Distribution-Aware Testing of Neural Networks Using Generative Models

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

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

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

***Abstract*—The reliability of software that has a Deep Neural Network (DNN) as a component is urgently important today given the increasing number of critical applications being deployed with DNNs. The need for reliability raises a need for rigorous testing of the safety and trustworthiness of these systems. In the last few years, there have been a number of research efforts focused on testing DNNs. However the test generation techniques proposed so far lack a check to determine whether the test inputs they are generating are valid, and thus invalid inputs are produced. To illustrate this situation, we explored three recent DNN testing techniques. Using deep generative model based input validation, we show that all the three techniques generate significant number of invalid test inputs. We further analyzed the test coverage achieved by the test inputs generated by the DNN testing techniques and showed how invalid test inputs can falsely inflate test coverage metrics.**

To overcome the inclusion of invalid inputs in testing, we propose a technique to incorporate the valid input space of the DNN model under test in the test generation process. Our technique uses a deep generative model-based algorithm to generate only valid inputs. Results of our empirical studies show that our technique is effective in eliminating invalid tests and boosting the number of valid test inputs generated.

***Index Terms*—deep neural networks, deep learning, input validation, test generation, test coverage**

I. Introduction

Deep Neural Networks (DNN) components are increasingly being deployed in mission and safety critical systems, e.g., [1], [2], [3], [4]. Similar to traditional *programmed* software components, these *learned* DNN components require significant testing to ensure that they are reliable and thus fit for deployment.

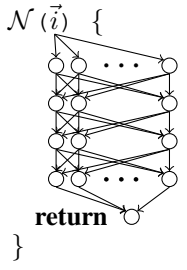
Yet DNNs differ from programmed software components in a variety of ways. (1) They generally do not have well-defined specifications and instead rely on a set of examples that represent intended component behavior. (2) These examples are used to train the parameters of a fixed implementation architecture resulting in implementation behavior encoded as values of the learned parameters. (3) The training process continues until the learned function is an accurate approximation of the intended behavior. Finally, (4) the accuracy of the learned function is intended to generalize to the set of valid inputs comprised of the data distribution of which the training examples are representative.

The above characteristics of DNNs present challenges for applying existing software testing methods to DNNs. For example, the lack of specifications makes it most challenging to develop a rich test oracle, as well as the fact that parameter values encode behavior which renders traditional structural code coverage ineffective. The growing body of research on DNN testing has begun to address some of these characteristics. While structural code coverage metrics are ineffective for DNNs, methods that cover combinations of computed DNN neuron values have been developed to assess and drive DNN testing [5], [6], [7]. Also, variations of metamorphic testing have been developed to check critical continuity properties across the learned function approximations helping to fill the oracle gap [8], [9], [10]. In this paper, we focus on the challenges that DNN generalization presents to testing, and in particular how current DNN testing techniques treat valid and invalid inputs. To understand these challenges, consider the implementation of a traditional software component , which is developed to meet a specification $\mathcal{S} : \mathbb{R}^n \rightarrow \mathbb{R}^m \cup e$, where e denotes the error behavior intended for *invalid inputs*. In this setting, the input domain $\mathbb{R}^n$ is partitioned into valid inputs, V, and invalid inputs, $\overline{V} = \mathbb{R}^n - V$, which should yield e.

The testing of selects a test set $T \subset \mathbb{R}^n$ and assesses whether $\forall t \in T : (t) = \mathcal{S}(t)$. As sketched in Fig. 1a, typically is comprised of *input validation*, which determines if an input value lies in $\overline{V}$ and then executes either *functional logic* which realizes the behavior of $\mathcal{S}$ on V, or *error processing* for invalid data. Developers have come to rely on the several intuitions about such software. First, input validation logic is

```
(i⃗)  {
 if (valid(i⃗))
  return logic(i⃗)
 else
  return error(i⃗)
}
```

(a) Code



(b) DNN

Fig. 1: Structure of code and DNN components and $\mathcal{N}$.

V	00000001000000000000000001001000111111111111111111	0.462
$\overline{V}$	00000001000000000000000010001000111111111111111111	0.442 (0.462)
V	11011101011010001100011010001100111111111111111111	0.692
$\overline{V}$	11111110000101001000110101001011111111111111011011	0.673 (0.808)

Fig. 2: Cumulative neuron coverage of LeNet-1 on the first 100 valid and invalid inputs generated by DLFuzz (top) and DeepXplore (bottom); coverage vectors (left) and ratios (right) for each set are shown along with the cumulative ratio (in parentheses)

distinct from functional logic, demanding testing approaches that exploit its properties [11], [12], [13], [14] to effectively support it [15], [16], [17]. Second, *test suites that achieve higher coverage are better* in that they exercise more of the validation, functional, and error logic.

Now, consider a DNN, $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, which is trained to accurately approximate the, possibly unavailable, specification $\mathcal{S}$. As sketched in Fig. 1b, $\mathcal{N}$ is comprised of layers of neurons that are cross-coupled by connections labeled with learned parameters. When the learned parameters for $\mathcal{N}$ are such that $Pr(\mathcal{N}(i) = \mathcal{S}(i) \mid i \in V) \geq 1 - \epsilon$, for a desired error ϵ, the network is expected to generalize to the valid input distribution, V. Even if $\mathcal{N}$ were trained to detect invalid data and respond appropriately, its structure does not force a distinction between input validation, functional logic, or error processing. In practice, this distinction is uncommon and in this case $\mathcal{N}$ does not even have an analog for e in its output domain. Because of the lack of this distinction, whether an input lies in V or $\overline{V}$, the computation performed by $\mathcal{N}$ overlaps to a large degree, e.g., common sets of neurons are activated.

Not distinguishing between valid and invalid input can be problematic for DNN testing in at least three ways. (1) Testing techniques that generate invalid inputs increase cost with little value added for testing the functional logic of $\mathcal{N}$. Fig. 3 depicts valid test inputs and selected invalid test inputs from two recently proposed DNN test generation techniques [5], [18]. As we show in §IV, across a range of testing approaches for DNNs [19], [5], [18], on average 42% of the generated tests are invalid and in the worst-case all generated tests by a given technique are invalid. (2) When a test case fails developer time is required to triage the failure. With high numbers of invalid test inputs, developers may be forced to look through large numbers of test inputs, similar to those depicted in Fig. 3, to make judgements about test validity. The high-rate of invalid inputs runs the risk that developers will avoid the use of these techniques, thereby negating their purported value. (3) Whereas for traditional software the coverage produced by invalid inputs is confined to the validation and error logic, for DNNs an analogous separation of coverage is not guaranteed. As depicted at the top of Fig. 2, the cumulative coverage from valid and invalid test sets can be almost identical – differing by as few as 1 of 52 neurons. Worse yet, as depicted in the bottom of Fig. 2, invalid tests can artificially boost coverage significantly beyond what is achieved by valid tests - from 0.692 to 0.808. This increase in coverage suggests that, unlike for traditional software, *DNN test suites that achieve higher coverage are not necessarily better*!

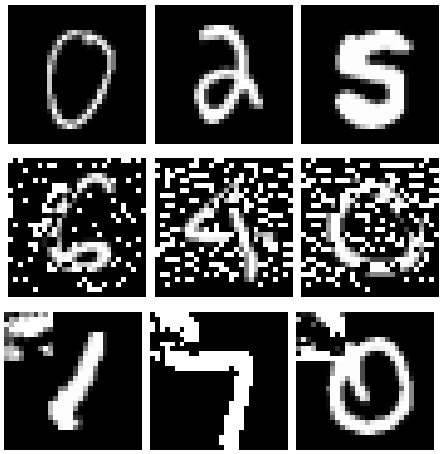


Fig. 3: Valid tests vs Invalid tests. Top Row: Valid tests from MNIST training dataset. Middle Row: Invalid tests from DeepConcolic. Bottom Row: Invalid tests from DeepXplore

In this paper, we study the effects of DNN test generation techniques not distinguishing between valid and invalid data and characterize the potential impact of the issues identified above. Our approach is to leverage a growing body of research from the Machine Learning (ML) community that learns models of the training distribution, V, from which the training data is drawn [20], [21], [22], [23]. While there are many such models, in this paper we employ the *variational auto-encoder* (VAE) – leaving the study of alternative models to future work.

Leveraging VAE models allows us to study techniques representative of the current state of DNN testing research and to make two important observations. First, we demonstrate that existing DNN testing techniques, such as DeepXplore [5], DLFuzz [19], and DeepConcolic [18], produce large numbers of test cases with invalid inputs, which increases test cost without a clear benefit. Second, we demonstrate that existing DNN test coverage metrics, e.g., [5], [6], are unable to distinguish valid and invalid test cases, which risks biasing test suites toward including more invalid inputs in pursuit of higher coverage.

Building on these observations, we present a novel approach that combines a VAE model with existing test generation techniques to produce test cases with only valid inputs. More specifically, we formulate the joint optimization of probability density of valid inputs and the objective of existing DNN test generation techniques, and use gradient ascent to generate valid tests. An experimental analysis on datasets used in the DNN testing literature [24], [25] shows the cost-effectiveness

of the proposed approach.

The primary contributions of this work lie in: (a) the identification of limitations in existing DNN test generation and coverage criteria in their treatment of invalid input data; (b) the development of a technique for incorporating an explicit model of the valid input space of a DNN into test generation to address those limitations; and (c) experimental evaluation that demonstrates the extent of the limitations and the effectiveness of our technique in mitigating them.

The remainder of this article is organized as follows. The following section, §II, describes the concepts that are used in this paper and related work. Our approach is detailed in §III. Experimental strategy and results are described in §IV. §V discusses the threats to validity of our study and §VI concludes.

II. Background and Related Research

A. Deep Neural Networks

Deep Neural Networks (DNNs) are a class of Machine Learning models that can extract high level features from raw input. Similar to the human brain, DNNs contain a large number of inter-connected elements called neurons. DNNs have multiple layers, and each layer contains a number of neurons. A typical DNN consists of an input layer, one or more hidden layers followed by an output layer. Connections between neurons are called edges and their associated weights are referred to as the model parameters. A neuron receives its input as a weighted sum over outputs of neurons from the previous layer. The neuron then applies a non-linear activation function on this input to generate its output. Overall, a DNN is a mathematical function over the model parameters for transforming inputs into outputs. The model learns its parameters by training on known input data called the training data. The objective of DNN training is to learn the model parameters in order to make accurate predictions on unseen data during deployment.

B. DNN testing techniques

DNN testing is an active research area with a number of testing techniques developed to address the challenges of testing these systems [26], [10] in terms of test coverage criteria, test generation and test oracles.

After training, DNN testing techniques use either natural inputs or adversarial inputs for testing. Adversarial inputs are test inputs that are generated by applying tiny perturbations on the original inputs, which cause the model to make false predictions [27]. There is another line of research that focuses on generating adversarial examples for exposing vulnerabilities of DNN models [27], [28], [29] without addressing test adequacy. However our work differs by focusing on coverage guided DNN testing techniques from the software engineering literature.

1) Coverage Criteria: In traditional software testing, coverage criteria are used to measure how thoroughly software is tested. Most practical coverage criteria e.g., [30], use the structure of the software system to make this assessment, e.g., the percentage of statements or branch outcomes covered by a test suite. Similar to structural software coverage criteria, coverage criteria for DNNs have been proposed by various research efforts, as follows.

Pei et al. [5] proposed neuron coverage (NC) as a test coverage criteria. For a given test suite, neuron coverage is measured as the ratio of the number of unique neurons whose output exceeds a specified threshold value to the total number of neurons present in the DNN.

Ma et al. [6] proposed a range of coverage criteria including: k-multisection neuron coverage (KMNC), neuron boundary coverage (NBC), and strong neuron activation coverage (SNAC). These coverage criteria can be used to determine whether a test case falls in the major functional region or corner case region of a DNN. Activation traces of all neurons are captured for the training data and lower and upper bounds of activations are measured for each of the neurons.

K-multisection coverage is calculated by dividing the interval between lower and upper bounds into k-bins and measuring the number of bins activated by the test inputs. For a test suite, k-multisection coverage is the ratio of the uniquely covered bins to the total number of bins in the model.

Neuron activations above the upper bound or below the lower bound are considered to be in corner case regions. Neuron boundary coverage is measured as a ratio of the number of covered upper and lower corner case regions to the total number of corner case regions of the model. Strong neuron activation coverage is the ratio of the number of covered upper corner case regions to the total number of upper corner case regions in the DNN. Top-k neuron coverage and top-k neuron patterns are based on top hyper-activate neurons and their combinations.

Modified Condition/Decision Coverage variants for DNNs [7] are proposed by Sun et al [7]. These metrics are based on sign and value change of a neuron's activation to capture the causal changes in the test inputs. Ma et al. [31] proposed combinatorial test coverage to measure the combinations of neuron activations and deactivations covered by a test suite.

In our work, we focus on the NC, KMNC, NBC, and SNAC criteria and we show that these metrics cannot differentiate between valid and invalid test inputs generated by existing DNN test generation techniques. We leave the analysis for other coverage metrics for future work.

2) DNN test generation: Research on DNN test generation is largely inspired by traditional software testing techniques such as metamorphic testing, fuzz based testing and symbolic execution. Below, we discuss the state of DNN test generation research.

DeepXplore [5] is a white-box differential test generation technique that uses domain specific constraints on inputs. This technique requires multiple DNN models trained on the same dataset as cross referencing oracles. The objective of DeepXplore is a joint optimization of neuron coverage and differences in the predictions of DNN models. Maximizing the objective generates tests that achieve high neuron coverage while simultaneously achieving erroneous predictions by the

DNN model. DeepXplore uses gradient ascent to solve the joint optimization. DeepTest [9] is another testing technique that generates test inputs by applying domain specific constraints on seed inputs. The major focus of DeepTest is to generate test inputs for testing autonomous vehicles. It uses greedy search driven by neuron coverage criteria.

Fuzzing is another traditional software testing technique that has been adapted for DNN test generation including DLFuzz [19], and TensorFuzz [32]. DLFuzz is an adversarial input test generation technique. It uses neuron coverage driven test generation similar to DeepXplore. However unlike DeepXplore, it does not require multiple DNN models. It also uses a constraint to keep the newly generated test inputs close to the original inputs. TensorFuzz is a coverage guided testing method for finding numerical issues in trained neural networks and disagreements between neural networks and their quantized versions.

DeepConcolic [18] uses the concolic testing approach for generating adversarial test inputs for DNN testing. Concolic execution is a coverage-guided testing technique that combines symbolic execution and path information from concrete execution for generating tests satisfying a coverage criteria. DeepConcolic supports neuron coverage and MC/DC variants for DNNs.

None of these DNN testing techniques check whether the test inputs they are generating follow the training distribution. They generate a significant number of invalid inputs that are outside the model's training distribution as shown in our evaluation section IV.

C. Out-of-Distribution Input Detection

Out-of-distribution input detection (OOD), also referred to as outlier or anomaly detection, is a well-studied problem in ML field[20], [21], [22], [33]. A recent survey [23] describes the state of deep learning based outlier detection research and classifies deep learning based outlier detection techniques into supervised, semi-supervised, unsupervised categories. Unsupervised models are preferred as labeling is expensive. We use an unsupervised generative model based approach for our work.

A generative model learns the distribution of the data and can predict how likely a test input is with respect to training distribution. This prediction can be used to identify invalid test inputs. A DNN classifier learns the conditional distribution of target variables with respect to observable variables. Even though such a classifier has high accuracy on data sampled from the training distribution, its accuracy on samples outside the training distribution cannot be guaranteed [34]. By training a generative model with the same data, its density predictions can be used to reject inputs with low densities. When a test input has low density it implies that the DNN classifier did not have enough samples around test input region in the training dataset.

Examples of generative models are autoencoders, variational autoencoders [35], generative adversarial networks (GAN) [36], and autoregressive models such as PixelCNN [37] and PixelCNN++ [38]. We primarily use the variational autoencoder based out-of-distribution detection technique in our work. Also, we repeat our experiments to identify invalid inputs generated by test generation techniques using a PixelCNN++ based validation approach. The study is described in section IV-B to show how sensitive invalid input identification is with respect to the out-of-distribution detection mechanism used.

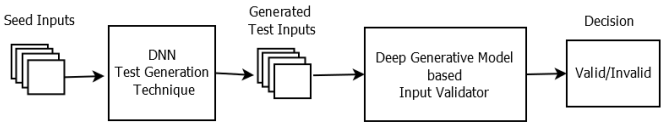


Fig. 4: Technique for identifying invalid test inputs

D. Variational Autoencoder

A variational autoencoder is a generative model that represents latent space as a probability distribution. It has an encoder, code layer and a decoder [35]. The encoder is responsible for mapping inputs to a lower dimensional latent space, and the decoder generates new inputs by sampling from the latent space. Latent space is modeled by a code layer, and it is generated from a prior distribution, e.g., a Normal Gaussian distribution. The encoder's objective is to learn the posterior distribution and decoder's objective is to learn the likelihood of the original input reconstructed by the decoder. A VAE model is trained by minimizing the difference between posterior and latent prior distributions and maximizing the likelihood estimation of the input. A trained VAE model will generate high probability density estimates for data belonging to the training data distribution when compared to out-of-distribution inputs. This key insight is used for validating test inputs generated by DNN test generation techniques in our research.

III. Approach

In this section, we describe our approach to (1) identifying limitations of existing DNN test generation techniques, and (2) generating valid test inputs for testing DNNs.

A. Analysis of Existing DNN Test Generation Techniques

The methodology for analysing test inputs generated by existing test generation techniques is depicted in Fig. 4. DNN(s) under test and the deep generative model are trained on the same dataset. Test inputs generated by existing DNN test generation techniques for the DNN(s) under test are passed as inputs to the deep generative model which estimates their densities. These densities are used by the decision logic to classify inputs as valid or invalid.

For our experiments, we use a VAE for expressing the deep generative model logic, and in particular, the model proposed by An and Cho [20] where the decoder of a VAE outputs distribution parameters for the samples generated by the encoder. The probability of generating the original test input from a latent variable is calculated using these distribution parameters. This probability is referred to as reconstruction

probability. Valid inputs have higher reconstruction probability when compared to invalid inputs.

For a dataset under test, which we call the valid dataset, we identify another dataset which has a different distribution. The inputs from this dataset are considered as invalid inputs. Invalid dataset selection is guided by two factors: (1) the dataset should have same input dimensions as the valid dataset, and (2) invalid and valid datasets should model disjoint data categories.

After identifying an invalid dataset, we compute the reconstruction probability threshold for identifying invalid inputs. Reconstruction probabilities are calculated for inputs from both valid and invalid datasets. We generate a range of thresholds from the combined reconstruction probability values of valid and invalid inputs. We compute the F-measure, which is a measure of a test's accuracy, for these threshold values. The F-measure is the harmonic mean of precision and recall. A good F-measure balances precision and recall and results in a fewer number of both false positives and false negatives. In our case, this means fewer valid inputs are falsely classified as invalid and fewer invalid inputs are falsely classified as valid. The threshold value with the highest F-measure is selected for our experiments. When classifying test inputs generated by DNN test generation techniques, test inputs with reconstruction probability less than the selected threshold are classified as invalid by the VAE classifier.

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

B. Our Test Generation Technique

We present a technique to generate valid test inputs in this section. Our workflow is described in Fig. 5. Our approach leverages existing gradient ascent based test generation technique's objective formulation. The objective of existing test generation techniques is modeled to increase test coverage and produce inputs that cause the model to make incorrect predictions. We augment this objective with probability density estimated by a generative model. Gradient ascent is used to solve the joint optimization. Maximizing the joint optimization will result in inputs that follow the distribution of the training data of the DNN under test along with satisfying objective of the baseline testing technique.

We provide a detailed description of our test generation algorithm using a VAE as the generative model in Algorithm 1. The decoder of the VAE outputs the distribution parameters ($\mu_{\hat{x}}$, $\sigma_{\hat{x}}$) for the samples generated by the encoder as per the OOD detection algorithm proposed in [20]. The algorithm requires a DNN under test, an objective function of a baseline gradient ascent based test generation technique $obj1$, a probabilistic encoder and decoder as inputs and produces both a test suite of valid inputs and their test coverage as output. For every input of the seed set, the probabilistic encoder generates parameters in latent space as shown in line 4 of the

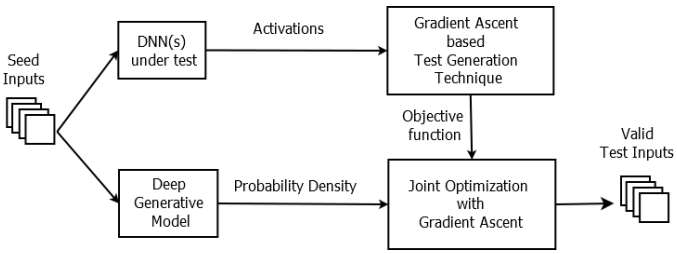


Fig. 5: Technique for generating valid test inputs

Algorithm 1. In lines 5-7, a sample from the latent space is used by the decoder to calculate the reconstruction probability of the input. The objective is modeled as a weighted sum of $obj1$ and reconstruction probability in line 8. Lines 9-11 show the gradient ascent. The gradient is calculated for the objective and at this stage, domain constraints, if any, are applied to the gradient and a new test input is generated. In lines 12-13, the generated test is tested for validity. If this test input causes the model to mispredict and has a reconstruction probability higher than the threshold, then on lines 14-15 the coverage is updated and input is added to the generated test suite. The procedure continues until all seeds are processed. We evaluate this technique using DeepXplore as a baseline test generation technique in Section IV.

Algorithm 1 Valid test input generation using VAE

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

```
1:  gen_test = {}
2:  for x in X do
3:      for i=1 to max_iterations do
4:          μ_z, σ_z = f_θ(z|x)
5:          draw sample from z ~ N(μ_z, σ_z)
6:          μ_x̂, σ_x̂ = g_φ(x|z)
7:          obj2 = p_θ(x|μ_x̂, σ_x̂)
8:          obj = obj1 + λ × obj2
9:          gradient = ∂obj/∂x
10:         gradient = Constraints(gradient)
11:         x = x + s × gradient
12:         p = Reconstruction_Probability(x, f_θ, g_φ)
13:         if Counter_Example(DNN, x) and p ≥ α then
14:             gen_test.add(x)
15:             update coverage
16:             break
17:         end if
18:     end for
19: end for
```

Dataset	Name	Architecture		Accuracy
		Source	#l:#n:#p	
MNIST	MNI-1	LeNet-1 [39]	3:52:7206	98.66%
	MNI-2	LeNet-4 [39]	4:148:69362	99.03%
	MNI-3	LeNet-5 [39]	5:268:107786	99.08%
	MNI-4	Custom [18]	7:1300:312202	99.03%
SVHN	SVH-1	ALL-CNN-A [40]	7:2248:1.2M	96%
	SVH-2	ALL-CNN-B [40]	9:2824:1.3M	95.67%
	SVH-3	ALL-CNN-C [40]	9:2824:1.3M	95.98%
	SVH-4	VGG19 [41]	19:28884:38M	94.69%

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

IV. Evaluation

The design and evaluation of experiments for studying existing techniques and demonstrating effectiveness of our approach are described in this section. We answer the following research questions:

RQ1: Do existing test generation techniques produce invalid inputs?

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

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

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

A. Evaluation Setup

All experiments are conducted on servers with one Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz processor with 32 cores, 62GB of memory, and 4 NVIDIA TITAN Xp GPUs. The software that supports our evaluation as well as all of the data described below is available at https://github.com/swa112003/DistributionAwareDNNTesting.

1) Test Generation Frameworks: We study three state of the art test generation techniques: DeepXplore [5], DLFuzz [19], and DeepConcolic [18] to demonstrate the limitations of existing techniques in terms of generating valid test inputs and satisfying test coverage criteria. The choice of these frameworks is guided by the categorization of test input generation techniques presented in a recent survey [26] and the availability of open source code. The survey categorizes test generation frameworks into three algorithmic families; we choose one technique from each family. DeepXplore is selected from domain-specific test input synthesis, DLFuzz from fuzz and search based test input generation and DeepConcolic from symbolic execution based test input generation categories.

2) Test Coverage Criteria: DeepXplore and DLFuzz use neuron coverage [5] as the test adequacy criteria whereas DeepConcolic can be used with neuron coverage [5], neuron boundary coverage [6] and MC/DC coverage criteria for DNNs [7]. We use neuron coverage as the test adequacy criteria for generating tests using all three frameworks. Resulting test inputs from test generation are analyzed using neuron coverage and extended neuron coverage metrics, i.e, k-multisection neuron coverage, neuron boundary coverage and strong neuron activation coverage. We leave the remaining coverage criteria discussed in these works [6], [7] for future study.

3) Datasets and DNN Models: We use two popular datasets MNIST [24] and SVHN [25] for the experiments. Generative models can assign higher densities to datasets whose distributions are different from their training datasets in some cases[42]. For example, a VAE trained on CIFAR10 [43] can assign higher densities to inputs from SVHN dataset. When such a model is used for invalid input identification, it might result in high densities being assigned to invalid inputs which will result in false negatives. Also selecting the threshold density for deciding invalid inputs becomes challenging in such scenarios. This problem is actively being addressed by ML research community[44]. Generative models trained on MNIST and SVHN do not have this issue [42], so we selected these two datasets for our research.

MNIST is a collection of grayscale images of handwritten digits with 60000 training images and 10000 test images. All three frameworks that we are studying support test generation for MNIST dataset. Similar to DeepXplore, we use LeNet-1, LeNet-4 and LeNet-5 networks from LeNet family [39] and a custom architecture used in the DeepConcolic work [18] for MNIST classification. All the four models are convolutional networks with max-pooling layers and the number of layers ranging from 3 to 7.

SVHN contains color images of digits in natural scenes and the dataset has 73257 training images and 26032 test images. We implemented SVHN support for all three frameworks. We trained SVHN classification models with the ALL-CNN-A, ALL-CNN-B and ALL-CNN-C network architectures proposed in [40] and VGG19 [41] for our experiments. These models are convolutional networks with dropout and either global average pooling or max-pooling layers and the number of layers range from 7 to 19. The models are summarized in Table I where we report measures of their architecture and test accuracy.

4) VAE Models: For MNIST, we trained the VAE that outputs distribution parameters using the model architecture described in [20]. The FashionMNIST dataset [45], is similar to MNIST and contains 28x28 grey scale images. However the distribution is different from that of MNIST as FashionMNIST contains clothing images. We use the FashionMNIST as the invalid input space for calculating the reconstruction probability threshold. Since the VAE is not trained on FashionMNIST distribution and FashionMNIST clothing inputs are semantically unrelated to MNIST digit inputs, the VAE should output lower reconstruction probabilities for test inputs from the FashionMNIST dataset.

We experimented with different variations of the generator architecture used in [46] for selecting a VAE network for the SVHN dataset. For each of the variants, the encoder is created by transposing the generator network as suggested in [46]. The network that achieved the highest F-measure for identifying

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

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

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

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

invalid inputs is selected for our experiments. CIFAR10 [43] is used as the invalid input dataset for calculating reconstruction probability threshold of VAE trained on SVHN. F-measure values and percentage of false positives for MNIST and SVHN test datasets are given in Table II.

B. Results and Research Questions

In this section, we present results of our experiments we used to answer the research questions.

RQ1. Do existing test generation techniques produce invalid inputs?

We generated test inputs for MNIST and SVHN classifiers using the DeepXplore, DLFuzz and DeepConcolic techniques. The DeepXplore framework supports three types of input transformations: lightening, occlusion and blackout. We generated tests for all three transformations to answer RQ1.

We randomly sampled 500 seed inputs from each MNIST and SVHN test dataset for DeepXplore and DLFuzz. DeepXplore and DLFuzz use gradient ascent for test generation, and we used the hyperparameters reported in their respective works [5], [19] for our study. Similarly, we selected the neuron coverage threshold of 0.25 as it is commonly used in DeepXplore and DLFuzz experiments in their original work. The DeepConcolic tool uses a single seed input for test generation for neuron coverage, and a timeout of 12 hours is used for test generation in the primary work [18]. We used the same strategy, and the framework is run with the global optimisation approach. Generated tests are classified as valid or invalid by using the reconstruction probability metric of VAE. The top row of Fig. 6 shows the percentage of invalid test inputs generated by these frameworks for MNIST and SVHN DNN models.

The percentage of tests generated by DeepXplore varies depending on the constraint used. For all the four MNIST classifiers, occlusion constraint produced a high percentage of

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

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

invalid test inputs i.e., greater than 90% while blackout constraint generated less than 1% invalid inputs. The lightening constraint generated 94% and 63% invalid inputs for models MNI-1 and MNI-3 and less than 1% for other two. DLFuzz generated invalid inputs in the range 36% to 46% for MNI-1, MNI-2 and MNI-3 classifiers while less than 1% for MNI-4.

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

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

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

We measured neuron coverage(NC), multi-granularity coverage criteria i.e., k-multisection neuron coverage (KMNC), neuron boundary coverage (NBC) and strong neuron activation coverage (SNAC) of both valid and invalid tests generated by the three frameworks. The k-value of 100 is used for measuring KMNC coverage. We also measured the cumulative neuron coverage of valid and invalid test inputs. Results are presented in Tables III and IV for neuron coverage metric and Tables V and VI have data for multi-granularity coverage criteria.

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

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

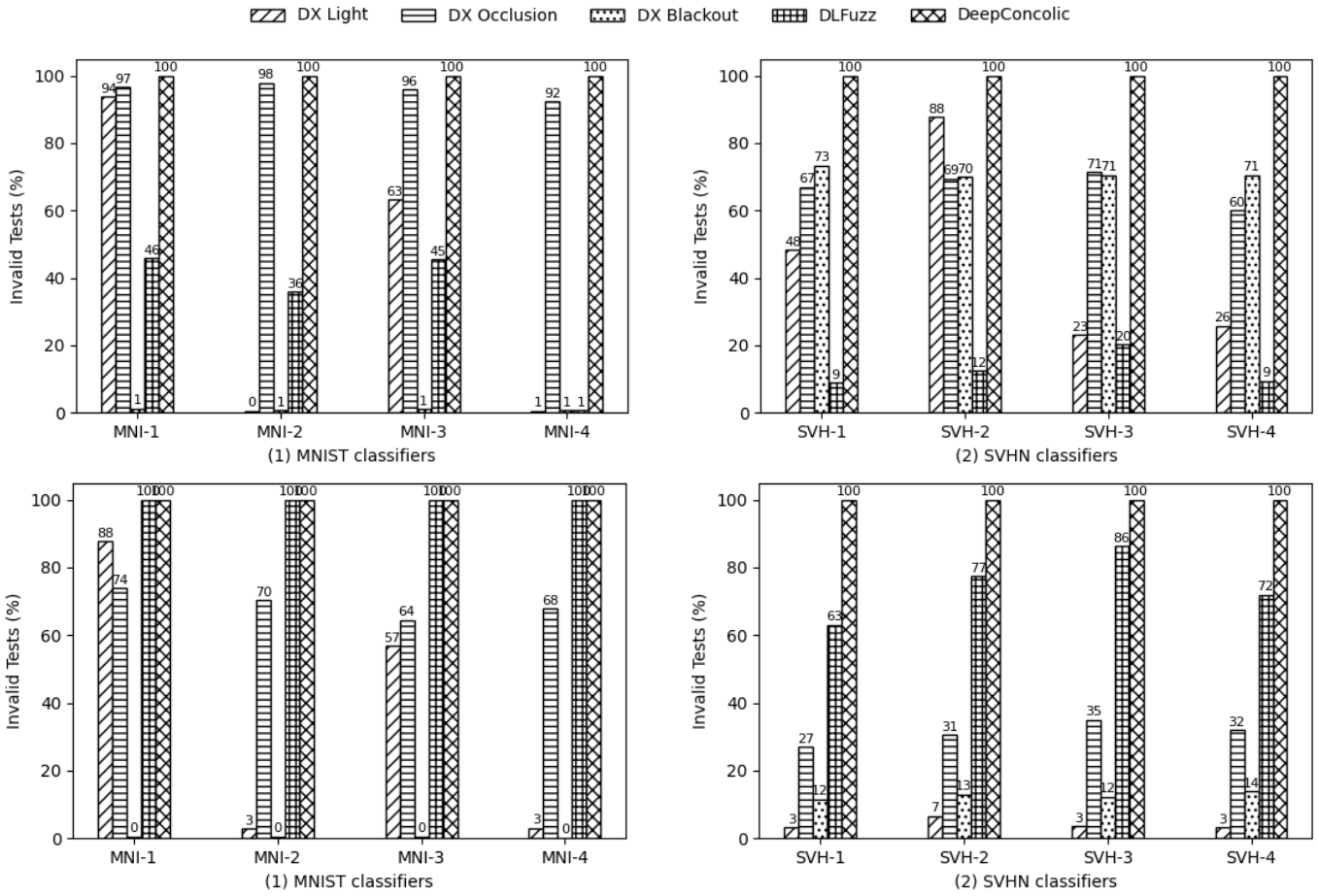


Fig. 6: Percentage of invalid test inputs identified by VAE (top pair) and PixelCNN++ (bottom pair) input validation techniques.

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

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

We measured the number of valid tests generated along with their neuron coverage for our technique and the baseline to demonstrate the effectiveness of our technique. The validity of the inputs is measured with respect to the OOD detection algorithm used, i.e., the VAE in this case. Our technique generates only valid test inputs. Since baseline generates both valid and invalid test inputs, we added the input validation module to the baseline to capture only the valid test inputs.

Neuron coverage achieved by the baseline technique and our technique are presented in Figures 7 and 8 for MNIST and SVHN classifiers respectively. The plots show the coverage over a range of 200 seed inputs. Our technique achieved neuron coverage greater than or equivalent to that of DeepXplore baseline for all the 8 DNN models. For the scenarios where baseline is able achieve neuron coverage comparable to ours, our technique outperformed the baseline in terms of the number of valid inputs generated. Fig. 9 contains a comparison of the number of valid inputs generated by the baseline and our technique for MNIST and SVHN classifiers. The total valid inputs generated by our technique for the MNIST models are 5.6 times the valid inputs generated by the baseline. For SVHN dataset, our technique generated 1.6 times more valid inputs when compared to the baseline. Hence, having VAE in the test objective guides gradient ascent effectively in searching for valid inputs.

Table VII shows the performance data of DeepXplore+VAE and DeepXplore algorithms for 200 seed inputs. Every iteration of these algorithms has two components, 1) gradient ascent, and 2) input validation. For each seed input, gradient

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

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

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

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

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

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

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

To answer RQ4, we use a PixelCNN++ based input validation technique. PixelCNN++ is an autoregressive deep generative model [38]. The advantage of using this model for out-of-distribution detection is that the model outputs the probability density explicitly. We trained PixelCNN++ models for MNIST and SVHN datasets. For each dataset, we find the threshold for identifying invalid inputs by using an invalid dataset and F-measure analysis similar to VAE based detection technique described in Section III-A. The F-measure, precision and recall of the selected thresholds for both the datasets are presented in Table VIII.

The percentage of test inputs generated by DeepXplore, DLFuzz and DeepConcolic for the MNIST and SVHN classification models that are classified as invalid by PixelCNN++ based input classifier are presented on the bottom row of Fig. 6. PixelCNN++ for the MNIST models, classified a high percentage of test inputs generated by DeepXplore's light and occlusion constraints as invalid and classified all test inputs as valid for blackout constraint. For the SVHN classifiers, occlusion and blackout constraints result in higher number of

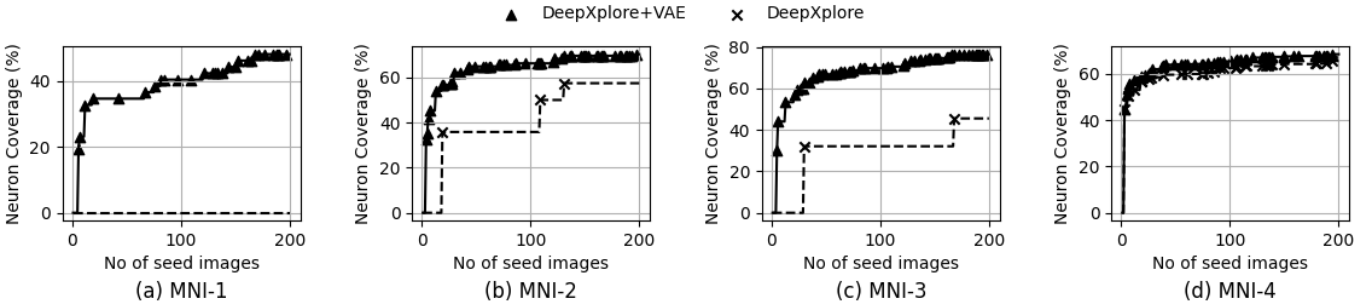


Fig. 7: Neuron Coverage of valid inputs generated by DeepXplore and DeepXplore extended with VAE for MNIST models

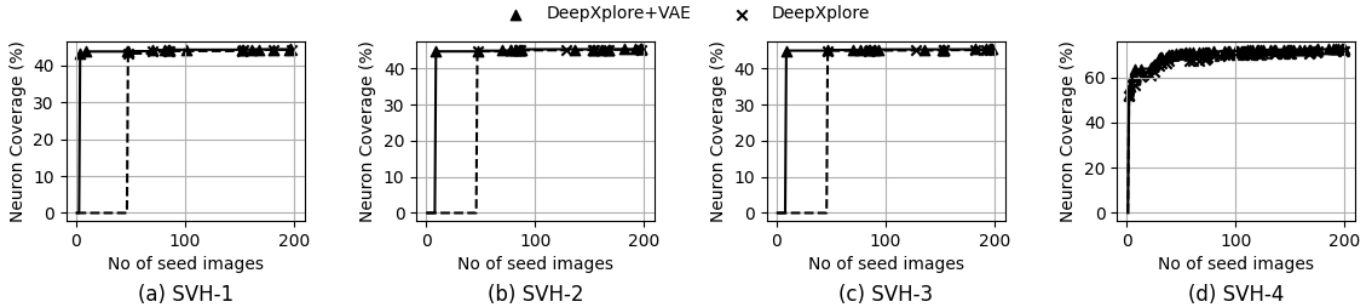


Fig. 8: Neuron Coverage of valid inputs generated by DeepXplore and DeepXplore extended with VAE for SVHN models

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

TABLE VII: Run-time analysis of test generation algorithms of DeepXplore+VAE and DeepXplore for MNIST and SVHN classifiers

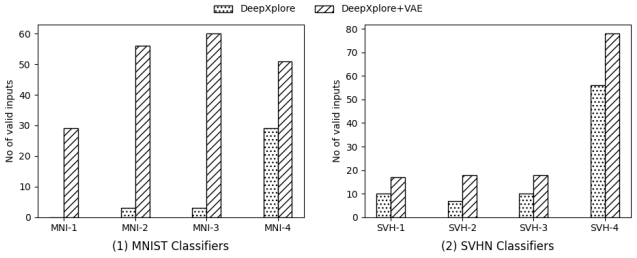


Fig. 9: Number of valid inputs generated by DeepXplore and DeepXplore extended with VAE for MNIST and SVHN models

invalid inputs when compared to the light constraint.

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

The results follow the same trend as observed by VAE based classifier. However the percentage of test inputs classified as invalid by PixelCNN++ is less when compared to that of VAE

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

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

for DeepXplore generated tests. For DLFuzz, the PixelCNN++ approach resulted in more invalid tests when compared to the VAE based classifier. Both the VAE and PixelCNN++ based techniques classified all test inputs generated by DeepConcolic as invalid.

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

V. Threats to Validity

We designed our study to provide a degree of generalizability by spanning all of the algorithmic families of DNN

test generation approaches that have been developed to date, as well as 2 datasets, 8 models, 4 coverage criteria, and 2 approaches to out-of-distribution detection. Moreover, the datasets and models that we have chosen are those that have been used in prior research – which was both a convenience choice and a means of promoting comparison among methods, e.g., against baselines. Despite these measures, our findings may be dependent on these choices.

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

We ran all of our experiments multiple times and cross-checked them with prior work, e.g., that we achieved the same level of coverage for baseline techniques as was reported in prior work. We took these measures to assure the quality of the data reported here and we made the code available in github for transparency and replicability.

VI. Conclusions

This paper demonstrates that existing DNN test generation and test coverage techniques do not consider the valid input space, which can have several deleterious effects. It can lead DNN test methods to generate large numbers of invalid inputs – those that lie off the training distribution as judged by state-of-the-art techniques – thereby reducing the efficiency of the test generation process and, even worse, producing large numbers of tests that might be rejected as invalid during fault triage processes. It can lead test coverage techniques to value invalid tests inappropriately by achieving or improving on coverage from valid tests – this has the potential to bias test generation results.

We demonstrate that existing out of distribution detection techniques can be coupled with test generation algorithms to address this problem. In this work, we focused on VAE-based OOD detection and incorporating such models into optimization-based test generation. Our study shows this to be effective in significantly boosting the number of valid test inputs generated and in eliminating invalid tests. While promising, more work is needed to explore the potential for other OOD models to inform test generation and to incorporate such models into constraint-based and fuzzing test generators.

```
N_defensive(i) {
 if (!OOD(i))
  return N(i)
 else
  return error(i)
}
```

Fig. 10: Defensive DNN

Finally, we plan to explore how the well-understood concept of defensive programming for traditional programs, as sketched in Fig. 1a, can be adapted to DNNs. Fig. 10 sketches a possibility suggested by the findings in this paper, where the role of input validation is played by an OOD detector. In such an architecture, testing of $\mathcal{N}$ should be restricted to inputs that are not out of distribution, but testing of the OOD itself must be conducted over a broader input space as is the case with prior work on input validation testing [15], [16], [17]. With such an architecture, DNN test suites that achieve higher coverage of OOD and $\mathcal{N}$ are *better*, thereby reestablishing the long held intuitions about test coverage for traditional software.

Acknowledgements

This material is based in part upon work supported by National Science Foundation awards 1900676 and 2019239.

REFERENCES

[1] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016. [Online]. Available: http://arxiv.org/abs/1604.07316

[2] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[3] N. Smolyanskiy, A. Kamenev, J. Smith, and S. Birchfield, "Toward low-flying autonomous mav trail navigation using deep neural networks for environmental awareness," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2017, pp. 4241–4247.

[4] A. Loquercio, A. I. Maqueda, C. R. D. Blanco, and D. Scaramuzza, "Dronet: Learning to fly by driving," *IEEE Robotics and Automation Letters*, 2018.

[5] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 1–18.

[6] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu *et al.*, "Deepgauge: Multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 120–131.

[7] Y. Sun, X. Huang, D. Kroening, J. Sharp, M. Hill, and R. Ashmore, "Testing deep neural networks," *arXiv preprint arXiv:1803.04792*, 2018.

[8] X. Xie, J. W. K. Ho, C. Murphy, G. E. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *J. Syst. Softw.*, vol. 84, no. 4, pp. 544–558, 2011. [Online]. Available: https://doi.org/10.1016/j.jss.2010.11.920

[9] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, 2018, pp. 303–314.

[10] X. Huang, D. Kroening, W. Ruan, J. Sharp, Y. Sun, E. Thamo, M. Wu, and X. Yi, "A survey of safety and trustworthiness of deep neural networks: Verification, testing, adversarial attack and defence, and interpretability," *Computer Science Review*, vol. 37, p. 100270, 2020.

[11] J. H. Hayes and J. Offutt, "Input validation analysis and testing," *Empirical Software Engineering*, vol. 11, no. 4, pp. 493–522, 2006.

[12] N. Li, T. Xie, M. Jin, and C. Liu, "Perturbation-based user-input-validation testing of web applications," *Journal of Systems and Software*, vol. 83, no. 11, pp. 2263–2274, 2010.

[13] H. Liu and H. B. K. Tan, "Covering code behavior on input validation in functional testing," *Information and Software Technology*, vol. 51, no. 2, pp. 546–553, 2009.

[14] K. Taneja, N. Li, M. R. Marri, T. Xie, and N. Tillmann, "Mitv: multiple-implementation testing of user-input validators for web applications," in *Proceedings of the IEEE/ACM international conference on Automated software engineering*, 2010, pp. 131–134.

[15] S. Sinha and M. J. Harrold, "Analysis and testing of programs with exception handling constructs," *IEEE Transactions on Software Engineering*, vol. 26, no. 9, pp. 849–871, 2000.

[16] P. Zhang and S. Elbaum, "Amplifying tests to validate exception handling code: An extended study in the mobile application domain," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 1–28, 2014.

[17] A. Goffi, A. Gorla, M. D. Ernst, and M. Pezzè, "Automatic generation of oracles for exceptional behaviors," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 213–224.

[18] Y. Sun, M. Wu, W. Ruan, X. Huang, M. Kwiatkowska, and D. Kroening, "Concolic testing for deep neural networks," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 109–119.

[19] J. Guo, Y. Jiang, Y. Zhao, Q. Chen, and J. Sun, "Dlfuzz: Differential fuzzing testing of deep learning systems," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 739–743.

[20] J. An and S. Cho, "Variational autoencoder based anomaly detection using reconstruction probability," *Special Lecture on IE*, vol. 2, no. 1, 2015.

[21] H. Xu, W. Chen, N. Zhao, Z. Li, J. Bu, Z. Li, Y. Liu, Y. Zhao, D. Pei, Y. Feng *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 2018 World Wide Web Conference*, 2018, pp. 187–196.

[22] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar, "Efficient gan-based anomaly detection," *arXiv preprint arXiv:1802.06222*, 2018.

[23] R. Chalapathy and S. Chawla, "Deep learning for anomaly detection: A survey," *arXiv preprint arXiv:1901.03407*, 2019.

[24] Y. LeCun, "The mnist database of handwritten digits," *http://yann. lecun. com/exdb/mnist/*, 1998.

[25] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[26] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Transactions on Software Engineering*, 2020.

[27] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[28] A. Kurakin, I. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," *arXiv preprint arXiv:1607.02533*, 2016.

[29] N. Carlini and D. Wagner, "Towards evaluating the robustness of neural networks," in *2017 ieee symposium on security and privacy (sp)*. IEEE, 2017, pp. 39–57.

[30] E. J. Weyuker, "The evaluation of program-based software test data adequacy criteria," *Communications of the ACM*, vol. 31, no. 6, pp. 668–675, 1988.

[31] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang, "Combinatorial testing for deep learning systems," *arXiv preprint arXiv:1806.07723*, 2018.

[32] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "Tensorfuzz: Debugging neural networks with coverage-guided fuzzing," in *International Conference on Machine Learning*, 2019, pp. 4901–4911.

[33] D. Hendrycks, M. Mazeika, and T. Dietterich, "Deep anomaly detection with outlier exposure," *arXiv preprint arXiv:1812.04606*, 2018.

[34] A. Nguyen, J. Yosinski, and J. Clune, "Deep neural networks are easily fooled: High confidence predictions for unrecognizable images," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 427–436.

[35] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[36] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[37] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves *et al.*, "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, 2016, pp. 4790–4798.

[38] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications," *arXiv preprint arXiv:1701.05517*, 2017.

[39] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[40] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

[41] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[42] E. Nalisnick, A. Matsukawa, Y. W. Teh, D. Gorur, and B. Lakshminarayanan, "Do deep generative models know what they don't know?" *arXiv preprint arXiv:1810.09136*, 2018.

[43] A. Krizhevsky, G. Hinton *et al.*, "Learning multiple layers of features from tiny images," 2009.

[44] J. Ren, P. J. Liu, E. Fertig, J. Snoek, R. Poplin, M. Depristo, J. Dillon, and B. Lakshminarayanan, "Likelihood ratios for out-of-distribution detection," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 707–14 718.

[45] H. Xiao, K. Rasul, and R. Vollgraf. (2017) Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms.

[46] M. Rosca, B. Lakshminarayanan, and S. Mohamed, "Distribution matching in variational inference," *arXiv preprint arXiv:1802.06847*, 2018.