CIT4DNN: Generating Diverse and Rare Inputs for Neural Networks Using Latent Space Combinatorial Testing

Swaroopa Dola
University of Virginia
Charlottesville, Virginia, USA
sd4tx@virginia.edu

Rory McDaniel
University of Virginia
Charlottesville, Virginia, USA
rory@virginia.edu

Matthew B. Dwyer
University of Virginia
Charlottesville, Virginia, USA
matthewbdwyer@virginia.edu

Mary Lou Soffa
University of Virginia
Charlottesville, Virginia, USA
soffa@virginia.edu

ABSTRACT

Deep neural networks (DNN) are being used in a wide range of applications including safety-critical systems. Several DNN test generation approaches have been proposed to generate fault-revealing test inputs. However, the existing test generation approaches do not systematically cover the input data distribution to test DNNs with diverse inputs, and none of the approaches investigate the relationship between rare inputs and faults. We propose CIT4DNN, an automated black-box approach to generate DNN test sets that are feature-diverse and that comprise rare inputs. CIT4DNN constructs diverse test sets by applying combinatorial interaction testing to the latent space of generative models and formulates constraints over the geometry of the latent space to generate rare and fault-revealing test inputs. Evaluation on a range of datasets and models shows that CIT4DNN generated tests are more feature diverse than the state-of-the-art, and can target rare fault-revealing testing inputs more effectively than existing methods.

CCS CONCEPTS

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

KEYWORDS

deep neural networks, test generation, test coverage, combinatorial interaction testing

ACM Reference Format:
Swaroopa Dola, Rory McDaniel, Matthew B. Dwyer, and Mary Lou Soffa. 2024. CIT4DNN: Generating Diverse and Rare Inputs for Neural Networks Using Latent Space Combinatorial Testing. In *2024 IEEE/ACM 46th International Conference on Software Engineering (ICSE '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3597503.3639106

ICSE '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0217-4/24/04.
https://doi.org/10.1145/3597503.3639106

1 INTRODUCTION

Deep Neural Networks (DNN) are being developed for use in mission and safety critical systems, e.g., [6, 36, 61]. Similar to traditional programmed software components, these learned components require significant testing to ensure that they are fit for deployment.

DNNs learn from observed data that comprises multiple factors of variation – referred to as *features*, combinations of which give rise to the *diversity* in the data [7, 13, 17, 31, 38]. For example, consider an input domain including handwritten zeros, for which stroke thickness and slant are two features. A combination of these two features results in diverse samples with varying stroke thickness and slant as shown in Figure 1. A study conducted on the real faults of deep learning systems identified that DNNs make incorrect predictions when they process inputs that are underrepresented in the training data [32, 70, 71]. Finding these faults requires methods that *adequately test DNNs with diverse inputs* that are representative of the feature combinations of the observed data distribution.

Feature combinations occur with varying probabilities in the observed data, and the inputs with a low probability of occurrence are referred to as *rare inputs*. Failing to test the system behavior for rare inputs can make deep learning systems unsafe for real-world deployment. For example, in a fatal Tesla crash, the Autopilot was not able to detect the white side of a tractor-trailer against a brightly lit sky and a GM autonomous vehicle crashed into a bus in this rare circumstance [2, 5]. There is a need for methods that *adequately test DNNs with rare inputs* to ensure the safety and trustworthiness of deep learning systems.

Much of the prior work in neural network test input generation has focused on fault detection and has targeted neither diverse nor rare inputs. DNN test generation methods that apply pixel-level manipulations on seed images to generate test inputs, e.g., [29, 49, 57, 60, 67], do not yield feature-diverse inputs [26, 70]. More recent work has targeted the generation of diverse test inputs using two approaches. First, tests can be generated using a manually constructed model of features and their interactions, but this does not scale to complex datasets [16, 27, 70]. Second, tests can be generated using feature space models that are trained from observed data [14, 15, 35, 68], but these provide no information about the degree to which test inputs cover that feature space. While, in principle, some of these techniques, e.g., [14, 35], could generate rare inputs through rejection sampling approaches, i.e., by generating test inputs first and then rejecting the ones that are not

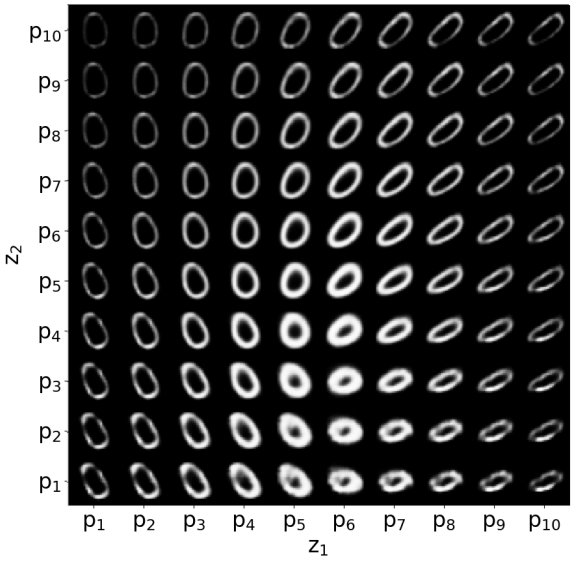

Figure 1: Diversity of digit 0 with variation in the slant and stroke thickness.

rare, this would be cost-prohibitive for complex datasets. Moreover, such approaches cannot systematically explore a subspace of the observed data distribution that comprises the *target density* of rare inputs.

In this research, we present a black-box test generation algorithm, CIT4DNN, that addresses the limitations of the existing methods discussed above. The research objective of CIT4DNN is to automatically generate diverse and rare inputs that systematically cover a target density of the feature space of the observed data distribution.

CIT4DNN meets the research objectives by adapting ideas from recently proposed work on *input distribution coverage* (IDC) [26]. IDC uses a generative model to automatically learn a low-dimensional representation of the features of the training data, called the *latent space*. IDC partitions the latent space and applies combinatorial interaction testing (CIT) [20] on the partitioned latent space to measure test coverage proportional to the t-way feature combinations present in a test set. While IDC measures test coverage, CIT4DNN applies CIT on the partitioned latent space to generate covering arrays containing combinations of partitions, called *test descriptions* from which test inputs can be generated. The test descriptions cover all the t-way feature combinations resulting in test adequacy with respect to the diversity of the observed data distribution.

Rare inputs are spread out on the low-probability regions of the latent space. To support testing with rare inputs, CIT4DNN formulates constraints on the latent space and presents a constrained CIT algorithm to generate test descriptions that belong to the required target density.

The test descriptions generated by CIT4DNN are dependent only on the dimensionality of the latent space. As a result, the test descriptions can be reused for DNNs trained on different datasets as long as the dimensionality of the latent space of the generative models of the datasets are the same. This results in test description generation time being amortized across testing many different DNNs under test. In §3 we describe how permuting test descriptions can further increase test diversity with low overhead.

An evaluation of CIT4DNN, in §4, shows that CIT4DNN yields higher feature diversity than the state-of-the-art DNN testing approach DeepHyperion-CS [70]. In comparison to the state-of-the-art generative model-based testing approach SINVAD [35], CIT4DNN yields 9 times more coverage and detects 6.5 times more faults while running 90 times faster. For rare test input generation, CIT4DNN improves on SINVAD 107-fold for coverage and 111-fold for fault-detection, and in comparison to random sampling, CIT4DNN improves on coverage while generating 6 orders of magnitude fewer test inputs and running 56 times faster.

The contributions of this work include:

(1) CIT4DNN– a black-box DNN test generator capable of automatically producing diverse and rare test inputs while achieving 100% IDC test adequacy;
(2) a constraint-based method to generate rare inputs without costly rejection sampling;
(3) a method for applying CIT to the latent space of a generative model that enables reusing test descriptions to generate inputs with increasing diversity; and
(4) the results of an evaluation across a range of datasets, DNNs, and instantiations of CIT4DNN that demonstrates its beneficial characteristics.

2 BACKGROUND

2.1 Deep Generative Models

The latent space is a low-dimensional embedding that represents the factors of variation comprising the observed data [7, 40]. Machine Learning research has shown that deep generative models are effective in learning the latent space of real-world datasets [13, 17, 28, 38, 39]. We use a deep generative model called variational autoenoder in this work [39].

A variational autoencoder (VAE) is comprised of a pair of networks – an encoder, $V_{\mathcal{E}}$, and a decoder, $V_{\mathcal{D}}$ – that is trained to accurately reconstruct inputs from the data distribution, $\mathcal{X}$, with the training objective, $\min_{\boldsymbol{x}\in\mathcal{X}}\|\boldsymbol{x} - V_{\mathcal{D}}(V_{\mathcal{E}}(\boldsymbol{x}))\|$ [39]. $V_{\mathcal{E}}$ converts samples from the data into vectors in the latent space whereas $V_{\mathcal{D}}$ is a generator that converts samples from the latent space into inputs in the data space. The latent space of a VAE defines the parameters of a multivariate distribution of size equal to the dimension of the latent space. A loss term in training biases that distribution to match an assumed prior, commonly a standard Normal distribution. While early VAEs were known to suffer in the quality of their reconstructions, modern VAEs exceed the performance of many other generative models [44].

2.2 Combinatorial Interaction Testing

Thorough black-box testing of software with a high-dimensional input space is challenging. Applying methods, such as category-partitioning [47], to construct a finite partition for each input helps to a degree, but the combinatorics generally preclude complete coverage of input-partition combinations. Combinatorial interaction testing (CIT) is a method to generate test suites that systematically cover a partitioned input space up to a user-specified arity, t, which is referred to as *combinatorial strength* [20]. CIT methods have been

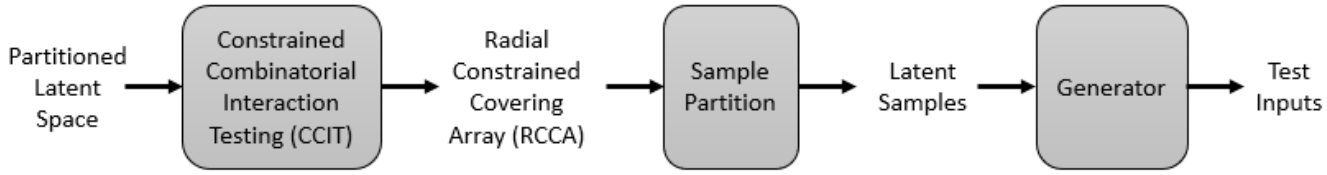

Figure 2: Overview of CIT4DNN.

used to both generate test suites that achieve a desired strength [20] and to measure the strength of a given test suite to define coverage criteria [41].

Central to CIT is the notion of a *covering array* (CA), a matrix with a column for each input where the cells hold the appropriate partition values for the input. A row constitutes a test description defining a possible set of values, defined by the partitions, for each input. A t-way CA includes every possible combination of input-partition pairs in some row and thereby assures that any interaction among inputs up to t has a chance to be exposed by tests generated from the CA. For example, Figure 1 shows inputs $z_1, z_2 \in [-6, 6]$ partitioned into 10 intervals $\{p_1, p_2, .., p_{10}\}$ representing two features of the handwritten digit zero. A 1-way CA for this configuration would have 10 test descriptions, $\{(p_i, p_i) : i \in [1, 10]\}$, that cover the diagonal inputs, whereas a 2-way CA for this results in 100 test descriptions, $\{(p_i, p_j) : i, j \in [1, 10]\}$, that cover all the inputs shown in the Figure 1.

In many systems, inputs are not completely independent, requiring that CIT methods take into account constraints over input-partition combinations [62]. Cohen et al. [21] defined a general framework that permits propositional constraints over such combinations to be incorporated into greedy CIT test input generation methods [20]. The resulting constrained CIT (CCIT) approach efficiently generates a *constrained covering array* (CCA), which is a CA whose rows are consistent with the constraints.

2.3 Input Distribution Coverage

A CIT coverage metric, called *input distribution coverage* (IDC), has been defined for DNN testing [26]. IDC's coverage domain is the latent space of a trained VAE with a standard Normal prior, $\mathcal{N}(0, 1)$. IDC defines an *equal density partition*, $\mathcal{P}$, of $\mathcal{N}(0, 1)$ as a set of intervals that each contain the same probability density. Choosing the size of the partition, $|\mathcal{P}|$, allows the set of intervals to be computed using the quantile function for $\mathcal{N}(0, 1)$.

IDC allows coverage to be computed over the portion of the latent space defined by a user-defined *target density*, d. This density defines a shell with inner, r_i, and outer, r_o, radii. IDC maps a test input to the latent space and then converts it to a hyper-rectangle defined as the product of the elements of $\mathcal{P}$ corresponding to the latent coordinates. The total t-way coverage metric measures how well a test set exercises combinations of partitions by accumulating the combinations present in the hyper-rectangles derived from the test inputs. Test suites with higher t-way coverage were shown to be more feature diverse, as judged against human-defined ground truth, and improve fault-detection effectiveness [26].

A key challenge in IDC is to compute the ratio of the count of t-way combinations in a test set to the *feasible t-way feature combinations*, which allows coverage to be reported as a percentage. To calculate this quantity, it is necessary to account for the fact that a hyper-rectangle defined by the product of partitions may not overlap with the target density shell, in which case it is infeasible. A step in this calculation involves checking a quadratic distance constraint: $\exists c \in \langle p_1, \ldots, p_k \rangle : ||c|| \in [r_i, r_o]$, where c is a latent space coordinate that is consistent with the $p_i \in \mathcal{P}$ intervals. To side-step the nonlinearity of this constraint, IDC uses an efficient SMT-encoding that expresses the constraints over the *squared latent coordinates* which yields a linear constraint that is efficient to solve.

Whereas IDC only measures coverage, CIT4DNN is the first DNN test generation approach to guarantee t-way coverage of a generated test suite. We describe how CIT4DNN builds on the concepts introduced in IDC in the next section.

3 APPROACH

The goal of CIT4DNN is to systematically generate diverse and rare inputs from a target density of the observed data distribution – Figure 2 sketches the components of CIT4DNN. CIT4DNN uses a generative model with a standard Normal prior, $\mathcal{N}(0, 1)$, to learn a representation of the features of the observed data in the latent space. CIT4DNN leverages the properties of $\mathcal{N}(0, 1)$ to partition the latent space into equal-density partitions and parameterizes the geometry of the latent space to represent a target density using radial constraints. The *Constrained Combinatorial Interaction Testing* (CCIT) module shown in Figure 2 uses the partitioned latent space and radial constraints that express the target density to generate a *Radial Constrained Covering Array* (RCCA) comprising the *test descriptions* representing partition combinations on the required target density. The *Sample Partition* module converts the RCCA into a set of latent samples, such that each sample belongs to the corresponding test description of the RCCA. The *Generator* module converts latent samples into test inputs, and, by construction, the generated test inputs achieve 100% t-way combinatorial coverage of the target density in the latent space.

3.1 Radial Constrained Covering Arrays

The latent space with a multivariate standard Normal prior has its probability concentrated in an annulus [9], and the probability density along the radial dimension of the annulus is described by a Chi distribution. Hence a target density can be converted to a pair of radii using the interval function of the Chi distribution [26], where the interval is the smallest one containing the target density. These radial bounds are used to specify the constraints for the CCIT and

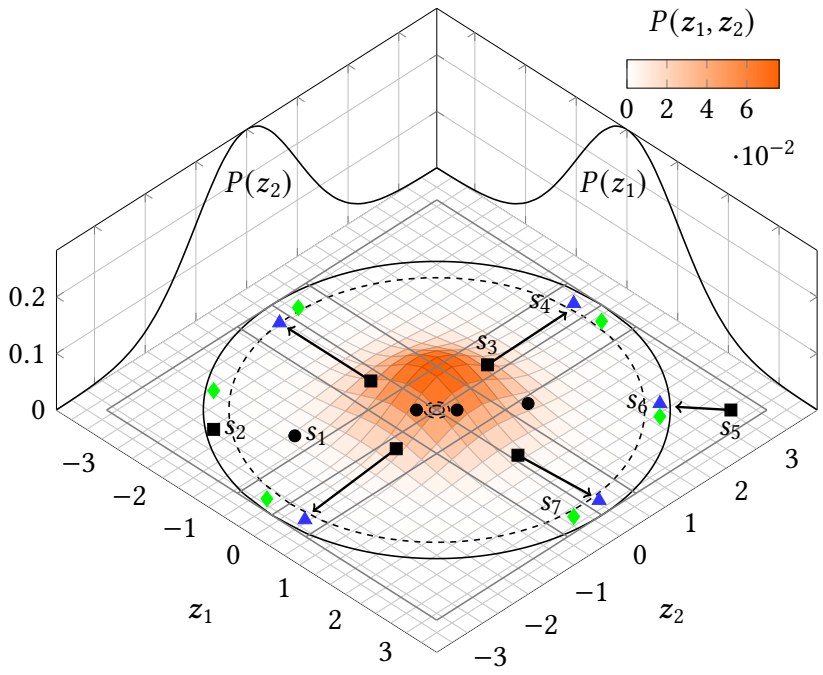

Figure 3: Depiction of CIT4DNN applied to a 2-dimensional latent space partitioned 4-ways ($\mathcal{P}$ = $\{[-3.25, -0.67), [-0.67, 0), [0, 0.67), [0.67, 3.25]\}$).

the resulting covering array is referred to as a *Radial Constrained Covering Array* (RCCA).

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

The properties of RCCA ensure that CIT4DNN systematically covers a target density of the feature space to test DNNs with diverse and rare inputs; CIT4DNN generates rare inputs when target density represents a low-probability region of the latent space.

RCCA differ from existing CCA in two ways. First, each of the columns of the array can take on the same set of values, $\mathcal{P}$. Second, they employ the quadratic distance constraints from IDC to model feasibility relative to a target density – specifically, we use the checkConstraint function from Alg. 1 [26]. We adapted the greedy AETG-SAT covering array generation algorithm from [21] with this modification to generate RCCA.

Example 3.1. Figure 3 depicts a 2-dimensional latent space – the surface comprised of light-gray grid lines with densities shown as intensities of orange and marginal distributions for each dimension shown separately, $P(z_i)$. The gray rectangular regions on the plane show the 4-way partitioned latent dimensions within an outer target density of 0.99 (solid circle). The four black circles, e.g., s_1, depict samples from a 1-way RCCA that covers each dimension partition.

Restricting the target density to the range $[0.97, 0.99]$ – the shell between the dashed and solid circles – requires a larger RCCA. The black square coordinates depict samples, e.g., s_2, from a 6-row RCCA. Some of these samples, e.g., s_3, s_5, fall outside of the shell and as a result, they are infeasible with respect to the target density. These samples are projected (arrows) to the blue triangle coordinates, s_4, s_6, using the SAMPLE-PARTITION method discussed in Alg. 1 thereby producing latent samples on the target density.

3.1.1 Reusing RCCA. CIT4DNN is designed to be applicable to any input domain for which a high-quality generator with a standard Normal prior can be trained, but it is largely independent of the learned mapping between the domain and the generator's latent space, i.e., CCIT and Sample Partition modules in Figure 2 are independent of the Generator module. Instead, it relies on the dimension of the latent space and the fact that the latent distribution is a close match to an isotropic standard normal prior.

Whereas prior CIT work focuses mostly on 2-way combinatorial strength [58], for DNN test generation, 3-way coverage yields more diverse test suites [26]. This presents a challenge since the cost of generating CAs grows combinatorially with t. Here we can leverage the fact that a t-way RCCA for a k-dimensional latent space partitioned p ways in a target density range $[d_i, d_o]$ is independent of the VAE's encoder and decoder. Thus, an RCCA for an input domain modeled by a k-dimensional latent space can be pre-computed and reused, saving time.

Moreover, the isotropic latent space means that the column-wise permutation of an RCCA yields an RCCA that, with high probability, has distinct rows.

Proposition 3.2 (RCCA Permutation Closure). *The set of RCCA$(t, k, \mathcal{P}, [r_i, r_o])$ is closed under column-wise permutation.*

Figure 4: Test inputs generated by RCCA permutation.

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

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

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

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

In general, there are $k!$ possible permutations, but repeated partition values in rows reduce the number of distinct permutations. If there are c repeated values, the number of distinct permutations is reduced by $c! - 1$, but the probability of c repeated values is $\frac{1}{p}^{c-1}$ which indicates that the probability of identical permutations is very low. Thus, reusing and permuting RCCA columns effectively guarantees distinct test descriptions in time that is linear in k – the time to permute column indices.

Example 3.3. In Figure 3, a second sample for the $[0.97, 0.99]$ shell would permute RCCA columns to generate the green diamond coordinates which generates additional diverse tests. For example, the coordinate $s_4 = (-0.3, 3.0)$ when permuted yields $s_7 = (3.0, -0.3)$.

To illustrate the diversity of column permutation, we randomly selected a row of a 3-way covering array for MNIST with $k = 9$, $p = 20$, $d_i = 0.9999$, and $d_o = 0.999999$. Figure 4 shows the images generated from 8 random permutations of this row and shows the diversity possible from column permutation. Only two columns share a common partition value meaning that there are $k!-(c!-1) = 362879$ distinct column permutations.

3.2 CIT4DNN Algorithm

Alg. 1 defines CIT4DNN as a pair of functions: CIT4DNN and SAMPLE-PARTITION. The entry point is the function CIT4DNN which first extracts the latent dimension and computes their partition – lines 2-3. Lines 4-11 employ *RCCA* to compute the covering array handling two cases. The simpler case is where d_i is zero and a single covering array is constructed – line 10 – based on the radii calculated for d_o – line 4. The more complex case – lines 6-8 – deals with the case where an inner target density is defined. Here a pair of covering arrays are constructed – lines 7-8 – where one uses the inner radii for d_i and d_o and the other uses the outer radii to define the shells used to compute *RCCA*. We require that $d_i < d_o$ which means that their corresponding shells – whose radii are computed on lines 4 and 6 – are concentric, requiring that the roles of the radii be transposed when computing the inner covering array – line 7. Lines 12-18 generate n samples from each row of CA. To yield more diverse test samples, we implement proposition 3.2 and generate n column-wise permutations of the covering array – line 14 – and draw a single sample from each row of each permutation – line 16. Since rows of the covering array(s) may be associated with different radii, we record radii along with each row to generate latent samples – line 15. The set of generated latent samples, S, is decoded to generate a set of test inputs – line 19.

Algorithm 1 Latent Space CIT Sampling for DNN Test Generation

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

1: **function** CIT4DNN(G, d_i, d_o, t, p, n)
2: $k \leftarrow dim(G)$
3: $\mathcal{P} \leftarrow$ PARTITION-EQ-DENSITY(p, d_o)
4: $r_i, r_o \leftarrow Chi.interval(d_o, k)$
5: **if** $d_i > 0$ **then** ▷ Construct two shells
6: $r_i', r_o' \leftarrow Chi.interval(d_i, k)$
7: $CA \leftarrow \{(c, r_i, r_i') : c \in RCCA(t, k, \mathcal{P}, [r_i, r_i'])\}$
8: $CA \leftarrow CA \cup \{(c, r_o', r_o) : c \in RCCA(t, k, \mathcal{P}, [r_o', r_o])\}$
9: **else** ▷ Construct one shell
10: $CA \leftarrow \{(c, r_i, r_o) : c \in RCCA(t, k, \mathcal{P}, [r_i, r_o])\}$
11: **end if**
12: $S \leftarrow \emptyset$
13: **for** $i \in [1, n]$ **do**
14: $CA \leftarrow$ PERMUTE-COLUMNS(CA)
15: **for** $(c, r_i, r_o) \in CA$ **do**
16: $S \leftarrow S \cup$ SAMPLE-PARTITION(c, r_i, r_o)
17: **end for**
18: **end for**
19: **return** $\{G(s) : s \in S\}$ ▷ Decode latent samples
20: **end function**
21: **function** SAMPLE-PARTITION(c, r_i, r_o)
22: $s \leftarrow \langle s_1, \ldots, s_{|c|} \rangle : s_i \sim [l_i, u_i] \wedge (l_i, u_i) = c_i$
23: **if** $\|s\| < r_i \vee \|s\| > r_o$ **then**
24: $q \leftarrow \langle q_1, \ldots, q_{|c|} \rangle : q_i = ite(s_1 < 0, -1, 1)$
25: $vs \leftarrow$ GET-FRESH($|c|$)
26: $\psi \leftarrow true$
27: **for** $i \in [1, |c|]$ **do**
28: $(l, u) \leftarrow c_i$
29: $\psi \leftarrow \psi \wedge vs[i] \geq ite(u > 0, l^2, u^2)$ ▷ Lower bound
30: $\psi \leftarrow \psi \wedge vs[i] < ite(u > 0, u^2, l^2)$ ▷ Upper bound
31: **end for**
32: $\psi \leftarrow \psi \wedge (\sum_{i \in [1,|c|]} vs[i]) \geq (r_i)^2$
33: $\psi \leftarrow \psi \wedge (\sum_{i \in [1,|c|]} vs[i]) \leq (r_o)^2$
34: $m \leftarrow$ GET-MODEL(ψ)
35: $s \leftarrow \langle \text{SQRT}(m_1) \cdot q_1, \ldots, \text{SQRT}(m_{|c|}) \cdot q_{|c|} \rangle$
36: **end if**
37: **return** s
38: **end function**

Example 3.4. A call to CIT4DNN($\mathcal{G}$, 0.97, 0.99, 1, 4, 1), with a 2-dimensional latent space, is depicted in Figure 3. The 4-way, equal-density partition is shown as $\mathcal{P}$ along the back z_2 axis. The pairs of interval partitions subdivide the plane into 16 rectangles. To simplify the example, we consider just the case of the outer radii computed on lines 4 and 6 which are 3.255 and 2.898, respectively. The call to *RCCA* on line 8 results in: $\{(p_1, p_1), (p_2, p_4), (p_3, p_1), (p_1, p_2), (p_4, p_3), (p_4, p_4)\}$, which covers each partition in each dimension. Six rows are required because the four inner rectangles, e.g., (p_2, p_2), do not intersect the target outer density shell and are thus infeasible.

The SAMPLE-PARTITION function samples coordinates, s, using the density associated with the partition interval, $[l_i, u_i]$, for each dimension, i – line 22. When inner and outer densities target a rare portion of the input distribution a sample within the partition interval may not lie between the target radii – line 23. This can be observed in Figure 3 where a small arc of the $[0.97, 0.99]$ shell intersects with any of the rectangular partition combinations. Lines 24-35 compute a sample that satisfies the partition and radial constraints using SMT. We build on the squared-distance constraint formulation used in the CHECKCONSTRAINT function from Alg. 1 [26], which works because distance constraints are insensitive to the orthant[1] within which a partition lies. We record a sample's orthant, q, as a vector that holds the polarity of each sample coordinate – line 24. Lines 26-33 compute squared-distance constraints and then solve them for a model in line 34. The use of radial constraints in *RCCA* generation guarantees these constraints are solvable. The model generated holds the squared coordinates and lies in the positive orthant. Line 35 recovers the coordinates and maps them to the recorded orthant and updates the sample s.

Example 3.5. Sampling from row (p_2, p_4) results in coordinate s_3 in Figure 3. The radius of this coordinate falls short of 2.898, the dotted circle. The orthant is recorded, $q = \langle -1, 1 \rangle$, and the following squared radial constraint is formulated:

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

where $vs[1] = (z_1)^2$ and $vs[2] = (z_2)^2$. Solving the constraint yields a satisfying model where $vs[1] = 0.04$ and $vs[2] = 9$. Taking the square root and recovering the orthant yields the blue coordinate $s_4 = (-0.2, 3)$. A similar calculation projects s_5 to the blue triangle s_6 in the target shell. Permuting the columns of the covering array to draw a second sample for s_4 results in the row (p_4, p_2) which is sampled and solved to yield the green diamond s_7.

4 EVALUATION

We designed a set of experiments to explore the effectiveness of CIT4DNN in generating realistic, feature-diverse, and rare inputs for testing DNNs by exploring a series of research questions:

RQ1: How realistic are tests generated by CIT4DNN?
RQ2: How effective is CIT4DNN in generating feature-diverse tests?
RQ3: How does fault density vary with the latent distribution?
RQ4: How cost-effective is CIT4DNN in targeting normal and rare inputs?

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

Dataset	Architecture	#Parameters	Metric	Value
MNIST	LeNet-4 [43]	69362	Accuracy	99.05%
	LeNet-5 [43]	107786	Accuracy	98.53%
Fashion	Custom [51]	1.6M	Accuracy	93.58%
	Custom [64]	3.3M	Accuracy	92.26%
SVHN	ALL-CNN-A [55]	1.2M	Accuracy	96%
	ALL-CNN-B [55]	1.3M	Accuracy	95.67%
TaxiNet	Taxi1 [63]	650	MSE	63.31
	Custom	794	MSE	37.79
Udacity	Dave-2 [10]	2.8M	MSE	0.014
	Epoch [1]	104.9M	MSE	0.016

Table 1: Models used in our studies with number of parameters, test accuracy or MSE (Mean Squared Error); "M" denotes millions of parameters.

4.1 Experimental Setup

Three Classification datasets, MNIST [42], FashionMNIST [65], SVHN [46], and two regression datasets, TaxiNet [34] and Udacity [33], are used in the studies. MNIST and SVHN are selected as they are used in the experimental studies of the baselines of our work, DeepHyperion-CS [71] and SINVAD [35]. FashionMNIST, TaxiNet, and Udacity are considered as they represent domains different from that of MNIST and SVHN. For each of the datasets, we train VAEs for instantiating CIT4DNN as shown in Table 2 and two DNN models as shown in Table 1 to study fault-revealing test inputs in RQ3 and RQ4.

4.1.1 Datasets and DNN Models. MNIST contains greyscale images of handwritten digits; it has 60k training inputs and 10k test inputs. We train LeNet-4 and LeNet-5 DNNs for this dataset [43]. FashionMNIST contains 28x28 greyscale images of fashion products belonging to 10 categories. We use the FashionMNIST networks used in the IDC work [26]. SVHN contains 32x32 color images of digits in natural scenes; it has 73257 training inputs and 26032 test inputs. We train All-CNN-A and All-CNN-B networks for this dataset[55].

TaxiNet contains aircraft runway images with 16x32 resolution with cross-track position and heading angle for each. The TaxiNet dataset has 80k training and 20k test inputs. We use the network from an open-sourced artifact [63] as one of the models for this dataset. Since a second model for TaxiNet is not available, we developed a custom model by adding two extra fully connected layers to the first model and using ELU activation in one of the layers. We used the same training hyperparameters for training both models. The Udacity dataset is a self-driving car dataset generated in a simulation environment as open-sourced by DeepCrime [33]. This dataset has 9800 training inputs and 2451 test inputs where each input is a 160x320 color image. We train two models, NVIDIA's Dave-2 [10, 33] and Epoch [1] to output steering angles for the inputs.

4.1.2 Test Oracles. CIT4DNN uses a differential test oracle similar to DeepXplore [49] for identifying fault-revealing test inputs. For classification datasets, the test oracle fails when the two DNN models trained on the same dataset predict different classes. For the

regression datasets, we use the steering angle outputs of DNNs for Udacity and the heading angle outputs of the DNNs for TaxiNet for formulating the test oracle. The test oracle fails when the outputs of the two models have different signs, and the difference in their predictions is greater than 5% of the output range of the test dataset. There are other test oracles proposed in the literature which we plan to study in our future work [22, 56, 59, 66].

4.1.3 Quantitative Model Metrics. We use quantitative metrics to measure both how realistic a given test is and how diverse a set of tests is. For a test to be completely *realistic*, a consumer must be unable to discern whether the test came from a generative technique or from the original data. For a test set to be perfectly *diverse*, all possible underlying feature combinations must be expressed in the set. Unfortunately, evaluation of the fidelity of generative models is a difficult problem with significant active research [12].

The three metrics we use to assess our models are FID [30] (which is used in prior works [15, 68]), Coverage, and Density [45]. While prior works have also used Inception Score [15], we contend that this is not an effective metric for this use case - Inception Score does not handle mode collapse well and is improved on by FID [12].

FID, the Fréchet-Inception Distance, is the 2-Wasserstein distance of two distributions taken from a lower-dimensional "embedding" space; the embedding used is Inception v3 [30]. FID is highly variable across implementations - we use the torchmetrics FID implementation on Inception v3's 2048 layer. A critical issue with FID and related single-valued metrics is their inability to differentiate a lack of diversity from other failure-modes. A recent line of work introduces a 2-dimensional metric: *Density and Coverage* [45]. Similar to FID, Density and Coverage use an embedding space, but instead of measuring the distribution distance, they measure how often generated points (in the embedding space) occur near real points, where "near" means within the manifold created by taking the k-nearest-neighbor ball of each point in the original dataset. Intuitively, Coverage is the percentage of real points that have at least one generated point near them. Density follows as the rate at which generated points are near real points. We include these as measures of CIT4DNN's capability to generate realistic inputs in §4. We use the reference implementation[3] for these with K=5 and torchvision's pretrained vgg16 Imagenet model as the embedding. For datasets smaller than 32x32, we upscale by repeating the undersized dimension until the image is >32x32; for single channel datasets we repeat the greyscale channel 3 times.

4.2 CIT4DNN Instantiation

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

Generator CIT4DNN uses a pre-trained Generator, and we use the decoder networks of variational autoencoders (VAE) as the Generator networks in the experiments. Since using higher-quality VAEs is beneficial, we explore the combination of two recent innovations in VAE architecture and training: a two-stage VAE [23], which ensures a better match to the prior, and a σ-VAE [54], which optimizes the balance between loss terms that govern reconstruction accuracy and matching the prior.

We use three VAE architectures in the study: a basic VAE [39], denoted K, a basic VAE trained with optimal variance estimate [54], denoted $K\sigma$, and a Two-Stage VAE [24] trained with optimal variance estimate, denoted 2σ. VAEs for the MNIST dataset are trained using the network configuration used in Burgess et al. [13] and all other VAEs are trained using the InfoGAN [18] network architecture. The network input layers are modified to fit the input sizes of the respective datasets. Each VAE configuration is trained for five datasets (MNIST, FashionMNIST, SVHN, TaxiNet, and Udacity), resulting in the 15 VAE configurations shown in Table 2. We report the non-noise latent dimension, k, for 2σ since using non-noise latent dimensions is recommended by IDC for formulating the test coverage domain [26].

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

CIT Parameters and n We vary p as well as t in RQ2 as that question studies aspects of the diversity of test sets. We fix $t = 3$ for RQ3/RQ4 and $p = 20$ for RQ1/RQ3/RQ4 since these were shown to be good choices for assessing test suite coverage [26]. We use n=1 in all of the research questions except for RQ4, where we also explore the fault-revealing capability of the test sets generated for increasing n.

4.3 Results and Research Questions

4.3.1 ***RQ1:*** *How realistic are tests generated by CIT4DNN?* The inputs generated by the testing techniques should be representative of the input data distribution for testing to be effective [8, 25]. We conduct a study to investigate whether the inputs generated by CIT4DNN are realistic. We show that our selected models generate viable outputs with random sampling; we then show that sampling with CIT4DNN preserves the fidelity of the underlying 2σ VAE.

We compute FID and Coverage scores relative to the full test set for each dataset to select the best VAE, and these values are presented in Table 2. While FID is sensitive to test set size, this is not an issue since we are only comparing within our trained model architectures.

We find that VAEs with optimal variance perform consistently better than those without on both metrics. Additionally, for each metric, 2σ performs better for 4/5 datasets. To reduce the cost of subsequent experiments, we use the 2σ VAE to instantiate CIT4DNN.

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

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

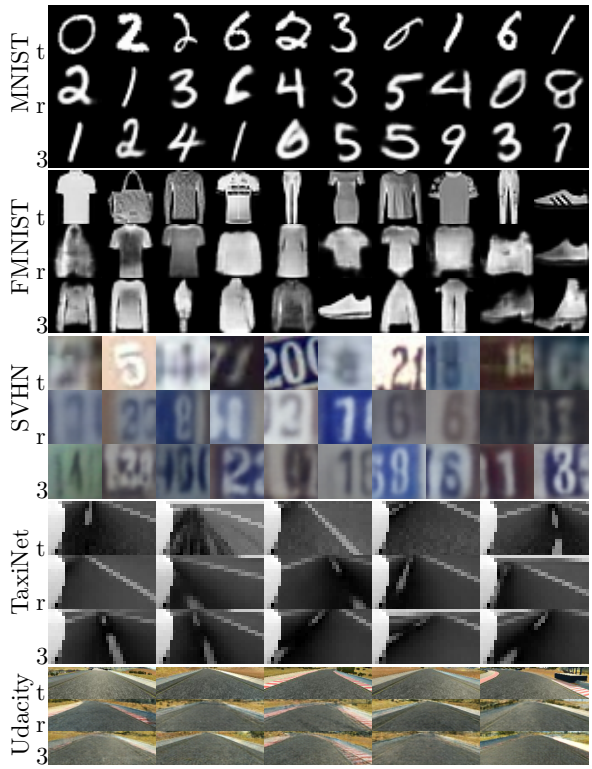

Figure 5: Sample images for each dataset from test set (t), 2σ (r), and CIT4DNN (3).

It is a common practice in the generative model literature to perform a visual study to verify the quality of the generated inputs [11, 54]. Using the same approach, all the co-authors manually checked the random samples generated by the VAEs for visual similarity with their training inputs and any visual anomalies. Figure 5 shows random samples for visual inspection. We also performed a qualitative comparison of randomly generated outputs from each VAE to random test samples. This analysis confirmed the quality of the 2σ; we show random generated samples from this VAE in the **r** rows of Figure 5 and show random test samples in the **t** rows.

We then use Density, Coverage, and FID to assess the impact of selecting tests using CIT4DNN as compared to randomly sampling the VAE. We are limited by the small size of CIT4DNN's t=2 test sets (100% 2-way coverage is achieved with 528-832 tests depending on dataset), so we test with random samples of size 500 from each set and repeat the trails 100 times each. The results are shown in Figure 6.

We find that sampling with CIT4DNN, for either value of t, preserves the realism of the 2σ VAE as measured by Density, Coverage, and FID. We confirmed this with a qualitative analysis. We provide example CIT4DNN tests with $t = 3$ for each dataset in the rows labeled **3** in Figure 5 and compare them with randomly generated tests in the rows labeled **r**.

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

4.3.2 ***RQ2:*** *How effective is CIT4DNN in generating feature-diverse tests?* To provide insight into the diversity of the tests generated by CIT4DNN with respect to human interpretable features, we need ground truth features for the datasets used in our studies (which are unknown). DeepHyperion-CS [70, 71] is a recently published approach that uses human interpretable features for generating diverse tests. Their evaluation shows that DeepHyperion-CS is superior to DeepHyperion [70], DeepJanus [52] and DLFuzz [29] with respect to feature diversity. For these reasons, we use DeepHyperion-CS as a baseline in this study.

DeepHyperion-CS uses human assessors to identify the features of the input data; the experimental studies include MNIST [42] and BeamNG [71] datasets. Of these datasets, DeepHyperion-CS transforms the inputs directly only for MNIST, so we compare CIT4DNN on that dataset. DeepHyperion-CS generates feature maps of the generated test sets and uses the metrics, Filled Cells (FC), and Coverage Sparseness (CS), computed over the feature maps to measure the feature diversity of the test sets. We use these two metrics for comparing the feature diversity of the tests generated by CIT4DNN and DeepHyperion-CS to answer RQ2.

DeepHyperion-CS uses Luminosity (Lum), Moves (Mov), and Orientation (Or) as the features of the MNIST digits. We ran CIT4DNN and DeepHyperion-CS for an hour each and generated feature maps for test inputs generated by the two approaches for pairs of feature combinations, (Or, Mov), (Or, Lum), (Mov, Lum). CIT4DNN is run with p=20, t=3, n=1, and $[d_i, d_o] \in \{D1, D5, D1 + D5\}$ to include both normal and rare inputs in the study. We ran DeepHyperion-CS for all 10 classes of MNIST and limited the overall runtime of the tool to one hour. CIT4DNN runs for less than 2 minutes across $D1$ and $D5$; DeepHyperion-CS runs for one hour. We note that CIT4DNN could have been run for a full hour by using $n = 25$, but we only

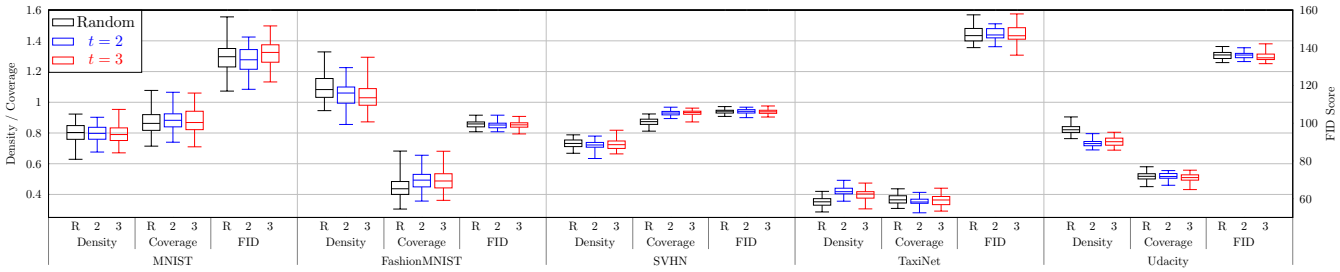

Figure 6: Density, Coverage and FID for test sets generated by CIT4DNN (2σ) with $t \in \{2, 3\}$ compared with random sampling.

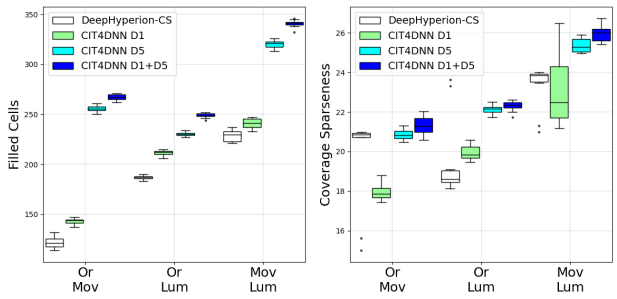

Figure 7: Filled Cells and Coverage Sparseness of test sets generated by DeepHyperion-CS and CIT4DNN (2σ, d_i, d_o, 3, 20, 1), where $[d_i, d_o] \in \{D1, D5, D1 + D5\}$, for the MNIST dataset.

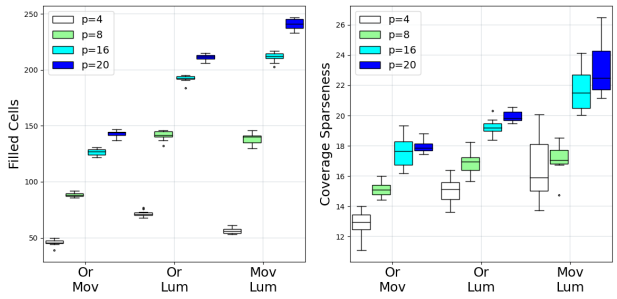

Figure 8: Filled Cells and Coverage Sparseness of test sets generated by CIT4DNN CIT4DNN (2σ, 0, .99, 3, p, 1), where $p \in \{4, 8, 16, 20\}$, for the MNIST dataset.

used $n = 1$ for these studies; therefore, the results presented underestimate the feature diversity that could have been achieved by CIT4DNN given an hour.

This experiment is repeated 10 times with the *FC* and *CS* metrics being measured for each of the feature maps. Box plots of the results are shown in Figure 7. We performed a Mann-Whitney U Test and the results show that rare inputs ($D5$) have better *FC* and *CS* values when compared to normal inputs ($D1$). Additionally, a test set with both rare and normal inputs ($D1 + D5$) outperforms both DeepHyperion-CS and test sets generated for normal or rare inputs alone. These all hold with significance; p-values for all less than 0.05.

Using FC and CS metrics, we conduct two studies to demonstrate how the partition granularity, p, and the strength of CIT, t, of CIT4DNN impact diversity. For the partition granularity study, test sets are generated for $p \in \{4, 8, 16, 20\}$ while keeping $[d_i, d_o]$, t and n fixed to (0, .99), 3 and 1 respectively. Figure 8 shows the FC and CS measured for the generated test sets. We used Mann-Whitney U Test to study whether $M(p = 4) < M(p = 8)$, $M(p = 8) < M(p = 16)$ and $M(p = 16) < M(p = 20)$ where $M \in \{FC, CS\}$. All three tests are performed for 3 feature combinations, resulting in 9 Mann-Whitney U Tests for each metric. The test passed for 9 out of 9 configurations for FC and 7 out of 9 configurations for CS with p-values less than 0.05. The tests failed for $CS(p = 4) < CS(p = 8)$ for (Mov, Lum) and $M(p = 16) < M(p = 20)$ for (Or, Mov) feature combinations. Similarly to the partition granularity study, to study the effect of t on diversity, test sets are generated for $t \in \{1, 2, 3\}$ while keeping $[d_i, d_o]$, p and n fixed to (0, .99), 20 and 1 respectively. Mann-Whitney U Tests for $M(t = 1) < M(t = 2)$, and $M(t = 2) < M(t = 3)$ where $M \in \{FC, CS\}$ indicate that diversity increases with t for all the feature combinations tested with p-values less than 0.05.

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

4.3.3 ***RQ3:*** *How does fault density vary with the latent distribution?* For this study, we used different target density regions, $D1 - D5$, to generate normal and rare inputs. These density ranges include a broad range of cumulative densities in the latent space. We measure the number of fault-revealing test inputs in each density partition to study the distribution of faults in the latent space. MNIST, Fashion, SVHN, TaxiNet, and Udacity are used in the study. The study varies $[d_i, d_o]$ while p, t and n are set to 20, 3, and 1 respectively. Table 3 shows the average number of faults identified by the differential test oracles for each of the density partitions across 10 repetitions of the experiment. Due to the low standard deviation of the number of faults across all the configurations, which is at most 45, we show only average values in the table.

Since $D3 \subseteq D2 \subseteq D1$, we note that faults detected in $D1$ might include faults present in $D3$. Nevertheless, we see that across the datasets there is an increase in the number of faults detected moving from $D1$ to $D3$, except for Udacity. Generally, the test set size

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

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

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

Table 4: Comparison of CIT4DNN (2σ, d_i, d_o, 3, 20, 1) to Random baselines in terms of number of generated tests, number of feasible tests, 3-way coverage, and test generation time for $D1$ and $D5$.

increases across this range, except for Udacity which could explain the lack of increased fault detection. Recall that varying the target density can change the RCCA size and the extent of that change will vary with k, which varies across the VAEs used for these datasets (Table 2). Comparing rare input to normal density region ($D5/D1$) reveals that the increase in test size alone does not explain the increase in the number of faults detected. This is especially true for datasets like MNIST and SVHN.

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

4.3.4 ***RQ4:*** *How cost-effective is CIT4DNN in targeting normal and rare inputs?* We demonstrate the effectiveness of CIT4DNN in generating normal and rare inputs by comparing against random sampling in the latent space and SINVAD [35]. We consider SINVAD as a baseline since it is a recently published generative-model based approach, and its implementation is available. While manifold-based test generation [15] is also relevant and open-sourced, it only generates normal inputs, and it would be unfair to use it as a baseline for rare input testing. Our primary cost metric is test generation time, but we also report the number of tests generated by each technique since this can influence clients of the generated tests, e.g., when running a test set multiple times. Our primary effectiveness metric is total 3-way coverage, which, as we have shown in RQ1 and RQ2, is capable of generating realistic and feature-diverse tests, but in the second part of this RQ we also report fault detection as an effectiveness metric.

Random Baseline The random baseline generates tests by drawing n samples from the Gaussian prior of a VAE, V, as Byun et al. [15] proposed, and is equivalent to CIT4DNN$(V, 0, 1, 1, 1, n)$. For each approach, the experiment terminates either when it achieves 100% total 3-way coverage or the runtime exceeds one hour. We used a server with an AMD EPYC 7742, 2.25GHz CPU with 128GB of memory for running the experiment. We formulated the random baseline test generation using a strategy to ensure that the coverage measurement overhead does not dominate the study results. The strategy first incrementally adds 1000 test inputs to the test set until the algorithm meets the termination criteria. When the random sampling fails to terminate with 100% total 3-way coverage, a strategy that adds one million tests on each increment is used.

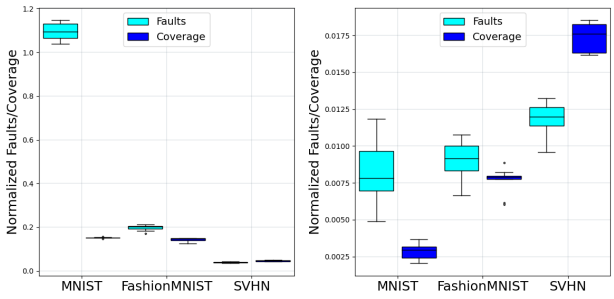

Figure 9: Number of fault-revealing inputs (Faults) and total 3-way coverage (Coverage) of the tests generated by SINVAD normalized with respect to the values measured for CIT4DNN across datasets for $D1$ (left) and $D4$ (right).

CIT4DNN is configured for two target densities, $D1$ and $D5$, to study normal and rare inputs respectively with p=20, t=3, and n=1. Table 4 reports the metrics for CIT4DNN and the random baseline for five datasets. The number of tests is divided into the number sampled and the number that is feasible relative to the target density range. For the high-density region, $D1$, most random samples are feasible, but this is not the case for the low-density region, $D5$. The hit rate of random sampling allows it to achieve 100% coverage, reported as a 1 in the table, within the timeout for $D1$, but requires between 3.3 and 7.2 times the number of tests generated by CIT4DNN. Moreover, CIT4DNN generates those tests up to 7 times faster. In the low-density space, the story is very different. Random sampling cannot hit the target density range frequently causing it to timeout and fail to achieve 100% coverage. In contrast, while CIT4DNN incurs increased cost in $D5$, those costs are comparable to the time for random on $D1$ for 4 of the datasets. CIT4DNN achieves 100% coverage by construction and the projection technique never fails to map a sampled input to the target density range.

SINVAD Since SINVAD only supports classification datasets, only MNIST, Fashion, and SVHN are used in the experiment. CIT4DNN is configured for two target densities, $D1$ and $D4$, to study

normal and rare inputs respectively with p=20, t=3, and n=1. We use the differential testing algorithm and the DNN networks provided in the SINVAD artifact for test generation and the test oracles in the study [4]. We ran each technique until either the generated tests achieve 100% total 3-way coverage or the total runtime exceeds five hours. Since SINVAD has high runtime costs, we ran it on a server with an 11GB Nvidia GTX1080Ti GPU, an Intel Xeon E5-2620 2.10GHz CPU and 128GB RAM for the experiment. SINVAD timed out for all runs, whereas CIT4DNN for n=1 ran in less than 15 minutes just on a CPU across each dataset.

Figure 9 reports the detected faults and coverage ratios of SINVAD relative to CIT4DNN metrics across $D1$ and $D4$ for 10 repetitions of the experiment. The $D1$ results show that CIT4DNN yields a 5-fold increase in total 3-way coverage when compared to SINVAD and that CIT4DNN is better than SINVAD with respect to the number of faults for 2 out of 3 cases in the D1 study. For MNIST, SINVAD detected 842 faults and CIT4DNN only 671. We then ran CIT4DNN with $n = 2$ for these datasets and found that it detected 1339 faults, an improvement of 1.6 times relative to SINVAD. Running with $n = 2$ involves permuting the columns of the covering array, allowing tests to be generated for $D1$ in 50 seconds – 360 times faster than SINVAD which ran for 5 hours. More broadly, we find that the cost of CIT4DNN grows linearly with n as does its fault detection effectiveness. The $D4$ results in Figure 9 indicate that CIT4DNN yields more than a 50-fold improvement in faults detected and total 3-way coverage across all datasets.

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

4.4 Threats to Validity

To promote replicability, we release our implementation as open-source[2].

To mitigate the threats to internal validity, we reused an existing CIT algorithm as a starting point as well as existing DNN model architectures and baselines. We used the default hyperparameters when training the models and for running the baselines. For the components we developed, we manually cross-checked the results for any anomalies in the data.

To mitigate the threats to external validity, we designed the experimental studies to explore a range of configuration parameters of CIT4DNN and recently published baselines. The study presented in §4 guides users in selecting VAEs for generalizing CIT4DNN to other datasets. We used five datasets representing different data domains. However, all the datasets use image inputs; generative models are available for other domains [50, 69], and we plan to extend CIT4DNN to speech and text datasets in the future.

5 RELATED WORK

DNN test generation approaches generate test inputs by exploring either the input space or the feature space [53]. Techniques such as DeepXplore [49], DeepTest [57], DLFuzz [29], and BET [60] work on the input space and generate test inputs by applying pixel-level transformations on seed inputs. The diversity of the generated tests is limited by the diversity of the seed inputs used by these methods and they can generate out-of-distribution inputs [8, 25].

Test generation approaches that work on the feature space produce in-distribution tests. However, these approaches require a model representing the features of the input data distribution. With such a model technique, DeepHyperion [70, 71], DeepJanus [52], and methods that apply traditional CIT to ML [19, 27, 48] have been shown to be effective in covering the input feature space and revealing faults. When such a model is available these approaches can be effective, but they are costly for domain-experts to construct and challenging to produce for high-dimensional input spaces like those found in image DNNs.

In contrast, generative model-based approaches such as Manifold-Based Test Generation [15], SINVAD [35], and DeepTraversal [68] use the latent space of the generative models as a feature domain for test generation, sidestepping the need for a human-defined model. CIT4DNN falls into the generative model-based test category but differs from existing approaches since it (a) guarantees systematic latent space coverage which yields a form of systematic feature diversity coverage and (b) has the ability to efficiently target low-probability input regions which may harbor faults.

Khadka et al. developed a method that applies CIT on the partitioned latent space of a VAE to generate synthetic datasets to train Machine learning models [37]. However, their goal is training data generation whereas CIT4DNN focuses on DNN testing. While their work uses CIT, CIT4DNN uses constrained CIT on the geometry of the latent space which means that, unlike CIT4DNN, their work cannot generate rare inputs on a target density of the latent space.

6 CONCLUSIONS AND FUTURE WORK

CIT4DNN applies constrained combinatorial interaction testing [21] to the latent space of a generative model to produce diverse test inputs. Our experimental studies show that CIT4DNN is effective in generating feature-diverse test sets when compared to the state-of-the-art approaches, is cost-effective for generating rare inputs, and is effective in revealing faults.

CIT4DNN is the first cost-effective approach for validating model behavior on rare inputs. We plan to further study the relationship between input and fault density by investigating a broader set of models, test oracles, and target density shells. We expect that moving too far out on the tails of the distribution will yield inputs that do not resemble training inputs and we plan to investigate methods to estimate the range of target density shells for which CIT4DNN can produce valuable tests.

ACKNOWLEDGEMENTS

We thank Myra Cohen for sharing the implementation of AETG-SAT algorithm proposed in [21] with us. This material is based in part upon work supported by National Science Foundation awards 2019239, 2129824, and 2217071; by The Air Force Office of Scientific Research under award number FA9550-21-0164; by DARPA AIE Geometries of Learning Program under contract HR0011229007; and by Lockheed Martin Advanced Technology Laboratories.

[2]https://github.com/less-lab-uva/CIT4DNN

REFERENCES

[1] 2016. Epoch model. https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23.

[2] 2016. A Tragic Loss. https://www.tesla.com/blog/tragic-loss.

[3] 2020. Reliable Fidelity and Diversity Metrics for Generative Models. https://github.com/clovaai/generative-evaluation-prdc.

[4] 2020. SINVAD replication package. https://github.com/coinse/SINVAD.

[5] 2023. GM Cruise recalls 300 robotaxis after crash involving San Francisco Muni bus. https://www.cbsnews.com/sanfrancisco/news/gm-cruise-recalls-300-robotaxis-after-crash-involving-bus/.

[6] Somil Bansal and Claire J Tomlin. 2021. Deepreach: A deep learning approach to high-dimensional reachability. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1817–1824.

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[8] 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.

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

[10] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, et al. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[11] Sam Bond-Taylor, Adam Leach, Yang Long, and Chris G Willcocks. 2021. Deep generative modelling: A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *IEEE transactions on pattern analysis and machine intelligence* (2021).

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

[13] 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).

[14] 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.

[15] Taejoon Byun, Abhishek Vijayakumar, Sanjai Rayadurgam, and Darren Cofer. 2020. Manifold-based test generation for image classifiers. In *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 15–22.

[16] Jaganmohan Chandrasekaran, Yu Lei, Raghu Kacker, and D Richard Kuhn. 2021. A combinatorial approach to testing deep neural network-based autonomous driving systems. In *2021 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 57–66.

[17] Ricky TQ Chen, Xuechen Li, Roger B Grosse, and David K Duvenaud. 2018. Isolating sources of disentanglement in variational autoencoders. *Advances in neural information processing systems* 31 (2018).

[18] 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.

[19] Tyler Cody, Erin Lanus, Daniel D Doyle, and Laura Freeman. 2022. Systematic training and testing for machine learning using combinatorial interaction testing. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 102–109.

[20] 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.

[21] 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.

[22] Charles Corbière, Nicolas Thome, Avner Bar-Hen, Matthieu Cord, and Patrick Pérez. 2019. Addressing failure prediction by learning model confidence. *Advances in Neural Information Processing Systems* 32 (2019).

[23] Bin Dai and David Wipf. 2019. Diagnosing and enhancing VAE models. *arXiv preprint arXiv:1903.05789* (2019).

[24] 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*.

[25] 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.

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

[27] Christoph Gladisch, Christian Heinzemann, Martin Herrmann, and Matthias Woehrle. 2020. Leveraging combinatorial testing for safety-critical computer vision datasets. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 324–325.

[28] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. 2016. *Deep learning*. Vol. 1. MIT press Cambridge.

[29] 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.

[30] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. 2018. GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium. arXiv:1706.08500 [cs.LG]

[31] Irina Higgins, Loic Matthey, Arka Pal, Christopher Burgess, Xavier Glorot, Matthew Botvinick, Shakir Mohamed, and Alexander Lerchner. 2016. beta-vae: Learning basic visual concepts with a constrained variational framework. In *International conference on learning representations*.

[32] 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.

[33] 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.

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

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

[36] Haresh Karnan, Garrett Warnell, Xuesu Xiao, and Peter Stone. 2022. Voila: Visual-observation-only imitation learning for autonomous navigation. In *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2497–2503.

[37] Krishna Khadka, Jaganmohan Chandrasekaran, Yu Lei, Raghu N Kacker, and D Richard Kuhn. 2023. Synthetic Data Generation Using Combinatorial Testing and Variational Autoencoder. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 228–236.

[38] Hyunjik Kim and Andriy Mnih. 2018. Disentangling by factorising. In *International Conference on Machine Learning*. PMLR, 2649–2658.

[39] 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*.

[40] Diederik P Kingma, Max Welling, et al. 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning* 12, 4 (2019), 307–392.

[41] 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.

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

[43] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proc. IEEE* 86, 11 (1998), 2278–2324.

[44] Kevin P. Murphy. 2023. *Probabilistic Machine Learning: Advanced Topics*. MIT Press. http://probml.github.io/book2

[45] Muhammad Ferjad Naeem, Seong Joon Oh, Youngjung Uh, Yunjey Choi, and Jaejun Yoo. 2020. Reliable Fidelity and Diversity Metrics for Generative Models. https://doi.org/10.48550/arXiv.2002.09797 arXiv:2002.09797 [cs, stat].

[46] 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).

[47] Thomas J. Ostrand and Marc J. Balcer. 1988. The category-partition method for specifying and generating fuctional tests. *Commun. ACM* 31, 6 (1988), 676–686.

[48] Ankita Ramjibhai Patel, Jaganmohan Chandrasekaran, Yu Lei, Raghu N Kacker, and D Richard Kuhn. 2022. A combinatorial approach to fairness testing of machine learning models. In *2022 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 94–101.

[49] 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.

[50] Victor Prokhorov, Ehsan Shareghi, Yingzhen Li, Mohammad Taher Pilehvar, and Nigel Collier. 2019. On the importance of the Kullback-Leibler divergence term in variational autoencoders for text generation. *arXiv preprint arXiv:1909.13668* (2019).

[51] Kashif Rasul. 2017. Fashion-MNIST-CNN. https://gist.github.com/kashif/76792939dd6f473b7404474989cb62a8

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

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

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

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

[56] Andrea Stocco, Michael Weiss, Marco Calzana, and Paolo Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering*. 359–371.

[57] 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.

[58] Rachel Tzoref-Brill. 2019. Advances in combinatorial testing. *Advances in Computers* 112 (2019), 79–134.

[59] Huiyan Wang, Jingwei Xu, Chang Xu, Xiaoxing Ma, and Jian Lu. 2020. Dissector: Input validation for deep learning applications by crossing-layer dissection. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 727–738.

[60] Jialai Wang, Han Qiu, Yi Rong, Hengkai Ye, Qi Li, Zongpeng Li, and Chao Zhang. 2022. BET: black-box efficient testing for convolutional neural networks. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. 164–175.

[61] Risheng Wang, Tao Lei, Ruixia Cui, Bingtao Zhang, Hongying Meng, and Asoke K Nandi. 2022. Medical image segmentation using deep learning: A survey. *IET Image Processing* 16, 5 (2022), 1243–1267.

[62] Huayao Wu, Changhai Nie, Justyna Petke, Yue Jia, and Mark Harman. 2019. A survey of constrained combinatorial testing. *arXiv preprint arXiv:1908.02480* (2019).

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

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

[65] 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]

[66] Yan Xiao, Ivan Beschastnikh, David S Rosenblum, Changsheng Sun, Sebastian Elbaum, Yun Lin, and Jin Song Dong. 2021. Self-checking deep neural networks in deployment. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 372–384.

[67] Xiaofei Xie, Lei Ma, Haijun Wang, Yuekang Li, Yang Liu, and Xiaohong Li. 2019. DiffChaser: Detecting Disagreements for Deep Neural Networks. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 5772–5778. https://doi.org/10.24963/ijcai.2019/800

[68] Yuanyuan Yuan, Qi Pang, and Shuai Wang. 2021. Enhancing Deep Neural Networks Testing by Traversing Data Manifold. https://doi.org/10.48550/ARXIV.2112.01956

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

[70] 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.

[71] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2023. Efficient and effective feature space exploration for testing deep learning systems. *ACM Transactions on Software Engineering and Methodology* 32, 2 (2023), 1–38.