAR10 the brightness transformation is included because the resultant inputs lie on the data distribution.

Intuitively an effective test coverage measure should have a non-trivial increase in test coverage when new test inputs are added to a test suite. All the figures use a dashed line to indicate a 1% increase in test coverage as a reference point in comparing the sensitivity of different test coverage metrics. Recall that the experiment begins at the coverage level achieved, for each metric, after running the FTS.

The leftmost plots of all three figure show that the total 3-way coverage computed by IDC is more sensitive to new tests than NC, as it always leads to a greater increase and most of those increases are non-trivial, i.e., greater than 1%.

The middle plots of the figures show a more nuanced situation where, for some types of tests, certain criteria yield greater percentage increases than IDC total 3-way coverage. For example, Fashion-MNIST DT-Contrast yields tests that produce larger changes in SNAC and NBC than the total 3-way coverage. In contrast, for many of the test generation techniques, the white box coverage techniques produce very small changes. For 7 of the 9 test generation techniques on CIFAR10, the increase in coverage is well below 1%. The total 3-way coverage computed by IDC exhibits a broader degree of sensitivity to test inputs, only dropping below the 1% threshold in 5 of the 25 cases.

The rightmost plots of the figures again show a nuanced picture, but there is significant variation with the data set. For MNIST, both LSC and DSC exhibit higher sensitivity than IDC total 3-way coverage for 3 test generation techniques, whereas for Fashion MNIST, higher sensitivity is observed for a single test generation technique, DLFuzz. The CIFAR10 data distinguishes DSC as more sensitive than LSC across the board and more sensitive than the total 3-way coverage for 7 of 9 test generation techniques. The statistical significance at 0.05 level is measured using the Wilcoxon signed-rank test for comparing the sensitivity of IDC with other metrics across all three datasets. The results indicate that IDC has higher sensitivity when compared to NC, KMNC, NBC, and LSC with p-values 2.98e-8, 1.49e-7, 0.018, and 0.005 respectively whereas the test failed for SNAC and DSC with p-values 0.0704 and 0.552 respectively. While IDC cannot be said to be

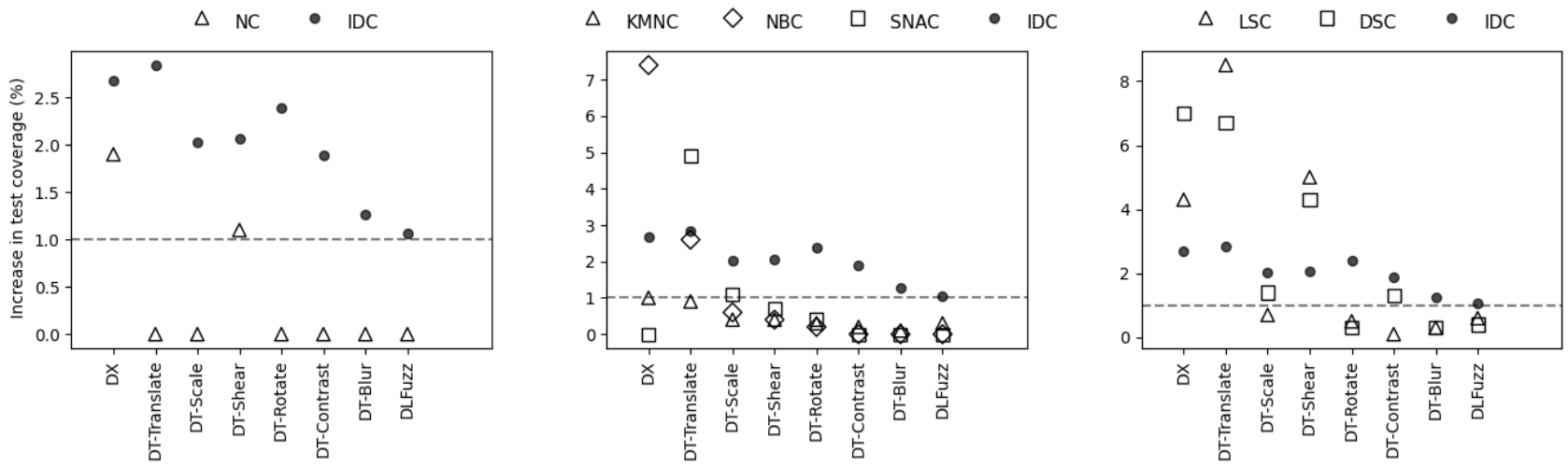


Figure 5.13: Increase in test coverage obtained by augmenting test dataset of MNIST with test inputs generated by DeepXplore (DX), DeepTest (DT), and DLFuzz. All values are in percentages.

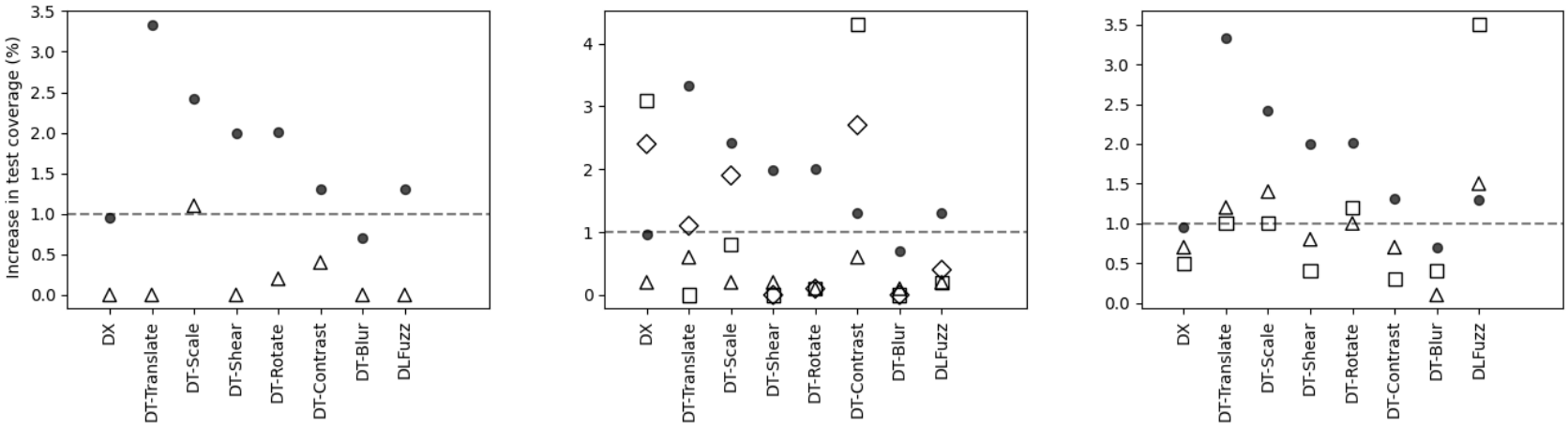


Figure 5.14: Increase in test coverage obtained by augmenting test dataset of Fashion-MNIST with test inputs generated by DeepXplore (DX), DeepTest (DT), and DLFuzz. All values are in percentages.

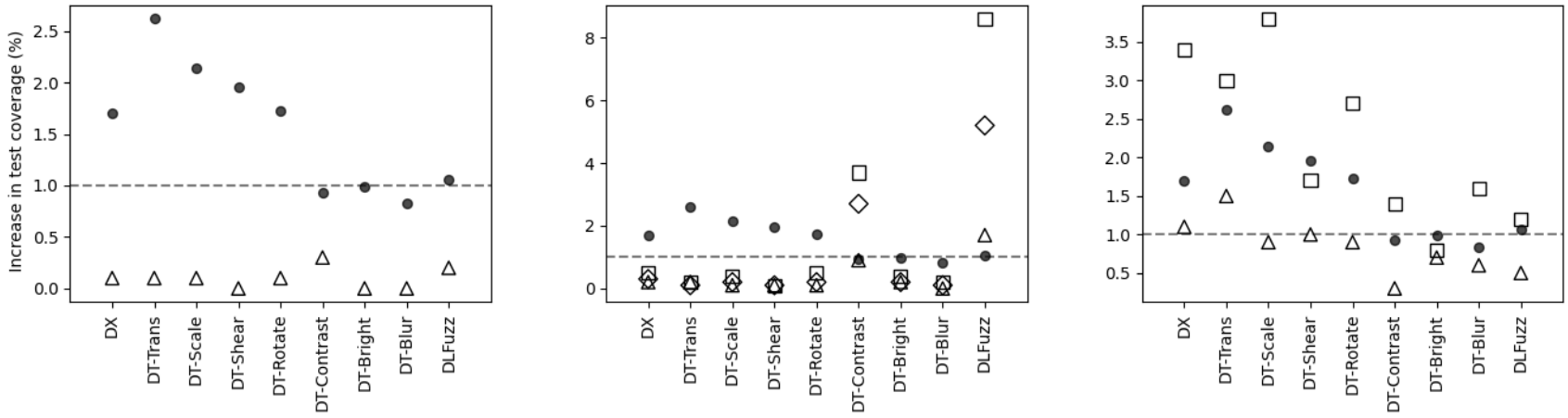


Figure 5.15: Increase in test coverage obtained by augmenting test dataset of CIFAR10 with test inputs generated by DeepXplore (DX), DeepTest (DT), and DLFuzz. All values are in percentages.

Dataset	Metric	LF1	LF2	FTS
MNIST	Mutation Score	100	93.33	100
	Total 3-way cov.	36.34	32.81	59.34
Fashion	Mutation Score	91.93	80.64	100
	Total 3-way cov.	25.17	36.67	56.95

Table 5.7: Total 3-way coverage computed by IDC vs Mutation Score computed by DeepCrime for different test suites. All values are in percentages.

more sensitive than SNAC and DSC, the results show that the total 3-way coverage falls below the 1% threshold in 5 of 25 cases, whereas SNAC and DSC fall below the threshold in 19 and 8 cases, respectively. These findings motivate further study in order to better understand how white-box and black-box DNN coverage techniques can complement each other.

Result for RQ5: IDC is sensitive to the test inputs generated using nine test generation techniques across three datasets. The test coverage metric computed by IDC is more sensitive to novel test inputs when compared to NC, KMNC, NBC, and LSC, whereas in comparison to SNAC and DSC it exceeds the 1% threshold more frequently.

RQ6: How does IDC compare with DNN mutation testing with respect to feature diversity in a test set?

To answer this question, the study uses an experiment to compare the DNN Mutation Score with the total 3-way coverage metric in regards to their sensitivity in detecting feature diversity. In testing traditional software, mutation testing has proven effective as a metric for test adequacy. Recent work has adapted mutation testing to DNNs to determine if yielding a similarly effective adequacy measure is applicable to DNN testing. DNN mutation testing techniques assess test suites by hypothesizing a fault model and measuring the extent to which faults related to that model when added to the DNN development process yield DNNs that can be detected by the test suite. Conceivably a fault model could be related to the features present in the data distribution, but none of the approaches proposed to date [58, 85, 56, 115] defined mutation operators for testing the features diversity used in testing. Consequently, there is no reason to expect that the existing DNN mutation operators will be particularly sensitive to feature diversity in a test set.

This experiment compares the sensitivity of DNN Mutation Score (MS) in reflecting the feature diversity of the inputs using the mutation operators available in the literature [58]. For the study, test suites are constructed to have different degrees of feature diversity for the MNIST and Fashion-MNIST datasets. A number of strategies are possible for creating test suites with different feature diversities, such as selecting the tests by controlling the regions of the latent dimensions covered as shown in RQ2, using human expertise to create the test suites, and by creating test suites with specific class categories. The study uses two different strategies to create the test suites. In the first approach, the study performs a latent traversal of the VAE trained on the MNIST dataset for different seed images and identifies human recognizable features. Orientation is identified as a feature for the MNIST dataset, and the test dataset of MNIST is split based on the orientation of the digit images. All the images that are tilted towards left are in the first test suite **LF1**, and the rest of the images tilted towards right are in the second test suite **LF2**. Manual verification is used to ensure that all the inputs in each of the test suites, LF1, and LF2 have the same feature. The study considers the full test dataset, **FTS** as a test set with high feature diversity for each of the DNN models. Since the latent dimensions might not always learn human recognizable features which is the case with the Fashion-MNIST models used in our study, the study uses a different strategy for the Fashion-MNIST dataset. The Fashion-MNIST test dataset is split based on the type of object; **LF1** for Fashion-MNIST images of footwear and bags, while **LF2** contains all clothing images.

The study uses recently published research, DeepCrime [58], for generating mutants to test the DNNs. DeepCrime supports 24 mutation operators (MO) [58]. Mutants are generated for all of the MOs and selected the MOs that are killable, non-trivial, and non-redundant for the MNIST and Fashion-MNIST datasets. DeepCrime required an average of 2680 models to be trained for each of the five different datasets used in their studies. The CIFAR10 dataset is excluded in this study since training a CIFAR10 model thousands of times is very costly in terms of the runtime.

The sensitivity of IDC's total 3-way coverage and the MS are studied with respect to the features present in the test suites: *LF1*, *LF2*, and *FTS*. The MS and total 3-way coverage measured using IDC for all three test suites for both MNIST and Fashion-MNIST are presented in Table 5.7. As in RQ5, this study also uses the β-TCVAE configurations, MNI-B1, FMN-B1, and CIF-B1 from Table 5.2.

The average change in the MS for *LF1*, and *LF2* with respect to *FTS* is 3.3% for MNIST and 13.7% for the Fashion-MNIST dataset. The average change in the total 3-way coverage for *LF1*, and *LF2* with respect to *FTS* is 41.7% for MNIST and 45.7% for the Fashion-MNIST dataset. The results show that DNN mutation testing using the existing mutation operators is much less sensitive to feature diversity when compared to IDC.

As discussed above, this is not surprising since DNN mutation testing uses operators that arise from a fault model that does not consider diversity. These findings suggest that DNN mutation techniques might benefit from incorporating concepts from IDC to develop feature diversity-related mutation operators. For example, an operator might filter training data based on patterns detected in IDC feature vectors. The filtered data sets could be used to train models that are insensitive to specific feature combinations. Such trained models would only be killed by test suites that cover those feature combinations.

Result for RQ6: The mutation operators used in existing DNN mutation testing techniques do not target feature diversity thereby making mutation testing significantly less sensitive to feature diversity than IDC. IDC suggests strategies for mutation operators to enhance DNN mutation testing in this regard.

5.3.3 Threats to Validity

The findings of these studies depend on a number of choices that are made in instantiating IDC, applying those instantiations, and comparing the results of IDC to existing baselines. IDC is open-sourced for the replicability of this research at `https://github.com/less-lab-uva/InputDistributionCoverage`.

Efforts were made to mitigate threats to internal validity by reusing component implementations wherever possible, e.g., white-box coverage techniques, test generation techniques, VAE, OOD, and CCM components, and by cross-validating IDC results using different instantiations of IDC with those components. In addition, the learned latent representations were qualitatively spot checked to understand whether they were capturing feature variation in training datasets.

Threats to external validity were mitigated by exploring a wide range of experimental settings and baselines, e.g., data sets, test generation techniques, and white-box coverage techniques. The consideration of test generation and white-box coverage techniques is quite thorough in relation to

the published literature. However, only pixel-level transformation based test generation approaches are used in the experiments, and more studies are required to evaluate the effectiveness of IDC to test inputs generated by feature level manipulations using testing frameworks such as [155] and [48]. While this research considered three widely used data sets from the DNN testing literature, the datasets are used in the image classification domain. There is always room to improve generalizability by reproducing this research on additional datasets that are used in text based classification and regression domains. With regard to the ability of IDC to generalize this research considered 4 state-of-the-art VAE configurations per dataset finding relatively stable results across that space.

IDC metric is not directly comparable to existing white-box DNN coverage techniques and there does not exist another black-box DNN coverage technique to compare against. To mitigate questions of construct validity the study is conducted using a synthetic benchmark, dSprites, for which the feature interactions and diversity can be manipulated and ground truth is available. Admittedly there are few such benchmarks available, which raises questions of generalizability, but the dSprites study allows exact computation of the feature interaction metric providing a clear baseline for comparison with the IDC metric.

Chapter 6

CIT4DNN: Latent Space Combinatorial Test Generation

This chapter describes CIT4DNN, a test generation algorithm to systematically test DNNs with diverse and rare inputs that satisfy the feature interaction test adequacy. CIT4DNN is developed using the latent space, generator, and constrained covering array generator components of the DisTest framework. CIT4DNN like IDC uses the latent space of a generative model as the testing domain. CIT4DNN applies CIT on the partitioned latent space to generate covering arrays containing combinations of partitions, called *test descriptions*, that cover all the t-way feature combinations resulting in test diversity. To support testing with rare inputs, CIT4DNN formulates constraints on a target density of the latent space and uses a constrained CIT algorithm extended with an SMT solver to generate test descriptions that belong to the required target density. CIT4DNN leverages the generator networks of deep generative models that convert samples in the latent space into inputs in the training data domain which corresponds to mapping test descriptions into test inputs.

CIT4DNN is black-box and only depends on the training inputs, which makes it reusable across multiple DNN models trained on the same dataset. Also, the test descriptions generated by the CIT are dependent only on the dimensionality of the latent space. As a result, the test descriptions can be reused across multiple DNN models trained on different datasets and instantiations of CIT4DNN with different generative model architectures as long as the generative models have the

same latent dimensionality. This results in test description generation time being amortized across testing many different DNNs under test. Section 6.4 describes how permuting test descriptions can further increase test diversity with low overhead. Unlike IDC, CIT4DNN does not require an encoder module which makes it portable across a variety of generative models including GANs [44] that do not contain encoder modules.

The research makes the following contributions:

1. CIT4DNN – a black-box DNN test generator capable of efficiently and automatically producing diverse and rare test inputs,
2. a method for applying CIT to the latent space of a generative model that enables reusing test descriptions to generate inputs with increasing diversity,
3. first test generation method to achieve feature interaction test adequacy on the latent space,
4. a constraint-based method to generate rare inputs without costly rejection sampling, and
5. the results of an evaluation across a range of datasets, DNNs, and instantiations of CIT4DNN that demonstrates its beneficial characteristics.

6.1 Overview

The goal of CIT4DNN is to automatically and efficiently generate diverse inputs from a specific target density of the data distribution. Figure 6.1 describes the overview, parameters, and a working example of CIT4DNN in the top, middle and bottom rows of the figure respectively. CIT4DNN uses a generative model with a standard Normal prior, $\mathcal{N}(0, 1)$, with the latent space representing the features of the training data distribution. CIT4DNN leverages the properties of $\mathcal{N}(0, 1)$ to partition the latent space into equal-density partitions and formulates radial constraints to cover a specific target density of the latent space. CIT4DNN has three modules as shown in the top row of Figure 6.1, 1) the *Constrained Combinatorial Interaction Testing* (CCIT) module, 2) the *Sample Partition* module, and 3) the *generator* module.

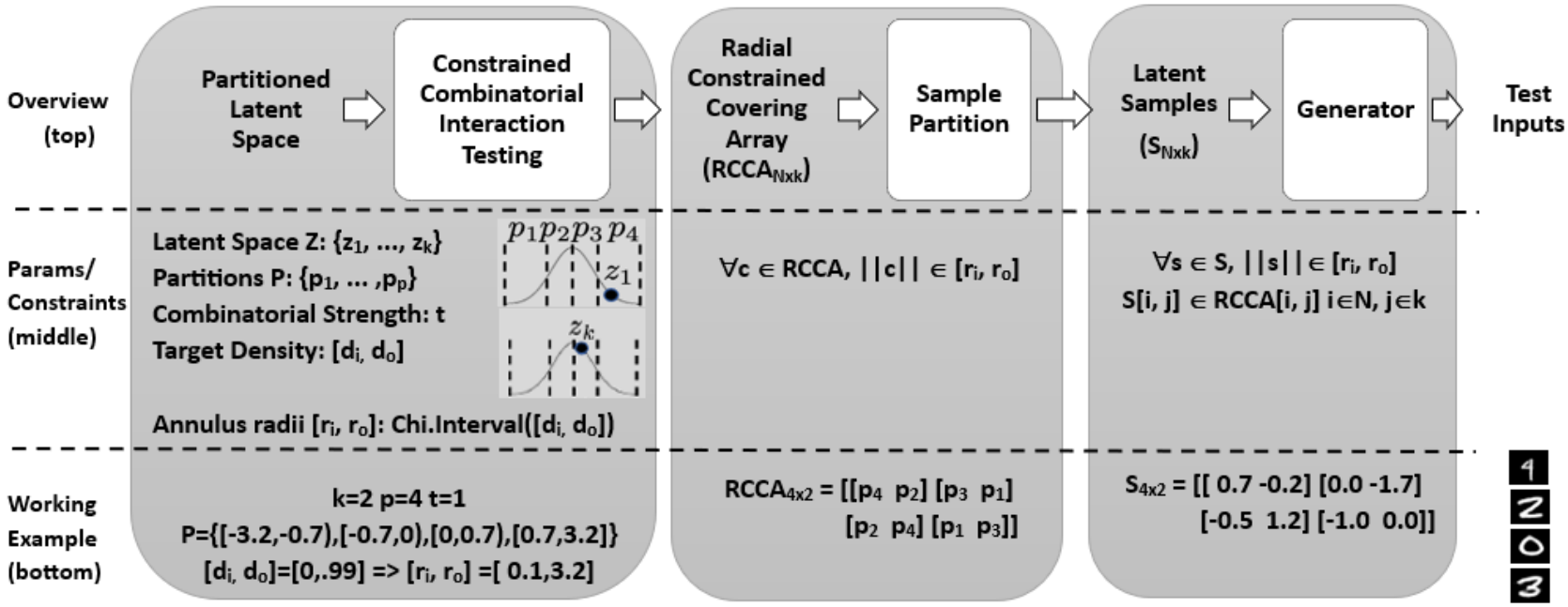


Figure 6.1: Overview of CIT4DNN.

6.2 Parameters

The middle row of Figure 6.1 shows the parameters of CIT4DNN. The input parameters are the latent space $\mathcal{Z}$ with dimensionality k, partitions along each latent dimension represented by $\mathcal{P}$, strength of CIT, t, and the target density $[d_i, d_o]$. The target density is converted into a pair of radii using the interval function of a Chi distribution and these radii are used for formulating the CIT constraints. The CCIT module uses these parameters to generate an RCCA that contains the *test descriptions*. Each row of the RCCA is a test description containing k elements, each element a partition of the corresponding latent dimension. Any t columns of the RCCA contain all the t-way partition combinations of the corresponding latent dimensions. Further, each row of the RCCA satisfies a constraint that there should at least be one latent sample per test description that belongs to the target density. The Sample partition module converts the RCCA into a set of latent samples S, such that each element of the S belongs to the partition represented by the corresponding element of the RCCA and also each sample of S belongs to the target density. The generator module converts latent samples into test inputs, and the generated test inputs are guaranteed to achieve 100% t-way combinatorial coverage of the target density in the latent space.

6.3 Working Example

This section describes a working example to demonstrate the CIT4DNN workflow. The example uses the MNIST dataset that contains images of handwritten digits [75]. The bottom row of Figure 6.1 shows a 2-dimensional latent space partitioned into 4 intervals along each of the dimensions, $k = 2$ and $|\mathcal{P}| = 4$. Constraints are formulated using $[r_i, r_o] = [0.1, 3.2]$ that represents a shell including a target density of .99 of $\mathcal{N}(0, 1)$. This is obtained using the interval function of a Chi distribution for a cumulative density of .99 and 2 degrees of freedom. CIT4DNN then applies CCIT to generate an RCCA whose rows contain test descriptions with 1-way feature combinations, i.e., each column in the RCCA contains all four partitions of $\mathcal{P}$ occurring at least once. CIT4DNN then converts the test descriptions into samples in the latent space. Finally, CIT4DNN uses a generator ($\mathcal{G}$) to convert samples in the latent space into test inputs.

6.4 Approach

6.4.1 Radial Constrained Covering Arrays

The latent space with a multivariate standard Normal prior has its probability concentrated in an annulus [6], and the probability density along the radial dimension of the annulus is described by a Chi distribution. Hence a target density can be converted to a pair of radii using the interval function of the Chi distribution, which generates bounds of an interval with equal areas around the median containing the target density [35]. These radii are used to specify the constraints for the CCIT and the resulting covering array is referred to as a *Radial Constrained Covering Array* (RCCA).

Definition 10 (Radial Constrained Covering Array)**.** *A radial constrained covering array, $RCCA(t, k, \mathcal{P}, [r_i, r_o])$, is an $N \times k$ array where: 1) Each column $1 \leq i \leq k$ contains an element of $\mathcal{P}$; 2) the rows of each $N \times t$ subarray cover all t-way feature combinations that are feasible for the given radii, $[r_i, r_o]$, at least once; and 3) all rows are feasible combinations for the radii.*

An $RCCA$ differs from Cohen et al.'s [29] CA in two ways. First, RCCAs are value-symmetric which means that each of the columns of the array can take on exactly the same set of values, $\mathcal{P}$. Second, whereas the prior work formulated satisfiability constraints over propositional constraints, CIT4DNN employs the quadratic distance constraints from IDC to model the feasibility relative to a target density – specifically, CIT4DNN uses the checkConstraint function from Alg. 2. CIT4DNN adapts the greedy AETG-SAT covering array generation algorithm from [29] with this modification to generate $RCCA$.

Continuing the example, Figure 6.2 depicts the 2-dimensional latent space – the surface comprised of light-gray grid lines with densities shown as intensities of orange and marginal distributions for each dimension shown separately, $P(\boldsymbol{z}_i)$. The gray rectangular regions on the plane show the 4-way partitioned latent dimensions within an outer target density of 0.99 (solid circle). The four black circles, e.g., s_1, depict a 1-way covering array that covers each partition of each dimension.

Restricting the target density to the range $[0.97, 0.99]$ – the shell between the dashed and solid circles – requires a larger covering array. The black square coordinates depict samples, e.g., s_2, from a 6-row covering array. Some of these samples, e.g., s_3, s_5, fall outside of the shell and are projected

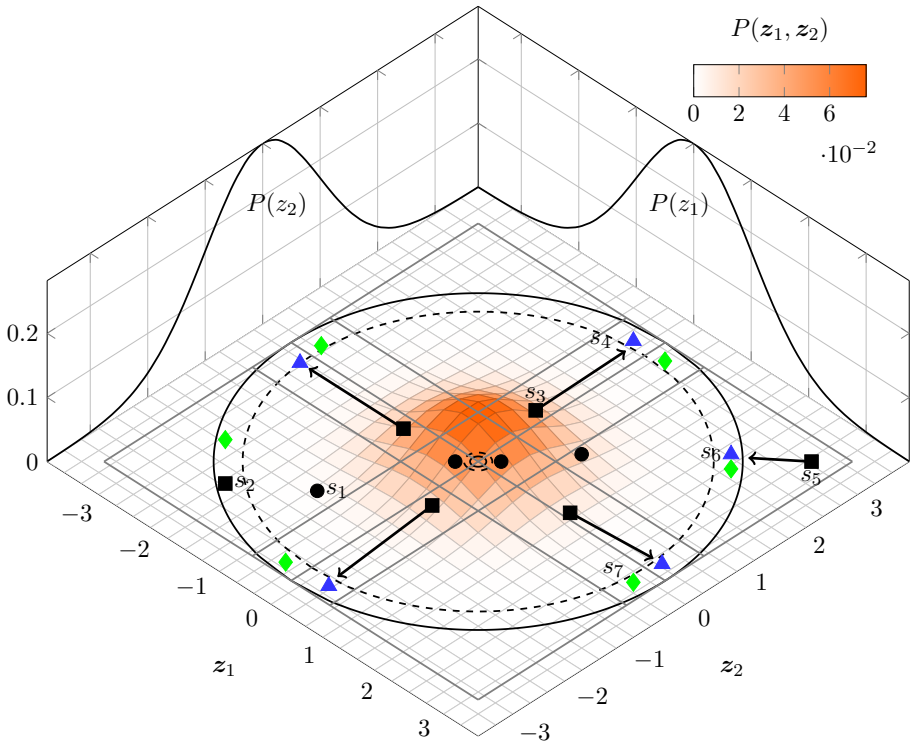


Figure 6.2: Depiction of CIT4DNN applied to a 2-dimensional latent space partitioned 4-ways ($\mathcal{P} = \{[-3.25, -0.67), [-0.67, 0), [0, 0.67), [0.67, 3.25]\}$). The dashed and dotted circles depict target densities of 0.99 and 0.97, respectively. The coordinates, s_1, and others represented by black-filled circles are samples from the 4 rows of a 1-way covering array for target densities within $[0, 0.99]$ based on $\mathcal{P}$. Decoding those coordinates yields 4 test inputs. Targeting rare inputs in the density range $[0.97, 0.99]$ – the shell bounded by the circles – increases the chance that samples, s_2, s_3, s_5, and others represented as black-filled squares, fall outside the shell, e.g., s_3, and must be projected to it, s_4. Projected samples from a covering array, shown in blue, can be further diversified by permuting the array and resampling, shown in green.

(arrows) to the blue triangle coordinates, s_4, s_6, using the method discussed in subsection 6.4.3, corresponding to the lines 23-36 of Alg. 3.

6.4.2 Reusing RCCA

CIT4DNN is designed to be applicable to any input domain for which a high-quality generator can be trained, but it is largely independent of the learned mapping between the domain and the generator's latent space, i.e., CCIT and Sample Partition modules in Figure 6.1 are independent of the generator module. Instead, CIT4DNN relies on the dimension of the latent space and the fact that the latent distribution is a close match to an isotropic standard normal prior. The benefits of this decoupling of CIT4DNN from the input domain are described below.

Whereas prior CIT work focuses mostly on 2-way combinatorial strength [131], for DNN test generation 3-way coverage yields much more diverse test suites. This presents a challenge since the cost of generating CAs grows combinatorially with t. CIT4DNN leverages the fact that a t-way RCCA for a k-dimensional latent space partitioned p ways in a target density range $[d_i, d_o]$ is independent of the VAE's encoder and decoder. Thus, an RCCA for an input domain modeled by a k-dimensional latent space can be pre-computed and reused, which saves execution time. Reuse might work against diversity, but as explained below, the isotropic latent space means that that column-wise permutation of an $RCCA$ yields an $RCCA$ that, with high probability, has distinct rows.

Proposition 1 (RCCA Permutation Closure)**.** *The set of $RCCA(t, k, \mathcal{P}, [r_i, r_o])$ are closed under column-wise permutation.*

Proof. Let $r \in RCCA(t, k, \mathcal{P}, [d_i, d_o])$ and $p : [1, k] \to [11, k]$ be a surjective function defining a column permutation of r.

For $i \in [1, k]$, let $t(i)$ be the set of t-tuples present in r involving column i. Since r is a covering array, $t(i)$ is the complete set of tuples pairing values of i with all other column-value pairs.

A tuple in $t(i)$ is of the form $(v_{j_1}, \ldots, v_i, \ldots, v_{j_{t-1}})$ and is mapped by p to $(v_{p(j_1)}, \ldots, v_{p(i)}, \ldots, v_{p(j_{t-1})})$. Every tuple in $t(i)$ is mapped in this way, and since p is surjective, it generates a complete set of tuples for column $p(i)$. Applying this mapping for all columns means that the permutation of r yields a covering array.

Figure 6.3: Diverse test inputs generated by column permutations of a row of an $RCCA$.

Radial constraints are formulated in terms of the sum of the intervals associated with the partitions in a row. Since addition commutes, the constraints hold on any permutation of a row. □

In general, there are $k!$ possible permutations, but repeated partition values in rows will reduce the number of distinct permutations. If there are c repeated values, then the number of distinct permutations is reduced by $c! - 1$, but the probability of c repeated values is $\frac{1}{p}^{c-1}$ which indicates that the vast majority of the $k!$ will be distinct.

In Figure 6.2, a second sample for the $[0.97, 0.99]$ shell would permute its columns to generate the green diamond coordinates which generates additional diverse tests. For example, the coordinate $s_4 = (-0.3, 3.0)$ when permuted yields $s_7 = (3.0, -0.3)$.

To illustrate the diversity of column permutation, a row of a 3-way covering array is randomly selected for MNIST with $k = 9$, $p = 20$, $d_i = 0.9999$, and $d_o = 0.999999$. The row's partitions are: $\langle p_{17}, p_8, p_{11}, p_5, p_{16}, p_1, p_{17}, p_2, p_{15} \rangle$. Note that marginal density is centered in partitions p_{10} and p_{11}, so this row has a number of partitions far from the center of mass which is required by the target density range. Only two columns share a common partition value meaning that there are $k! - (c! - 1) = 362879$ distinct column permutations. Figure 6.3 shows the images generated from 8 random permutations of this row and shows the diversity possible from column permutation.

6.4.3 CIT4DNN Algorithm

Alg. 3 defines CIT4DNN as a pair of functions: CIT4DNN and SAMPLE-PARTITION. The entry point is the function CIT4DNN which first extracts the latent dimension and computes their partition – lines 2-3. Lines 4-11 employ $RCCA$ to compute the covering array handling two cases. The simpler case is where d_i is zero and a single covering array is constructed – line 10 – based on the radii calculated for d_o – line 4. The more complex case – lines 6-8 – deals with the case where an inner target density is defined. Here a pair of covering arrays are constructed – lines 7-8 – where one uses the inner radii for d_i and d_o and the other uses the outer radii to define the shells used to compute $RCCA$. CIT4DNN requires that $d_i < d_o$ which means that their corresponding shells

– whose radii are computed on lines 4 and 6 – are concentric, requiring that the roles of the radii be transposed when computing the inner covering array – line 7. Lines 12-18 generate n samples from each row of the covering array CA. To yield more diverse test samples, rather than drawing multiple samples from the same covering array, n column-wise permutations of the covering array are generated – line 14 – and draw a single sample from each row of each permutation – line 15. Since rows of the covering array(s) may be associated with different radii, the radii along with each row are recorded to generate latent samples – line 15. The set of generated latent samples, S, is decoded to generate a set of test inputs – line 16.

A call to $CIT4DNN(\mathcal{G}, 0.97, 0.99, 1, 4, 1)$, with a 2-dimensional latent space, is depicted in Figure 6.2. The 4-way, equal-density partition is shown as $\mathcal{P}$ along the back z_2 axis. The pairs of interval partitions subdivide the plane into 16 rectangles. To simplify the example, consider just the case of the outer radii computed on lines 4 and 6 which are 3.255 and 2.898, respectively. The call to $RCCA$ on line 8 results in:

$$\{(p_1, p_1), (p_2, p_4), (p_3, p_1), (p_1, p_2), (p_4, p_3), (p_4, p_4)\}$$

which covers each partition in each dimension with 6 rows. The rows are required because the four inner rectangles, e.g., (p_2, p_2), do not intersect the target outer density shell and are thus infeasible.

The SAMPLE-PARTITION function samples coordinates, s, using the density associated with the partition interval, $[l_i, u_i]$, for each dimension, i – line 22. When inner and outer densities target a rare portion of the input distribution a sample within the partition intervals may not lie between the target radii – line 23. This can be observed in Figure 6.2 where there only a small arc of the outer shell intersects with any of the rectangular partition combinations. Lines 24-35 compute a sample that satisfies the partition and radial constraints using SMT. CIT4DNN builds on the squared-distance constraint formulation used in the CHECKCONSTRAINT function from Alg. 2, which works because distance constraints are insensitive to the orthant[1] within which a partition lies. CUT4DNN records a sample's orthant, q, as a vector that holds the polarity of each sample coordinate – line 24. Lines 26-33 compute squared-distance constraints and then solve them for a model in line 34.

[1] An orthant is the high-dimensional analog of a quadrant.

The use of radial constraints in $RCCA$ generation guarantees these constraints are solvable. The model generated holds the squared coordinates and lies in the positive orthant. Line 35 recovers the coordinates and maps them to the recorded orthant and updates the sample s to be returned.

Sampling from row (p_2, p_4) results in coordinate s_6 in Figure 6.2. The radius of this coordinate falls short of 2.898, the dotted circle. The orthant is recorded, $q = \langle -1, 1 \rangle$, and the following squared radial constraint is formulated:

$$(0 < vs[1] \leq (-0.674)^2) \wedge ((0.674)^2 \leq vs[2] \leq (3.255)^2) \wedge$$
$$((2.898)^2 \leq vs[1] + vs[2] \leq (3.255)^2)$$

where $vs[1] = (z_1)^2$ and $vs[2] = (z_2)^2$. Solving the constraint yields a satisfying model where $vs[1] = 0.04$ and $vs[2] = 9$. Taking the square root and recovering the orthant yields the blue coordinate $s'_6 = (-0.2, 3)$. A similar calculation projects $s_7, \ldots, s_{10}$ to the blue coordinates $s'_7, \ldots, s'_{10}$ in the target shell. Permuting the columns of the covering array to draw a second sample results in the row (p_4, p_2) which is sampled and solved to yield the red coordinate s'_{11}. Rows of a permuted covering array need not be distinct, e.g., (p_1, p_1), but as this example illustrates the vast majority of the rows, e.g., 5 of the 6, will be distinct especially as k and p increase.

Algorithm 3 Latent Space CIT Sampling for DNN Test Generation

Input:
G ← Generator
$[d_i, d_o]$ ← Target density
t ← strength of CIT
p ← No. of partitions
n ← No. of samples per Covering Array (CA) row
Output: Set of test inputs

function (G, d_i, d_o, t, p, n)
  $k \leftarrow dim(G)$
  $\mathcal{P} \leftarrow$ PARTITION-EQ-DENSITY(p, d_o)
  $r_i, r_o \leftarrow Chi.interval(d_o, k)$
  if $d_i > 0$ **then** ▷ Construct two shells
    $r'_i, r'_o \leftarrow Chi.interval(d_i, k)$
    $CA \leftarrow \{(c, r_i, r'_i) : c \in RCCA(t, k, \mathcal{P}, [r_i, r'_i])\}$
    $CA \leftarrow CA \cup \{(c, r'_o, r_o) : c \in RCCA(t, k, \mathcal{P}, [r'_o, r_o])\}$
  else ▷ Construct one shell
    $CA \leftarrow \{(c, r_i, r_o) : c \in RCCA(t, k, \mathcal{P}, [r_i, r_o])\}$
  end if
  $S \leftarrow \emptyset$
  for $i \in [1, n]$ **do**
    $CA \leftarrow$ PERMUTE-COLUMNS(CA)
    for $(c, r_i, r_o) \in CA$ **do**
      $S \leftarrow S \cup$ SAMPLE-PARTITION(c, r_i, r_o)
    end for
  end for
  return $\{G(s) : s \in S\}$ ▷ Decode latent samples
end function
function SAMPLE-PARTITION(c, r_i, r_o)
  $s \leftarrow \langle s_1, \ldots, s_{|c|} \rangle : s_i \sim [l_i, u_i] \wedge (l_i, u_i) = c_i$
  if $\|s\| < r_i \vee \|s\| > r_o$ **then**
    $q \leftarrow \langle q_1, \ldots, q_{|c|} \rangle : q_i = ite(s_1 < 0, -1, 1)$
    $vs \leftarrow$ GET-FRESH$(|c|)$
    $\psi \leftarrow true$
    for $i \in [1, |c|]$ **do**
      $(l, u) \leftarrow c_i$
      $\psi \leftarrow \psi \wedge vs[i] \geq ite(u > 0, l^2, u^2)$ ▷ Lower bound
      $\psi \leftarrow \psi \wedge vs[i] < ite(u > 0, u^2, l^2)$ ▷ Upper bound
    end for
    $\psi \leftarrow \psi \wedge (\sum_{i \in [1,|c|]} vs[i]) \geq (r_i)^2$
    $\psi \leftarrow \psi \wedge (\sum_{i \in [1,|c|]} vs[i]) \leq (r_o)^2$
    $m \leftarrow$ GET-MODEL(ψ)
    $s \leftarrow \langle \text{SQRT}(m_1) \cdot q_1, \ldots, \text{SQRT}(m_{|c|}) \cdot q_{|c|} \rangle$
  end if
  return s
end function

6.5 Evaluation

Experiments are designed to demonstrate the effectiveness of CIT4DNN in generating realistic, feature-diverse, and rare inputs for testing DNNs by exploring a series of research questions:

RQ1: How realistic are tests generated by CIT4DNN?

RQ2: How effective is CIT4DNN in generating feature-diverse tests?

RQ3: How does fault density vary with the latent distribution?

RQ4: How cost-effective is CIT4DNN in targeting normal and rare inputs?

6.5.1 Experimental Setup

Three Classification datasets, MNIST [75], FashionMNIST [141], SVHN [94], and two regression datasets, TaxiNet [60] and Udacity [58], are used in the studies. MNIST and SVHN are selected as they are used in the experimental studies of the baselines of our work, DeepHyperion-CS [156] and SINVAD [61]. FashionMNIST, TaxiNet, and Udacity are considered as they represent domains different from that of MNIST and SVHN. For each of the datasets, VAEs are trained for instantiating CIT4DNN as shown in Table 6.2 and two DNN models as shown in Table 6.1 to study fault-revealing test inputs in RQ3 and RQ4.

Datasets and DNN Models

MNIST contains greyscale images of handwritten digits; it has 60k training inputs and 10k test inputs. LeNet-4 [76] and LeNet-5 [76] DNNs trained for this dataset are used in the studies. FashionMNIST contains 28x28 greyscale images of fashion products belonging to 10 categories. FashionMNIST networks used in the IDC research are used in this work. SVHN contains 32x32 color images of digits in natural scenes; it has 73257 training inputs and 26032 test inputs. All-CNN-A and All-CNN-B networks are trained for this dataset [122].

TaxiNet contains aircraft runway images with 16x32 resolution with cross-track position and heading angle for each. The TaxiNet dataset has 80k training and 20k test inputs. The experiments use the network from an open-sourced artifact as one of the models for the differential test oracle [138]. Since a second model for TaxiNet is not available, a custom model is developed by adding two extra fully connected layers to the first model and using ELU activation in one of the

layers. The same training hyperparameters are used for training both models. The Udacity dataset is a self-driving car dataset generated in a simulation environment as open-sourced by DeepCrime work [58]. This dataset has 9800 training inputs and 2451 test inputs where each input is a 160x320 color image. Two models, NVIDIA's Dave-2 and Epoch are trained to output steering angles for the inputs [7, 58, 36].

Test Oracles

Since CIT4DNN does not use any seed inputs, it requires a test oracle that works with unlabelled data for identifying fault-revealing test inputs. Similar to DeepXplore, this research uses a differential test oracle [99]. For classification datasets, the test oracle fails when the two DNN models trained on the same dataset predict different classes. For the regression datasets, the steering angle outputs of DNNs for Udacity and the heading angle outputs of the DNNs for TaxiNet are used for formulating the test oracle. The test oracle fails when the outputs of the two models have different signs, and the difference in their predictions is greater than 5% of the output range of the test dataset. There are other test oracles proposed in the literature study of which is left for future work [142, 124, 134, 31].

6.5.2 CIT4DNN Instantiation

CIT4DNN has five configuration parameters that yield a large experimental space. To control costs, the studies consider a range of parameter combinations selected to explore the RQs and leave a fuller consideration of the parameter space to future work.

Generator CIT4DNN uses a pre-trained Generator, and the studies use the decoder networks of variational autoencoders (VAE) as the Generator networks in the experiments. Since using higher-quality VAEs is beneficial, this research explores a combination of two recent innovations in VAE architecture and training: a two-stage VAE [32], which ensures a better match to the prior and a σ-VAE [110], which optimizes the balance between loss terms that govern reconstruction accuracy and matching the prior.

The quality of tests generated by CIT4DNN is dependent on the quality of VAE used. This research performs a study that engineers can use to choose an appropriate VAE for the application of CIT4DNN. This involves measuring the quality of generated images. Prior work used either

qualitative analysis or FID score to assess image quality [61, 17, 149]; this research extends those by incorporating the state-of-the-art coverage and density metrics for generative models [91].

The study uses three VAE architectures: a basic VAE [68], denoted K, a basic VAE trained with optimal variance estimate [110], denoted $K\sigma$, and a Two-Stage VAE [32] trained with optimal variance estimate, denoted 2σ. VAEs for the MNIST dataset are trained using the network configuration used in Burgess et al. [13] and all other VAEs are trained using InfoGAN [22] network configuration. The network layers are modified to fit the input sizes of the respective datasets. Each VAE configuration is trained for five datasets (MNIST, FashionMNIST, SVHN, TaxiNet, and Udacity), resulting in the 15 VAE configurations shown in Table 6.2. The non-noise latent dimension, k, for 2σ is reported since using non-noise latent dimensions is recommended by IDC for formulating the test coverage domain.

Target Density Across the experiments, a range of target densities are considered that include both high-density and low-density regions of the latent distribution to study both normal and rare input test generation. More specifically, the studies use the following overlapping higher-density regions: $D1 = [0, 0.99], D2 = [0.49, 0.99], D3 = [0.94, 0.99]$, and a disjoint set of low-density regions: $D4 = [0.99, 0.9999], D5 = [0.9999, 0.999999]$. The target densities are distributed across the research questions to control the experimental cost. RQ1 uses only $D1$, which includes 99% of the distribution since the aim of the study is to explore normal inputs. RQ2 studies the diversity of normal inputs using $D1$ and also the improvement in diversity obtained by adding rare inputs from the tail of the distribution, $D5$ with a density of .99e-4. RQ3 uses all of the regions as it studies the prevalence of faults across the regions with varying densities. RQ4 has two sub-studies that use both normal and rare inputs, which led to selecting D1,D5, and D1,D4 in order to include both low-density regions in the experiments respectively.

CIT Parameters and n RQ1 and RQ2 vary p and t as they study aspects of the diversity of test sets. For RQ3 and RQ4, $p = 20$ and $t = 3$ are used since these were shown to be good choices for assessing test suite coverage [35]. n=1 is used in all of the research questions except for RQ4, which explores the fault-revealing capability of the test sets generated for increasing n.

Dataset	Architecture	#Parameters	Metric	Value
MNIST	LeNet-4 [76]	69362	Accuracy	99.05%
	LeNet-5 [76]	107786	Accuracy	98.53%
Fashion	Custom [102]	1.6M	Accuracy	93.58%
	Custom [140]	3.3M	Accuracy	92.26%
SVHN	ALL-CNN-A [122]	1.2M	Accuracy	96%
	ALL-CNN-B [122]	1.3M	Accuracy	95.67%
TaxiNet	Taxi1 [138]	650	MSE	63.31
	Custom	794	MSE	37.79
Udacity	Dave-2 [7]	2.8M	MSE	0.014
	Epoch [36]	104.9M	MSE	0.016

Table 6.1: Models used in our studies with number of parameters, test accuracy or MSE (Mean Squared Error); "M" denotes millions of parameters.

6.5.3 Results and Research Questions

RQ1: How realistic are tests generated by CIT4DNN?

The inputs generated by the testing techniques should be representative of the input data distribution for the testing to be effective [34]. This research conducts a study to investigate whether the inputs generated by CIT4DNN are realistic. The study first shows that the selected models generate viable outputs with random sampling; it then shows that sampling with CIT4DNN preserves the fidelity of the underlying 2σ VAE. Realism, as defined in Chapter 2, is difficult to measure directly. The study uses qualitative comparison and FID scores as previous work [61] has and also uses a newer, 2-dimensional metric: Density & Coverage.

The study computes FID and Coverage scores relative to the full test set for each dataset to select the best VAE. While FID is sensitive to test set size, this is not an issue since the study only compares within model architectures trained for the study. FID is also highly variable across implementations - the study uses the torchmetrics FID implementation on Inception v3's 2048 layer. For Coverage the study uses the reference implementation[103] with K=5 and torchvision's pretrained vgg16 Imagenet model as the embedding. For datasets smaller than 32x32, the study upscales by repeating the undersized dimension until the image is 32x32; for single channel datasets the study repeats the greyscale channel 3 times. The results are provided in Table 6.2.

The study results indicate that VAEs with optimal variance perform consistently better than

Dataset	Size	Type	k	Coverage	FID
MNIST	10000	K		0.145	89
		$K\sigma$		**0.756**	44
		2σ	9	0.751	**43**
Fashion	10000	K		0.322	161
		$K\sigma$		0.518	105
		2σ	12	**0.574**	**92**
SVHN	26032	K		0.452	137
		$K\sigma$		0.742	**49**
		2σ	30	**0.769**	62
TaxiNet	13502	K		0.003	379
		$K\sigma$		0.807	68
		2σ	8	**0.897**	**43**
Udacity	2451	K		0.059	154
		$K\sigma$		0.183	139
		2σ	23	**0.274**	**120**

Table 6.2: Coverage and FID scores for VAEs of different types for each dataset evaluated relative to test set of given size. Best metric values are in bold. Latent dimension, k, shown for 2σ VAE.

those without on both metrics. Additionally, for each metric, 2σ performs better for 4/5 datasets. To reduce the cost of subsequent experiments, the 2σ VAE is used to instantiate CIT4DNN.

It is a common practice in the generative model literature to perform a visual study to verify the quality of the generated inputs [110, 9]. Using the same approach, the random samples generated by the VAEs are manually checked for visual similarity with their training inputs and any visual anomalies. Figure 6.5 shows random samples for visual inspection. The study also performed a qualitative comparison of randomly generated outputs from each VAE to random test samples. This analysis confirmed the quality of the 2σ; randomly generated samples from this VAE are shown in the **r** rows of Figure 6.5 and test samples are shown in the **t** rows.

The study also uses Density, Coverage, and FID to assess the impact of selecting tests using CIT4DNN as compared to randomly sampling the VAE. Density is run with the same configuration as Coverage. The study is limited by the small size of CIT4DNN's t=2 test sets (100% 2-way coverage is achieved with 528-832 tests depending on dataset), so the study tests with random samples of size 500 from each set and repeat the trails 100 times each. The results are shown in Figure 6.4.

The study results indicate that sampling with CIT4DNN, for either value of t, preserves the realism of the 2σ VAE as measured by Density, Coverage, and FID. This is confirmed with a qualitative analysis. Example CIT4DNN tests with $t = 3$ are provided for each dataset in the rows

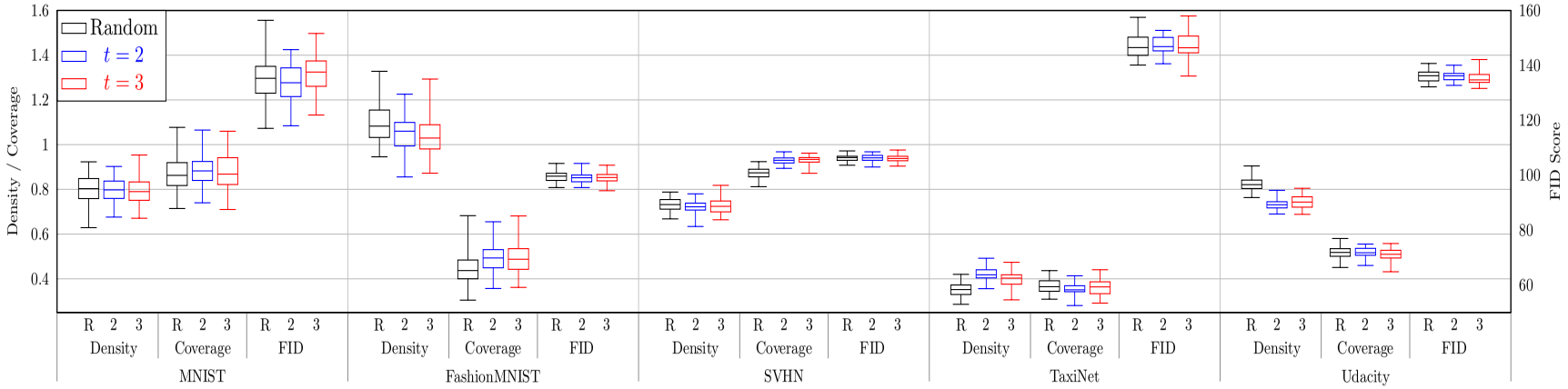


Figure 6.4: Density, Coverage and FID for test sets generated by CIT4DNN(2σ) with $t \in \{2, 3\}$ compared with random sampling.

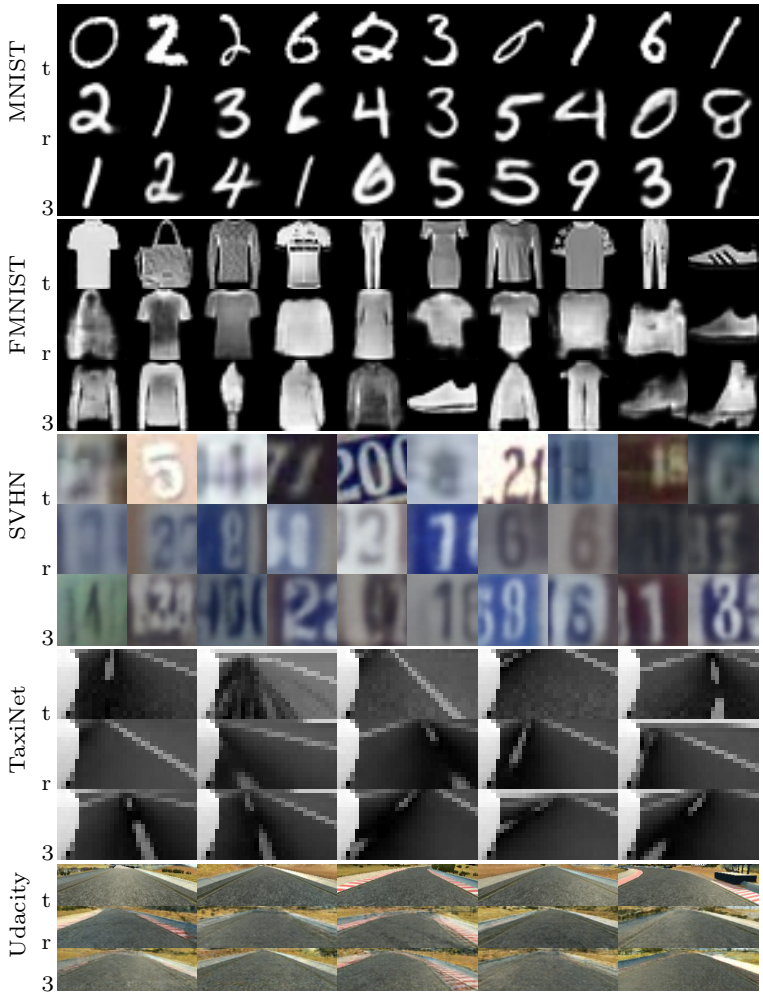


Figure 6.5: Sample images for each dataset from test set (t), 2σ (r), and CIT4DNN (3).

labeled **3** in Figure 6.5, and they are compared with randomly generated tests in the rows labeled **r**.

RQ1 Finding: Based on both FID and qualitative analysis used by prior work and the state-of-the-art Coverage and Density metrics, the study results show that CIT4DNN generates tests that are as realistic as any that can be generated by the VAE used to instantiate it.

RQ2: How effective is CIT4DNN in generating feature-diverse tests?

To provide insight into the diversity of the tests generated by CIT4DNN with respect to human interpretable features, ground truth features are required for the datasets used in the studies (which are unknown). DeepHyperion-CS [156, 155] is a recently published approach that uses human

interpretable features for generating diverse tests. Their evaluation shows that DeepHyperion-CS is superior to DeepHyperion [155], DeepJanus [106] and DLFuzz [47] with respect to feature diversity. For these reasons, DeepHyperion-CS is used as a baseline in this study.

DeepHyperion-CS uses human assessors to identify the features of the input data; the experimental studies include MNIST [75] and BeamNG [156] datasets. Of these datasets, DeepHyperion-CS transforms the inputs directly only for MNIST, so this study compares CIT4DNN on that dataset. DeepHyperion-CS generates feature maps of the generated test sets and uses the metrics, Filled Cells (FC), and Coverage Sparseness (CS), computed over the feature maps to measure the feature diversity of the test sets. These two metrics are used for comparing the feature diversity of the tests generated by CIT4DNN and DeepHyperion-CS to answer RQ2.

DeepHyperion-CS uses Luminosity (Lum), Moves (Mov), and Orientation (Or) as the features of the MNIST digits. The study ran CIT4DNN and DeepHyperion-CS for an hour each and generated feature maps for test inputs generated by the two approaches for pairs of feature combinations, $(Or, Mov), (Or, Lum), (Mov, Lum)$. CIT4DNN is run with p=20, t=3, n=1, and $[d_i, d_o] \in \{D1, D5, D1 + D5\}$ to include both normal and rare inputs in the study. The study ran DeepHyperion-CS for all 10 classes of MNIST and limited the overall runtime of the tool to one hour. The number of CIT4DNN generated test inputs and the time to generate them are shown in Table 6.4. CIT4DNN runs for 26 seconds for $D1$ and 116 seconds for $D5$; DeepHyperion-CS runs for one hour. Eventhough CIT4DNN could have been run for a full hour by using $n = 25$, the study only used $n = 1$ for these studies; therefore, the results presented underestimate the feature diversity that could have been achieved by CIT4DNN given an hour.

This experiment is repeated 10 times with the FC and CS metrics being measured for each of the feature maps. Box plots of the results are shown in Figure 6.6. The study performed a Mann-Whitney U Test and the results show that rare inputs ($D5$) have better FC and CS values when compared to normal inputs ($D1$). Additionally, a test set with both rare and normal inputs ($D1 + D5$) outperforms both DeepHyperion-CS and test sets generated for normal or rare inputs alone. These all hold with significance; p-values for all less than 0.05.

Using FC and CS metrics, the research conducts two studies to demonstrate how the partition granularity, p, and the strength of CIT, t, of CIT4DNN impact diversity. For the partition granularity study, test sets are generated for $p \in \{4, 8, 16, 20\}$ while keeping $[d_i, d_o]$, t and n fixed to

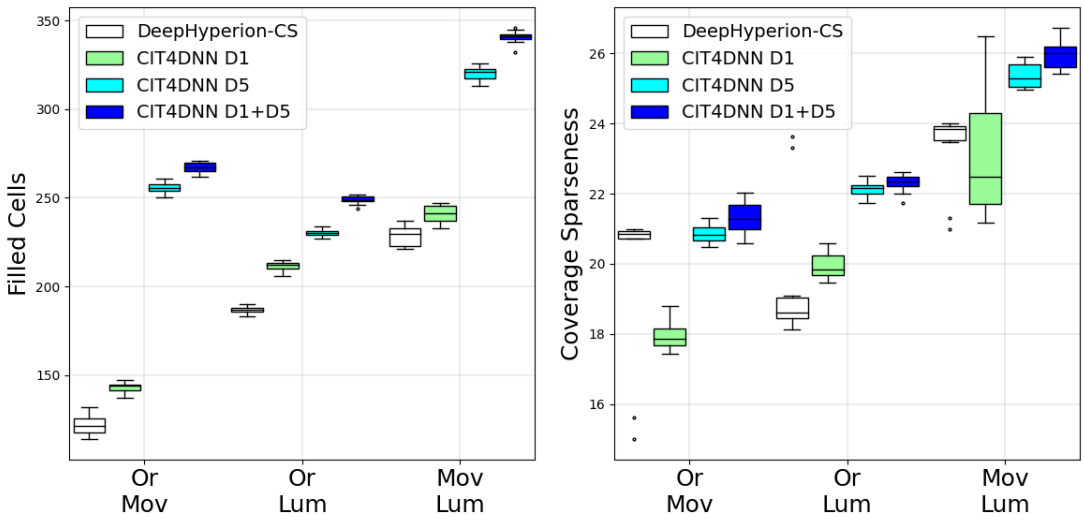


Figure 6.6: Filled Cells and Coverage Sparseness of test sets generated by DeepHyperion-CS and CIT4DNN(2σ, d_i, d_o, 3, 20, 1), where $[d_i, d_o] \in \{D1, D5, D1 + D5\}$, for the MNIST dataset.

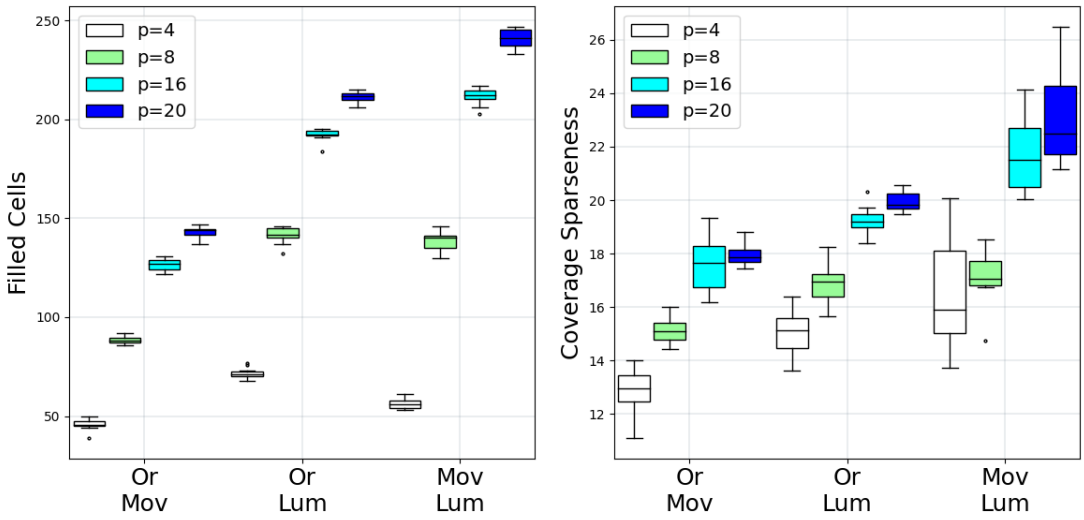


Figure 6.7: Filled Cells and Coverage Sparseness of test sets generated by CIT4DNNCIT4DNN(2σ, 0, .99, 3, p, 1), where $p \in \{4, 8, 16, 20\}$, for the MNIST dataset.

$(0, .99)$, 3 and 1 respectively. Figure 6.7 shows the FC and CS measured for the generated test sets. Mann-Whitney U Test is used to study whether $M(p = 4) < M(p = 8)$, $M(p = 8) < M(p = 16)$ and $M(p = 16) < M(p = 20)$ where $M \in \{FC, CS\}$. All three tests are performed for 3 feature combinations, resulting in 9 Mann-Whitney U Tests for each metric. The test passed for 9 out of 9 configurations for FC and 7 out of 9 configurations for CS with p-values less than 0.05. The tests failed for $CS(p = 4) < CS(p = 8)$ for (Mov, Lum) and $M(p = 16) < M(p = 20)$ for (Or, Mov) feature combinations. Similarly to the partition granularity study, to study the effect of t on diversity, test sets are generated for $t \in \{1, 2, 3\}$ while keeping $[d_i, d_o]$, p and n fixed to $(0, .99)$, 20 and 1 respectively. Mann-Whitney U Tests for $M(t = 1) < M(t = 2)$, and $M(t = 2) < M(t = 3)$ where $M \in \{FC, CS\}$ indicate that diversity increases with t for all the feature combinations tested with p-values less than 0.05.

RQ2 Finding: CIT4DNN generates diverse test inputs when compared to DeepHyperion-CS. For the configurations of p and t studied in the research question, diversity increases with both p and t, with increasing t leading to better diversity than increasing p.

RQ3: How does fault density vary with the latent distribution?

This study uses different target density regions, $D1 - D5$, to generate normal and rare inputs. These density ranges include a broad range of cumulative densities in the latent space. The study measures the number of fault-revealing test inputs in each density partition to study the distribution of faults in the latent space. MNIST, Fashion, SVHN, TaxiNet, and Udacity are used in the study. The study varies $[d_i, d_o]$ while p, t and n are set to 20, 3, and 1 respectively. Table 6.3 shows the average number of faults identified by the differential test oracles for each of the density partitions across 10 repetitions of the experiment. Due to the low standard deviation of the number of faults across all the configurations, which is at most 45, only average values are shown in the table.

Since $D3 \subseteq D2 \subseteq D1$, the faults detected in $D1$ might include faults present in $D3$. Nevertheless, the results show that across the datasets there is an increase in the number of faults detected moving from $D1$ to $D3$, except for Udacity. Generally, the test set size increases across this range, except for Udacity which could explain the lack of increased fault detection. Recall that varying the target density can change the RCCA size and the extent of that change will vary with k, which

Dataset	Metric	D1	D2	D3	D4	D5	D5/D1
MNIST	#tests	14658	14668	15421	15825	16073	1.09
	#faults	768	795	1266	1792	3145	4.09
Fashion	#tests	17777	17791	18037	18227	18366	1.03
	#faults	4341	4324	4575	4659	4953	1.14
SVHN	#tests	29702	29698	29703	29711	29717	1.00
	#faults	5537	5519	5998	6492	7320	1.32
TaxiNet	#tests	13504	13534	14660	15171	15446	1.14
	#faults	1739	1720	2254	2480	2626	1.51
Udacity	#tests	26028	26014	26025	26056	26071	1.00
	#faults	1567	1565	1560	1503	1559	0.99

Table 6.3: Number of tests (#tests) and faults (#faults) generated by CIT4DNN(2σ, d_i, d_o, 3, 20, 1), where $[d_i, d_o] \in D1 - D5$.

varies across the VAEs used for these datasets (Table 6.2). Comparing rare input to normal density region ($D5/D1$) reveals that the increase in test size alone does not explain the increase in the number of faults detected. This is especially true for datasets like MNIST and SVHN.

RQ3 Finding: The research results indicate an increase in faults detected that is out of proportion to the increase in test set size with decreasing input probability, which suggests that fault density increases as input density decreases.

RQ4: How cost-effective is CIT4DNN in targeting normal and rare inputs?

This study demonstrates the effectiveness of CIT4DNN in generating normal and rare inputs by comparing against random sampling in the latent space and SINVAD [61]. The study considers SINVAD as a baseline since it is a recently published generative-model based approach, and its implementation is available. While manifold-based test generation [17] is also relevant and open-sourced, it only generates normal inputs, and it would be unfair to use it as a baseline for rare input testing. The primary cost metric of the study is test generation time, but the number of tests generated is also reported by each technique since this can influence clients of the generated tests, e.g., when running a test set multiple times. The primary effectiveness metric used by the study is total 3-way coverage, which, as shown in RQ1 and RQ2, is capable of generating realistic and feature-diverse tests, but the second part of this RQ also reports fault detection as an effectiveness metric.

Random Baseline The random baseline generates tests by drawing n samples from the Gaussian prior of a VAE, V, as Byun et al. [17] proposed, and is equivalent to $CIT4DNN(V, 0, 1, 1, 1, n)$.

For each approach, the experiment terminates either when it achieves 100% total 3-way coverage or the runtime exceeds one hour. A server with an AMD EPYC 7742, 2.25GHz CPU with 128GB of memory is used for running the experiment. Two random sampling configurations are used in the study to ensure that the coverage measurement overhead of random sampling does not dominate the study results. The first configuration, Random-1K, incrementally adds 1000 test inputs to the test set until the algorithm meets the termination criteria, whereas the second configuration, Random-1M, adds test inputs at one million granularity. Random-1M is used only when Random-1K fails to terminate with 100% total 3-way coverage. CIT4DNN is configured for two target densities, $D1$ and $D5$, to study normal and rare inputs respectively with p=20, t=3, and n=1.

Table 6.4 reports the metrics for CIT4DNN and the random baseline for five datasets. The number of tests is divided into the number sampled and the number that is feasible relative to the target density range. For the high-density region, $D1$, most random samples are feasible, but this is not the case for the low-density region, $D5$. The hit rate of random sampling allows it to achieve 100% coverage, reported as a 1 in the table, within the timeout for $D1$, but requires between 3.3 and 7.2 times the number of tests generated by CIT4DNN. Moreover, CIT4DNN generates those tests up to 7 times faster. In the low-density space, the story is very different. Random sampling cannot hit the target density range frequently causing it to timeout and fail to achieve 100% coverage. In contrast, while CIT4DNN incurs increased cost in $D5$, those costs are comparable to the time for random on $D1$ for 4 of the datasets. CIT4DNN achieves 100% coverage by construction and the projection technique never fails to map a sampled input to the target density range.

SINVAD Since SINVAD only supports classification datasets, only MNIST, Fashion, and SVHN are used in the experiment. CIT4DNN is configured for two target densities, $D1$ and $D4$, to study normal and rare inputs respectively with p=20, t=3, and n=1.The differential testing algorithm and the DNN networks provided in the SINVAD artifact [119] are used for test generation and the test oracles in the study. The study ran each technique until either the generated tests achieve 100% total 3-way coverage or the total runtime exceeds five hours. Since SINVAD has high runtime costs, it was run on a server with an 11GB Nvidia GTX1080Ti GPU, an Intel Xeon E5-2620 2.10GHz CPU and 128GB RAM for the experiment. SINVAD timed out for all runs, whereas CIT4DNN for n=1 ran in less than 15 minutes just on a CPU across each of dataset.

Figure 6.8 reports the detected faults and coverage ratios of SINVAD relative to CIT4DNN

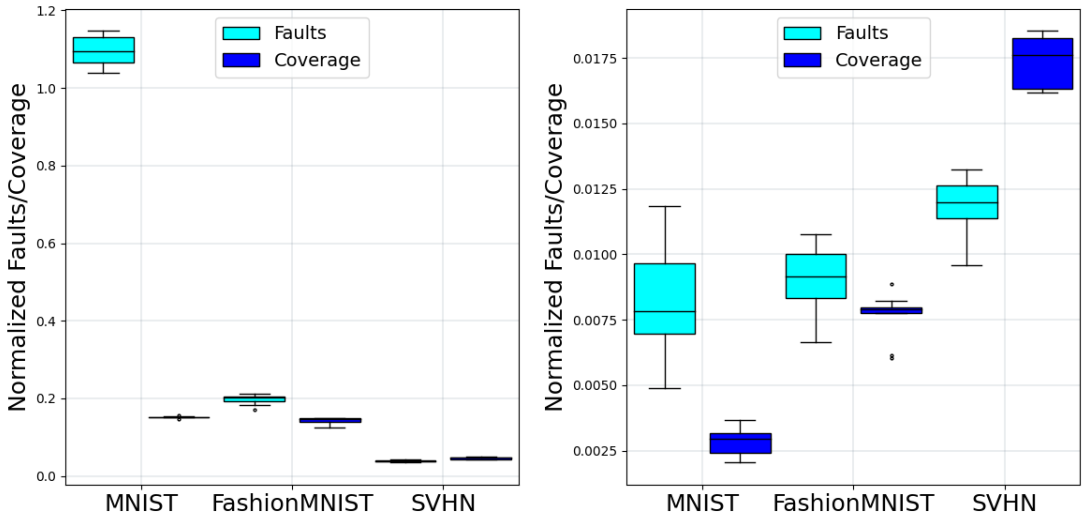


Figure 6.8: Number of fault-revealing inputs (Faults) and total 3-way coverage (Coverage) of the tests generated by SINVAD normalized with respect to the values measured for CIT4DNN across datasets for $D1$ (left) and $D4$ (right).

Dataset	Approach	D1			D5		
		Feasible/ Generated	3-way Cov.	Time (s)	Feasible/ Generated	3-way Cov.	Time (s)
MNIST	CIT4DNN	14.6k/14.6k	1	15	16.1k/16.1k	1	74
	Random	98.9k/100k	1	137	158k/1.5B	0.88	3602
Fashion	CIT4DNN	17.8k/17.8k	1	24	18.4k/18.4k	1	110
	Random	98.6k/99.6k	1	172	132k/1.3B	0.93	3602
SVHN	CIT4DNN	29.7k/29.7k	1	95	29.7k/29.7k	1	481
	Random	97.6k/98.6k	1	847	53.9k/543M	0.97	3606
TaxiNet	CIT4DNN	13.5k/13.5k	1	13	15.4k/15.4k	1	64
	Random	96.1k/97.1k	1	124	169k/1.7B	0.85	3602
Udacity	CIT4DNN	26.0k/26.0k	1	65	26.1k/26.1k	1	312
	Random	98.4k/99.4k	1	455	72.5k/733M	0.97	3604

Table 6.4: Comparison of CIT4DNN(2σ, d_i, d_o, 3, 20, 1) to Random baselines in terms of number of generated tests, number of feasible tests, 3-way coverage, and test generation time for $D1$ and $D5$.

metrics across $D1$ and $D4$. The box plots are computed with 10 repetitions of the experiment. The $D1$ results show that CIT4DNN yields a 5-fold increase in total 3-way coverage when compared to SINVAD and that CIT4DNN is better than SINVAD with respect to the number of faults for 2 out of 3 cases in the D1 study. For MNIST, SINVAD detected 842 faults and CIT4DNN only 671. The study then ran CIT4DNN with $n = 2$ and found that it detected 1339 faults, an improvement of 1.6 times relative to SINVAD. Running with $n = 2$ involves permuting the columns of the covering array, which allows tests to be generated for $D1$ in 50 seconds – 360 times faster than SINVAD. More broadly, the study results show that the cost of CIT4DNN grows linearly with n as does its fault detection effectiveness. The $D4$ results in Figure 6.8 indicate that CIT4DNN yields more than a 50-fold improvement in faults detected and total 3-way coverage across all datasets used in the study.

RQ4 Finding: CIT4DNN is cost-effective in generating normal and rare inputs and achieving 100% total 3-way coverage when compared to random sampling and SINVAD. While CIT4DNN is not a fault-directed technique it outperforms SINVAD in fault-detection while running more than 140 times faster.

6.5.4 Threats to Validity

To mitigate the threats to internal validity, this research reused an existing CIT algorithm as a starting point[2] as well as existing DNN model architectures and baselines. This research used the default hyperparameters when training the models and for running the baselines. For the components developed for CIT4DNN, the results were manually cross-checked for any anomalies in the data. For example, the AETG-SAT algorithm [29] was extended with an SMT solver for generating the covering arrays. To ensure the validity of the modified artifact, it was verified whether the generated covering arrays meet the test adequacy criterion and latent samples generated from the covering arrays satisfy the SMT constraints.

To mitigate the threats to external validity, this research designed the experimental studies to explore a range of configuration parameters of CIT4DNN and recently published baselines. The study presented in Section 6.5 guides users in selecting VAEs for generalizing CIT4DNN to other datasets. Five datasets representing different data domains are used in the studies. However, all

[2]We thank Myra Cohen for sharing the implementation in [29] with us.

the datasets use image inputs; generative models are available for other domains [101, 153], and future work could explore extending CIT4DNN to speech and text datasets.

Chapter 7

Related Work

DNN testing is an active research area with a number of testing techniques developed to address the challenges of testing these systems in terms of test coverage criteria, test generation, and test oracles [152, 57, 99, 129, 47, 144, 155, 135, 106, 61, 17]. A majority of the research is focused on white-box test coverage metrics and test generation algorithms guided by these coverage metrics. The white-box test coverage metrics are developed over the domain of the neuron activations of the DNN under test [99, 84, 65]. There is also recent research on using deep generative models for DNN testing with research objectives similar to this research [17, 149, 61]. This section describes the existing research on DNN testing techniques and their limitations.

7.1 Test Coverage Criteria

In traditional software testing, test coverage criteria are used to measure how thoroughly a software is tested. Most practical coverage criteria, e.g., [137], use the structure of the software system to make this assessment, e.g., the percentage of statements or branch outcomes covered by a test suite. Similar to structural software coverage criteria, coverage criteria for DNNs have been proposed by various research efforts.

7.1.1 Using Neuron Activation Traces

Pei et al. proposed neuron coverage (NC) as a metric for measuring the test coverage of DNNs [99]. For a given test suite, neuron coverage is measured as the ratio of the number of unique neu-

rons whose output exceeds a specified threshold value to the total number of neurons present in the DNN. Ma et al. proposed extended neuron coverage criteria including k-multisection neuron coverage (KMNC), neuron boundary coverage (NBC), and strong neuron activation coverage (SNAC) [84]. These coverage criteria can be used to determine whether a test case falls in the major functional region or corner case region of a DNN. The major and corner case regions are defined with respect to the neuron activation traces of the training dataset. DeepCover proposed Modified Condition/Decision Coverage variants for DNNs that are based on the sign and value change of a neuron's activation to capture the causal changes in the test inputs [126]. Ma et al. proposed combinatorial test coverage to measure the combinations of neuron activations and deactivations covered by a test suite in DeepCT [83].

Surprise Adequacy proposed two white-box test adequacy measures, *Likelihood-based Surprise Adequacy* (LSA) and *Distance-based Surprise Adequacy* (DSA) for measuring test coverage [65]. Surprise Adequacy measures how surprising a test input is with respect to the training dataset using kernel density estimation for Likelihood-based Surprise Adequacy, and distance between the neuron activations of the test input and the training data for Distance-based Surprise Adequacy. The test coverage is calculated by partitioning the domain of the surprise values into intervals and measuring the fraction of intervals covered by a test suite. A performance-optimized implementation of Surprise Adequacy and a Mahalanobis distance-based Surprise Adequacy metrics are proposed that are cost-effective when compared to LSA [136, 66]. While the initial evaluation of Surprise Adequacy was carried out using image classification datasets, another work extended Surprise Adequacy to test a Natural Language Processing model [67].

Since both valid and invalid inputs can have similar neuron activation patterns, these techniques cannot distinguish between valid and invalid inputs as described by the challenge *C2*. Also, these techniques do not assess how much of the input data distribution is covered systematically thereby making the testing inadequate with respect to the feature diversity used in testing which implies that they cannot address the challenge *C2*. IDC addresses both these challenges by using an *OOD Filter* to avoid using invalid inputs and measuring test adequacy over the systematic coverage of the feature interactions of the input data distribution.

DeepCT [83] similar to IDC, applies Combinatorial Interaction Testing to measure DNN test coverage. Unlike IDC, it is a white-box coverage technique and the key difference between IDC

and DeepCT is that for IDC, the combinatorial space is a partitioned latent space that represents the input distribution which makes it black-box whereas DeepCT uses the latent space of the DNN under test which makes it white-box. Also, IDC has a known mathematical structure for the latent space to compute the feasible coverage domain, and to partition the latent space in a meaningful way. DeepCT does not have a notion of a feasible coverage domain since the structure of the latent space of a DNN can be an arbitrary manifold.

Similar to this research, there have been multiple research studies demonstrating the limitations of the DNN test coverage criteria [49, 78, 112, 148]. While this current research demonstrated the limitations of existing coverage metrics in not distinguishing between valid and invalid inputs, other work also studied the correlation of the test coverage metrics with the model quality. Sekhon et al. demonstrated that neuron coverage can easily be saturated and faults exist beyond 100% neuron coverage [112]. Harel-Canada et al. conducted a study on neuron coverage and their results demonstrate that neuron coverage does not correlate with fault detection, naturalness of inputs, and output impartiality [49]. Li et al. studied coverage metrics proposed in DeepXplore [99], DeepCT [83], and DeepCover [126], and the study results demonstrated that the high coverage reported by the coverage metrics is due to adversary-oriented search but not the structural coverage, and the coverage metrics are not effective in detecting faults in natural inputs [78]. Shenao et al. studied the correlation between model quality and coverage criteria using the coverage metrics proposed in DeepXplore [99], DeepGauge [84], and Surprise Adequacy [65] [148]. Their study results show that the coverage metrics have a low correlation with the model quality in the presence of adversarial inputs, cannot distinguish between benign and adversarial samples, and adversarial test generation guided by these criteria adds larger perturbations when compared to gradient-based methods resulting in unrealistic inputs [86, 18].

While IDC addresses the challenge of the naturalness and diversity of inputs by using the valid input domain for testing, it is not designed to measure the adversarial robustness of DNNs. Hence this research cannot be compared with other coverage metrics using robustness studies. Unlike neuron coverage, IDC does not saturate easily and correlates with diversity and faults in natural inputs as shown in Section 5.3. The fault detection capability of IDC can further be improved by using finer partition granularity and higher strength of CIT which is left for future work.

Recent research proposed Neuron Path Coverage (NPC) that creates a decision graph of the

neuron activations resulting from the training data and uses it as a domain to define control-flow and data-flow based test coverage metrics [146]. NPC demonstrated a positive correlation with fault detection using natural inputs and output impartiality. Neural Coverage (NLC) is another recent work that showed a positive correlation with the diversity and fault-revealing capability of test sets [150]. The research also showed that test generation guided by NLC resulted in mutations that preserved the naturalness of the inputs. Studies are required to understand the correlation between IDC and these recently proposed DNN test coverage metrics.

7.1.2 Using Latent Space

Byun et al. were the first to propose using the latent space of generative models as a test coverage domain [15, 16]. Their work computes combinatorial coverage of the input data manifold by exploiting the latent space of a VAE [68] and, similar to this research, they partition the dimensions of the latent space to achieve this. Their technique suffers from several drawbacks - none of which are discussed in their paper. First, they do not consider the presence of noise dimensions in the latent space of the VAE which can occur with some frequency, and when they do they degrade the accuracy of coverage information. Second, they completely ignore the notion of the feasibility of the latent space used in testing that arises from the concentration of the latent distribution which means that they cannot accurately compute the total size of the coverage domain. Third, they compute coverage for out-of-distribution inputs which can yield completely invalid coverage information since the VAE encoder will compute a latent vector for any input – whether in or out of the distribution. Finally, and most importantly, they propose to measure full combinatorial coverage which is intractable. While Byun et al.'s short paper neither describes in detail nor conducts a substantial evaluation of manifold coverage, it is clear that the limitations described here render it both intractable and inaccurate as a black-box coverage criterion for DNNs.

IDC addresses all these challenges by avoiding noise dimensions in measuring test coverage, uses Alg. 2 to compute the total size of the feasible coverage domain, uses an *OOD Filter* to avoid invalid inputs, and uses t-way combinatorial coverage instead of full combinatorial coverage to reduce the test complexity.

7.2 Test Generation

DNN test generation approaches generate test inputs by exploring either the input space or the feature space [107]. Techniques such as DeepXplore, DeepTest, DLFuzz, and BET work on the input space and generate test inputs by applying pixel-level transformations on seed inputs [99, 129, 47, 135]. The diversity of the generated tests is limited by the diversity of the seed inputs used by these methods, and they can generate invalid inputs [34, 5]. As a result, these methods cannot address the challenges *C3* and *C1*. DisTest addresses the challenge *C1* by using the latent space as a testing domain, which ensures the validity of the generated inputs. CIT4DNN addresses the challenge *C3* by generating test inputs through a systematic exploration of the partitioned latent space that results in feature-diverse test inputs.

Test generation approaches that work on the feature space produce in-distribution tests. However, these approaches require a model representing the features of the input data distribution. With such a model technique, DeepHyperion [155, 156], DeepJanus [106], and methods that apply traditional CIT to ML [40, 27, 98] have been shown to be effective in covering the input feature space and revealing faults. When such a model is available these approaches can be effective, but they are costly for domain experts to construct and challenging to produce for high-dimensional input spaces like those found in image DNNs as described by the challenge *C4*. DisTest uses a generative model to automatically learn the input features thereby addressing the challenge *C4*.

In contrast to the above approaches, generative model-based approaches such as Manifold-Based Test Generation, SINVAD, and DeepTraversal use the latent space of the generative models as a feature domain for test generation, sidestepping the need for a human-defined model [17, 61, 149]. These techniques rely on the fact that a generative model learns to embed input space features in the latent space of the model. CIT4DNN falls into the generative model-based test category but differs from existing approaches since it (a) guarantees systematic latent space coverage which yields a form of systematic feature diversity coverage addressing the challenge *C3* and (b) has the ability to efficiently and systematically target low-density input regions which may harbor faults that address the challenge *C5*.

7.3 Mutation Testing

Mutation testing is used to evaluate the quality of a test suite. DeepMutation, MuNN, and DeepCrime propose mutation operators for testing DNNs [85, 115, 58]. The mutation operators used by DeepCrime are based on an analysis of faults that can be introduced during the development of deep learning models, which have the potential to better reflect real faults than those of DeepMutation and MuNN [59]. DeepCrime addresses the stochastic nature of the DNN training in generating mutants and can be used to generate killable, non-trivial, and non-redundant mutants for testing DNNs [59, 58]. DeepCrime is also effective in identifying weak test sets with respect to the confidence with which the network makes predictions for the inputs.

IDC research conducted a study on DeepCrime to understand the sensitivity of mutation operators used by DeepCrime to feature diversity in test suites. The study results demonstrate the necessity to incorporate mutation operators that test for failures in feature adequacy test sets. DeepMetis is a test generation framework proposed to generate test inputs for achieving mutation score adequacy [108]. Comparing DisTest and DeepMetis is left for future work.

7.4 Distribution-Aware Testing

There has been research on using the distribution of the observed data for both testing and training purposes that was developed either concurrently with this research or after this research is published. Berend et al. conducted empirical studies similar to this research to identify the limitations of DNN methods that use invalid inputs [5]. Both their work and the research conducted in Chapter 3 were published concurrently. However, the experimental treatment and the research questions have little overlap. Also, this research proposed a method to incorporate data distribution in the test generation which is not included in Berend et al.'s work.

Toledo et al. developed a method to incorporate input data distribution in DNN verification and Validation [130]. Their approach results in enforcing the existing DNN falsification and verification tools to generate counter-examples that belong to the input data distribution thereby reducing the false property violations. Their technique similar to CIT4DNN uses the latent space and generator to generate test inputs. However, they treat the latent space symbolically unlike CIT4DNN which generates test descriptions resulting in systematic coverage of the partitioned latent space.

Another work used the research conducted in Chapter 3 as one of the baselines and conducted a study to assess the accuracy of the input validators, referred to as out-of-distribution detectors in this work, against human validators [107]. Their study results show that DNN test input generators produce invalid inputs, and automated input validators have a good consensus with human validators except for either too simple or too feature-rich datasets.

Khadka et al. developed a method to generate synthetic datasets to train Machine learning models [63]. Their approach similar to the CIT4DNN, includes training a VAE and applying CIT on the partitioned latent space to generate synthetic datasets with t-way coverage. However, their goal is data generation for training machine learning models, whereas CIT4DNN focuses on generation inputs for testing DNNs. While their work uses CIT, CIT4DNN uses constrained CIT on the geometry of the latent space which means that, unlike CIT4DNN, their work cannot generate inputs on a target density of the latent space. CIT4DNN does not require that the generative model should have an encoder network that allows instantiating CIT4DNN with a broader variety of generative models when compared to Khadka et al.'s work.

7.5 Test Oracles

A test oracle is an important aspect of software testing that helps in identifying the erroneous behaviors of a system under test. A test oracle is a predicate relation formulated on the output and the expected output of a software system. A test oracle passes a test when the output matches the expected output and fails the test otherwise. For traditional software, expected outputs are identified from the specifications. However, DNNs do not have well-defined specifications which makes it challenging to develop test oracles for DNN testing.

Research has explored using differential and metamorphic test oracles for identifying the erroneous behaviors of DNNs. Research such as DeepXplore, MIT, and DISSECTOR use differential test oracles [99, 123, 134]. They use multiple models or sub-models trained on the same dataset and cross-reference their outputs to formulate the differential test oracle. DeepTest [129] and DeepHyperion [155] apply domain-specific transformations on the inputs to test DNNs using metamorphic test oracles under the assumption that the transformations preserve the class category of the test inputs.

SelfOracle is a test oracle developed for testing autonomous driving scenarios [124]. This work uses the reconstruction error as a metric for passing or failing the test inputs. SelfChecker uses the outputs of the internal layers of the DNN under test as a cross-referencing test oracle [142]. A recent work ICOS formulates properties representing likely invariants from input data, training data, and the machine learning algorithm to identify the violations for detecting the erroneous inputs [46].

CIT4DNN is instantiated with a differential test oracle in this research. Studying the effectiveness of CIT4DNN for other test oracles is left for future work.

Chapter 8

Conclusion and Future Work

DNN models are being widely used in applications such as healthcare, recommender systems, speech recognition, and autonomous systems. Effective testing methodologies are required to assess the quality of these models. Several methods have been proposed in the software engineering literature to test DNNs including [99, 129, 125, 47, 95, 155, 106, 17, 61]. These methods can be categorized into pixel-level manipulation based methods that apply pixel-level transformations on the seed inputs and feature-level manipulation based methods that test DNNs by varying the features of the observed data. Pixel-level manipulation based methods test DNNs with both valid and invalid inputs. Using a large number of invalid inputs increase testing cost with little value, and invalid inputs inflate test coverage thereby providing a false notion of test adequacy to the developers. Also, pixel-level manipulation based methods cannot adequately test DNNs with feature-diverse inputs. So the DNN testing research proposed feature-level manipulation based methods that complement the pixel-level manipulation based methods to ensure the reliability of DNNs [155, 106, 59]. State-of-the-art feature-level manipulation based methods either use manual expertise or generative models to formulate the feature space of the observed data [155, 156, 61, 17, 15]. The methods that rely on manual expertise are costly and cannot be applied to complex datasets, whereas the methods that use generative models either do not systematically span the feature space or they use full combinatorial testing on the feature space which is costly. None of the existing methods can systematically test a target density of the feature space containing rare inputs of the observed data. To address these challenges with the existing methods, this research developed a comprehensive framework, DisTest, that incorporates a model of the input data distribution

in the testing process. Two applications, a black-box test adequacy criterion, IDC, and a test generation algorithm, CIT4DNN, are developed using DisTest to test DNNs systematically with valid, feature-diverse, and rare test inputs.

8.1 Contributions

First, this research demonstrated the limitations of using invalid inputs in testing through empirical studies and proposed a method for incorporating the data distribution in the testing process to address the challenges with invalid inputs. The results showed that the DNN testing methods that generate inputs by applying pixel-level manipulations on seed inputs produce a large number of invalid inputs – those that lie off the training distribution as judged by state-of-the-art techniques – thereby reducing the efficiency of the test generation process and, even worse, producing a large number of tests that might be rejected as invalid during fault triage processes. Using a large number of invalid inputs can lead test coverage techniques to value invalid tests inappropriately by achieving or improving coverage from valid tests – this has the potential to bias test generation results. This research proposed a method to address the limitations of testing methods that use invalid inputs by coupling existing out-of-distribution detection techniques with test generation algorithms. A study using a VAE-based OOD detection incorporated into an optimization-based test generation was conducted to demonstrate that the approach is effective in significantly boosting the number of valid test inputs generated and eliminating invalid tests.

Next, this research developed the DisTest framework and used it to develop a black-box test adequacy criterion, IDC, that measures test adequacy as a function of the t-way feature interactions present in a test set. IDC uses a generative model to automatically learn the features of the observed data in a low-dimensional latent space and calculates test adequacy as a measure of the systematic coverage of the partitioned latent space. As a result, the test adequacy measured by IDC reflects the diversity with respect to the feature interactions present in a test set. Experimental studies on IDC demonstrated that IDC is cost-effective in reflecting the feature interactions of a test suite, and it has a positive correlation with the fault-revealing capability of the test suites. IDC is more sensitive to the in-distribution test inputs generated using diverse test generation techniques when compared to the existing white-box metrics. While existing white-box and mutation testing techniques focus

on fault-detection effectiveness, IDC can complement these techniques by incorporating metrics that capture input feature diversity into the testing process.

Finally, this research proposed a test generation algorithm over the DisTest framework, CIT4DNN, that applies constrained combinatorial interaction testing [29] on a target density of the latent space to generate diverse inputs that satisfy the feature interaction adequacy. CIT4DNN also generates rare inputs by formulating constraints to focus testing on a target density representing the low-density region of the latent space where rare inputs are prevalent. CIT4DNN is the first cost-effective approach for validating model behavior on rare inputs. This research demonstrated that CIT4DNN is effective in generating feature-diverse test sets when compared to the state-of-the-art approaches, is cost-effective for generating rare inputs, and is effective in revealing faults.

To summarize, this research proposed methods using input data distribution to overcome the limitations of the existing DNN testing methods. This research resulted in two publications, and the artifacts of this work are made available in GitHub for replicability and reproducibility [34, 35].

8.2 Future Work

This research can be extended using the following ideas. DisTest is a modular framework comprising a generative model, out-of-distribution detector, combinatorial covering array generator, and combinatorial coverage measurement tool. There is scope for extending this research beyond the developed applications and module instantiations used in the experimental studies to improve the generalizability of the framework.

For example, current research used a VAE for instantiating the DisTest framework [68, 32, 110]. However, any generative model with an autoencoder-based architecture and a standard normal prior can be used for both IDC and CIT4DNN, such as autoencoder-based GANs [74, 87, 133, 23, 132]. Since CIT4DNN does not require an encoder module, it can be instantiated with a broader choice of generative models including GANs [44, 22, 11]. This research leveraged the properties of the standard normal distribution of the latent space whereas the recent advances in the Machine Learning literature proposed generative models that use distributions such as a hierarchical standard Normal distribution and discretized distribution for the latent space [132, 23, 133]. Future work can develop strategies to partition hierarchical and discretized latent priors

and apply combinatorial testing on these priors for measuring test adequacy and test generation.

While this research studied DNN models trained for computer vision datasets, e.g., [75, 141, 94, 60], the applicability of DisTest is not limited to computer vision domains. DisTest can be used for any datasets for which generative models can be trained. For example, generative models are available for speech and text datasets, and future work can study the generalizability of DisTest to these datasets [101, 153].

This research used differential test oracles in the studies. Future work can explore using other test oracles proposed in the literature to study the relationship between test inputs and fault density [142, 124, 134, 46]. This research can also be extended using conditional generative models, that generate inputs based on a user-specified class category [11, 121]. Using such models eliminates the need for an explicit test oracle in the DisTest framework. However, training conditional generative models requires labels for the training data which increases the framework setup cost.

Also, future research can exploit DisTest to develop novel approaches for defining test oracles by leveraging the structure of the latent space. Similar samples are adjacent to each other in the latent space and this property can be used for defining test oracles. For example, for input with a known label the adjacent inputs in the neighborhood of the input in the latent space also should have the same label. Such an oracle could be used to perform robustness testing of DNNs with respect to feature variations in the inputs similar to the adversarial testing methods that measure robustness with respect to pixel-level variations in the inputs. However, it is challenging to define the neighborhood for inputs that are at the decision boundaries of the DNN under test.

The CIT4DNN test generation algorithm developed using the DisTest framework is black-box. White-box test generation algorithms can be easily developed using the components of DisTest by developing methods that use feedback from the DNN under test to guide the test generation. Such methods have the capability to improve the fault-finding effectiveness of CIT4DNN.

Another potential research direction is to improve the scalability of the DisTest framework. Generative models for complex datasets such as ImageNet [33] use a high dimensional latent space; for example, BigGAN has a latent space with 128 dimensions [11]. A large number of latent dimensions increase the runtime and memory cost of DisTest when used with CIT tools developed for general-purpose testing. Future research could explore ideas to leverage the symmetry of the latent space to improve the scalability of the CIT tools used in DisTest.

Bibliography

[1] *A Tragic Loss*. https://www.tesla.com/blog/tragic-loss. 2016.

[2] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[3] Jinwon An and Sungzoon Cho. "Variational autoencoder based anomaly detection using reconstruction probability". In: *Special Lecture on IE* 2.1 (2015).

[4] Yoshua Bengio, Aaron Courville, and Pascal Vincent. "Representation learning: A review and new perspectives". In: *IEEE transactions on pattern analysis and machine intelligence* 35.8 (2013), pp. 1798–1828.

[5] David Berend et al. "Cats are not fish: Deep learning testing calls for out-of-distribution awareness". In: *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*. 2020, pp. 1041–1052.

[6] Avrim Blum, John Hopcroft, and Ravindran Kannan. *Foundations of data science*. Cambridge University Press, 2020.

[7] Mariusz Bojarski et al. "End to end learning for self-driving cars". In: *arXiv preprint arXiv:1604.07316* (2016).

[8] Mariusz Bojarski et al. "End to End Learning for Self-Driving Cars". In: *CoRR* abs/1604.07316 (2016). arXiv: 1604.07316. URL: http://arxiv.org/abs/1604.07316.

[9] Sam Bond-Taylor et al. "Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models". In: *IEEE transactions on pattern analysis and machine intelligence* (2021).

[10] Ali Borji. *Pros and Cons of GAN Evaluation Measures: New Developments*. arXiv:2103.09396 [cs]. Oct. 2021. URL: http://arxiv.org/abs/2103.09396.

[11] Andrew Brock, Jeff Donahue, and Karen Simonyan. "Large scale GAN training for high fidelity natural image synthesis". In: *arXiv preprint arXiv:1809.11096* (2018).

[12] Yaroslav Bulatov. *The notMNIST dataset*. http://yaroslavvb.com/upload/notMNIST. 2011.

[13] Christopher P. Burgess et al. "Understanding disentangling in β-VAE". In: *CoRR* abs/1804.03599 (2018).

[14] John Ashley Burgoyne and Stephen McAdams. "Non-linear scaling techniques for uncovering the perceptual dimensions of timbre". In: *ICMC*. 2007.

[15] Taejoon Byun and Sanjai Rayadurgam. "Manifold for machine learning assurance". In: *2020 IEEE/ACM 42nd International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE. 2020, pp. 97–100.

[16] Taejoon Byun, Sanjai Rayadurgam, and Mats P.E. Heimdahl. "Black-Box Testing of Deep Neural Networks". In: *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. 2021, pp. 309–320. DOI: 10.1109/ISSRE52982.2021.00041.

[17] Taejoon Byun et al. "Manifold-based test generation for image classifiers". In: *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE. 2020, pp. 15–22.

[18] Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks". In: *2017 ieee symposium on security and privacy (sp)*. Ieee. 2017, pp. 39–57.

[19] Raghavendra Chalapathy and Sanjay Chawla. "Deep learning for anomaly detection: A survey". In: *arXiv preprint arXiv:1901.03407* (2019).

[20] Jaganmohan Chandrasekaran et al. "A combinatorial approach to testing deep neural network-based autonomous driving systems". In: *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2021, pp. 57–66.

[21] Ricky T. Q. Chen et al. "Isolating Sources of Disentanglement in VAEs". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018.

[22] Xi Chen et al. "Infogan: Interpretable representation learning by information maximizing generative adversarial nets". In: *Advances in neural information processing systems* 29 (2016).

[23] Rewon Child. "Very deep vaes generalize autoregressive models and can outperform them on images". In: *arXiv preprint arXiv:2011.10650* (2020).

[24] Francois Chollet et al. *Keras*. 2015. URL: `https://github.com/fchollet/keras`.

[25] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. "A downsampled variant of imagenet as an alternative to the cifar datasets". In: (2017). arXiv: `1707.08819`.

[26] Tarin Clanuwat et al. *Deep Learning for Classical Japanese Literature*. Dec. 3, 2018. arXiv: `cs.CV/1812.01718 [cs.CV]`.

[27] Tyler Cody et al. "Systematic training and testing for machine learning using combinatorial interaction testing". In: *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2022, pp. 102–109.

[28] David M. Cohen et al. "The AETG system: An approach to testing based on combinatorial design". In: *IEEE Transactions on Software Engineering* 23.7 (1997), pp. 437–444.

[29] Myra B Cohen, Matthew B Dwyer, and Jiangfan Shi. "Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach". In: *IEEE Transactions on Software Engineering* 34.5 (2008), pp. 633–650.

[30] Myra B Cohen et al. "Constructing test suites for interaction testing". In: *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE. 2003, pp. 38–48.

[31] Charles Corbière et al. "Addressing failure prediction by learning model confidence". In: *Advances in Neural Information Processing Systems* 32 (2019).

[32] Bin Dai and David P. Wipf. "Diagnosing and Enhancing VAE Models". In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. 2019.

[33] Jia Deng et al. "Imagenet: A large-scale hierarchical image database". In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[34] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. "Distribution-aware testing of neural networks using generative models". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 226–237.

[35] Swaroopa Dola, Matthew B Dwyer, and Mary Lou Soffa. "Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing". In: *ACM Transactions on Software Engineering and Methodology* (2022). DOI: 10.1145/3576040. URL: https://doi.org/10.1145/3576040.

[36] *Epoch model*. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23. 2016.

[37] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. "Testing the manifold hypothesis". In: *Journal of the American Mathematical Society* 29.4 (2016), pp. 983–1049.

[38] Maurice Fréchet. "Sur la distance de deux lois de probabilité". In: *Annales de l'ISUP*. Vol. 6. 3. 1957, pp. 183–198.

[39] Abel G. *MNIST-Fashion-CNN*. 2017. URL: https://github.com/abelusha/MNIST-Fashion-CNN.

[40] Christoph Gladisch et al. "Leveraging combinatorial testing for safety-critical computer vision datasets". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 324–325.

[41] *GM Cruise recalls 300 robotaxis after crash involving San Francisco Muni bus*. https://www.cbsnews.com/sanfrancisco/news/gm-cruise-recalls-300-robotaxis-after-crash-involving-bus/. 2023.

[42] Alberto Goffi et al. "Automatic generation of oracles for exceptional behaviors". In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. 2016, pp. 213–224.

[43] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[44] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems* 27 (2014).

[45] Arthur Gretton et al. “A kernel method for the two-sample-problem”. In: *Advances in neural information processing systems* 19 (2006), pp. 513–520.

[46] Antonio Guerriero et al. “Assessing operational accuracy of CNN-based image classifiers using an oracle surrogate”. In: *Intelligent Systems with Applications* 17 (2023), p. 200172.

[47] Jianmin Guo et al. “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*. 2018, pp. 739–743.

[48] Fitash Ul Haq et al. “Automatic test suite generation for key-points detection dnns using many-objective search (experience paper)”. In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2021, pp. 91–102.

[49] Fabrice Harel-Canada et al. “Is neuron coverage a meaningful measure for testing deep neural networks?” In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 851–862.

[50] Jane Huffman Hayes and Jeff Offutt. “Input validation analysis and testing”. In: *Empirical Software Engineering* 11.4 (2006), pp. 493–522.

[51] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[52] Dan Hendrycks and Kevin Gimpel. “A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks”. In: *Proceedings of International Conference on Learning Representations* (2017).

[53] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. “Deep Anomaly Detection with Outlier Exposure”. In: *Proceedings of the International Conference on Learning Representations* (2019).

[54] Martin Heusel et al. “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017.

[55] Irina Higgins et al. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *5th International Conference on Learning Representations, ICLR*. 2017.

[56] Qiang Hu et al. "DeepMutation++: A mutation testing framework for deep learning systems". In: *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2019, pp. 1158–1161.

[57] Xiaowei Huang et al. "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability". In: *Computer Science Review* 37 (2020), p. 100270.

[58] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. "DeepCrime: mutation testing of deep learning systems based on real faults". In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2021, pp. 67–78.

[59] Nargiz Humbatova et al. "Taxonomy of real faults in deep learning systems". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 1110–1121.

[60] Kyle D Julian, Ritchie Lee, and Mykel J Kochenderfer. "Validation of image-based neural network controllers through adaptive stress testing". In: *2020 IEEE 23rd international conference on intelligent transportation systems (ITSC)*. IEEE. 2020, pp. 1–7.

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

[62] Tero Karras, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 4401–4410.

[63] Krishna Khadka et al. "Synthetic Data Generation Using Combinatorial Testing and Variational Autoencoder". In: *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2023, pp. 228–236.

[64] Hyunjik Kim and Andriy Mnih. “Disentangling by Factorising”. In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80. Proceedings of Machine Learning Research. PMLR, 2018.

[65] 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)*. IEEE. 2019, pp. 1039–1049.

[66] Jinhan Kim et al. “Reducing dnn labelling cost using surprise adequacy: An industrial case study for autonomous driving”. In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 1466–1476.

[67] Seah Kim and Shin Yoo. “Evaluating surprise adequacy for question answering”. In: *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 2020, pp. 197–202.

[68] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*. 2014.

[69] Diederik P Kingma, Max Welling, et al. “An introduction to variational autoencoders”. In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392.

[70] Alex Krizhevsky, Geoffrey Hinton, et al. “Learning multiple layers of features from tiny images”. In: (2009).

[71] D Richard Kuhn, Dolores R Wallace, and Albert M Gallo. “Software fault interactions and implications for software testing”. In: *IEEE transactions on software engineering* 30.6 (2004), pp. 418–421.

[72] D Richard Kuhn et al. “Combinatorial coverage measurement concepts and applications”. In: *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. IEEE. 2013, pp. 352–361.

[73] H. O. Lancaster. “The chi-squared distribution”. In: *Wiley, New York* (1969).

[74] Anders Boesen Lindbo Larsen et al. “Autoencoding beyond pixels using a learned similarity metric”. In: *Proceedings of The 33rd International Conference on Machine Learning*. Vol. 48. PMLR, 2016, pp. 1558–1566.

[75] Yann LeCun. “The MNIST database of handwritten digits”. In: *http://yann.lecun.com/exdb/mnist/* (1998).

[76] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[77] Nuo Li et al. “Perturbation-based user-input-validation testing of web applications”. In: *Journal of Systems and Software* 83.11 (2010), pp. 2263–2274.

[78] Zenan Li et al. “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)*. IEEE. 2019, pp. 89–92.

[79] Shiyu Liang, Yixuan Li, and R. Srikant. “Enhancing The Reliability of Out-of-distribution Image Detection in Neural Networks”. In: *International Conference on Learning Representations*. 2018.

[80] Hui Liu and Hee Beng Kuan Tan. “Covering code behavior on input validation in functional testing”. In: *Information and Software Technology* 51.2 (2009), pp. 546–553.

[81] Francesco Locatello et al. “Challenging common assumptions in the unsupervised learning of disentangled representations”. In: *international conference on machine learning*. PMLR. 2019, pp. 4114–4124.

[82] Antonio Loquercio et al. “Dronet: Learning to Fly by Driving”. In: *IEEE Robotics and Automation Letters* (2018). DOI: 10.1109/lra.2018.2795643.

[83] Lei Ma et al. “Deepct: Tomographic combinatorial testing for deep learning systems”. In: *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE. 2019, pp. 614–618.

[84] Lei Ma et al. “Deepgauge: Multi-granularity testing criteria for deep learning systems”. In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 120–131.

[85] Lei Ma et al. “Deepmutation: Mutation testing of deep learning systems”. In: *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE. 2018, pp. 100–111.

[86] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).

[87] Alireza Makhzani et al. “Adversarial Autoencoders”. In: *International Conference on Learning Representations*. 2016.

[88] Loic Matthey et al. *dSprites: Disentanglement testing Sprites dataset*. `https://github.com/deepmind/dsprites-dataset`. 2017.

[89] Joshua R Maximoff et al. “A method for analyzing system state-space coverage within a t-wise testing framework”. In: *2010 IEEE International Systems Conference*. IEEE. 2010, pp. 598–603.

[90] Muhammad Ferjad Naeem et al. *Reliable Fidelity and Diversity Metrics for Generative Models*. arXiv:2002.09797 [cs, stat]. June 2020. DOI: `10.48550/arXiv.2002.09797`. URL: `http://arxiv.org/abs/2002.09797` (visited on 01/24/2023).

[91] Muhammad Ferjad Naeem et al. “Reliable fidelity and diversity metrics for generative models”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7176–7185.

[92] Eric Nalisnick et al. “Do deep generative models know what they don’t know?” In: *arXiv preprint arXiv:1810.09136* (2018).

[93] Yuval Netzer et al. “Reading digits in natural images with unsupervised feature learning”. In: (2011).

[94] Yuval Netzer et al. “Reading digits in natural images with unsupervised feature learning”. In: (2011).

[95] Augustus Odena et al. “Tensorfuzz: Debugging neural networks with coverage-guided fuzzing”. In: *International Conference on Machine Learning*. 2019, pp. 4901–4911.

[96] Thomas J. Ostrand and Marc J. Balcer. “The category-partition method for specifying and generating fuctional tests”. In: *Communications of the ACM* 31.6 (1988), pp. 676–686.

[97] Adam Paszke et al. “Automatic differentiation in pytorch”. In: (2017).

[98] Ankita Ramjibhai Patel et al. "A combinatorial approach to fairness testing of machine learning models". In: *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE. 2022, pp. 94–101.

[99] Kexin Pei et al. "Deepxplore: Automated whitebox testing of deep learning systems". In: *proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 1–18.

[100] Scott Pendleton et al. "Perception, planning, control, and coordination for autonomous vehicles". In: *Machines* 5.1 (2017), p. 6.

[101] Victor Prokhorov et al. "On the importance of the Kullback-Leibler divergence term in variational autoencoders for text generation". In: *arXiv preprint arXiv:1909.13668* (2019).

[102] Kashif Rasul. *Fashion-MNIST-CNN*. 2017. URL: `https://gist.github.com/kashif/76792939dd6f473b7404474989cb62a8`.

[103] *Reliable Fidelity and Diversity Metrics for Generative Models*. `https://github.com/clovaai/generative-evaluation-prdc`. 2020.

[104] Jie Ren et al. "Likelihood ratios for out-of-distribution detection". In: *Advances in Neural Information Processing Systems*. 2019, pp. 14707–14718.

[105] Jie Ren et al. "Likelihood Ratios for Out-of-Distribution Detection". In: *Advances in Neural Information Processing Systems*. Vol. 32. 2019.

[106] Vincenzo Riccio and Paolo Tonella. "Model-based exploration of the frontier of behaviours for deep learning system testing". In: *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 2020, pp. 876–888.

[107] Vincenzo Riccio and Paolo Tonella. "When and Why Test Generators for Deep Learning Produce Invalid Inputs: an Empirical Study". In: *arXiv preprint arXiv:2212.11368* (2022).

[108] Vincenzo Riccio et al. "DeepMetis: Augmenting a Deep Learning Test Set to Increase its Mutation Score". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 355–367.

[109] Mihaela Rosca, Balaji Lakshminarayanan, and Shakir Mohamed. "Distribution matching in variational inference". In: *arXiv preprint arXiv:1802.06847* (2018).

[110] Oleh Rybkin, Kostas Daniilidis, and Sergey Levine. "Simple and effective VAE training with calibrated decoders". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 9179–9189.

[111] Tim Salimans et al. "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications". In: *arXiv preprint arXiv:1701.05517* (2017).

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

[113] Joan Serrà et al. "Input complexity and out-of-distribution detection with likelihood-based generative models". In: *CoRR* abs/1909.11480 (2019). arXiv: 1909.11480.

[114] Hang Shao, Abhishek Kumar, and P Thomas Fletcher. "The riemannian geometry of deep generative models". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 2018, pp. 315–323.

[115] Weijun Shen, Jun Wan, and Zhenyu Chen. "Munn: Mutation analysis of neural networks". In: *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE. 2018, pp. 108–115.

[116] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556* (2014).

[117] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: (2014). arXiv: 1409.1556.

[118] Saurabh Sinha and Mary Jean Harrold. "Analysis and testing of programs with exception handling constructs". In: *IEEE Transactions on Software Engineering* 26.9 (2000), pp. 849–871.

[119] *SINVAD replication package*. https://github.com/coinse/SINVAD. 2020.

[120] N. Smolyanskiy et al. "Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness". In: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017, pp. 4241–4247. DOI: 10.1109/IROS.2017.8206285.

[121] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. "Learning Structured Output Representation using Deep Conditional Generative Models". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes et al. Vol. 28. Curran Associates, Inc., 2015. URL: https://proceedings.neurips.cc/paper_files/paper/2015/file/8d55a249e6baa5c06772297520da2051-Paper.pdf.

[122] Jost Tobias Springenberg et al. "Striving for simplicity: The all convolutional net". In: *arXiv preprint arXiv:1412.6806* (2014).

[123] Siwakorn Srisakaokul et al. "Multiple-Implementation Testing of Supervised Learning Software." In: *AAAI Workshops*. 2018, pp. 384–391.

[124] Andrea Stocco et al. "Misbehaviour prediction for autonomous driving systems". In: *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 2020, pp. 359–371.

[125] Youcheng Sun et al. "Concolic testing for deep neural networks". In: *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 2018, pp. 109–119.

[126] Youcheng Sun et al. "Structural test coverage criteria for deep neural networks". In: *ACM Transactions on Embedded Computing Systems (TECS)* 18.5s (2019), pp. 1–23.

[127] Youcheng Sun et al. "Testing deep neural networks". In: *arXiv preprint arXiv:1803.04792* (2018).

[128] Kunal Taneja et al. "Mitv: multiple-implementation testing of user-input validators for web applications". In: *Proceedings of the IEEE/ACM international conference on Automated software engineering*. 2010, pp. 131–134.

[129] Yuchi Tian et al. "Deeptest: Automated testing of deep-neural-network-driven autonomous cars". In: 2018, pp. 303–314.

[130] Felipe Toledo et al. "Distribution models for falsification and verification of dnns". In: *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE. 2021, pp. 317–329.

[131] Rachel Tzoref-Brill. "Advances in combinatorial testing". In: *Advances in Computers* 112 (2019), pp. 79–134.

[132] Arash Vahdat and Jan Kautz. "NVAE: A deep hierarchical variational autoencoder". In: *Advances in neural information processing systems* 33 (2020), pp. 19667–19679.

[133] Aaron Van Den Oord, Oriol Vinyals, et al. "Neural discrete representation learning". In: *Advances in neural information processing systems* 30 (2017).

[134] Huiyan Wang et al. "Dissector: Input validation for deep learning applications by crossing-layer dissection". In: *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 2020, pp. 727–738.

[135] Jialai Wang et al. "BET: black-box efficient testing for convolutional neural networks". In: *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2022, pp. 164–175.

[136] Michael Weiss, Rwiddhi Chakraborty, and Paolo Tonella. "A Review and Refinement of Surprise Adequacy". In: *2021 IEEE/ACM Third International Workshop on Deep Learning for Testing and Testing for Deep Learning (DeepTest)*. IEEE. 2021, pp. 17–24.

[137] Elaine J. Weyuker. "The evaluation of program-based software test data adequacy criteria". In: *Communications of the ACM* 31.6 (1988), pp. 668–675.

[138] Haoze Wu et al. "Efficient neural network analysis with sum-of-infeasibilities". In: *Tools and Algorithms for the Construction and Analysis of Systems: 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2–7, 2022, Proceedings, Part I*. Springer. 2022, pp. 143–163.

[139] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST*. `https://github.com/zalandoresearch/fashion-mnist`. 2017.

[140] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST*. 2017. URL: `https://github.com/zalandoresearch/fashion-mnist/blob/master/benchmark`.

[141] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[142] Yan Xiao et al. "Self-checking deep neural networks in deployment". In: *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE. 2021, pp. 372–384.

[143] Zhisheng Xiao, Qing Yan, and Yali Amit. "Likelihood Regret: An Out-of-Distribution Detection Score For Variational Auto-encoder". In: *Advances in Neural Information Processing Systems*. Vol. 33. 2020.

[144] Xiaofei Xie et al. "Deephunter: a coverage-guided fuzz testing framework for deep neural networks". In: *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2019, pp. 146–157.

[145] Xiaofei Xie et al. "DiffChaser: Detecting Disagreements for Deep Neural Networks". In: *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, July 2019, pp. 5772–5778. DOI: 10.24963/ijcai.2019/800. URL: https://doi.org/10.24963/ijcai.2019/800.

[146] Xiaofei Xie et al. "Npc: N euron p ath c overage via characterizing decision logic of deep neural networks". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31.3 (2022), pp. 1–27.

[147] Haowen Xu et al. "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications". In: *Proceedings of the 2018 World Wide Web Conference*. 2018, pp. 187–196.

[148] Shenao Yan et al. "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*. 2020, pp. 775–787.

[149] Yuanyuan Yuan, Qi Pang, and Shuai Wang. "Enhancing Deep Neural Networks Testing by Traversing Data Manifold". In: *arXiv preprint arXiv:2112.01956* (2021).

[150] Yuanyuan Yuan, Qi Pang, and Shuai Wang. "Revisiting neuron coverage for dnn testing: A layer-wise and distribution-aware criterion". In: *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE. 2023, pp. 1200–1212.

[151] Houssam Zenati et al. "Efficient gan-based anomaly detection". In: *arXiv preprint arXiv:1802.06222* (2018).

[152] Jie M Zhang et al. "Machine learning testing: Survey, landscapes and horizons". In: *IEEE Transactions on Software Engineering* (2020).

[153] Lan Zhang, Wray Buntine, and Ehsan Shareghi. "On the Effect of Isotropy on VAE Representations of Text". In: *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 2022, pp. 694–701.

[154] Pingyu Zhang and Sebastian Elbaum. "Amplifying tests to validate exception handling code: An extended study in the mobile application domain". In: *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23.4 (2014), pp. 1–28.

[155] Tahereh Zohdinasab et al. "Deephyperion: exploring the feature space of deep learning-based systems through illumination search". In: *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 2021, pp. 79–90.

[156] Tahereh Zohdinasab et al. "Efficient and effective feature space exploration for testing deep learning systems". In: *ACM Transactions on Software Engineering and Methodology* (2022).

Appendix A

Additional Experimental Data

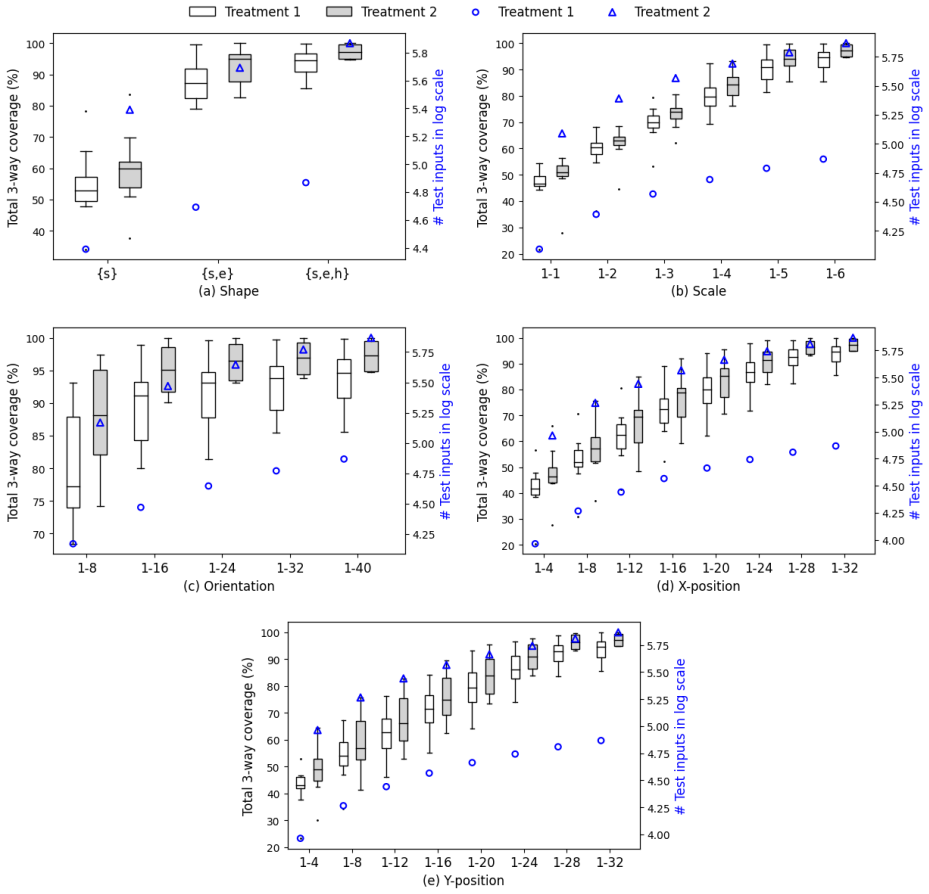


Figure A.1: Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the VAE configuration, DSP-B2.

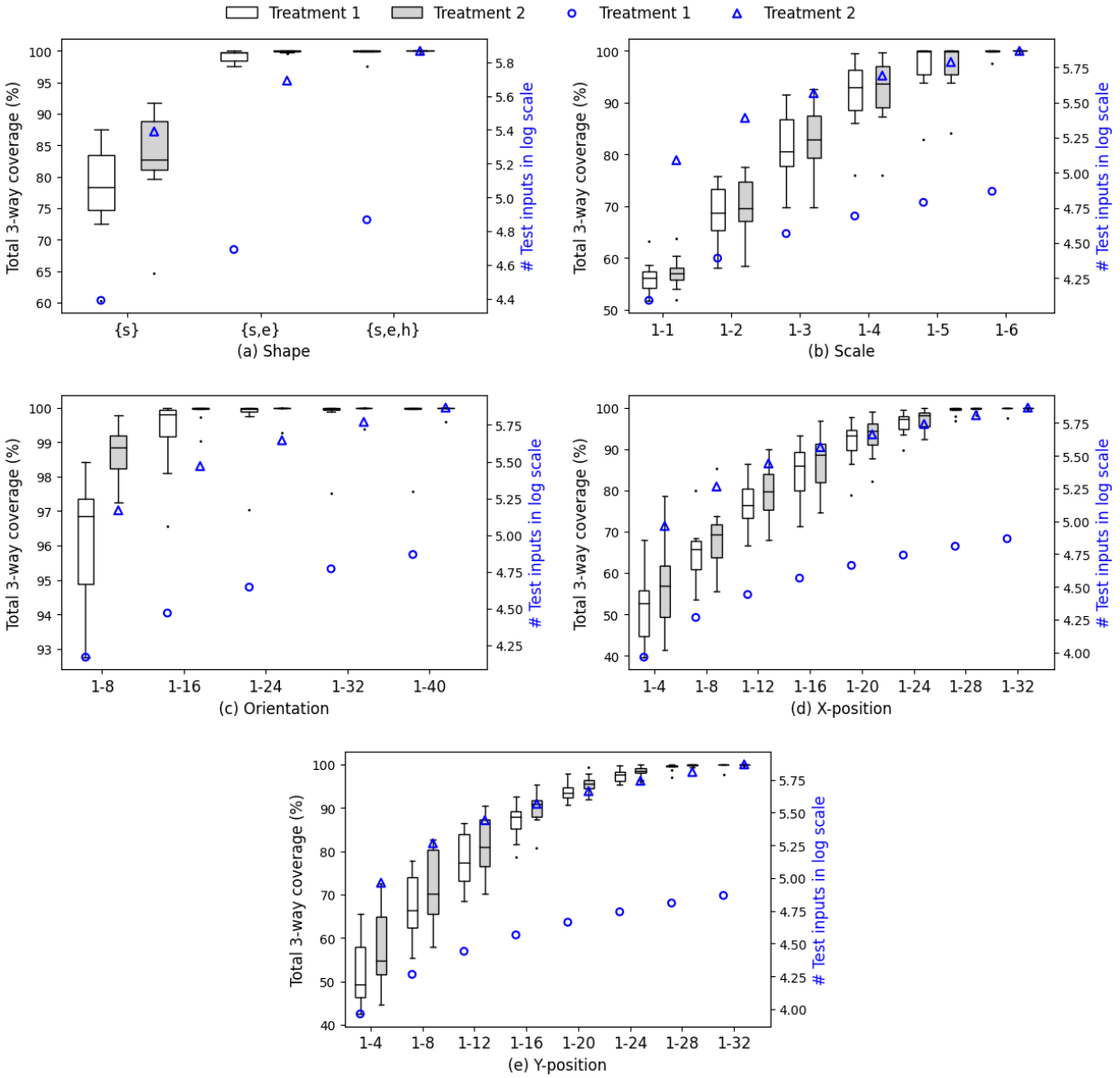


Figure A.2: Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the VAE configuration, DSP-F.

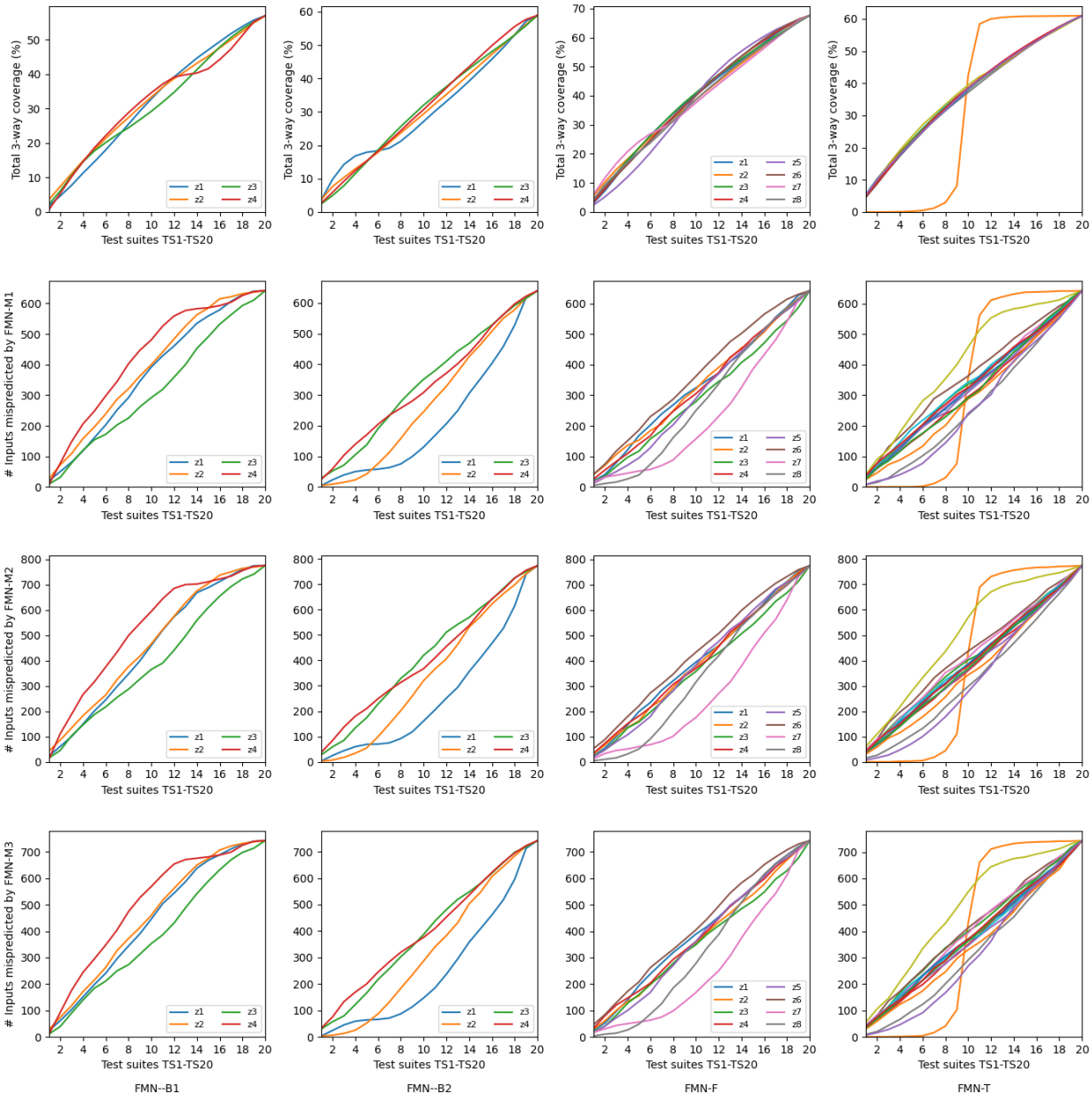


Figure A.3: The total 3-way coverage (top row) and the number of fault-revealing test inputs of the test suites (2-4 rows) created using Fashion-MNIST dataset for the latent dimensions (z) of VAE configurations FMN-B1, FMN-B2, FMN-F and FMN-T from Table 5.2. DNN models FMN-M1, FMN-M2 and FMN-M3 from Table 6.1 are used for calculating the number of fault-revealing test inputs in the test suites. Legend is not shown in the plots with more than 10 latent dimensions.

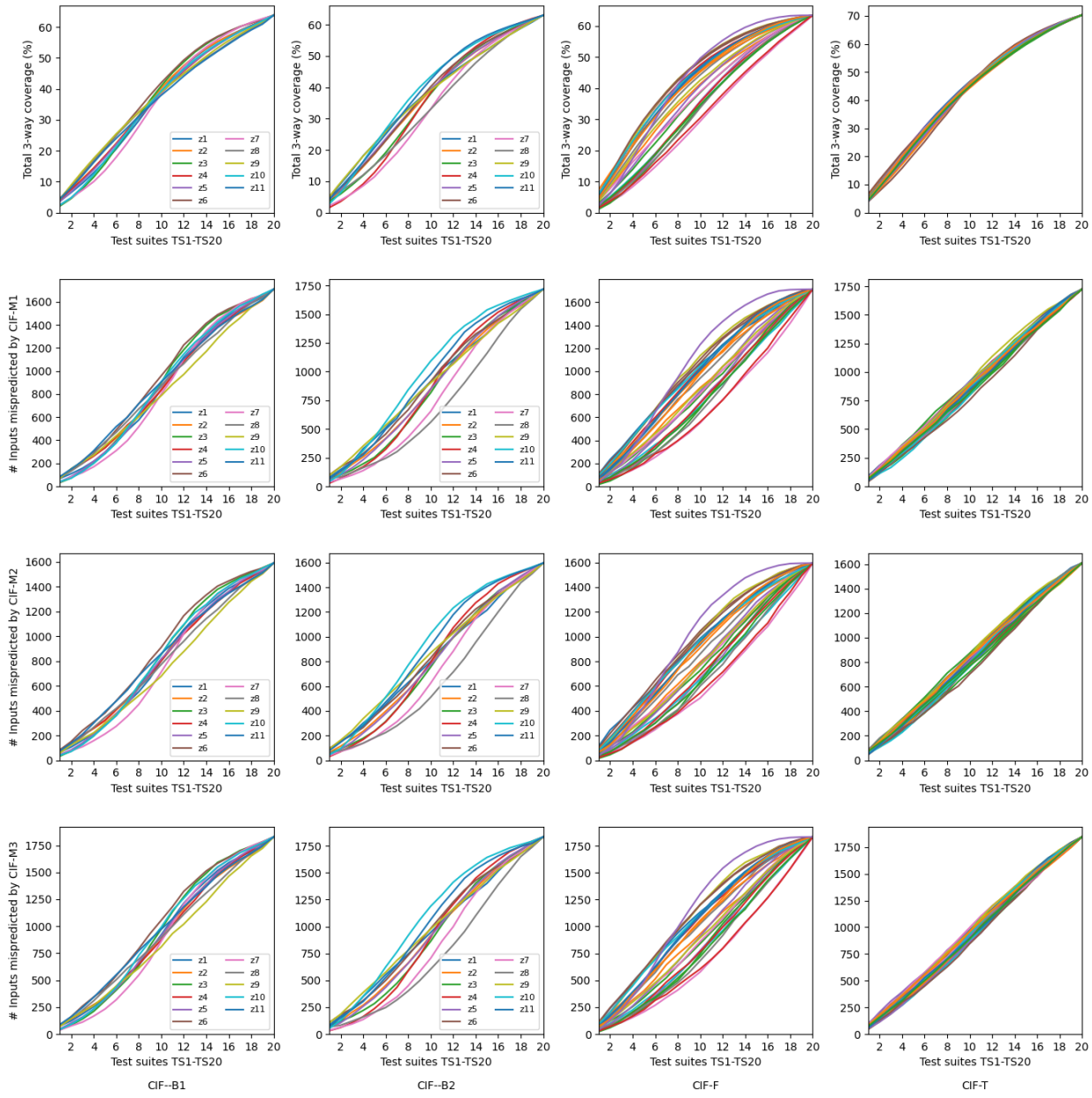


Figure A.4: The total 3-way coverage (top row) and the number of fault-revealing test inputs (2-4 rows) of the test suites created using CIFAR10 dataset for the latent dimensions (z) of VAE configurations CIF-B1, CIF-B2, CIF-F and CIF-T from Table 5.2. DNN models CIF-M1, CIF-M2 and CIF-M3 from Table 6.1 are used for calculating the number of fault-revealing test inputs in the test suites. Legend is not shown in the plots with more than 15 latent dimensions.