Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing

SWAROOPA DOLA, MATTHEW B. DWYER, and MARY LOU SOFFA, University of Virginia, USA

Testing **deep neural networks (DNNs)** has garnered great interest in the recent years due to their use in many applications. Black-box test adequacy measures are useful for guiding the testing process in covering the input domain. However, the absence of input specifications makes it challenging to apply black-box test adequacy measures in DNN testing. The **Input Distribution Coverage (IDC)** framework addresses this challenge by using a variational autoencoder to learn a low dimensional latent representation of the input distribution, and then using that latent space as a coverage domain for testing. IDC applies combinatorial interaction testing on a partitioning of the latent space to measure test adequacy. Empirical evaluation demonstrates that IDC is cost-effective, capable of detecting feature diversity in test inputs, and more sensitive than prior work to test inputs generated using different DNN test generation methods. The findings demonstrate that IDC overcomes several limitations of white-box DNN coverage approaches by discounting coverage from unrealistic inputs and enabling the calculation of test adequacy metrics that capture the feature diversity present in the input space of DNNs.

CCS Concepts: • **Computing methodologies** → **Machine learning**; **Artificial intelligence**; • **Software and its engineering** → **Software creation and management;**

Additional Key Words and Phrases: Software testing, deep neural networks, generative models, test coverage

ACM Reference format:
Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. 2023. Input Distribution Coverage: Measuring Feature Interaction Adequacy in Neural Network Testing. *ACM Trans. Softw. Eng. Methodol.* 32, 3, Article 81 (April 2023), 48 pages.
https://doi.org/10.1145/3576040

1 INTRODUCTION

Testing is the primary means of determining whether a software system behaves as intended. A fundamental challenge in testing is determining the adequacy of the selected set of test inputs, i.e., whether they have sufficient diversity to exercise the range of nominal and off-nominal behavior of a system implementation. Several decades of research on test adequacy have culminated in

This material is based in part upon work supported by National Science Foundation awards 2019239 and 2129824, by The Air Force Office of Scientific Research under award number FA9550-21-0164, and by Lockheed Martin Advanced Technology Laboratories.

Authors' addresses: S. Dola, 85 Engineer's Way, Department of Computer Engineering, University of Virginia, Charlottesville, Virginia, 22904, US; email: sd4tx@virginia.edu; M. B. Dwyer and M. L. Soffa, 85 Engineer's Way, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 22904, US; emails: {matthewbdwyer, soffa}@virginia.edu.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2023 Association for Computing Machinery.
1049-331X/2023/04-ART81 $15.00
https://doi.org/10.1145/3576040

two broad frameworks for addressing this challenge: (1) white-box coverage criteria which assess adequacy in terms of the internal behavior of a system implementation [3, 13, 17, 89], and (2) black-box coverage criteria which assess adequacy in terms of a system's input domain [52, 71]. White and black-box criteria are viewed as complementary [38] and combine to provide a measure of confidence that the software has been adequately tested.

Most traditional software coverage criteria are not directly applicable to testing **Deep Neural Networks (DNN)** due to fundamental differences in the context in which DNNs are employed and in their implementation structure. First, traditional black-box criteria rely on input specifications to determine how the input space should be partitioned, and since such an input specification is unavailable for DNNs, prior work is inapplicable. Second, traditional structural white-box criteria rely on the fact that the components of the implementation under test that are executed will vary across test inputs. There is no such variation with DNNs, making much of the prior work on structural coverage criteria inapplicable.

Recent years have seen the development of a range of white-box coverage approaches for DNNs that measure coverage in terms of predicates defining partitions of values resulting from subcomputations. Many approaches focus on the subcomputation performed by an individual neuron using different partitions of the computed results, e.g., [40, 63, 64, 73, 83]. These coverage approaches have a number of limitations as demonstrated by the recent studies [5, 10, 24, 25, 32, 58, 94]. Some of the limitations of white-box DNN coverage criteria demonstrated by these studies are related to their inability to capture a model of the *input distribution* – the space of inputs over which the model should generalize. It has been shown that coverage metrics do not account for variation across the input distribution [5, 32]. It has also been shown that coverage metrics do not suppress coverage for inputs that lie off of the input distribution, which can lead to significant testing inefficiencies [5, 24]. These findings point to the need for methods of measuring the adequacy of a DNN test suite that overcome these challenges.

In this paper, we present ***input distribution coverage*** **(IDC)** – a novel black-box coverage framework for DNN test adequacy. IDC defines adequacy criterion over the domain of realistic inputs, and measures the test coverage as a function of the feature combinations present in test inputs. The main challenge in measuring feature combinations is the lack of formal input specifications for DNNs. IDC addresses this by exploiting the fact that the training data for a DNN implicitly defines a space of features[1] that can play the role of the unavailable input specification. Features might describe "what kind of vehicle is in the image", "the size of the vehicle", "the horizontal coordinate of the vehicle", "the vertical coordinate of the vehicle", "the orientation of the vehicle", and so on. While there may be many such features, in general there are orders of magnitude fewer than the dimensions of the DNN's input domain.

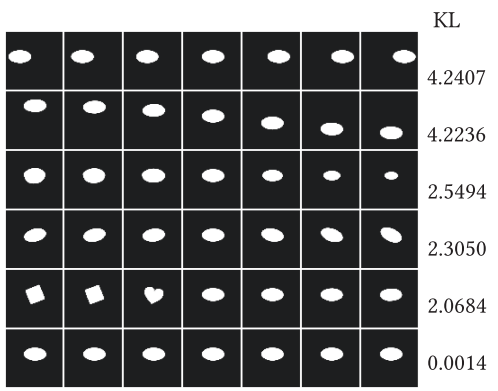

Fig. 1. Feature variation in the latent space.

To illustrate, Figure 1 depicts how varying different features for the dSprites dataset [67] 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 4,096 dimensions. The plot shows a small

[1]The machine learning literature uses the term "generative factor" to describe the sources of variation that give rise to a dataset [8]. We use the term "feature" for this concept since that is the term used in the software testing literature [18, 53].

neighborhood in the six dimensional latent space learned by training a **variational autoencoder (VAE)** [11] 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 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 five rows, to horizontal coordinate, vertical coordinate, scale, rotation, and shape variation. The bottom row shows a learned latent dimension that does not vary with 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 [8]. 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 five orthogonal features – the same features illustrated in Figure 1. In general, however, the set of features defining a dataset will not be known. Consider a DNN, $\mathcal{N} : \mathbb{R}^m \rightarrow \mathbb{R}^n$, 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 [26] 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 coverage metric for several reasons. First, due to measure concentration as $m - \dim \mathcal{X}$ grows, $\frac{|\mathcal{X}|}{|\mathbb{R}^m|}$ quickly approaches zero. Thus, *coverage that focuses on $\mathcal{X}$ has the potential to be more tractable* than coverage 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 *coverage that focuses on $\mathcal{X}$ will not be artificially inflated* by such inputs. Third, the dimensions of $\mathcal{X}$ can be understood as representing combinations of features [8]. Thus, *coverage that seeks to span the dimensions of $\mathcal{X}$ has the potential to better reflect the feature diversity of inputs* that a test suite might exercise and to which DNN behavior may be sensitive. Unfortunately, a precise definition of $\mathcal{X}$ is not available.

The IDC framework employs state-of-the-art VAEs that learn a latent representation of $\mathcal{X}$ through training on the inputs from the training data, $\boldsymbol{x} \in T$. As we describe in Section 2, 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 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. IDC relies on these properties, but leveraging a VAE's latent representation is only possible for inputs that lie on $\mathcal{X}$. To address this challenge, the IDC framework employs state-of-the-art **out-of-distribution (OOD)** detectors to filter any test inputs prior to employing a VAE.

Figure 2 depicts the high-level elements of IDC. IDC records coverage information only after filtering OOD inputs; in Figure 2 *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 4, 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

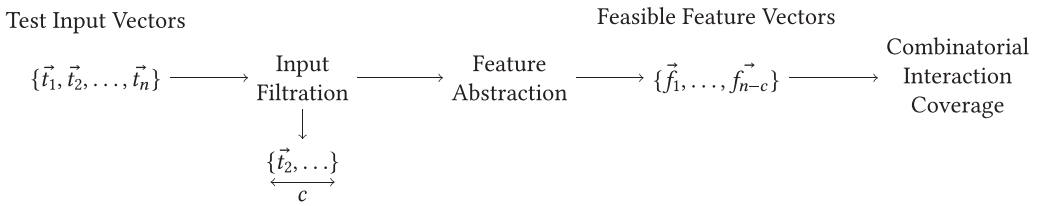

Fig. 2. Conceptual IDC workflow. Tests are filtered to focus coverage on feasible inputs. Feasibility is judged relative to a model of the input distribution. Combinatorial interaction coverage is computed in terms of abstracted input feature vectors.

combinatorial interaction coverage measures, e.g., [52], to compute coverage measures over the set of abstracted feature vectors.

As we describe in Section 3, the IDC framework is instantiated by selecting OOD and VAE components and then configuring a number of parameters adapted from **combinatorial interaction testing (CIT)** [18, 19] 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 the IDC 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 [21, 37, 46];
- 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 [30, 73, 85] across three data sets [51, 56, 91]; furthermore, demonstrating that IDC is more sensitive to variation in tests than state-of-the-art white-box DNN coverage criteria [47, 64, 73];
- 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 framework.

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. After describing background and related work, Section 3 provides a more detailed overview of the IDC framework, its instantiation, and its application.

2 BACKGROUND AND RELATED WORK

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 $\mathcal{N}$ [28]. Networks are defined as 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. A wide-range of non-linearities can be applied, but one of the more common functions is the *rectified linear unit*:

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{otherwise} \end{cases}$$

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 ability 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.

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 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 varies.

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 encoder, $\mathcal{E} : \mathbb{R}^m \to \mathbb{R}^k$, and decoder, $\mathcal{D} : \mathbb{R}^k \to \mathbb{R}^m$, networks that are trained to minimize reconstruction loss, $||\boldsymbol{x} - \mathcal{D} \circ \mathcal{E}(\boldsymbol{x})||$ [28]. 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. While autoencoders can learn an effective feature representation, the architecture and training process of an autoencoder is not designed to impose a clearly defined mathematical structure on the learned latent space. Without such a structure IDC cannot meaningfully partition the space, or compute the size of the feasible coverage domain.

A variational autoencoder (VAE) [50] 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, $||\boldsymbol{x} - \mathcal{D}_\mathcal{X} \circ \mathcal{E}_\mathcal{X}(\boldsymbol{x})||$. Despite this similarity, VAEs are designed to perform variational inference which demands a structure 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 3, 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 can be sampled, $\hat{x}$. The VAE is trained by maximizing the likelihood of

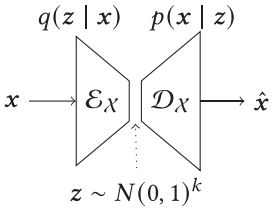

Fig. 3. Variational autoencoder.

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:

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

where β is 1.

The architecture and training process of a VAE impose a mathematical structure on the latent space that IDC 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 k grows. We use a well known property 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 [54]. 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 feasible coverage domain, as described in Section 4. 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 [78]. The density of the representation allows for a simple interval-based partitioning scheme to be applied.

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 and the lack of mode collapse – and their ability to match a prior. There are several variations of VAEs proposed in literature including β-VAE [37], FactorVAE[46], Total Correlation VAE (β-TCVAE) [11], and Two-Stage VAE [21]. One goal of VAE research has been to learn *disentangled* latent representations in which the dimensions are independent; while complete independence is rarely achievable, training can aim to reduce the correlation among latent dimensions and thereby reduce their "tangling". β-VAE learns disentangled latent dimensions by penalizing the KL-divergence term of Equation (1) using $\beta > 1$. β-TCVAE derives a three factor decomposition of the KL-divergence component of Equation (1) comprised of terms for: *index-code mutual information*, ***total correlation*** **(TC)**, and *dimension-wise KL*. It is observed that tuning the TC component is effective in learning disentangled latent dimensions [11]. FactorVAE uses an additional discriminator network for tuning the TC component. A Two-Stage VAE uses a second VAE to better approximate the posterior distribution and improve the quality of the reconstructions. The second stage VAE uses the samples from the posterior distribution of the first stage and trains to match the aggregate posterior to $\mathcal{N}(0,1)^k$. In our work, we use three of these state-of-the-art VAE architectures, namely β-TCVAE, FactorVAE, and Two-Stage VAE.

A key property of disentangled latent representations is that they learn a latent space that is the combination of linear subspaces where variation in each subspace corresponds to variation in a data generative factor, i.e., a feature, of the dataset [11]. To date, methods for accurately counting the number of features or identifying the features present in a latent representation are not available. While it would be potentially useful to have an accurate mapping from features to latent subspaces for interpretability, IDC does not depend on this. In Section 4 we define how IDC

partitions the latent space of a VAE by tiling it with hyper-rectangles. This implies that any latent linear subspace is also tiled by a subset of the hyper-rectangles partitions. In this way, IDC can compute coverage of interaction among features by measuring combinations of hyper-rectangles that tile the subspaces associated with those features.

There is strong empirical evidence that generative models, like VAEs, can learn rich and comprehensive representations of complex input spaces. For example, researchers have explored the ability 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 [45]. Measuring input realism is challenging, but a number of metrics have been developed for images, such as **structural similarity (SSIM)** [87] and **Fréchet inception distance (FID)** [36]. Recent results using VAEs show superior precision, as measured by SSIM and FID, on image generation tasks compared to alternative highly-tuned generative models, such as **generative adversarial networks (GAN)** [21, 86]. Unlike GANs, state-of-the-art VAEs offer superior recall, since they do not suffer from *mode collapse* where a model is only capable of producing a single image that includes a given feature [4].

2.3 Out-of-Distribution Detection

Test inputs that are in the very low-density regions of the training data distribution are referred to as **out-of-distribution (OOD)**; inputs in high-density regions are referred to as in-distribution. OOD detection is an active research area in Machine Learning [35, 59, 75, 77, 93]. 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 [34] and ODIN [59], use softmax scores of the classifiers for classifying OOD inputs. The generative model based approaches, such as Likelihood Regret [93] and Input-Complexity [77], use likelihood of the test inputs estimated by the generative models for classifying OOD inputs.

In this work we use an open-sourced unsupervised OOD technique proposed by Xiao et al., 2020 [93] for identifying OOD inputs. The technique works by calculating a **Likelihood-Regret (LR)** score for the test inputs using a VAE. The Likelihood-Regret score demonstrated better OOD performance over other OOD scores using VAEs such as Input Complexity [77], Likelihood-Ratio [75], and Latent Mahalanobis distance [23]. To calculate the Likelihood Regret score [75], first the log-likelihood of the test input is calculated using a trained VAE. Then the encoder of the VAE is further trained for a fixed number of iterations to optimize the log-likelihood for the test input. The difference in log-likelihoods is measured as the Likelihood-Regret score. An in-distribution input does not cause much variation in the log-likelihood when the encoder is further trained using the test input whereas the Likelihood Regret score will be higher for an OOD input. This is because the optimal encoder configuration for OOD input is much different from the encoder configuration optimized for the in-distribution inputs. To calibrate an LR detector for a given distribution, one selects other data sets that reflect out-of-distribution inputs and computes an LR score that maximizes the F-score for judging in-distribution inputs. For example, for the dSprites distribution one can define the following as OOD: MNIST [56], FashionMNIST [92], KMNIST [16], a dataset comprised of constant values, and a dataset comprised of Gaussian noise values. With this choice an LR threshold score of 40.3 maximizes the F-score for judging whether an input is in the dSprites distribution or out-of-distribution.

2.4 Combinatorial Interaction Testing

When testing a software system, it is not practical to test the full combinatorial space of the input variable configurations when the number of variables and their possible values are large. For example, for a system with 10 variables and five values for each variable, a total of $5^{10} = 9,765,625$

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 [18, 20, 53]. 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. The combinatorial sub-space is described in terms of a parameter, t, that describes the strength of the covering array [20]. A t-way covering array guarantees that all t-tuples of factor-level combinations are present in some row of the array. 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.

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. in [68] proposed combinatorial interaction test coverage metrics for measuring the test coverage of existing test suites, one of which is used in this work. We use the *total t-way coverage* which is also referred to as *Total variable-value configuration coverage* [52, 68].

Definition 1 (Total Variable-value Configuration Coverage [52]). 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 [19]. As we describe in Section 4, IDC directly leverages existing solutions for computing CIT coverage metrics and it builds on the ideas of constrained-CIT to compute the size of the feasible coverage domain.

2.5 DNN Testing

In this section, we discuss the state of research for test coverage metrics, test generation, and mutation testing of DNNs that are relevant to our work.

2.5.1 Test Coverage Metrics. In traditional software conditional statements, such as *if-then-else*, indicate that only a subset of statements may execute for a given input. In DNNs all neurons perform a computation for any given input. While this renders traditional white-box coverage metrics inapplicable to DNNs, researchers have observed that a form of conditional execution is present in DNNs that use the ReLU activation function. When a neuron's output is greater than zero, the neuron is said to be *active* which allows that neuron's input value to impact subsequent computation in the network; an *inactive* neuron outputs a zero value which suppresses any contribution of the neuron.

A range of coverage criteria based on this active-inactive distinction have been developed. *Neuron coverage* [73] measures coverage as the fraction of active neurons with respect to the total number of neurons of the DNN. Ma et al. proposed extensions to neuron coverage such as *K-multisection neuron coverage*, *neuron boundary coverage* [64], and *strong neuron activation coverage*. These metrics measure the lower, lb, upper boundaries, ub, of the magnitude of activations for all neurons using the inputs from the training dataset. The range $[lb, ub]$ is called the major function region of a neuron, and $(-\infty, lb)$ and $(ub, +\infty)$ are called the corner-case regions.

K-multisection neuron coverage is measured by dividing the major function region of each neuron into k-sections and measuring the fraction of sections covered by a test suite across all the neurons. Neuron boundary coverage is the fraction of corner-case regions covered, and strong neuron activation coverage is the fraction of upper corner regions covered by the test suite across all the neurons.

Surprise Adequacy [47] proposed two white-box test adequacy measures, ***Likelihood-based Surprise Adequacy*** **(LSA)** and ***Distance-based Surprise Adequacy*** **(DSA)** for measuring test coverage. 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 is proposed in [88], and a Mahalanobis Distance based Surprise Adequacy metric is proposed in [48] which is cost-effective when compared to LSA. While the initial evaluation of Surprise Adequacy was carried out using image classification datasets, [49] extended the Surprise Adequacy work to testing a Natural Language Processing model.

For complex coverage metrics, such as data flow criteria [17], a well known challenge is that it is difficult to accurately calculate the size of the feasible coverage domain. For instance, def-use coverage uses an overapproximation computed by static analysis that does not account for path executability [17]. Without knowing the feasible coverage domain, one cannot compute a normalized metric, i.e., one that is reported as percent coverage, and instead one must resort to reasoning about saturation [80] which is unable to make absolute statements about test adequacy for a given metric.

DeepCT [63] applies Combinatorial Interaction Testing to measure DNN test coverage. Unlike IDC, it is a white-box coverage technique where the t-way interactions among neuron activations are measured to calculate three test coverage metrics: *t-way combination sparse coverage*, *t-way combination dense coverage*, and *(p, t)-completeness*. 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 to 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.

All of the white-box DNN test adequacy and coverage measures described above measure test coverage for both in-distribution and OOD inputs. As reported by a study using neuron coverage and the extended neuron coverage metrics, such test coverage measures are inflated by OOD inputs [24]. Also the metrics are not designed to measure the feature diversity in the test suites. Both the challenges are addressed by IDC.

Byun and Rayadurgam were the first to propose computing coverage over the data manifold in a short paper [9]. Their work computes combinatorial coverage of the input data manifold by exploiting the latent space of a VAE [50] and, like IDC, they partition the dimensions of the latent space to achieve this. Their technique suffers from several drawbacks. First, they do not consider the presence of noise dimensions in the latent space of the VAE which, as we show in Section 5, can occur with some frequency and when they do they degrade the accuracy of coverage information. Second, they ignore the notion of the feasibility of partition combinations which 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 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. For several of the data sets discussed in Section 5 the size of the full combinatorial space is on the order of 10^{81}. While Byun and Rayadurgam's short-paper describes a novel concept, the work was not developed further and its limitations make it impractical. IDC addresses all of these limitations and, as shown in the evaluation in Section 5, yields a cost-effective DNN coverage metric that is sensitive to input feature variation.

2.5.2 Test Generation. Several test generation techniques have been developed in recent years for DNNs [30, 62, 73, 85]. DeepXplore [73], DLFuzz [30], and DeepConcolic [84] are coverage guided white-box test generation techniques. DeepXplore and DLFuzz use gradient ascent based optimization whereas DeepConcolic applies concolic testing on the DNNs for test generation. TensorFuzz [70] generates new inputs by mutating the inputs, and the testing is guided by a test coverage metric measured over the activation traces of the neurons. DeepTest [85] applies image transformations on the seed inputs, and it uses neuron coverage as a test coverage measure. Researchers developed a feature-guided black-box test generation to produce adversarial test inputs [90]. In this work, a feature is defined to be a region of the input that a prediction models pays particular attention to – as judged by saliency techniques [62] – and where changes are likely to lead to mis-classification. In contrast, IDC's notion of feature corresponds not to a region in the input, but rather a generative factor whose variation, in combination with variation in other such factors, gives rise to the data distribution.

DeepHyperion [95] is a DNN test generation technique that uses illumination search across the feature space to generate test inputs that are close to fault-revealing test inputs. However, DeepHyperion requires human expertise to identify the interpretable features of a dataset, whereas IDC does not need any human expertise, and the features learnt by the VAE component of IDC need not be human interpretable. A many-objective search is used by [31] to manipulate the features space of a dataset for generating fault-revealing test inputs for testing Key-Points detection DNNs [41, 44]. However, this technique also requires knowing the possible features of the training dataset of the DNN under test.

In our evaluation of IDC, we use three test generation frameworks: DeepXplore, DeepTest, and DLFuzz, that use algorithmically different approaches and thereby generate distinct test sets. The choice of these frameworks is guided by the availability of open-sourced models for the VAEs, the OOD input detection algorithms, and the test coverage metrics used in our studies and the feasibility of the evaluation cost across a range of research questions that we studied in Section 5. We plan to study the effectiveness of IDC with respect to other test generation techniques such as DeepHyperion [95] in future work.

2.5.3 Mutation Testing. Mutation testing is used to evaluate the quality of a test suite. DeepMutation [65], MuNN [79], and DeepCrime [43] propose mutation operators for testing DNNs. The mutation operators used by DeepCrime are based on an analysis of faults that can be introduced during the development of deep learning models [42], which has the potential to better reflect real faults than those of DeepMutation and MuNN. 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 [42, 43]. DeepCrime is also effective in identifying weak test sets with respect to the confidence with which the network makes predictions for the inputs. We explore DeepCrime to understand the sensitivity of DNN mutation testing techniques to feature diversity in the test suites. DeepMetis [76] is a test generation framework proposed to generate test inputs for achieving mutation score adequacy. We plan to extend the study we presented using DeepCrime in Section 5 with DeepMetis in future.

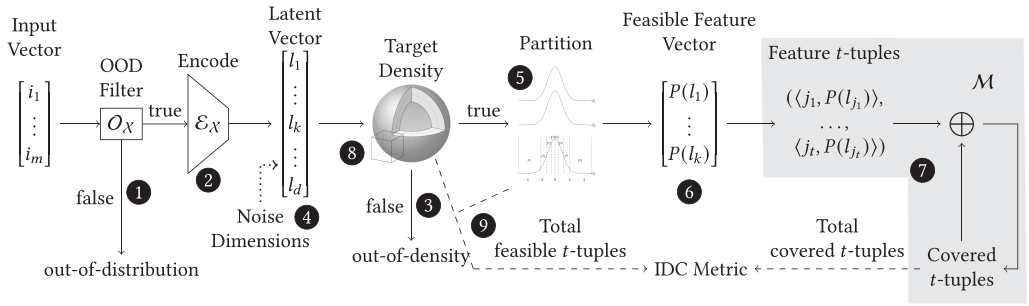

Fig. 4. Detailed IDC workflow for a single test input. Inputs are filtered using an OOD detector and relative to a chosen target density of the latent distribution. Unfiltered inputs are referred to as feasible tests and are abstracted by encoding them to a latent vector and partitioning non-noise dimensions 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.

3 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 4, 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. In the remainder of this section, we elaborate on how the elements in Figure 4 combine to realize the conceptual IDC workflow and illustrate their use on small test suite for dSprites [67] data as depicted in Figure 5.

3.1 Input Filtration

There are two distinct goals when filtering test inputs to compute the IDC metric. First, to avoid the problems of prior work [5, 24] 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., [35, 75, 93], to filter inputs that are judged not to be in $\mathcal{X}$. Thus, unlike prior work [5, 24], 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 4. As we discuss below, these radii also play an important role in defining the total feasible coverage domain for IDC.

3.2 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 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*, $[l_{k+1}, \ldots, l_d]$ (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 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.

3.3 Computing Interaction Coverage

IDC can be configured to use any number of combinatorial interaction coverage measures [52], $\mathcal{M}$, applied to the set of abstracted feature vectors. The grey background in Figure 4 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:

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

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 4 (❽). If the intersection of such a hypercube and the target density shell is empty, then we say that the feature combination associated with the hypercube is 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 number of *Total feasible t-tuples* – as depicted by dashed arrows in the figure (❾)

3.4 Working Example

We illustrate the end-to-end workflow of IDC for measuring the test adequacy of four test inputs using dSprites data in Figure 5. The example in Figure 5 depicts an instantiation of IDC using an OOD **likelihood-regret (LR)** test with a threshold of 40.3 as described in Section 2. For the four

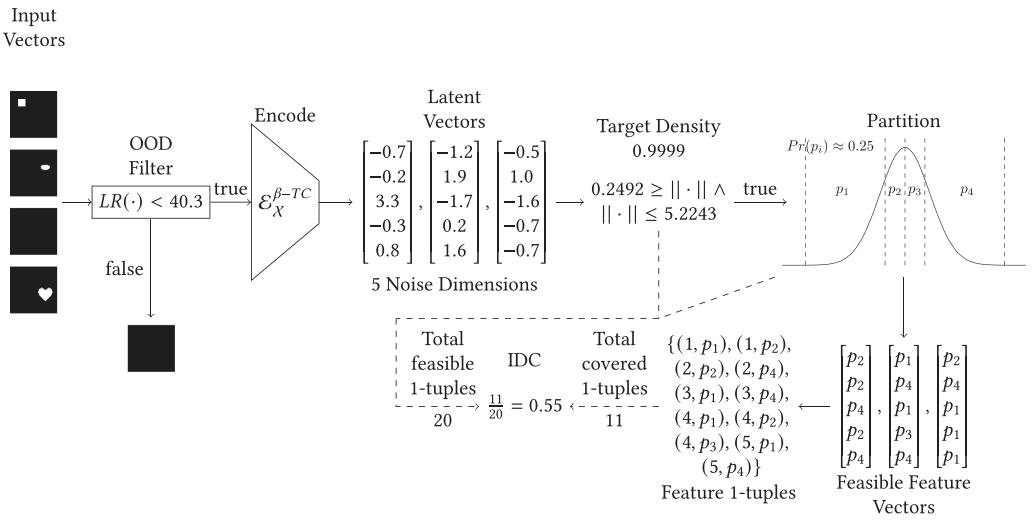

Fig. 5. Example of IDC applied to a suite of four tests for a model trained on the dSprites dataset using a likelihood-regret OOD filter with threshold of 40.3, a 10-dimensional Total Correlation VAE with five noise dimensions, a target density of 0.9999, a 4-way equal-density dimension partition, and computing total 1-way combinatorial coverage.

test inputs shown on the left of the figure, the entirely black input is filtered out and the three remaining inputs are processed by the rest of the IDC instantiation.

The Encode module is instantiated using a Total Correlation VAE (β-TCVAE) [11] with 10 latent dimensions. Despite training a 10-dimensional β-TCVAE, five were judged to be noise dimensions – not shown – leaving the five latent dimensions that are shown to define the feature space. As discussed in Section 4 – Definition 7 – the thoroughness of coverage can be configured by the user. Here a density of 0.9999 is chosen to ensure thorough coverage. Using the quantile function of the Chi distribution we compute the inner and outer radii of 0.25 and 5.22, respectively, for the five-dimensional shell in this problem. All three of the latent vectors lie within the target density shell and are therefore processed by the rest of the IDC instantiation.

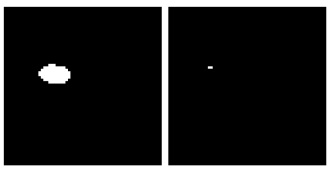

Fig. 6. Inputs generated using VAE decoder for improving 1-way coverage of Example in Figure 5.

Figure 5 shows a four-way equal density partition applied to each dimension where $Pr(p_i) \approx 0.25$. The latent vectors are converted into feasible feature vectors and three such vectors are shown in the lower right corner of Figure 5. For a simple case where $t = 1$, 11 1-tuples are covered by the three feasible test inputs. The total feasible t-tuples calculation – given in Algorithm 1 – determines that there are 20 feasible 1-tuples and, thus, IDC reports that just 55% of the feasible domain is covered by this set of three tests.

The example in this section illustrates a single set of choices for configuring IDC, but the framework can be varied by using a different OOD detector, VAE, different target density, partitioning scheme, choice of t, or combinatorial coverage measure. Moreover, the design of IDC naturally establishes a relationship between the input space and the feature space by way of the VAE. For measuring test adequacy, IDC only exploits the VAE's encoder, $\mathcal{E}_{\mathcal{X}}$, but a VAE's decoder learns to map latent vectors back to the input space. This opens up a range of future research directions building on the IDC workflow.

For example, by analyzing the nine missing 1-tuples in Figure 5 one can augment the test suite to achieve 100% coverage of 1-tuples by constructing just two feature vectors. Randomly sampling points from the intervals associated with the features in those vectors yields two latent vectors that can be extended with random values for the noise dimensions and run through the decoder to produce the test inputs shown in Figure 6. Adding these to the test in Figure 5 yields an adequate 1-way IDC test suite since it achieves 100% coverage of the domain.

4 APPROACH

IDC applies combinatorial interaction coverage measures on feature abstractions of feasible test inputs to measure DNN test adequacy. As depicted in Figure 4, IDC is a framework capable of generating test adequacy measures varying in their rigor. Instantiating the IDC framework 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 2 (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 we demonstrate in Section 5, while the choice of these components impacts the sensitivity of IDC to feature variation in tests, the framework 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.

4.1 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, $\mathcal{E}_\mathcal{X} : \mathbb{R}^m \rightarrow \mathbb{R}^d$, where $d \ll m$ as shown in Figure 4. 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., [11, 21, 37, 46], give rise to latent representations that meet these requirements. As described in Section 2 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 as we show in Section 5 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 multi-variate 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. We choose the mean values, μ_i, to represent the input in the latent space since they are the most likely values.

Definition 3 (Latent Vector). Given a trained VAE encoder, $\mathcal{E}_\mathcal{X} : \mathbb{R}^m \rightarrow \mathbb{R}^d \times \mathbb{R}^d$, 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.

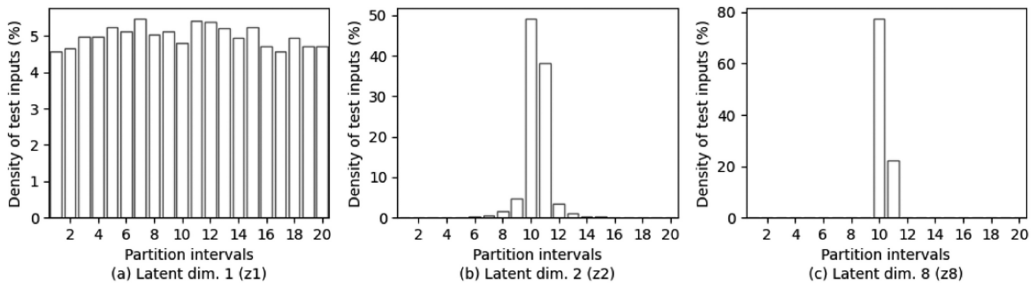

Fig. 7. The marginal empirical distributions of test inputs for latent dimensions 1, 2, and 8 of a Two-Stage VAE trained on the Fashion-MNIST dataset. The latent dimensions are partitioned into 20 intervals.

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 [8, 21, 61]. Such *noise dimensions* do not vary with the input and, consequently, they add no value in determining input coverage. We are not aware of any research on VAEs that aims to minimize the number of latent dimensions and noise dimensions while maximizing the disentangling of dimensions. Consequently, we choose an approach that identifies noise dimensions after the VAE has been trained.

To illustrate the challenge of noise dimensions, in Figure 7 we plot 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, we can eliminate such dimensions from our coverage calculation.

In IDC we identify noise dimensions by computing their KL-divergence relative to a standard Normal prior across a range of samples from $\mathcal{X}$ [8]. Low KL-divergence values indicate that little information is carried in those dimensions [8], since they match the standard Normal prior. Thought of another way, since we use just the mean values as latent coordinates, low KL-divergence means that the variance in the mean values is also very low.

Definition 4 (Noise Dimensions). A dimension, $i \in [1, d]$, of the latent space of a trained VAE encoder, $\mathcal{E}_{\mathcal{X}} : \mathbb{R}^m \rightarrow \mathbb{R}^d$, is considered noise if $KL(\mathcal{E}_{\mathcal{X}}(T)[i], \mathcal{N}(0, 1)) \approx 0$.

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 we calculate this threshold we plot the average KL-divergence for a variety of VAEs and datasets in Figure 8 on logarithmic scale; details of the VAEs are not important here and are provided in Section 5. By analyzing the gaps in KL-values and the marginal distribution of test inputs across dimensions (as in Figure 7), we empirically determined that a threshold of 10^{-2} was 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.

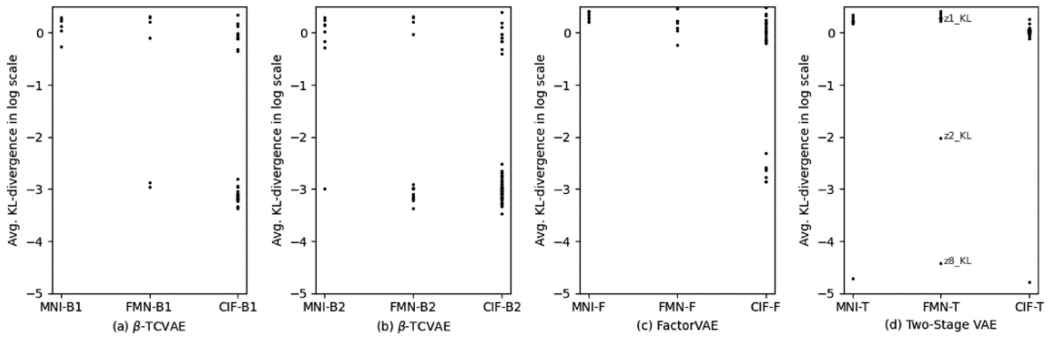

Fig. 8. Average KL-divergence of the latent dimensions of MNIST, Fashion-MNIST, and CIFAR10 VAEs trained using β-TCVAE, FactorVAE, and Two-Stage VAE configurations described in Table 2 plotted in logarithmic scale. Each marker in the scatter plot is the KL-divergence of a latent dimension. z1_KL, z2_KL 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.

For the Two-Stage VAE for Fashion MNIST (FMN-T in Figure 8) we show the KL values for dimensions 1, 2, and 8 as z1_KL, z2_KL, and z8_KL, respectively. Our threshold setting means that we would include dimension 2 as 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 we show in Section 5 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.

Our findings in Section 5 suggest that IDC can be instantiated with any state-of-the-art VAE, like the β-TCVAE, FactorVAE, and Two-Stage VAE described in Section 2. 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. We believe that 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.

4.2 Features as Latent Partitions

The features encoded in VAE latent dimensions need not correspond with a human-understandable meaning in order to calculate an informative test adequacy metric using IDC. 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 a test suite.

Coordinates within the latent space are continuous, but IDC discretizes the space by partitioning each dimension in order to produce a finite combinatorial space over which to formulate a coverage metric. Given the approximate nature of a trained VAE, one cannot expect that a 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

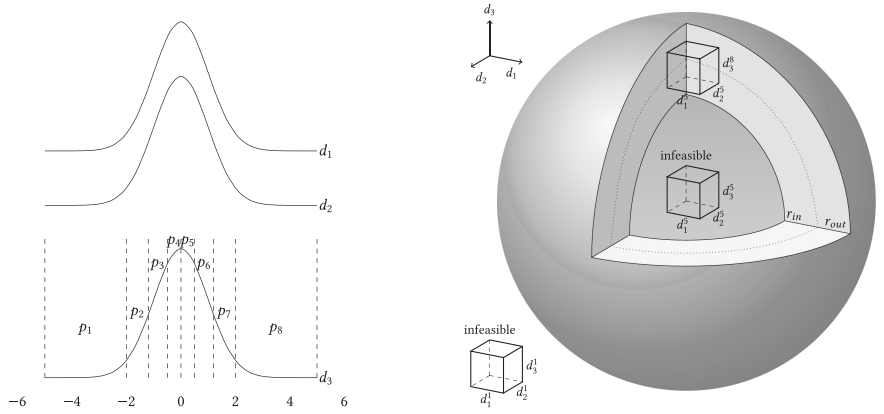

Fig. 9. Normally distributed dimensions with equal density partitions (left), Multi-variate spherical Gaussian with hyper-rectangle partitions (right).

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 IDC can manipulate to partition the latent space.

IDC 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 IDC requires that dimension partitions be symmetric around 0.

Definition 5 (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 IDC$(\mathcal{O}_\mathcal{X}, \mathcal{E}_\mathcal{X}, \mathcal{M}, d, \mathcal{P}, t)$ denotes an interval partition for a latent vector. 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 IDC 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)** [29] can be used to select VAEs whose latent distribution closely follows $\mathcal{N}(0, 1)^k$.

Figure 9 provides a detailed view of the Partition and the target density shell in Figure 4. Any symmetric partitioning scheme suffices for IDC, but one efficient and general partition simply

equally divides the space based on probability density within each dimension. As illustrated on the left side of Figure 9, the three dimensions of the sphere are partitioned, within the range $[-5, 5]$, into regions of equal probability density based on the assumed Normal prior. 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 6 (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:

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

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. Figure 9 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 we discuss below, IDC operates on a subset of partitions that comprise a target density within which adequacy measures are computed.

4.3 Feasible Feature Vectors

IDC 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 d parameter of IDC$(\mathcal{O}_{\mathcal{X}}, \mathcal{E}_{\mathcal{X}}, \mathcal{M}, d, \mathcal{P}, t)$ establishes a threshold on the density of this distribution which gives rise to a k-dimensional shell as depicted on the right side of Figure 9.

Definition 7 (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 quantile function of the Chi distribution with k degrees of freedom [6], i.e., $(r_{in}, r_{out}) = Chi.quantile(d, k)$. As k grows so will r_{in}, since little probability mass is located near zero, which gives rise to the shell-like structure.

In Figure 4, a latent vector, l, is checked to determine whether it lies within in 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.

4.4 Feasible Partition Combinations

The full combinatorial space of feasible feature vectors is intractable as a coverage domain so IDC($O_\chi, \mathcal{E}_\chi, \mathcal{M}, d, \mathcal{P}, t$) 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 4. This allows IDC to report coverage measures as a percentage of the feasible coverage domain. To explain how the number of feasible combinations are computed, we first consider the simpler unconstrained problem. 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 1 presents a more efficient calculation of the number of feasible t-way combinations that exploits the symmetry of the spherical standard Normal distribution.

We exploit the *dimension symmetry* of the sphere. 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.

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 we 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 are 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 sum of the squares of the coordinate variables, which can lead to poor performance by modern SMT solvers. To avoid this complexity, we formulate the constraint over *squared-coordinate*

ALGORITHM 1: Calculate the number of feasible t-way combinations

```
1:  function feasibleCombinations(t, k, 𝒫, [r_in, r_out])
2:      return (k choose t) · ftcCount(0, [], {}, t, 𝒫, [r_in, r_out])
3:  end function
4:  function ftcCount(depth, path, visited, t, 𝒫, [r_in, r_out])
5:      c ← 0                                              ▷ Count rooted at this path
6:      if d < t then                                      ▷ Extend path
7:          for p ∈ 𝒫 do
8:              lc ← 0                                     ▷ Local count for path extension
9:              bag = path ∪ p                             ▷ Order-independent path extension
10:             if bag ∉ visited then
11:                 newpath ← path + p                     ▷ Append partition to path
12:                 if checkConstraint(newpath, k, ⋃_{p∈𝒫} p, [r_in, r_out]) then
13:                     lc ← ftcCount(depth + 1, newpath, visited, t, 𝒫)
14:                 end if
15:                 visited ← visited ∪ (bag, lc)          ▷ Record bag with local count
16:             else
17:                 lc ← visited[bag]                      ▷ Retrieve local count for bag
18:                 c ← c + lc
19:             end if
20:         end for
21:     else
22:         c ← 1                                          ▷ Feasibility check already performed
23:     end if
24:     return c
25: end function
26: function checkConstraint({[l_1, u_1], ..., [l_n, u_n]}, k, [l_𝒫, u_𝒫], [r_in, r_out])
27:     vs ← fresh(k)                                      ▷ A fresh variable for each dimension
28:     φ ← true                                           ▷ Initialize constraint encoding
29:     for i ∈ [1, n] do
30:         φ ← φ ∧ (vs[i] ≥ ((u_i > 0)?(l_i)^2 : (u_i)^2))   ▷ Explicit squared lower bound
31:         φ ← φ ∧ (vs[i] < ((u_i > 0)?(u_i)^2 : (l_i)^2))   ▷ Explicit squared upper bound
32:     end for
33:     for i ∈ [n + 1, k] do
34:         φ ← φ ∧ (vs[i] ≥ 0 ∧ (vs[i] ≤ (u_𝒫)^2)        ▷ Maximal squared partition constraints
35:     end for
36:     φ ← φ ∧ (∑_{i∈[1,k]} vs[i]) ≥ (r_in)^2             ▷ Squared inner radius constraint
37:     φ ← φ ∧ (∑_{i∈[1,k]} vs[i]) ≤ (r_out)^2            ▷ Squared outer radius constraint
38:     return isSat(φ)
39: end function
```

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 and 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 we can directly square the inner and outer radii of the target density shell – 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. In Section 5 we study problems with $|\mathcal{P}| = 10$ and $t = 3$ and the use of Algorithm 1 reduces the number of explored paths, and SMT queries, from 1,000 to 220.

4.5 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 9 shows three dimensions partitioned eight 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 9 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, three 2-way combinations, and three 1-way combinations. Varying the strength of combinatorial coverage in IDC provides a means of controlling coverage granularity. In Section 5 we fix these parameters 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 EXPERIMENTAL EVALUATION

We explore the cost-effectiveness of IDC and how it compares to alternative approaches to assessing DNN testing techniques 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?

5.1 Dataset Selection and IDC Instantiation

To evaluate the IDC framework, we must instantiate IDC and apply it to compute the test adequacy of DNN test suites. As described in Section 4 there are six parameters to choose for instantiating IDC and several of those parameters depend on the choice of dataset. We begin by describing the rationale for selecting datasets to be used across our evaluation. The choice of datasets was influenced by the availability of other artifacts that we use to define experimental treatments or to serve as baselines in studies designed to answer the RQs.

5.1.1 Dataset and Artifact Selection. To answer RQ1 we require a dataset for which the ground truth set of features are known. This led us to choose the dSprites [67] 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 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 a 5-dimensional orthogonal feature space to images in the dataset. As we explain below, this permits the definition of experimental treatments for answering RQ1.

The application of IDC does not require knowing the ground truth set of feature, so for answering questions RQ2 through RQ6 we had more freedom in choosing datasets. To control costs in answering RQ2 through RQ6 we sought to develop a set of artifacts that could support studies across these questions. We wanted to choose multiple datasets since the set of features will vary with the dataset and we wanted to explore IDC's ability to handle that diversity. The breadth of the RQs also requires that we have 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 us to select MNIST [56], FashionMNIST [92], and CIFAR10 [51] for the experiments. MNIST is a collection of 28 by 28 gray-scale images of handwritten digits with 60,000 training images and 10,000 test images. FashionMNIST is a collection of 28 by 28 gray-scale images of articles of clothing with 60,000 training and 10,000 test images. CIFAR10 is a collection of 32 by 32 color images of a variety of animals and vehicles with 50,000 training and 10,000 test images. While there is no known ground truth set of features for these datasets, from our study of them their feature sets seem quite distinct, e.g., the thinness and tilt of a digit in MNIST, the arms and neckline of a blouse in FashionMNIST, 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 the machine learning and DNN testing literature so there were a wealth of existing artifacts we could reuse from prior work. For each of the datasets we select standard DNN models with different architectures from the literature as described in Table 1. LeNet-1, LeNet-4, and LeNet-5 [57] models are used for the MNIST dataset. Three of the benchmark models available in the Fashion-MNIST github repository [91] are used for the Fashion-MNIST dataset, and All-CNN-A, All-CNN-B, and All-CNN-C models [82] 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 [73], DeepTest [85], and DLFuzz [30] – are applicable to these datasets and models as are the implementations of white-box coverage frameworks – Neuron Coverage [73], extended Neuron Coverage metrics [64], and Surprise Adequacy [47]. We could have selected other test generation techniques that do not have available implementations, e.g., [9], but we wanted to reduce internal threats to validity by using implementations that were pre-built. For comparison with mutation approaches we selected DeepCrime [43] since it has been shown to be superior to prior work, e.g., DeepMutation [65], and has artifacts that support its application to the selected datasets. As we discuss in RQ6, we found its application to CIFAR10 to be cost prohibitive so we only considered its use with MNIST and FashionMNIST.

We also use ImageNet dataset [22] in RQ2 for demonstrating the applicability of IDC to complex datasets. ImageNet is a large scale dataset containing images from 1,000 object categories, with a training dataset of size 1,281,167 and test dataset of size 50,000. We used three DNN pretrained models available with the Keras framework [14], VGG16 [81], VGG19 [81], and ResNet50 [33] as

Table 1. Models used in our Studies with Number of Parameters (#p), and Test Accuracy; "M" Denotes Millions of Parameters

Dataset	Name	Architecture	#Parameters	Test Accuracy
MNIST [56]	MNI-M1	LeNet-1 [57]	7206	97.88%
	MNI-M2	LeNet-4 [57]	69362	98.52%
	MNI-M3	LeNet-5 [57]	107786	98.53%
Fashion [92]	FMN-M1	Custom [74]	1.6M	93.58%
	FMN-M2	Custom [91]	3.3M	92.26%
	FMN-M3	Custom [27]	211690	92.57%
CIFAR10 [51]	CIF-M1	ALL-CNN-A [82]	1.2M	82.67%
	CIF-M2	ALL-CNN-B [82]	1.3M	83.84%
	CIF-M3	ALL-CNN-C [82]	2.9M	81.45%
ImageNet [22]	IMG-M1	VGG16 [81]	138.4M	71.3%
	IMG-M2	VGG19 [81]	143.7M	71.3%
	IMG-M3	ResNet50 [33]	25.6M	74.9%

Names are shorthands used in subsequent discussion.

the DNN models under test, IMG-M1, IMG-M2, and IMG-M3, for the ImageNet experiment. The models are described in Table 1.

5.1.2 IDC Instantiation. Once a dataset has been selected, Definition 2 requires the choice of six parameters to instantiate the IDC framework: an OOD input detection technique (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 we made several pragmatic choices that allow us 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, we chose to configure IDC so 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., [52], the standard metric for assessing feature interactions is total 2-way interaction coverage. We explored the use of 2-way coverage in preliminary studies and determined that it also led to relatively early saturation. This led us to select $\mathcal{M}$ 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. 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 our choice for t we wanted to select a challenging coverage goal and, after some preliminary experimentation, we selected $d = 0.9999$. Fixing the choices for $\mathcal{M}$, t, and d reduces the breadth of our 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 of critical systems. We leave to future work experimentation with more rigorous instantiations that increase t and d.

Choosing 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 we do not employ an OOD filter in those experiments. For RQ3 and RQ5 we selected the most accurate OOD algorithm available – the state-of-the-art Likelihood Regret [93] (LR) score based technique. As described in Section 2, to use this technique one must define a set

Table 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

Dataset	Name	VAE		Latent Size	Non-Noise Latent Size
		Objective	Architecture		
MNIST	MNI-B1	β-TCVAE	Burgess et al. [8]	6	6
	MNI-B2	β-TCVAE	Burgess et al. [8]	8	7
	MNI-F	FactorVAE	Burgess et al. [8]	8	8
	MNI-T	Two-Stage VAE	InfoGAN [12]	10*	9
Fashion	FMN-B1	β-TCVAE	Burgess et al. [8]	6	4
	FMN-B2	β-TCVAE	Burgess et al. [8]	16	4
	FMN-F	FactorVAE	Burgess et al. [8]	8	8
	FMN-T	Two-Stage VAE	InfoGAN [12]	16*	15
CIFAR10	CIF-B1	β-TCVAE	Burgess et al. [8]	32	11
	CIF-B2	β-TCVAE	Burgess et al. [8]	64	11
	CIF-F	FactorVAE	Burgess et al. [8]	32	25
	CIF-T	Two-Stage VAE	InfoGAN [12]	64*	63
dSprites	DSP-B1	β-TCVAE	Burgess et al. [8]	5	5
	DSP-B2	β-TCVAE	Burgess et al. [8]	10*	5
	DSP-F	FactorVAE	Burgess et al. [8]	5	5
ImageNet	IMG-F	FactorVAE	Burgess et al. [8]	64	33

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.

of inputs that are considered to be out-of-distribution and then use that set to compute an LR score threshold. We choose the OOD inputs for the datasets used by LR in our work. For MNIST, we use a randomly sampled set of inputs, 1,000 each from Fashion-MNIST [92], CIFAR10 [51], SVHN [69], KMNIST [16], notMNIST [7], Noise, and Constant 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 [93]. For Fashion-MNIST, we replace Fashion-MNIST with MNIST for defining OOD inputs. For CIFAR10, we use MNIST, Fashion-MNIST, SVHN, CelebA [60], Noise, and Constant as OOD inputs. For each dataset we calculate the F-score 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.

In RQ3 we also selected two alternative OOD techniques, **Input-Complexity (IC)** [77] and ODIN [59], 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 us to explore a variety of different VAE models in answering RQ1 through RQ4; for RQ5 and RQ6 we used the most effective VAE per dataset as determined in the earlier studies. To explore how varying the VAE impacts IDC we selected three state-of-the-art families of VAE models – β-TCVAE [11], FactorVAE [46], and Two-Stage VAE [21] – to develop the models summarized in Table 2. As explained in Section 2, a VAE is comprised of the encoder and decoder architectures, the size of the latent space, and the training objective. We used the best-performing VAE published in the literature (marked with an asterisk in Table 2) as a starting point, e.g., a 16 dimensional latent space for the Two-Stage VAE yielded the lowest FID score for Fashion MNIST [21]. We then varied the objective function, architecture, and latent size by incorporating elements from prior work. The goal was to use a less sophisticated objective, a simpler architecture, or a smaller latent size in order to reduce

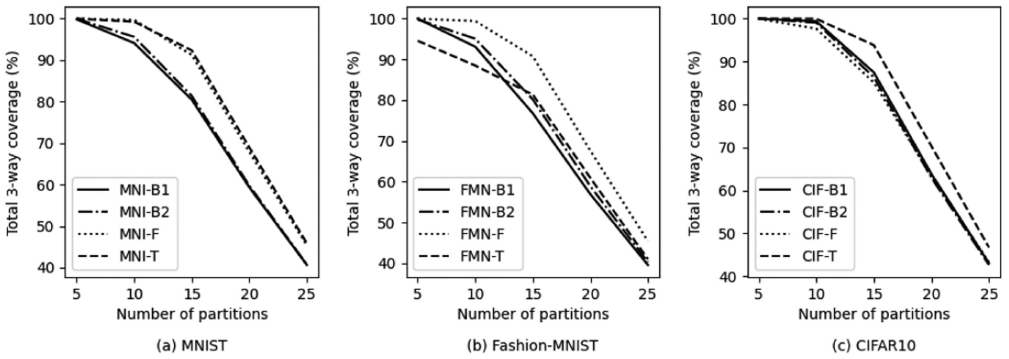

Fig. 10. Plots showing the variation in the total 3-way coverage with respect to the number of partitions of IDC.

the ability of the VAE to accurately model the data distribution. In this way, we could explore how IDC operates with a state-of-the-art VAE and with VAEs that fall short of the state-of-the-art.

In Section 4 we describe 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 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, for RQ1 we train each VAE in Table 2 ten times and provide a statistical characterization of our findings. For the remaining RQs we only train a VAE a single time, since the results from RQ1 suggest that it is not a significant factor influencing IDC performance.

Choosing $\mathcal{P}$. We chose to explore the equal density partitioning scheme from Definition 6 since it seeks to evenly balance the distribution across bins. Consistent with our choice for t and d we selected $|\mathcal{P}|$ so that the IDC metric did not prematurely saturate. This was especially relevant for RQ5 where our 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 we ran a preliminary study that varied the partition size between 5 and 25 and evaluated the total 3-way coverage computed using all of the VAEs in Table 2 for the MNIST, Fashion-MNIST, and CIFAR10 test sets. Based on the results, shown in Figure 10, we selected $|\mathcal{P}| = 20$ 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 we use this space as the domain for RQ5. While this choice was based on the experiment design for RQ5 we used the same partition size for RQ2, RQ3, RQ4, and RQ6 and leave the exploration of alternative partitioning schemes and sizes to future work.

5.2 Experimental Studies and Results

For all of the RQs, except RQ3, the cost to execute IDC is not relevant, so to facilitate experimentation we use a large and diverse set of machines to execute the IDC algorithms. For RQ3 where we study 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.

5.2.1 RQ1: How Well Does IDC Reflect the Feature Interactions Present in a Test Suite? To answer this question, we designed an experiment that allows us to manipulate the diversity of features present in a test suite and to control feature diversity while manipulating test suite size.

To manipulate feature diversity we require a dataset for which the ground truth set of features are known. Rather than generate a synthetic dataset, we select the dSprites dataset [67] which has been used in numerous studies of generative models in the ML community [8, 37, 46]. The feature model of dSprites defines five 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 six 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 $3 * 6 * 40 * 32 * 32 = 737,280$ 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 us to construct sets of inputs that vary with the dSprites ground truth features.

Our 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 three 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.

We measure 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 $\binom{5}{3}10^3 = 10,000$ 3-way combinations that can be observed.

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

We use 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) for this study and the configurations are described in Table 2. Each VAE architecture is trained 10 times to account for the stochastic variation in the training process. We use a 90%/10% training/test split which results in test sets with 73,728 images.

Figure 11 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 we show 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 Figure 11(a) the label "{s,e}" means shapes may be either a square or an ellipse and in Figure 11(c) the "1-16" label means the orientation may be in any of the first 16 intervals which corresponds 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.

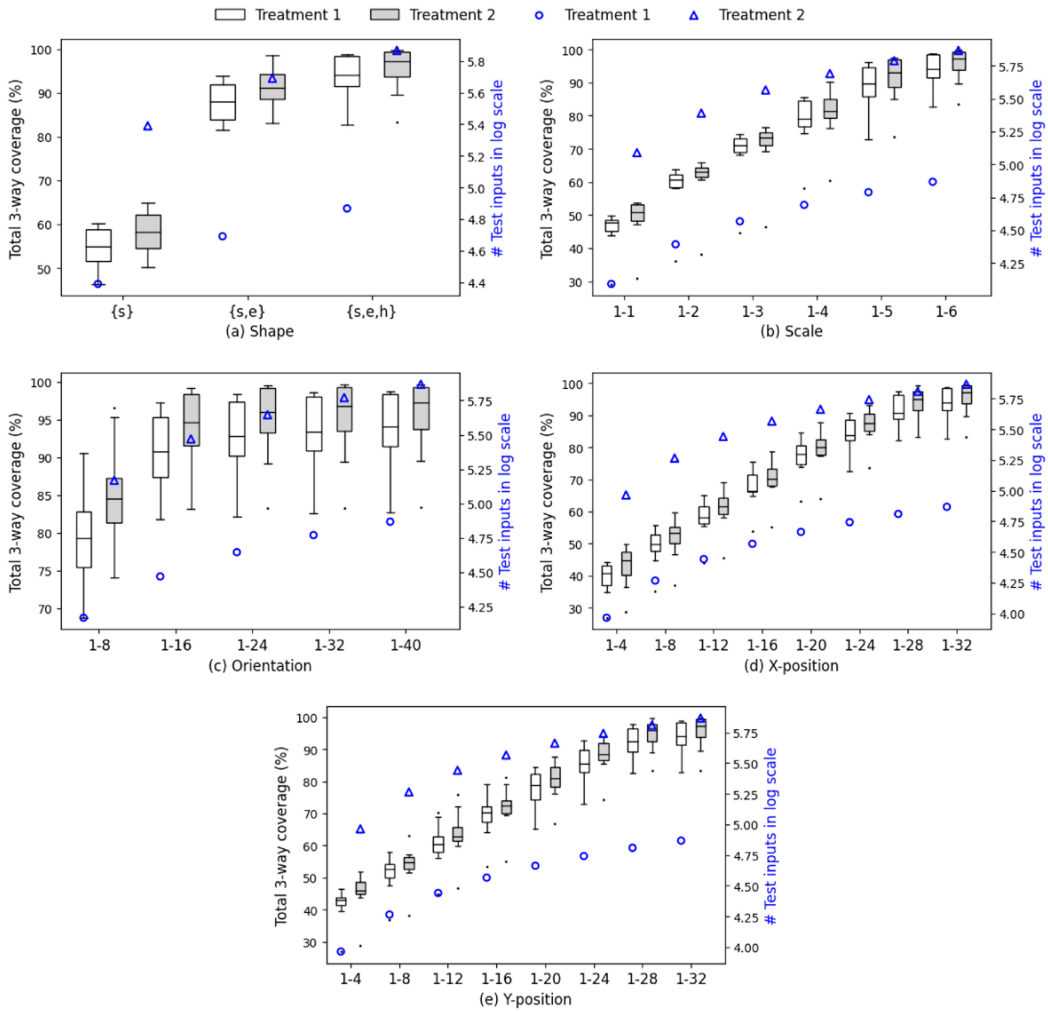

Fig. 11. Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the VAE configuration, DSP-B1.

For 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 Figure 11(b) consider the white box corresponding to "1-2" with a median total 3-way coverage value of 60.64%. Treatment 1 adds 12,278 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 nine 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.

We measured the statistical significance for the hypothesis that IDC is significantly more sensitive to increases in test diversity in comparison to increases in test size using 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 we 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 led us to explore the dSprites data further. While the dSprites feature model defines the five 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 Figure 11(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 Figure 11(a). We conjecture that this correlation of features in the image space led to our 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.

5.2.2 RQ2: Does IDC Correlate with the Fault Detection Capabilities of Test Suites? To address this question, we perform 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. We demonstrate the fault detection capability of IDC with an experimental study where we explore the number of fault-revealing test inputs contained in test suites with monotonically increasing test coverage values. We use three DNN models each for the MNIST, Fashion-MNIST, and CIFAR10 datasets (see Table 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. We created test suites with increasing total 3-way coverage for each of the VAE architectures shown in Table 2 for the three datasets. We explain the methodology in detail for creating test suites for the MNI-B1 VAE configuration. 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 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 12. Data for Fashion-MNIST and CIFAR10 datasets show the same trends and are provided in Figure B.1 and Figure B.2 in Appendix B. The curve showing the total 3-way coverage corresponding to the test suites TS_1-TS_{20} is plotted in the first row of Figure 12. The curve is monotonically increasing which confirms that the test generation approach produced a sequence of tests with increasing total 3-way coverage. We emphasize that 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 12. 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 second row indicate the number of test inputs of TS_{10} created for each latent dimension of the MNI-B1 VAE that are mispredicated by the DNN MNI-M1.

The lower three rows in Figure 12 show the number of mispredicted labels by test inputs corresponding to the sequence of test suites along the x-axis for different DNN under test with varying VAE across the columns. Note that the number of non-noise dimensions varies with the VAE.

We can observe from the plots in Figure 12 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, if we observed a plateau in these plots that would indicate that we increased the partitions covered and interactions appearing in a test suite 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. We measured 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 B.1 in Appendix B contain 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 5.1.2. This latent dimension has very few test inputs in

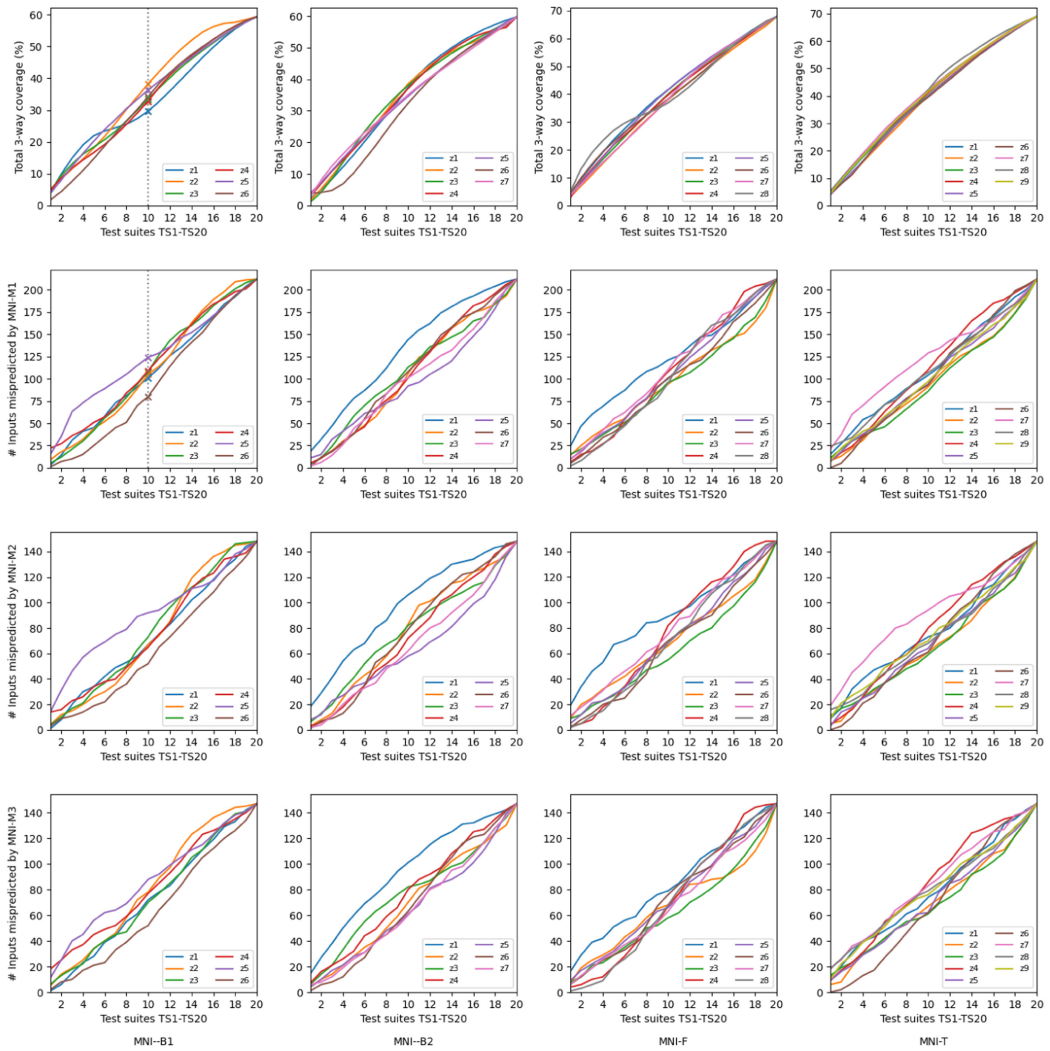

Fig. 12. The total 3-way coverage (top row) and the number of fault-revealing test inputs (2-4 rows) of the test suites created using MNIST dataset for the latent dimensions (z) of VAE configurations MNI-B1, MNI-B2, MNI-F, and MNI-T from Table 2. DNN models MNI-M1, MNI-M2 and MNI-M3 from Table 1 are used for calculating the number of fault-revealing test inputs in the test suites.

the first eight intervals because of which the number of fault-revealing test inputs and the total 3-way coverage follow the same trend.

We also performed a case study to demonstrate that IDC is effective for complex datasets by repeating the experiment for ImageNet [22]. We trained a FactorVAE configuration, IMG-F in Table 1, 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 1) are used in the experiment. While the DNN models are trained using 224×224 size inputs, we downsampled the ImageNet dataset size to 32×32 using the procedure described in [15] in order to speed up the VAE training.

Results of the experiment with the ImageNet dataset are presented in Figure 13. The plots show the total 3-way coverage and the number of test inputs mispredicted corresponding to the test

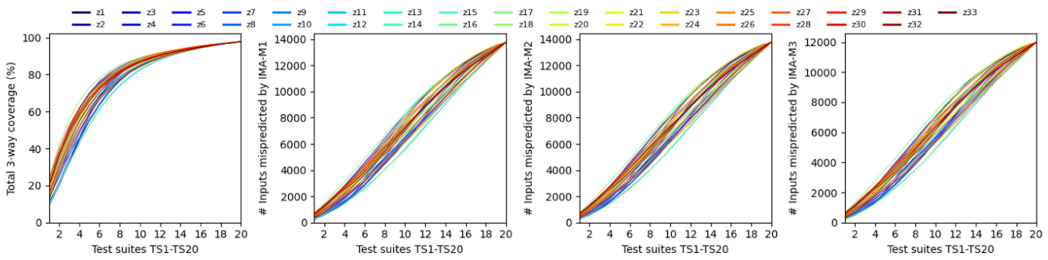

Fig. 13. The total 3-way coverage (leftmost) and the number of fault-revealing test inputs (three rightmost) of the test suites created using ImageNet dataset for the latent dimensions (z) of a FactorVAE configuration. DNN models IMG-M1, IMG-M2, and IMG-M3 from Table 1 are used for calculating the number 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.

suites TS_1-TS_{20}. The total 3-way coverage and the number of fault-revealing test inputs have a positive correlation with a 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.

5.2.3 RQ3: How Costly is it to Compute IDC?. To compute cost we measure the runtime of the major components of the IDC framework. While there are many components to the framework, in preliminary experiments we 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 1 and the combinatorial coverage 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 4, have runtimes that are orders of magnitude less than any of the above mentioned components, so we do not consider them in this study.

To evaluate the time performance of IDC, we measured the runtime to compute the total 3-way coverage for different VAE configurations, datasets, and OOD Filters. We use β-TCVAE and Two-Stage VAE configurations from Table 2 for MNIST, Fashion-MNIST, and CIFAR10 datasets. Table 3 provides a breakdown of the costs of each component of IDC using the LR OOD Filter, in seconds, for test sets of size 10,000. 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 we suspect that a significant factor is that the β-TCVAE configurations are implemented in Pytorch [72] whereas Two-Stage VAE configurations are implemented in TensorFlow [1]; 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 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 1 managed cost even in that case.

Table 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 10,000 Test Inputs

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 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

OOD Method	#In-distribution Inputs	Avg. Time per Input
Likelihood-Regret [93]	1686	1.17
Input Complexity [77]	1845	0.19
ODIN [59]	1628	0.09

The dataset consists of 2,000 test inputs generated for the CIFAR10 dataset.

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 we could use to instantiate IDC that vary in runtime and in their accuracy in detecting OOD inputs [5, 24]. We compare LR with two OOD techniques, **Input-Complexity (IC)** based OOD detection [77], and ODIN [59] by generating 2,000 CIFAR10 test inputs using DeepTest and DeepXplore in order to produce tests that are both in and out of the distribution. Table 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 six 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 11,047 to 893 seconds.

The paper reporting on LR OOD [93] 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 we believe that evaluating the accuracy of OOD alternatives on the target dataset is advisable. 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.

5.2.4 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 4, 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.

We designed a study where our treatments vary different parameters of the VAE and for randomly chosen sequences of test suites we study the variation in total 3-way coverage computed by IDC with each VAE. We study the VAEs in Table 2 for MNIST, Fashion-MNIST, and CIFAR10 datasets which vary with total latent dimension, encoder architectures, and training objective. We use a partial factorial design for each dataset that includes (1) two β-TCVAE configurations with

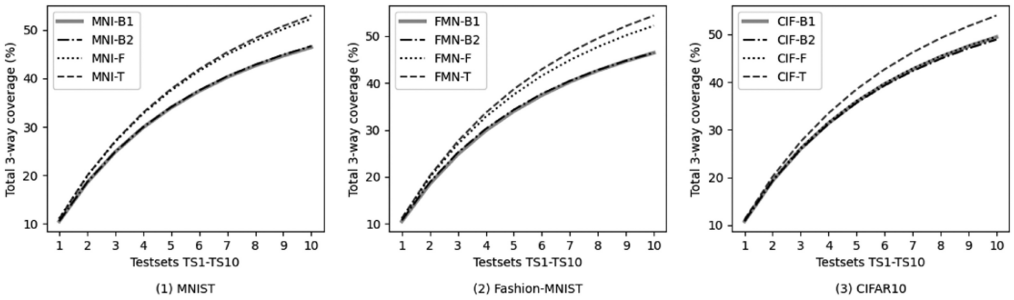

Fig. 14. Total 3-way coverage (%) computed using IDC for different VAE configurations using randomly sampled test suites from MNIST, Fashion-MNIST, and CIFAR10 test datasets averaged across 20 repetitions of the experiment.

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. We used the latent dimension of the best performing VAE for each dataset as a starting point and reduced that dimension for our other treatments. We note that 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 we report additional data to provide greater context for interpreting the coverage growth data.

We created 10 test suites, $TS_i : i \in [1, 10]$ by randomly sampling from the test datasets. TS_1 is created first by randomly selecting 1,000 inputs from the test dataset; $TS_i : i \in [2, 10]$, is created by augmenting TS_{i-1} with 1,000 random samples from the testset. We study the cumulative total 3-way coverage across this sequence of tests. Figure 14 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 1,000 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, we compute the **Maximum Mean Discrepancy (MMD)** [29] values of the VAE models as shown in Table 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 the IDC framework as it uses only non-noise dimensions for measuring test coverage.

To better characterize the variation in coverage across our experiment we calculated the **area under the curve (AUC)** of the total 3-way coverage versus the 10 test suites for all the 20 repetitions of the experiment across the three datasets and four VAEs. Figure 15 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.

We note that the best performing VAEs have the largest number of non-noise dimensions and the lowest non-noise MMD – see Table 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

Table 5. MMD Values Calculated using all the Latent Dimensions of VAEs and only the Non-Noise Dimensions of VAEs

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

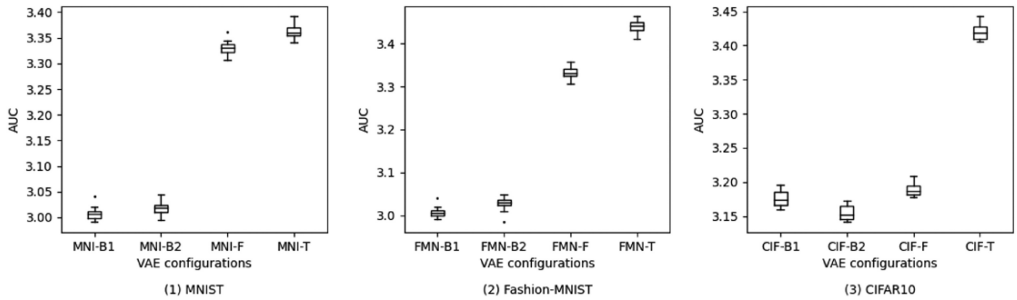

Fig. 15. AUC of curves in Figure 14 computed for different VAE configurations for MNIST, Fashion-MNIST, and CIFAR10 test datasets.

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. We measured the p-values using the Mann-Whitney U Test, which is a non-parametric test, to demonstrate the confidence in assessing 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 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.

Table 6. p-value Calculated for Different Tests using Mann–Whitney U Test for the AUC of the Curves shown in the Figure 15

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

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.

5.2.5 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, we experimentally explored the extent to which test coverage computed by IDC and existing white-box test coverage metrics are sensitive to different types of test inputs. In this experiment, we compare the percentage of coverage using total 3-way coverage metric for different test generation techniques. We focus on test inputs that are generated by a collection of different state-of-the-art DNN test generation techniques, namely DeepXplore [73], DeepTest [85], and DLFuzz [30]. 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 transformations 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 [24], these techniques generate test inputs that are both in-distribution and out-of-distribution. 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 1,000 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)** [73], **k-multisection neuron coverage (KMNC)** [64], **neuron boundary coverage (NBC)** [64], **strong neuron activation coverage (SNAC)** [64], **likelihood based surprise coverage (LSC)** [47], and **distance based surprise coverage (DSC)** [47]. We use a neuron activation threshold of 0.25 for measuring NC and parameter k value 100 for calculating KMNC. The number of buckets for measuring LSC and DSC is 1,000, the same value used in the prior works. As RQ4 demonstrated that test adequacy measured using IDC is sensitive to the VAE configuration used, we configured IDC using the MNI-B1, FMN-B1, and CIF-B1 VAEs, from Table 2, since these lead to the least sensitive instantiations of IDC as shown in Figure 14; using any other VAE would yield greater increases in test coverage for this study.

Figure 16 shows a panel of nine plots. The rows correspond to datasets and the columns to different comparison groups of coverage measures. The IDC data are repeated across the plots in the rows 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

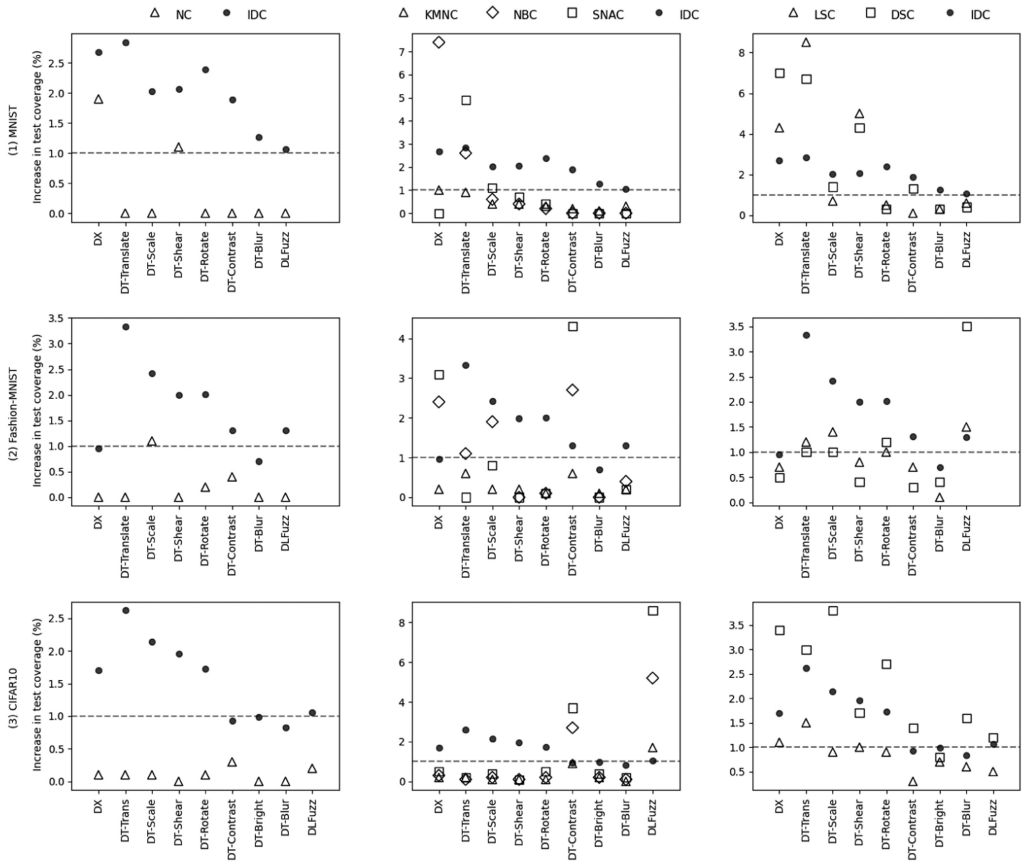

Fig. 16. Increase in test coverage obtained by augmenting test datasets of MNIST (top row), Fashion-MNIST (middle row), and CIFAR10 (bottom row) with test inputs generated by DeepXplore (DX), DeepTest (DT), and DLFuzz. All values are in percentages.

points are interpreted as the new coverage achieved by adding 1,000 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. In Figure 16, we 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.

In the first column of graphs, we observe 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 second column of graphs shows 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 seven of the nine 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 five of the 25 cases.

The third column again shows 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 forthree 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 seven of nine test generation techniques. We also measured the statistical significance at 0.05 level using the Wilcoxon signed-rank test for comparing the sensitivity of IDC with other metrics across all the 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, where as the test failed for SNAC and DSC with p-values 0.0704 and 0.552, respectively. While IDC cannot be said to be more sensitive than SNAC and DSC, we note that the total 3-way coverage falls below the 1% threshold in five of 25 cases, whereas SNAC and DSC fall below the threshold in 19 and eight cases, respectively. We believe that 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.

5.2.6 RQ6: How Does IDC Compare with DNN Mutation Testing with Respect to Feature Diversity in a Test Set? To answer this question, we conducted an experiment to compare the DNN Mutation Score with 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 [39, 43, 65, 79] 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.

In this experiment, we compare the sensitivity of DNN **Mutation Score (MS)** in reflecting the feature diversity of the inputs using the mutation operators available in the literature [43]. For the study, we use test suites 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. We use two different strategies to create the test suites for this study. In the first approach, we perform a latent traversal of the VAE trained on MNIST dataset for different seed images and identify human recognizable features. We identified orientation as a feature for the MNIST dataset, and we split the test dataset of MNIST based on the orientation of the digit images. All the images that are tilted towards left are in the first test

Table 7. Total 3-way Coverage Computed by IDC vs Mutation Score Computed by DeepCrime for Different Test Suites

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

All values are in percentages.

suite **LF1**, and the rest of the images tilted towards right are in the second test suite **LF2**. We manually verify that all the inputs in each of the test suites, LF1, and LF2 have the same feature. We consider the full test dataset, **FTS** as a testset 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, we use a different strategy for the Fashion-MNIST dataset. We split the Fashion-MNIST test dataset based on the type of object; **LF1** for Fashion-MNIST images of footwear and bags, while **LF2** contains all clothing images.

We use recently published research, DeepCrime [43], for generating mutants to test the DNNs. DeepCrime supports 24 **mutation operators (MO)** [43]. We generated mutants 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 2,680 models to be trained for each of the five different datasets used in their studies. We excluded the CIFAR10 dataset in this study since training a CIFAR10 model thousands of times is very costly in terms of the runtime.

We study the sensitivity of IDC's total 3-way coverage and the MS 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 the three test suites for both MNIST and Fashion-MNIST are presented in Table 7. As in RQ5 we use the β-TCVAE configurations, MNI-B1, FMN-B1, and CIF-B1 from Table 2 for this study.

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 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 use 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 Threats to Validity

The findings of these studies depend on a number of choices that we made in instantiating the IDC framework, in applying those instantiations, and in comparing the results of IDC to existing

baselines. We open-sourced the IDC framework for replicability of our studies 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 the framework with those components. In addition, we qualitatively spot checked the learned latent representations 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, white-box coverage techniques. Our 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 [95] and [31]. While we considered three widely used data sets from the DNN testing literature, the datasets are used in image classification domain. There is always room to improve generalizability by reproducing our study on additional datasets that are used in text based classification and regression domains. With regard to the ability of the IDC framework to generalize we considered four 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 we chose to ground our study 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.

6 CONCLUSION AND FUTURE WORK

In this paper, we propose the IDC framework for measuring black-box test adequacy of DNNs. IDC uses a VAE to convert the test inputs into feature vectors, which provide a coverage domain for applying CIT metrics to measure test coverage. We demonstrate through our experimental studies 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.

IDC relies on the VAE's learned latent representation and a VAE, like any machine learning model, can vary in quality. We advocate the use of state-of-the-art VAE models that use objectives that seek to disentangle dimensions in the latent space and demonstrate, through experimental studies, that IDC is robust to a number of variations in such models. For example, the stochastic variation due to training and the latent dimension. Since IDC exploits the structure of the assumed multivariate Gaussian prior of the latent space we advocate the use of VAEs that minimize the Maximum Mean Discrepancy [29] measure and show that these models yield the most informative IDC measures. While our studies did not sweep the full parameter space for instantiating IDC framework, they do provide clear evidence that instantiating IDC with VAEs like those described above yields effective measures of input feature interactions, coverage measures that correlate with fault detection, and that have the potential to complement existing white-box DNN coverage metrics and mutation testing techniques.

In our future work, we plan to explore other generative models such as autoencoder-based GANs [55, 66], and also study metrics such as precision/recall/fidelity [2] for guiding the selection of generative model for the IDC framework. While these metrics focus on properties of decoded images, the assumption is that precision and recall in the decoder output space is only possible when a high-fidelity latent representation has been learned.

An interesting feature of the IDC framework is its ability to target different regions of density. This offers the potential to assess coverage of rare events in testing. Differencing the shell for a target density of 0.99999 and 0.999999 defines two shells that together comprise a density of 0.000009 over which IDC metrics can be computed. We plan to investigate such an approach to evaluate testing of ML models used in critical systems.

Finally, IDC builds on the insights of prior research [9] identifying the potential of the latent space of generative models for DNN testing. We plan to explore leveraging the mathematical structure of that space to drive test generation to both augment existing test suites – as suggested in Section 3.4 – and to generate complete test suites that achieve 100% IDC coverage.

APPENDICES

A APPENDIX

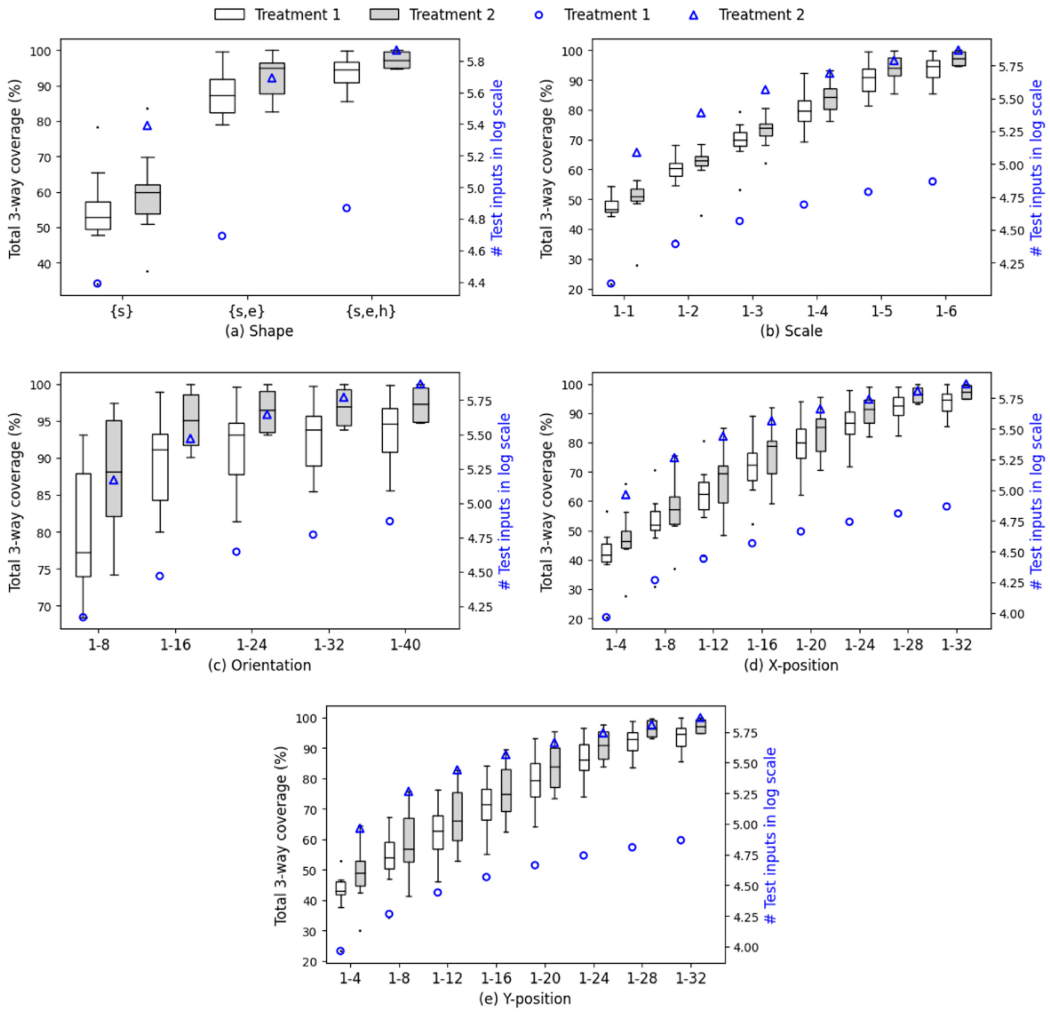

Fig. A.1. Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the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

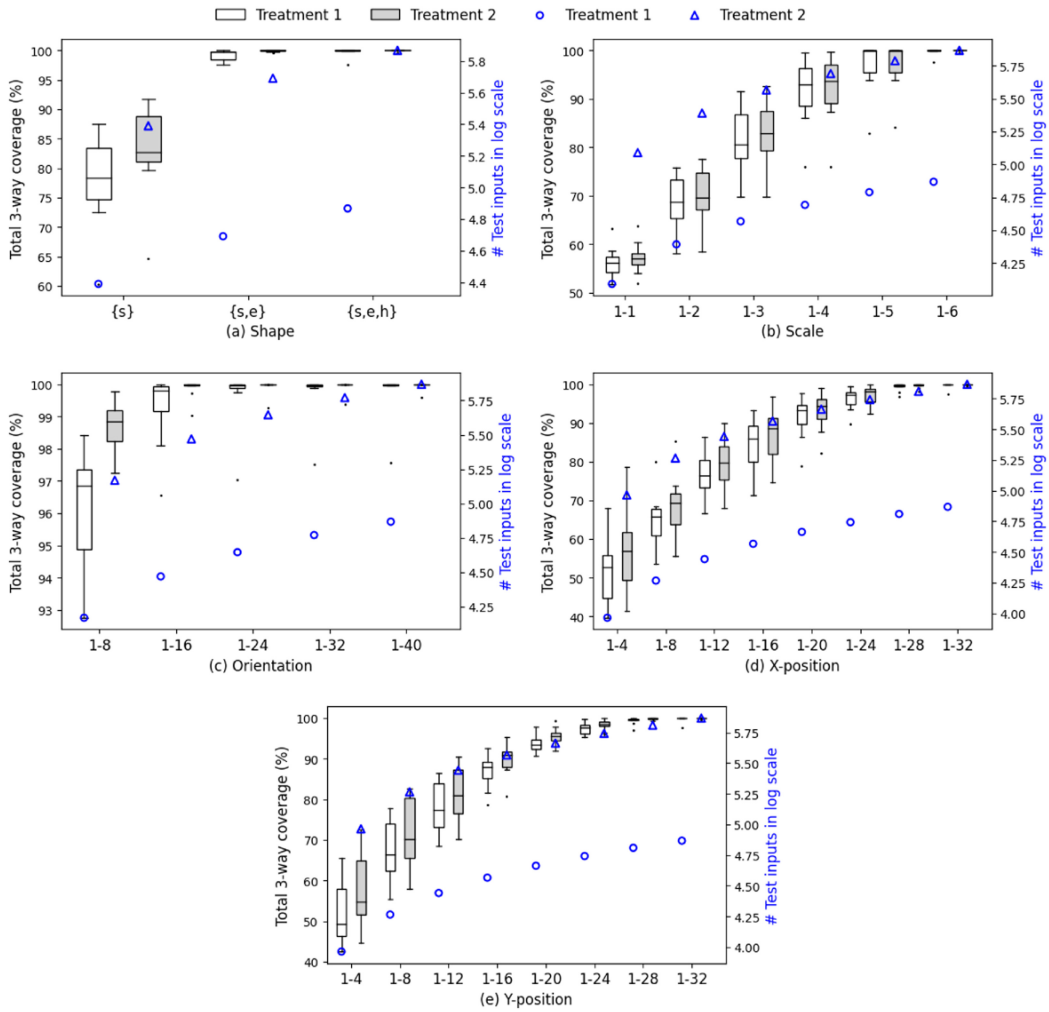

Fig. A.2. Total 3-way coverage and the number of test inputs obtained by augmenting the test inputs with a relatively small number of inputs with previously unseen features in Treatment 1 and a large number of inputs while holding the feature diversity fixed in Treatment 2. The experiment is using the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

B APPENDIX

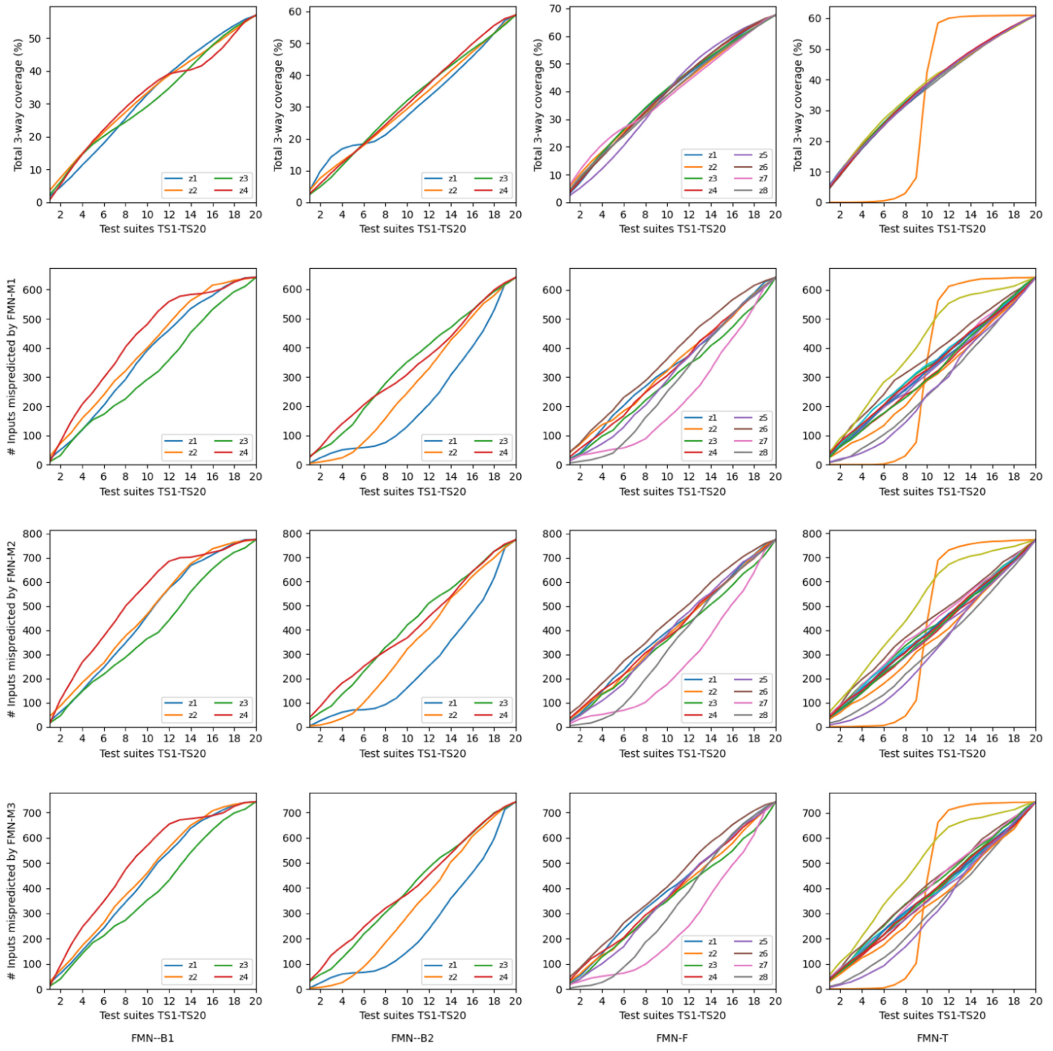

Fig. B.1. The total 3-way coverage (top row) and the number of fault-revealing test inputs of the test suites (2-4 rows) created using Fashion-MNIST dataset for the latent dimensions (z) of VAE configurations FMN-B1, FMN-B2, FMN-F, and FMN-T from Table 2. DNN models FMN-M1, FMN-M2, and FMN-M3 from Table 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.

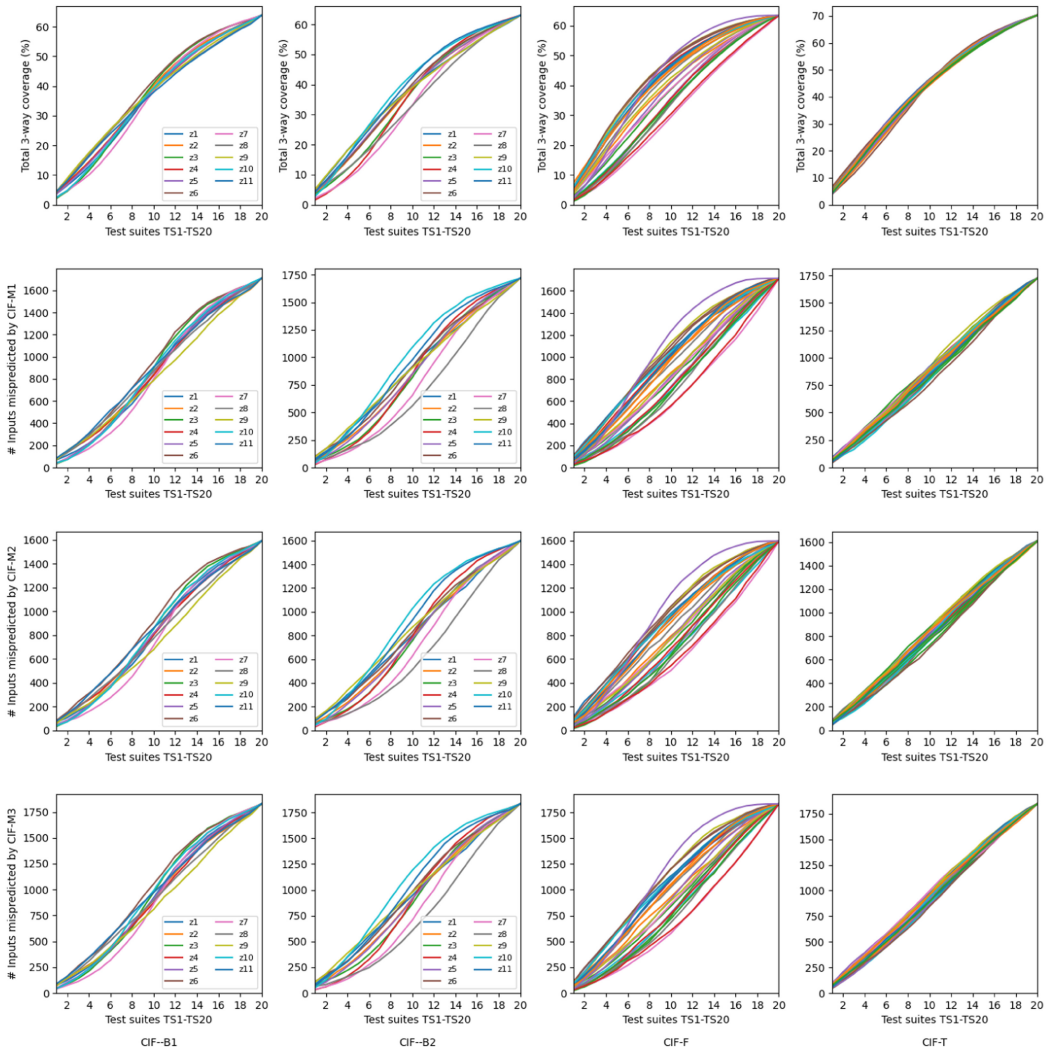

Fig. B.2. The total 3-way coverage (top row) and the number of fault-revealing test inputs (2-4 rows) of the test suites created using CIFAR10 dataset for the latent dimensions (z) of VAE configurations CIF-B1, CIF-B2, CIF-F, and CIF-T from Table 2. DNN models CIF-M1, CIF-M2, and CIF-M3 from Table 1 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.

ACKNOWLEDGMENTS

The authors would like to thank Tom Fletcher for helpful discussions about the geometry of the latent space.

REFERENCES

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar,

Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/. Software available from tensorflow.org.

[2] Ahmed M. Alaa, Boris van Breugel, Evgeny Saveliev, and Mihaela van der Schaar. 2021. How faithful is your synthetic data? Sample-level metrics for evaluating and auditing generative models. *CoRR* abs/2102.08921 (2021). arXiv:2102.08921.

[3] James H. Andrews, Lionel C. Briand, Yvan Labiche, and Akbar Siami Namin. 2006. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering* 32, 8 (2006), 608–624.

[4] David Bau, Jun-Yan Zhu, Jonas Wulff, William Peebles, Hendrik Strobelt, Bolei Zhou, and Antonio Torralba. 2019. Seeing what a GAN cannot generate. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 4502–4511.

[5] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. 2020. 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*. 1041–1052.

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

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

[8] Christopher P. Burgess, Irina Higgins, Arka Pal, Loïc Matthey, Nick Watters, Guillaume Desjardins, and Alexander Lerchner. 2018. Understanding disentangling in β-VAE. *CoRR* abs/1804.03599 (2018).

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

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

[11] Ricky T. Q. Chen, Xuechen Li, Roger Grosse, and David Duvenaud. 2018. Isolating sources of disentanglement in VAEs. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*.

[12] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. 2016. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*. 2180–2188.

[13] John Joseph Chilenski and Steven P. Miller. 1994. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal* 9, 5 (1994), 193–200.

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

[15] Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2017. A downsampled variant of ImageNet as an alternative to the CIFAR datasets. (2017). arXiv:1707.08819.

[16] Tarin Clanuwat, Mikel Bober-Irizar, Asanobu Kitamoto, Alex Lamb, Kazuaki Yamamoto, and David Ha. 2018. Deep learning for classical Japanese literature. (2018). arXiv:1812.01718.

[17] Lori A. Clarke, Andy Podgurski, Debra J. Richardson, and Steven J. Zeil. 1989. A formal evaluation of data flow path selection criteria. *IEEE Transactions on Software Engineering* 15, 11 (1989), 1318–1332.

[18] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23, 7 (1997), 437–444.

[19] Myra B. Cohen, Matthew B. Dwyer, and Jiangfan Shi. 2008. Constructing interaction test suites for highly-configurable systems in the presence of constraints: A greedy approach. *IEEE Transactions on Software Engineering* 34, 5 (2008), 633–650.

[20] Myra B. Cohen, Peter B. Gibbons, Warwick B. Mugridge, and Charles J. Colbourn. 2003. Constructing test suites for interaction testing. In *25th International Conference on Software Engineering, 2003. Proceedings.* IEEE, 38–48.

[21] Bin Dai and David P. Wipf. 2019. Diagnosing and enhancing VAE models. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[22] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 248–255.

[23] Taylor Denouden, Rick Salay, Krzysztof Czarnecki, Vahdat Abdelzad, Buu Phan, and Sachin Vernekar. 2018. Improving reconstruction autoencoder out-of-distribution detection with Mahalanobis distance. *CoRR* abs/1812.02765 (2018). arXiv:1812.02765.

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

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

[26] Charles Fefferman, Sanjoy Mitter, and Hariharan Narayanan. 2016. Testing the manifold hypothesis. *Journal of the American Mathematical Society* 29, 4 (2016), 983–1049.

[27] Abel G. 2017. MNIST-Fashion-CNN. https://github.com/abelusha/MNIST-Fashion-CNN.
[28] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
[29] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex Smola. 2006. A kernel method for the two-sample-problem. *Advances in Neural Information Processing Systems* 19 (2006), 513–520.
[30] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. 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*. 739–743.
[31] Fitash Ul Haq, Donghwan Shin, Lionel C. Briand, Thomas Stifter, and Jun Wang. 2021. 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*. 91–102.
[32] Fabrice Harel-Canada, Lingxiao Wang, Muhammad Ali Gulzar, Quanquan Gu, and Miryung Kim. 2020. 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*. 851–862.
[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 770–778.
[34] Dan Hendrycks and Kevin Gimpel. 2017. A baseline for detecting misclassified and out-of-distribution examples in neural networks. *Proceedings of International Conference on Learning Representations* (2017).
[35] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. 2019. Deep anomaly detection with outlier exposure. *Proceedings of the International Conference on Learning Representations* (2019).
[36] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2017. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *Advances in Neural Information Processing Systems*, Vol. 30.
[37] Irina Higgins, Loïc Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. 2017. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *5th International Conference on Learning Representations, ICLR*.
[38] C. Michael Holloway. 2012. Towards understanding the DO-178C/ED-12C assurance case. In *7th IET International Conference on System Safety, Incorporating the Cyber Security Conference 2012*. IET, 1–6.
[39] Qiang Hu, Lei Ma, Xiaofei Xie, Bing Yu, Yang Liu, and Jianjun Zhao. 2019. DeepMutation++: A mutation testing framework for deep learning systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1158–1161.
[40] Xiaowei Huang, Daniel Kroening, Wenjie Ruan, James Sharp, Youcheng Sun, Emese Thamo, Min Wu, and Xinping Yi. 2020. A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability. *Computer Science Review* 37 (2020), 100270.
[41] Yichao Huang, Xiaorui Liu, Lianwen Jin, and Xin Zhang. 2015. DeepFinger: A cascade convolutional neuron network approach to finger key point detection in egocentric vision with mobile camera. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE, 2944–2949.
[42] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 1110–1121.
[43] Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. 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*. 67–78.
[44] Rateb Jabbar, Khalifa Al-Khalifa, Mohamed Kharbeche, Wael Alhajyaseen, Mohsen Jafari, and Shan Jiang. 2018. Real-time driver drowsiness detection for Android application using deep neural networks techniques. *Procedia Computer Science* 130 (2018), 400–407.
[45] Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 4401–4410.
[46] Hyunjik Kim and Andriy Mnih. 2018. Disentangling by factorising. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*. PMLR.
[47] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 1039–1049.
[48] Jinhan Kim, Jeongil Ju, Robert Feldt, and Shin Yoo. 2020. 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*. 1466–1476.
[49] Seah Kim and Shin Yoo. 2020. Evaluating surprise adequacy for question answering. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. 197–202.

[50] Diederik P. Kingma and Max Welling. 2014. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14–16, 2014, Conference Track Proceedings*.
[51] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. Learning multiple layers of features from tiny images. (2009).
[52] D. Richard Kuhn, Itzel Dominguez Mendoza, Raghu N. Kacker, and Yu Lei. 2013. Combinatorial coverage measurement concepts and applications. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*. IEEE, 352–361.
[53] D. Richard Kuhn, Dolores R. Wallace, and Albert M. Gallo. 2004. Software fault interactions and implications for software testing. *IEEE Transactions on Software Engineering* 30, 6 (2004), 418–421.
[54] H. O. Lancaster. 1969. *The Chi-squared Distribution*. Wiley, New York (1969).
[55] Anders Boesen Lindbo Larsen, Søren Kaae Sønderby, Hugo Larochelle, and Ole Winther. 2016. Autoencoding beyond pixels using a learned similarity metric. In *Proceedings of the 33rd International Conference on Machine Learning*, Vol. 48. PMLR, 1558–1566.
[56] Yann LeCun. 1998. The MNIST database of handwritten digits. http://yann.lecun.com/exdb/mnist/.
[57] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.
[58] Zenan Li, Xiaoxing Ma, Chang Xu, and Chun Cao. 2019. 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, 89–92.
[59] Shiyu Liang, Yixuan Li, and R. Srikant. 2018. Enhancing the reliability of out-of-distribution image detection in neural networks. In *International Conference on Learning Representations*.
[60] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. 2015. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.
[61] Francesco Locatello, Stefan Bauer, Mario Lucic, Gunnar Raetsch, Sylvain Gelly, Bernhard Schölkopf, and Olivier Bachem. 2019. Challenging common assumptions in the unsupervised learning of disentangled representations. In *International Conference on Machine Learning*. PMLR, 4114–4124.
[62] David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision* 60, 2 (2004), 91–110.
[63] Lei Ma, Felix Juefei-Xu, Minhui Xue, Bo Li, Li Li, Yang Liu, and Jianjun Zhao. 2019. DeepCT: Tomographic combinatorial testing for deep learning systems. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 614–618.
[64] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 120–131.
[65] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. DeepMutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.
[66] Alireza Makhzani, Jonathon Shlens, Navdeep Jaitly, and Ian Goodfellow. 2016. Adversarial autoencoders. In *International Conference on Learning Representations*.
[67] Loic Matthey, Irina Higgins, Demis Hassabis, and Alexander Lerchner. 2017. dSprites: Disentanglement testing Sprites dataset. https://github.com/deepmind/dsprites-dataset/.
[68] Joshua R. Maximoff, D. Richard Kuhn, Michael D. Trela, and Raghu Kacker. 2010. A method for analyzing system state-space coverage within a t-wise testing framework. In *2010 IEEE International Systems Conference*. IEEE, 598–603.
[69] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. 2011. Reading digits in natural images with unsupervised feature learning. (2011).
[70] Augustus Odena, Catherine Olsson, David Andersen, and Ian Goodfellow. 2019. TensorFuzz: Debugging neural networks with coverage-guided fuzzing. In *International Conference on Machine Learning*. 4901–4911.
[71] Thomas J. Ostrand and Marc J. Balcer. 1988. The category-partition method for specifying and generating functional tests. *Commun. ACM* 31, 6 (1988), 676–686.
[72] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in Pytorch. (2017).
[73] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles*. 1–18.
[74] Kashif Rasul. 2017. Fashion-MNIST-CNN. https://gist.github.com/kashif/76792939dd6f473b7404474989cb62a8.
[75] Jie Ren, Peter J. Liu, Emily Fertig, Jasper Snoek, Ryan Poplin, Mark Depristo, Joshua Dillon, and Balaji Lakshminarayanan. 2019. Likelihood ratios for out-of-distribution detection. In *Advances in Neural Information Processing Systems*, Vol. 32.

[76] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. 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, 355–367.

[77] Joan Serrà, David Álvarez, Vicenç Gómez, Olga Slizovskaia, José F. Núñez, and Jordi Luque. 2019. Input complexity and out-of-distribution detection with likelihood-based generative models. *CoRR* abs/1909.11480 (2019). arXiv:1909.11480.

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

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

[80] Elena Sherman, Matthew B. Dwyer, and Sebastian Elbaum. 2009. Saturation-based testing of concurrent programs. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*. 53–62.

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

[82] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. 2014. Striving for simplicity: The all convolutional net. (2014). arXiv:1412.6806.

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

[84] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic testing for deep neural networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 109–119.

[85] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering*. 303–314.

[86] Arash Vahdat, Karsten Kreis, and Jan Kautz. 2021. Score-based generative modeling in latent space. (2021). arXiv:2106.05931.

[87] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. 2004. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing* 13, 4 (2004), 600–612.

[88] Michael Weiss, Rwiddhi Chakraborty, and Paolo Tonella. 2021. 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, 17–24.

[89] Elaine J. Weyuker. 1988. The evaluation of program-based software test data adequacy criteria. *Commun. ACM* 31, 6 (1988), 668–675.

[90] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 408–426.

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

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

[93] Zhisheng Xiao, Qing Yan, and Yali Amit. 2020. Likelihood Regret: An out-of-distribution detection score for variational auto-encoder. In *Advances in Neural Information Processing Systems*, Vol. 33.

[94] Shenao Yan, Guanhong Tao, Xuwei Liu, Juan Zhai, Shiqing Ma, Lei Xu, and Xiangyu Zhang. 2020. 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*. 775–787.

[95] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. 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*. 79–90.

Received 25 January 2022; revised 20 October 2022; accepted 23 November 2022