Distribution Models for Falsification and Verification of DNNs

***Abstract*—DNN validation and verification approaches that are input distribution agnostic waste effort on irrelevant inputs and report false property violations. Drawing on the large body of work on model-based validation and verification of traditional systems, we introduce the first approach that leverages environmental models to focus DNN falsification and verification on the relevant input space. Our approach, DFV, automatically builds an input distribution model using unsupervised learning, prefixes that model to the DNN to force all inputs to come from the learned distribution, and reformulates the property to the input space of the distribution model. This transformed verification problem allows existing DNN falsification and verification tools to target the input distribution – avoiding consideration of infeasible inputs. Our study of DFV with 7 falsification and verification tools, two DNNs defined over different data sets, and 93 distinct distribution models, provides clear evidence that the counter-examples found by the tools are much more representative of the data distribution, and it shows how the performance of DFV varies across domains, models, and tools.**

***Index Terms*—neural networks, verification**

I. Introduction

A deep neural network (DNN) is trained to accurately approximate a partial target function, $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. The domain of definition of f – referred to as the data distribution $\mathcal{D}$ – is typically an infinitesimal portion the full domain, $\frac{|\mathcal{D}|}{|\mathbb{R}^n|} \approx 0$. However, much of the recent literature on validation and verification of DNNs ignores the partiality of a DNN's definition with significant negative consequences. First, existing test generation techniques [1], [2], [3], [4], [5] have been shown to produce a majority of inputs that lie off of the data distribution [6], [7]. Second, white-box DNN coverage criteria [8], [2] do not take the distribution into account and this can drive coverage-directed test generators off the distribution [6] and give misleading reports of the coverage achieved [7]. Third, faults that are detected for off distribution inputs constitute false reports [6], [7] which can lead to wasted effort in fault triage, localization, and fixing.

Whereas recent research has begun to explore how to leverage models of the data distribution for testing [9], [7], in this paper we present the first approach to use such models to support techniques for DNN verification and falsification – a form of property-driven validation [10]. Our *distribution-based falsification and verification* (DFV) approach for DNNs draws inspiration from the large body of research exploiting environmental models of the feasible input domain for software systems to focus V&V. These models are typically built from the system requirements and can be expressed in a variety of forms, e.g., simulations [11], state-machines [12], or logical specifications [13]. Such *environment models* [14] have become an essential component of validation and verification approaches for software systems [15], [16], [17], [18], [19] and this has led them to be adopted in several domains [20].

To be amenable for V&V, environment models must satisfy three requirements. First, they must be *accurate* in defining the set of feasible inputs. For example, for an underapproximating analysis, e.g., [18], an underapproximating model is required to guarantee feasible counter-examples; dually an overapproximating analysis requires an overapproximating environment model. Second, they must be *generative*, providing the ability to be executed, interpreted, or solved, so they can be leveraged to generate feasible inputs. For example, generating feasible counter-examples when verifiers or falsifiers detect property violations [21]. Third, for verification they must be *amenable to constraint-based encoding* in a form that can be leveraged by the verification algorithm. For example, for a SMT-based verification method, e.g., [18], an environment model must be convertible to logical formulae in a supported theory. For abstract interpretation, e.g., [22], an environment model must be convertible to supported abstract domains.

In this paper, we adapt the concept of an environment model to support the existing verification and falsification techniques [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [10] of specified properties of DNNs. To do this, we have to address a challenge that it is intractable: the specification of an accurate model of the feasible inputs for a complex DNN – like those that process images captured by a forward facing camera (see Figure b [39]). A key insight of this work is that we can leverage the rich body of research that the machine learning (ML) community has developed for learning generative models of the data distribution, which we use as environment models.

DFV transforms a DNN and a correctness property into a falsification or verification problem focused on the data distribution in three steps. First, a generative model of the data distribution for a DNN is trained [40], [41]. Unlike manually developed environment models for traditional software V&V, these environment models are constructed automatically through an unsupervised training process. The design and training of the model of the data distribution can leverage ML best-practices to produce a suitably accurate model [42], [43]. Second, the approach modifies the original DNN to use the appropriate component of the trained generative model, e.g., the decoder of a variational autoencoder (VAE), as a set of prefix layers to the DNN under analysis. This forces all inputs to the DNN to come from the learned data distribution. Third, the approach reformulates the correctness property over

the input space of the generative model. For verification, the feasible input space can be overapproximated, whereas for falsification techniques, the feasible input space can be underapproximated. DFV supports the reporting of feasible on-distribution counter-examples, when property violations are detected, and reporting that specified subsets of the data distribution are free of violations, when verifiers are able to discharge such proofs.

We evaluate DFV on DNNs trained to recognize images of clothing [44] and trained to control a drone from image data [39], for a range of challenging correctness properties. We find clear evidence that DFV enables existing falsification and verification techniques to produce counter-examples that are much more representative of the data distribution than are computed otherwise – both visually and in terms of standard measures of similarity. While scaling of verification techniques is challenging, we also find evidence that distribution models can enable them to prove properties over the data distribution. Building on these promising findings, we study how varying the architecture of the model and how shifting between different families of generative models impacts the effectiveness of the technique. Our results can be used to guide the development of models to support DFV.

The primary contributions of this work are the: (1) formulation of the first model-based verification and falsification method for DNNs; (2) demonstration that distribution models yield substantially better counter-examples from verification and falsification; and (3) exploration of different models of the data distribution and their trade-offs.

II. Background

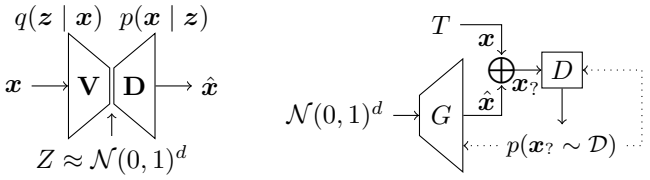
In this section we provide background on deep neural networks, DNN verification and falsification, and DNN testing approaches that exploit the data distribution.

A. Deep Neural Networks

A deep neural network, $\mathcal{N}$, is a type of machine learning model that is trained to approximate a partial target function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$. For example, f may classify some n-dimensional input (e.g., an image) as one of m possible classes (e.g., a digit in the range 0 to 9). f is partial in the sense that it is trained to generalize to a target data distribution, $\mathcal{D} \subseteq \mathbb{R}^n$. For inputs off of the distribution, $\boldsymbol{x} \not\sim \mathcal{D}$, its behavior, $\mathcal{N}(\boldsymbol{x})$, should be considered as undefined.

DNNs are comprised of *layers*, $l_0, ..., l_k$, each of which performs some computation on their input (e.g., matrix multiplication or convolution). A typical linear architecture defines a DNN as the composition of layers, $\mathcal{N} = l_k \circ \cdots \circ l_1 \circ l_0$. Layers are comprised of *neurons*. The input of a neuron is defined as the weighted sum of the outputs of a set of neurons in a preceding layer where the connections between neurons have trainable *parameters*. The output of a neuron applies a non-linear activation function to the input.

Training involves initializing the parameters and then applying $\mathcal{N}$ to samples, $(\boldsymbol{x}, \boldsymbol{y})$, from the training set, T, and repeatedly updating parameters based on $\|\mathcal{N}(\boldsymbol{x}) - \boldsymbol{y}\|$. While the goal of training is to learn the partial function f defined over $\mathcal{D}$, $\mathcal{D}$ is generally unmanageably large (e.g., the set of road images visible on a forward facing camera). Consequently, the training set is defined as a representative sample of the data distribution, $T \sim \mathcal{D}$. A well-trained network is said to *generalize* to the data distribution [45].



(a) VAE (b) GAN

Fig. 1: Generative latent distribution models.

B. Models of the Data Distribution

The field of machine learning has long understood the importance of modeling the data distribution. Broadly speaking, the field has developed two types of approaches. Out-of-distribution detectors [46] are designed to determine whether a data point lies on the data distribution, but are generally unable to generate new daa from the distribution. In contrast, *generative* models are designed to generate unseen samples from the data distribution. There are three broad classes of generative models: variational autoencoders (VAE) [40], generative adversarial networks (GAN) [41], and autoregressive models such as PixelCNN++ [47]. Among these, VAEs and GANs can be classified as *latent variable* models since they make explicit the mathematical structure of the learned latent space which models $\mathcal{D}$. We leverage generative latent variable models of the data distribution in this work.

Fig. 1a depicts a VAE as comprised of a pair of trainable models: an encoder, $\mathbf{V}$, and a decoder, $\mathbf{D}$ [40]. The encoder, or inference network, is trained to learn the parameters of the latent distribution, $q(\boldsymbol{z} \mid \boldsymbol{x})$, that, through a regularization term, seeks to match a given prior distribution - usually a multivariate Gaussian, $\mathcal{N}(0,1)^d$. The decoder, or generative network, is trained to learn the likelihood of an input given values in the latent space, $p(\boldsymbol{x} \mid \boldsymbol{z})$. These networks are trained together on inputs drawn from the data distribution, $\mathcal{D}$, by minimizing the difference between posterior and latent prior and maximizing the likelihood estimation of the input. A VAE is generative in the sense that one can sample from the latent space, $\boldsymbol{z} \sim N(0,1)^d$, and then run the decoder, $\mathbf{D}(\boldsymbol{z}) = \hat{\boldsymbol{x}}$, to produce a sample that lies on the data distribution. VAEs can be leveraged for out-of-distribution detection by exploiting the fact that for $\boldsymbol{x} \sim \mathcal{D}$, $\mathbf{V}(\boldsymbol{x})$ produces a distribution that can be sampled to generate inputs, $\hat{\boldsymbol{x}}$. Computing $\|\boldsymbol{x} - \hat{\boldsymbol{x}}\|$ for a number of samples yields the *encoder-stochastic reconstruction error* (ESRE) [48]. We adapt ESRE to use the structural similarity index measure (SSIM) [49] to assess the quality of generated image data in §IV.

Fig. 1b depicts a GAN as comprised of a pair of trainable generator, $G : \mathbb{R}^d \rightarrow \mathbb{R}^n$, and discriminator, $D : \mathbb{R}^n \rightarrow \mathbb{R}$, [41]. The generator produces an input, $\hat{\boldsymbol{x}}$, from a set

of latent variables. The discriminator predicts the probability that an input is from the true data distribution, $p(\boldsymbol{x} \sim \mathcal{D})$. The GAN is trained by presenting generated, $\hat{\boldsymbol{x}}$, and training inputs, $\boldsymbol{x}$, to the discriminator without disclosing their source, $\boldsymbol{x}_?$. The generator loss function is high when the generated data is classified as generated data by the discriminator, i.e., $p(\hat{\boldsymbol{x}} \sim \mathcal{D})$ is low. The loss function of the discriminator is high when it incorrectly classifies data, i.e., $p(\hat{\boldsymbol{x}} \sim \mathcal{D})$ is high or $p(\boldsymbol{x} \sim \mathcal{D})$ is low, and a low value when it is correct. The weights of the generator and discriminator are updated to decrease their respective loss values. Through this process, the generator learns to produce data close to the data distribution.

There is a rich literature on the design of VAEs and GANs exploring the impact of latent dimension, complexity of model architectures, and variation in loss functions on the accuracy of the learned model. Generally GANs are thought to possess better precision, i.e., produce sharper images, but suffer from poor recall, whereas VAEs are thought to be the opposite, i.e., good recall, but produce blurry images. Leveraging distribution models for V&V requires a measure of both, but the ML literature continues to improve in this regard. For example, VAEs can now achieve precision that outperforms well-tuned GANs while retaining good recall [50].

C. DNN Verification and Falsification

A correctness problem is a pair, $\psi = \langle \mathcal{N}, \phi \rangle$, of a DNN, $\mathcal{N} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, and a property specification ϕ, formed to determine whether $\mathcal{N} \models \phi$ is valid or invalid. The property specification defines a set of constraints over the inputs, $\phi_{\mathcal{X}}$ – the pre-condition, and a set of constraints over the outputs, $\phi_{\mathcal{Y}}$ – the post-condition. Verification of $\phi(\mathcal{N})$ seeks to prove or falsify: $\forall \boldsymbol{x} \in \mathbb{R}^n : \phi_{\mathcal{X}}(x) \rightarrow \phi_{\mathcal{Y}}(\mathcal{N}(x))$. Falsification seeks only to falsify that formula.

Two common types of DNN properties are *robustness* and *reachability*. Robustness originated with the study of adversarial examples [51], [52], and specifies that inputs from a given input region are all classified the same. This type of property is common for evaluating verifiers [53], [26], [36], [31]. Reachability properties define the post-condition using constraints over output values, specifying that inputs from a given input region reach outputs within a given safe output region. This type of property has been used to evaluate several DNN verifiers [53], [35], [54].

A recent survey on DNN verification [55] classifies approaches for verifying DNN correctness problems based on their type: reachability, optimization, or search, or a combination of these. Tools implementing a range of these approaches and their combination have been developed over the past three years [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37]. Despite the significant research into this topic scalability remains a challenge, but usability has been improved by the availability of frameworks like DNNV [38] which we use in our work.

Complementary to verification, falsification checks properties of DNNs by attempting to find examples that violate the specification for a given model. Two categories of techniques that have been developed for falsifying DNN correctness problems are adversarial attacks and fuzzing. *Adversarial attacks* are methods optimized for detecting violations of robustness properties [52]. *Fuzzing* methods randomly generate inputs within a given input region, and checking whether the outputs they produce violate the post-condition. Fuzzing techniques include TensorFuzz [3] and DeepHunter [4]. More recently, the range of applicability of adversarial attacks and fuzzing has been increased to correctness properties by DNNF, which reduces general DNN correctness properties to robustness properties [10], [56].

D. Distribution-aware DNN Testing

Recent work in testing has begun to explore models of the input domain to support DNN testing. Riccio and Tonella construct explicit models of the input space using domain knowledge to generate test cases in order to characterize the space of DNN misbehaviors [57]. Dola et al. show that not considering the input distribution can bias the assessment of DNN testing techniques, and then use VAEs to model the input distribution and to augment the objective functions of test generation techniques to remedy that bias [7]. Byun and Rayadurgam also describe how to model the input distribution with a VAE and use it to generate test inputs [58]. Our approach differs in that it composes the distribution model, e.g., a VAE decoder, with the DNN under analysis and, because we focus on verification and falsification, develops constraint-based encodings that over and under-approximate designated portions of the input space of that model.

III. Approach

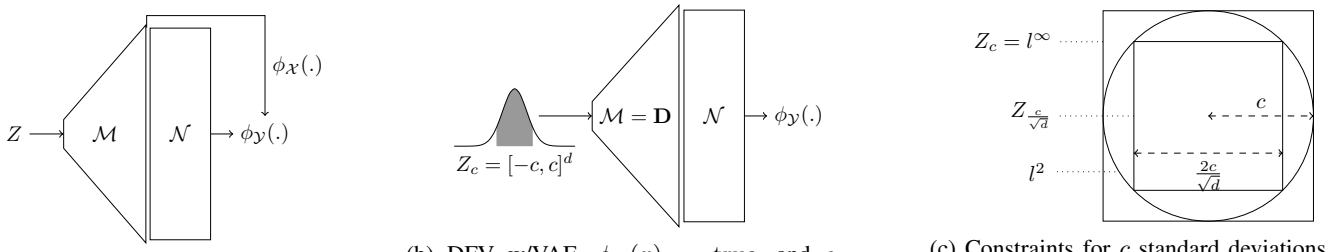
We present DFV, our approach for focusing DNN verification and falsification techniques on the data distribution.

A. DFV *Overview*

Fig. 2a depicts the generic structure of DFV. Our insight is that properties should be verified not over the entire input domain, $\mathbb{R}^n$, of a DNN, but rather over the data distribution, $\mathcal{D}$, used to train the DNN, i.e., its domain of definition. Since $\mathcal{D}$ is a small subset of the domain there is the potential to enable property verification when violating inputs lie off of the distribution. Moreover, by restricting verification to inputs on the distribution, $\boldsymbol{x} \sim \mathcal{D}$, counter-examples will be feasible and worth fixing.

To define $\mathcal{D}$, we advocate the use of a latent variable generative model, $\mathcal{M} : \mathbb{R}^d \rightarrow \mathbb{R}^n$ where $d \ll n$. With $\mathcal{M}$ analyses can be formulated over the low-dimensional latent space, $Z \approx \mathcal{N}(0,1)^d$, to reason about the behavior of systems on $\mathcal{D}$. Two classes of such models that we explore, introduced earlier in §II, are variational autoencoders (VAE) and generative adversarial networks (GAN).

Our goal is to define the set of inputs that lay on the data distribution and that satisfy the property pre-condition, i.e., $\boldsymbol{x} \sim \mathcal{D} \wedge \phi_{\mathcal{X}}(\boldsymbol{x})$. Fig. 2a depicts the enforcement of these constraints using two mechanisms: the use of $\mathcal{M}$ as a prefix network, and the forwarding of generated inputs to



(a) General DFV Framework

(b) DFV w/VAE, $\phi_{\mathcal{X}}(x) = \texttt{true}$, and c-bound on $\mathcal{M}$

(c) Constraints for c standard deviations of $\mathcal{M}$.

Fig. 2: Distribution-based Falsification and Verification Framework with Bounding Constraints

enable the enforcement of the pre-condition prior to checking post-conditions – as described in [10]. We note that it is also possible to combine $\mathcal{M}$ and $\phi_{\mathcal{X}}$ by defining a generative model capable only of producing inputs satisfying the precondition, in which case the forwarding generated inputs is not needed. With these elements the verification problem can be reformulated as $\forall \boldsymbol{z} \in Z : \phi_{\mathcal{X}}(\mathcal{M}(\boldsymbol{z})) \rightarrow \phi_{\mathcal{Y}}(\mathcal{N}(\mathcal{M}(\boldsymbol{z})))$.

In this paper, we explore in detail an instantiation of DFV, depicted in Fig. 2b, that uses a VAE as the generative model and targets output properties of the DNN, i.e., where the precondition is true. As explained in §II, the VAE's encoder and decoder are trained together, but we only exploit the decoder, $\mathcal{M} = \mathbf{D}$, for DFV.

Since the dimension of the latent space of the VAE is generally much smaller than the input dimension, the generated set of inputs, $\{\mathcal{M}(\boldsymbol{z}) : \boldsymbol{z} \in Z\}$, takes up an infinitesimal portion of the ambient input space. For falsification, this allows the problem to be reformulated from the input space, $\exists \boldsymbol{x} \sim \mathcal{D} : \neg\phi_{\mathcal{Y}}(\mathcal{N}(\boldsymbol{x}))$, to the latent space, $\exists \boldsymbol{z} \in Z : \neg\phi_{\mathcal{Y}}(\mathcal{N}(\mathcal{M}(\boldsymbol{z})))$. More importantly, since existing falsification algorithms cannot test that $\boldsymbol{x} \sim \mathcal{D}$, this approach is the first to yield counter-examples that lie on the data distribution – subject to the precision of $\mathcal{M}$ in modeling $\mathcal{D}$. Similarly verification can be reformulated to the latent space, $\forall \boldsymbol{z} \in Z : \phi_{\mathcal{Y}}(\mathcal{N}(\mathcal{M}(\boldsymbol{z})))$.

Existing verification and falsification tools require that their input space be bounded. When using $\mathcal{M}$ that input space is the latent space of the distribution model and its Gaussian structure allows us to formulate a meaningful bound. For a d-dimensional latent space, the l^2 d-ball of radius c, as depicted in Fig. 2c, contains all points within c standard deviations of the mean. The value of c can be specified to contain an arbitrarily large portion of the distribution, e.g., $c = 5$ specifies verification of 99.99994% of the distribution. However, existing DNN falsifiers and verifiers do not support the non-linear constraints necessary for defining the l^2 d-ball, so we formulate hypercube approximations.

As depicted in Fig. 2c, the l^∞ d-ball is the smallest hypercube that overapproximates the l^2 d-ball. We denote with Z_c the hypercube with radius c – side-length $2c$. Formulating constraints that restrict each of the d dimensions to the interval $[-c, c]$ yields a verification problem, $\forall \boldsymbol{z} \in Z_c : \phi_{\mathcal{Y}}(\mathcal{N}(\mathcal{M}(\boldsymbol{z})))$, that will soundly verify c standard deviations in Z. Dually, for falsification, we can use the largest hypercube that underapproximates the l^2 d-ball. We denote with $Z_{\frac{c}{\sqrt{d}}}$ the hybercube with radius $\frac{c}{\sqrt{d}}$. Restricting each latent dimension to lie in the interval $[-\frac{c}{\sqrt{d}}, \frac{c}{\sqrt{d}}]$ guarantees that any counter-example detected will lie within c standard deviations in Z. This follows from defining the inscribed hypercube with side-length s, equating the diameter of the l^2 d-ball, $2c$, with the longest diagonal of that hypercube, $s\sqrt{d}$, and solving for s.

Algorithm 1: DFV

Input: Correctness problem $\psi = \langle \mathcal{N}, \phi \rangle$, Distribution model $\mathcal{M}$, Verifier/Falsifier V, Radius c
Output: $\{violation$:ce, $valid$, $unknown\}$

```
begin
    N' ← N ∘ M
    φ' ← ∀z ∈ ℝ^{M.d} : (z ∈
      [−c, c]^{M.d}) ∧ φ_X(M(z)) →
                        φ_Y(N'(z))
    ψ' ← ⟨N', φ'⟩
    result ← check(V, ψ')
    if result = violation then
        return violation :
          M(getCounterExample(result))
    return result
```

B. DFV *Algorithm*

Algorithm 1 defines DFV through a series of transformations followed by the invocation of the verifier or falsifier.

The DFV algorithm accepts a correctness problem, comprised of a DNN, $\mathcal{N}$, and correctness property, ϕ. In addition, it takes a model, $\mathcal{M}$ of the data distribution, $\mathcal{D}$, a verifier or falsifier, V, and a radius, c, which defines how much of the data distribution should be subjected to analysis. Its output indicates that either a property $violation$ has been detected, the property is $valid$ within radius c (for verifiers), or the result is $unknown$ – due to limitations in the verifier or falsifier used. When a violation is reported a counter-example, ce, is returned as well. We now describe the algorithm in more detail.

Decoupling the Training of the Distribution Model from DFV. Algorithm 1 consumes $\mathcal{M}$, separating the training of $\mathcal{M}$ from DFV. This is important because there are many degrees of freedom when training $\mathcal{M}$, often dependent on the type of model and training used. For example, for a VAE, such as those used in §IV, the effectiveness of the learned model depends on many factors, including the architecture of the model (e.g., number, type, and configuration of layers) and the training parameters (e.g., optimizer, batch size, and learning

rate). Independent of the model and training process, the goal of this pre-stage to DFV is for $\mathcal{M}$ to approximate $\mathcal{D}$. We note that the development of distribution models that have high precision and recall is an active area of ML research [42], [43], and that recent research has defined high-precision VAEs [50]. We note, however, that models with lower levels of precision can still be quite valuable. As we show in §IV, rather simple VAE models can yield much more meaningful counter-examples than those produced without using a distribution model. In this work we explore VAE variations, but we leave to future work a broader study of how model accuracy impacts the cost and benefit of DFV.

Problem Transformation. DFV transforms the correctness problem as described in lines 2-5 of Algorithm 1. Line 2 modifies the original DNN by prefixing it with a generative model, as shown in Fig. 2a. This transformation prefixes the original DNN with a latent variable generative model, such as the decoder of a VAE. Because the generative model maps inputs from a known distribution to the learned data distribution, this step ensures that verification and falsification will only check inputs from the data distribution learned by the prefixed model. That is, the tools can focus on the inputs that are within the distribution. Line 3 replaces the input pre-condition of the original property with a new pre-condition specifying that inputs come from the latent space of the generative model – $\mathcal{M}.d$ denotes its dimension – and that these inputs satisfy the pre-condition. Because the verifiers and falsifiers require inputs to be bounded, we assert bounds on the latent space. When the latent space distribution of the generative model is Gaussian, we require z to be within c standard deviations of the mean. We approximate this with a hypercube of radius c centered at the origin. Line 5 joins the outcomes of the transformation from line 2 and 3 to redefine the correctness problem on the data distribution.

Verification and Falsification. After transforming the problem, on lines 6-9, falsifiers and verifiers can be run on the modified correctness problem, $\psi' = \langle \mathcal{N}', \phi' \rangle$. If a counter-example to ψ' is found, then it can be mapped to a valid counter-example of the original property by performing inference with the generative model, $\mathcal{M}$. DFV will report, $violation$, along with the counter-example, otherwise it will report the $valid$ or $unknown$ result returned by V.

IV. Study

In this section we assess the cost-effectiveness and scalability of DFV by applying it in conjunction with multiple falsifiers and verifiers. Our evaluation will answer the following research questions:

1) What is the cost-effectiveness of applying falsification and verification with DFV?
2) How does the configuration of the model used by DFV affect the quality and quantity of counter-examples?
3) How well does DFV scale to more complex input domains that required more sophisticated models?

(a) FashionMNIST training images.

(b) DroNet training images.

Fig. 3: Samples from FashionMNIST and DroNet training sets.

A. Design

We now describe the problem benchmarks, generative models, falsifiers, verifiers, and metrics that constitute the three experiments in our study. We describe the experimental procedures using these items under each research question.

1) Problem Benchmarks: We evaluate DFV on two benchmarks of correctness problems. The *GHPR-FMNIST* benchmark is a new DNN correctness problem benchmark, based on the GHPR-MNIST benchmark from the evaluation of DNNF [10]. The benchmark consists of 20 global reachability properties applied to a small FashionMNIST [44] network. A sample of images from the FashionMNIST training set is shown in Fig. 3a. The network used is based on the architecture of the small MNIST network from the evaluation of the Neurify verifier [53]. There are 2 formulations of properties in this benchmark. The first 10 properties, which we will refer to as type A, are of the form: for all inputs, if class a has the maximal value, then the output values for classes a and b are closer to one another than the output values for classes a and c. For example, one of the properties states that for all inputs, if that input is classified as a $sneaker$, then the class $sandal$ will be ranked higher than class $shirt$. The other 10 properties (type B) are weaker variations that drop the maximal value constraint.

The *GHPR-DroNet* benchmark, introduced in DNNF [10], consists of 10 global reachability properties applied to the DroNet DNN [39], which predicts a steering angle and probability of collision for a quadrotor from 200 by 200 black and white images. DroNet is a large DNN model consisting of 3 residual blocks and over 475,000 neurons. A sample of images from the DroNet training set is shown in Fig. 3b. The properties are of the form: for all inputs, if the probability of collision is between p_{min} and p_{max}, then the steering angle is within d degrees of 0. As p_{min} increases, so does d, capturing the intuition that if the probability of collision is low, then the quadrotor vehicle should not make sharp turns.

2) Generative Models: We consider two powerful types of latent variable generative models to learn the data distribution of the training set – VAEs and GANS. We selected these models because: 1) they meet the requirements of the approach, 2) they are among the most popular unsupervised learning approaches to encode a data distribution, and 3) they work

TABLE I: Generative Models

RQ	Name	Latent Dim.	Layers/ Neurons	Output Activation
1,2	VAE_{MRS}	100	2/768	Sigmoid
1	VAE_{RQ1}	2	1/24	∼Sigmoid
2	$VAE_{d,l,h}$	1-32	1-4/16-1024	Sigmoid
3	FC-VAE_{DroNet}	512	5/4608	Sigmoid
3	Conv-VAE_{DroNet}	512	7/202729	Sigmoid
3	GAN_{DroNet}	512	2/3840000	Sigmoid

in different ways and provide different tradeoffs. Given the number of variables involved in our experiments, we chose VAEs for RQ1 and RQ2, and incorporated GANs for RQ3. Through the study we explored a total of 93 models, 91 to characterize the distribution of *GHPR-FMNIST* and 2 to characterize the distribution of *GHPR-DroNet*. All models used in the study are shown in Table I. Details for the configuration of those models is provided in the experimental procedures for each of the research questions.

3) Falsifiers and Verifiers: For the falsifiers, we will use four common adversarial techniques included in the DNNF tool [10]. DNNF reduces correctness problems to adversarial robustness problems to allow them to be falsified by off-the-shelf adversarial attacks. We chose to use FGSM [59], Basic Iterative Method (BIM) [60], DeepFool [61], and Projected Gradient Descent (PGD) [62] as they were the top performing falsifiers in the DNNF study. We use the same parameters for each adversarial attack method as used in that study.

We will also use three top performing DNN verifiers. We chose to use Neurify [53], VeriNet [63], and nnenum [64], which are all supported by DNNV [38], and have performed well in recent benchmarks [65], [66]. In addition, each of these verifiers have the ability to return counter-examples.

4) Metrics: For each run of the falsifiers and verifiers we report the number of counter-examples found and the time to find each counter-example. To judge the quality of each counter-example we compute the mean reconstruction similarity (MRS) which, as discussed in §II, adapts ESRE to use the SSIM metric. Given a reference VAE, $\mathcal{V}$, MRS computes for a given input, $\boldsymbol{x}$, the expected similarity for a set of reconstructed inputs, $MRS(\boldsymbol{x}, \mathcal{V}) = \frac{1}{N}\sum_{i=1}^{N} SSIM(\mathcal{V}(\boldsymbol{x}), \boldsymbol{x})$. In this work, we estimate the mean using a sample size, N, of 100 reconstructions.

For each problem domain we also require a VAE model to use as a ground truth for measuring the MRS. For Fashion-MNIST we trained a fully-connected VAE model, VAE_{MRS}, with a 100-dimensional latent space, and symmetric encoder and decoder, each with two hidden layers, one of 256 neurons and one of 512 neurons, and ReLU activations. The decoder uses a Sigmoid activation so that output values are in the range 0 to 1. We chose to use a model significantly larger than those used for DFV for evaluating MRS under the assumption that a larger model would be able to better model the distribution and thus provide accurate MRS measures for all models tested. For DroNet we trained a convolutional VAE model, Conv-VAE_{DroNet}, with symmetric encoder and decoder, and a 512 dimensional latent space. The decoder consists of 8 blocks, each composed of a convolutional transpose operation followed by batch normalization and an ELU activation, except for the final block, which uses a Sigmoid activation so that output values are in the range 0 to 1. We chose to use this model as the baseline for MRS, since we expected a convolutional model to perform well on the image data of the DroNet benchmark.

5) Computing Resources: The experiments in this work were run on nodes with Intel Xeon Silver 4214 processors at 2.20 GHz and 512GB of memory. For RQ1 and RQ2 each job was allowed to use 1 processor core and unrestricted memory, and had a time limit of 1 hour, while falsification jobs in RQ2 – exploring the factors of the VAE – had a time limit of 5 minutes. For RQ3, each job was allowed to use 2 processor cores and had a time limit of 1 hour.

V. Results

A. RQ-1: on DFV efficacy

In this first experiment, we quantitatively and qualitatively assess the effectiveness of DFV and its costs when applied in conjunction with 4 falsifiers and 3 verifiers.

Experimental Procedure. To answer RQ1, we use the GHPR-FMNIST benchmark. We run both the verifiers and falsifiers on this benchmark, with and without our DFV with VAE_{RQ1}. We designed VAE_{RQ1} so that all existing tools could successfully run on it. This meant that we had to constrain its size and type of activation functions so that existing verifiers could process it. More specifically, we design VAE_{RQ1} with a single hidden layer of 24 neurons in the decoder, and instead of a Sigmoid activation on the output, it uses an approximation of the Sigmoid function with ReLU activations, since, of the verifiers explored in this work, only VeriNet supported non-ReLU activation functions. The approximation used is as follows $Sigmoid(x) \approx ReLU(-ReLU(-0.25{*}x{+}0.5){+}1)$. We run each tool 5 times on every problem to account for random noise and we record the number of problems that return a `sat` result, indicating that a counter-example was found, as well as the MRS of each counter-example. Each falsification and verification job had a timeout of 1 hour and used a radius of 3 in the latent space.

Analysis and Findings. We start by examining the mean reconstruction similarity (MRS) measures for the counter-examples generated by DFV. The MRS values are computed based on their reconstruction with VAE_{MRS}. Fig. 4 shows box plots representing the distribution of the MRS of the counter-examples found by each of the 7 tools (x-axis) when applied to the original DNN (red) and the DNN with the VAE_{RQ1} decoder (blue) generated by DFV. We find that, across all tools, the use of a model with DFV renders counter-examples that are reconstructed better by VAE_{MRS} than those found in the original DNN. Indeed, the median MRS for the counter-examples found in the original DNN is under 0.1, while the median MRS for the tools applied with DFV is above 0.6. This implies that they are closer to the distribution learned by VAE_{MRS} and thus may be closer to the true input

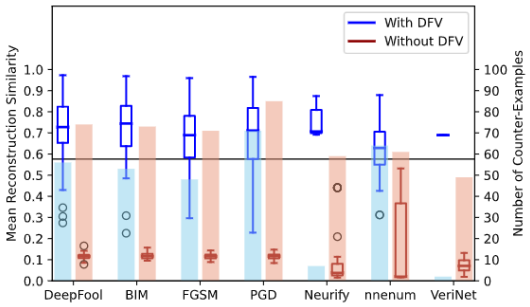


Fig. 4: MRS for generated counter-examples across falsifiers and verifiers, with and without DFV. Solid horizontal line indicates the median MRS of the test set images reconstructed with VAE_{MRS}. The shaded columns, measured in the y2-axis, represent the number of counter-examples found.

distribution. A statistical analysis of variance with the Kruskal-Wallis method[1] confirmed that the differences between using and not using DFV on any given tool are significant at p=0.05.

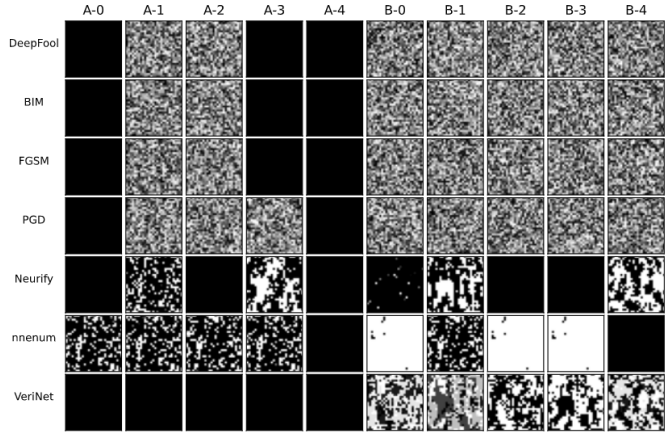
Fig. 4 also includes a horizontal line representing the median MRS of the FashionMNIST test set. This line provides another guideline to judge the quality of the counter-examples. Counter-examples found by the tools without DFV seem to be well below the median MRS of the test data, indicating that they are constructed poorly, likely due to being far from the distribution. Counter-examples found with DFV tend to have MRS higher than the median of the test set, indicating that they come from the distribution.

The shaded columns in Fig. 4, measured on the y2-axis, show the number of counter-examples found. These data show that, as expected, the number of counter-examples found when a tool is applied with a generative model decreases as irrelevant parts of the input space are pruned. For example, DeepFool found 74 counter-examples on the original DNN and 56 when applying DFV[2].

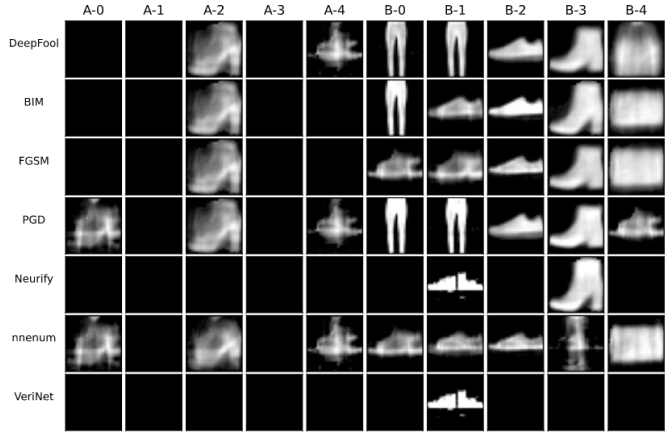
This portion of the study also revealed an interesting opportunity for verifiers. Based on the property design, we expected that the verifiers would not be able to prove a property of type B, and were unlikely to prove one of type A on the original DNN, which was indeed the case. However, when we used DFV, the nnenum verifier was able to prove 25 problems that held under the reduced input space encoded by the VAE_{RQ1}. This observation points to an opportunity for enabling verification to prove properties that may not hold over the whole input space but may certainly hold over the relevant input space as per the training distribution. In order for such an approach to be effective, further studies are needed to guarantee that the generative model encodes an overapproximation of the input distribution. We discuss this further in future work.

[1]We had to perform the non-parametric Kruskal-Wallis test given the different standard deviations observed across the distributions.

[2]One exception to this trend was nnenum, which reported a floating point error when attempting to verify many properties on the original DNN but it did not do the same with DFV. We conjecture that this is because DFV may be steering the the tool away from inputs that cause the failure. We will contact the developers to address this issue.



(a) Without DFV



(b) With DFV

Fig. 5: Counter-examples with highest MRS found for GHPR-FMNIST. Rows correspond to tools, columns represent properties.

We now qualitatively examine the counter-examples generated with and without DFV. The tabulated images in Fig. 5a are the counter-examples with the highest SSIM generated by each tool on the DNN, and the ones in Fig. 5b with DFV. Without using DFV, we see in Fig. 5a the images generated by all the falsifiers look like random noise, while the images generated by the verifiers have a bit more structure, with larger blocks of similarly valued pixels, but still have little discernible pattern. On the other hand, most of the counter-examples generated with DFV in Fig. 5b bear some resemblance to the training images (e.g., boots, pants, sandals), and some of them are clearly identifiable. We also notice that the counter-examples found with DFV for some properties correspond to distinct classes. We argue that when no counter-examples are found for a property with a model, but are found for the original DNN, like for Property A-1 and A-3, those counter-examples are likely to be invalid as they reside outside the data distribution. By the same token, when counter-examples are found with a model but not found without a model, like for Property A-4, we argue that the model reduction enables tools to explore the pruned space more extensively, enabling

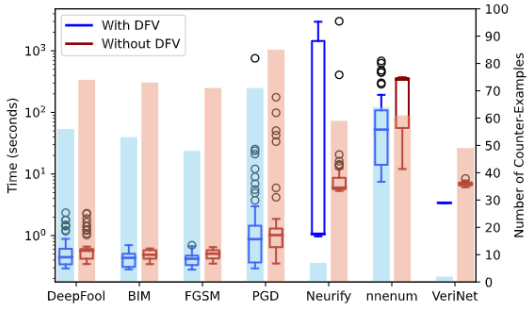


Fig. 6: Times to find counter-examples by each tool. Blue box plots represent the times of our approach, while red box plots represent the times for a DNN alone. The shaded columns, measured in the y2-axis, represent the number of counter-examples found.

their generation.

Last, we briefly examine the time distribution for each tool to generate the counter-examples. Fig. 6 presents box plots for each of the tools, and we again plot the number of counter-examples on the y2-axis. As expected, falsifiers are faster than verifiers. Looking at the 0.75 quartiles of the times spent by the different tools, we can see that all falsifiers took under 1.5 seconds, while the verifiers took up to 1444.5 seconds. PGD detected the most counter-examples, 85 on the original DNN and 71 with the approach, while its median execution time was just over a second. When comparing the boxes within a tool, we find that incorporating DFV did not have a major impact on the time taken by any of the tools[3].

Major Findings: Tools applied in conjunction with DFV generate fewer counter-examples that have a **x4** increase in MRS, in negligible time, and that visually appear to be much better aligned with the training distribution.

B. RQ-2: on VAE structure effects on DFV

We now explore the effects of the VAE's latent space size, number and size of layers, and radius from the center of the latent space distribution on the efficacy of DFV.

Experimental Procedure. To answer RQ2, we again use the GHPR-FMNIST benchmark while exploring several factors that may affect the efficacy of DFV. Given that there is a large number of configurations to explore, that the larger VAE configurations are not runnable by all verifiers, and that the performances of all falsifiers were similar in the first experiment, we selected to run all configurations and DFV with only the PGD falsifier. We first explore factors related to the VAE architecture, varying the size of the latent space, the number of hidden layers, and the size of each layer. We explored latent space sizes of 1, 2, 4, 8, 16, and 32; hidden layer counts of 1, 2, and 4; and layer sizes of 16, 32, 64, 128, and 256. For each combination of factors, we trained a VAE on the Fashion-MNIST data and transformed the correctness problems using DFV, and ran PGD on the resulting problems. We will refer to each as $VAE_{d,l,h}$, where d is the latent space size, l is the number of hidden layers, and h is the size of each hidden layer. The model $VAE_{8,2,256}$ has an 8-dimensional latent space with 2 hidden layers, each with 256 neurons. Second, we explore how the size of latent space region of the model affects the quality of the found counter-examples. We will specify the size of the input region by restricting the radius of the l^∞ d-ball in the latent space of the VAE. We will explore this factor with radii of 0.25 to 4, in 0.25 increments. To reduce the number of experiments, we use only the VAE that performed best in the first part of the experiment – with a high number of counter-examples found and high MRS. For this question, each falsification job was run 5 times to reduce the effects of random noise and each job was given a timeout of 5 minutes. For each combination of factors we report the number of counter-examples found, as well as the MRS of each counter-example.

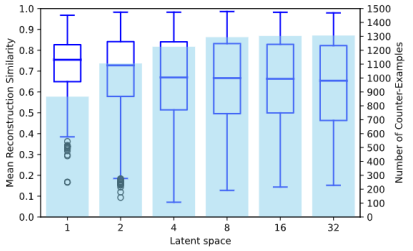
Analysis and Findings. We explored a total of 90 VAE configurations that work in conjunction with DFV. To control for randomness, we run each configuration five times.

We start by examining the effect of the latent space size, across all 15 of the VAE architectures, on the quality of counter-examples found, as measured by the MRS and the number of counter-examples found. Fig. 7a shows, across the latent spaces, that the median MRS varies between 0.65 and 0.75, and the number of counter-examples between 866 and 1306. The maximum possible number of counter-examples in this plot is 1500, since there are 15 VAE architectures for each latent space size, and 20 properties that were checked 5 times each for each architecture.

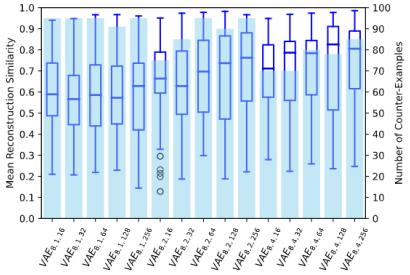
We observe that smaller latent spaces (LS=$\{1,2\}$) appear to generate counter-examples with slightly higher MRS, mainly because the model renders a less diverse set of images but of really good quality. The differences in MRS are confirmed with an ANOVA test of significance and a multiple comparison of latent space means with a Bonferroni correction across the latent spaces. More specifically, the MRS for LS=1 is significantly different from the rest of the latent spaces, and a LS=2 is significantly different from the rest, at p=0.05. We conjecture that larger spaces are able to encode richer data distributions enabling the generation of more and more diverse counter-examples that are sometimes farther from the distribution (i.e., a sandal that appears as printed on a shirt). Still, for this particular benchmark the gains in the number of counter-examples found and the losses in MRS seem to saturate after latent spaces of size 8. Across all of the latent spaces sizes, PGD required a median of 0.5 seconds to find counter-examples. The timing data for these experiments is available in the appendix.

We then selected the VAE architectures with LS=8, which contained the architecture with the tying highest MRS with the most counter-examples, to examine their variance. The x-axis of Fig. 7b contains the 15 VAE architectures we explored specified in the x-axis by the latent space size, the number of layers, and the number of neurons. We note that the architectures with more layers appear to be able to produce counter-examples with higher MRS. For example, the median

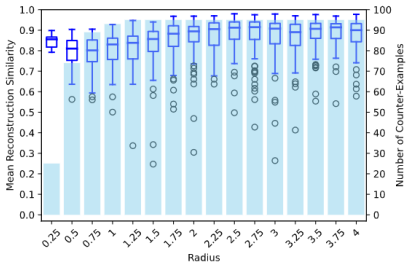
[3]The larger variation for Neurify can be attributed to the smaller number of counter-examples it generated.



(a) MRS of the counter-examples found by PGD using DFV for each latent space size. The blue columns show the number of counter-examples found.



(b) MRS of counter-examples found by PGD using different decoder architectures with a latent space of size 8.



(c) MRS of counter-examples found by PGD using DFV with $VAE_{8,2,256}$ across different radii.

Fig. 7: Study of impact of varying dimension, architecture, and radius of VAE in DFV.

for the architectures with 1 layer was 0.58, with 2 layers was 0.68, and with 4 layers was 0.79. An ANOVA confirms that the differences across architectures are significant at p=0.05, and a pair-wise comparison with a Bonferroni correction reveals that all the architectures with 1 layer are significantly different from the ones with 4 layers. The figure also seems to indicate that, given the same number of layers, having more neurons would render slightly higher MRS. For example, the median for the architecture with 16 neurons was 0.59 and for the three architectures with 256 neurons it was 0.71. We notice, however, that the number of counter-examples found was higher when fewer layers are used. We conjecture that having more layers further restricts the size of the input space learned by the VAE, perhaps due to the extra expressive power of the additional layers. As with latent spaces, the time to find counter-examples did not vary significantly across architectures, with most of them requiring a median of less than 1 second to find a counter-example. The timing data for these experiments is available in the appendix.

The last piece of this experiment explores changing the radius of the constraints in the latent space. We examine the effect of such changes on $VAE_{8,2,256}$, the architecture with the most counter-examples and greatest MRS in Fig. 7b. Fig. 7c shows that the MRS slightly increases from the lowest to the highest bounds, from 0.85 for a radius of 0.25 to 0.91 for a radius of 2.75. An ANOVA test across radii was significant at p=0.05, and a multple comparison with a Bonferroni correction showed that values between 0.05 and 1.25 were deemed non-significantly different from each other but significantly different from the higher radii (more details are provided in the appendix). We also note that the number of counter-examples found increases as the space to explore around the captured data distribution increases from a radius of 0.25 (25 counter-examples found) to 1.25 (95 counter-examples found).

Major Findings: VAE configurations with very limited capacity (in layers, neurons, or latent space size) can have a noticeable effect on the DFV effectiveness, specially in the number of counter-examples being found. If more counter-examples are desirable, then one should increase the dimensionality of the latent space, reduce the number of layers, and increase the radius. If higher-quality counter-examples are more desirable, then favoring a lower-dimensional latent space and smaller radius, and reduce the number of layers.

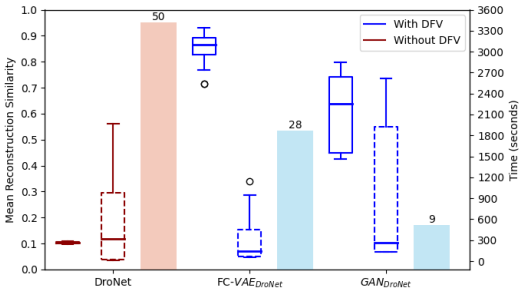


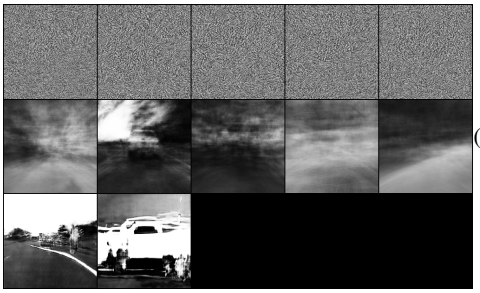
Fig. 8: The mean reconstruction similarity (solid box), time (dashed box), and numbers of counter-examples (color bar) to the DroNet properties.

C. *RQ-3: on* DFV *Scalability*

In this experiment, we assess the scalability of DFV by applying it to a large DNN model for autonomous UAV control using 3 different input distribution models.

Experimental Procedure. To answer RQ3, we use the larger and more complex GHPR-DroNet benchmark. We apply the PGD falsifier to the benchmark, both as is, and using DFV both with a VAE model, as well as with a GAN as the generative model. We train a fully-connected VAE, FC-VAE_{DroNet} with a symmetric encoder and decoder. The decoder of FC-VAE_{DroNet} has 6 hidden layers in the decoder with sizes 512, 512, 512, 512, 1024, and 2048, all with ELU activations, except the final layer which uses a Sigmoid activation. For GAN_{DroNet} we train a DCGAN [67] model on the DroNet dataset [39] with a Sigmoid on the final layer. Both models use a 512 dimensional latent space. As before, we run each falsifier 5 times to account for random noise and we record the number of counter-examples found and the time to find each counter-example. Each job had a timeout of 1 hour.

Analysis and Findings. Fig. 8 shows box plots with solid outlines for the distributions of the reconstruction similarities of counter-examples found using PGD on the DroNet DNN without DFV, as well as using DFV with the decoder of FC-VAE_{DroNet} and the generator of GAN_{DroNet}. Fig. 8 also shows the number of counter-examples found using each



(a) No model

(b) FC-VAE_{DroNet}

(c) GAN_{DroNet}

Fig. 9: Counter-examples for the first 5 DroNet properties with 3 distinct input models from the final run of the falsifier on each property.

model using bars with the count labeled above each bar. We find that, for DFV with both models, while fewer counter-examples are found, they clearly have higher reconstruction similarities than those found using the DroNet model alone. Indeed, the MRS differences between DroNet, FC-VAE_{DroNet}, GAN_{DroNet} are shown to be statistically significant overall by a Kruskal-Wallis test with p=0.05, and so do their pairwise differences. Corroborating the previous findings, this implies that the counter-examples found using DFV are closer to the distribution learned by Conv-VAE_{DroNet}, the model used to compute the MRS values, and thus may be closer to the actual input distribution. Without DFV, violations were found for all 10 properties across all 5 seeds. Using FC-VAE, 28 violations were found for 6 properties. Using GAN, 9 violations were found across 2 properties. While the lowest MRS for a counter-example found using DFV was 0.42, the MRS without DFV never exceeded 0.11.

We now proceed to visually examine the counter-examples generated with and without DFV for 5 properties. Fig. 9 shows counter-examples generated by PGD. The images generated without DFV look like random noise, while the images generated with DFV, independent of the chosen model, have structure and contain features seen in the training images such as roads, trees, or horizon lines. The model used for DFV has an impact on the images produced. While the VAE model tended to produce blurrier counter-examples, the GAN model produced counter-examples with sharper lines, but fewer recognizable road features.

Finally, Fig. 8 shows box plots, with dashed outlines, of the time to generate each counter-example using each model. The median time to falsify DroNet alone was 321 seconds, while DFV with FC-VAE_{DroNet} took 146 seconds and DFV with GAN_{DroNet} took 259 seconds, but there is enough performance variance that those differences are not deemed statistically significant.

Major Findings: DFV can be applied with various models without significant time penalty, while also producing counter-examples with up to **9x** gain in reconstruction similarity.

D. Threats to Validity

External Validity. Three threats to the generalization of our findings are our choice of tools, benchmarks, and generative models to evaluate DFV. We mitigate the concern about tools generality by selecting multiple falsifiers and verifiers as part of the first research question. For the next questions we traded generality across tools for more insights about the performance of DFV under different models, which implied that we had to drop DNN verifiers from the rest of the assessment because they did not scale to the networks and models we were targeting. Regarding benchmarks, we selected ones from different domains, one a classification task, and the other being a regression task, with very different architectures and training data. To mitigate the threat about model selection, we explored an extensive set of models in a systematic way. Still, the examination of more generative models is part of the future work.

Construct Validity. Our choice of MRS as a quality measure and our personal qualitative judgment of generated counter-examples pose a threat in that the relevance of a counter-example could be judged by many means. We mitigated this threat by basing MRS on a popular measure, ESRE, and specializing it to images with SSIM. We also provide results using ESRE in the appendix, and we will explore additional measures in the future and perform studies with users to help us judge the counter-examples quality.

Internal Validity. Our training processes for the networks and the models constitute a threat to the internal validity of the study as their correctness could have affect the findings. We have documented and programmed those processes when possible through scripts to facilitate their reproduction. We also mitigate this threat by making our data and scripts for running our experiments and analyzing our results publicly available[4]. Another threat to validity is the randomness involved in training of networks and models, and in the tools' performance. We mitigated that threat by running those tools multiple times and showing their variability.

VI. Conclusion

This work introduces a novel approach, DFV, which enables existing DNN verification and falsification techniques to target the data distribution. DFV composes learned latent variable generative distribution models with the DNN under analysis, reformulating the problem so that generated counter-examples are on the data distribution and that counter-examples that are off the distribution are not reported. We explore different data distribution models and find that using even simple models yield substantially better counter-examples across a range verification and falsification techniques for two different benchmarks.

These findings along with recent work on distribution-aware testing [9], [7], suggest that models of the data distribution can play an important role in V&V of DNNs. We plan to pursue further work along these lines. For example, how performance metrics for latent variable generative models that assess their precision and recall [43], can guide the development of distribution models that are customized to best suite different V&V activities for DNNs.

[4]https://anonymous.4open.science/anonymize/DFV-Artifact-1164

REFERENCES

[1] 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, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, 2018, pp. 303–314. [Online]. Available: http://doi.acm.org/10.1145/3180155.3180220

[2] 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, Shanghai, China, October 28-31, 2017*, 2017, pp. 1–18. [Online]. Available: http://doi.acm.org/10.1145/3132747.3132785

[3] A. Odena, C. Olsson, D. Andersen, and I. Goodfellow, "TensorFuzz: Debugging neural networks with coverage-guided fuzzing," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, K. Chaudhuri and R. Salakhutdinov, Eds., vol. 97. Long Beach, California, USA: PMLR, 09–15 Jun 2019, pp. 4901–4911. [Online]. Available: http://proceedings.mlr.press/v97/odena19a.html

[4] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: A coverage-guided fuzz testing framework for deep neural networks," in *28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2019.

[5] 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.

[6] D. Berend, X. Xie, L. Ma, L. Zhou, Y. Liu, C. Xu, and J. Zhao, "Cats are not fish: Deep learning testing calls for out-of-distribution awareness," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 1041–1052.

[7] S. Dola, M. B. Dwyer, and M. L. Soffa, "Distribution-Aware Testing of Neural Networks Using Generative Model," in *Proceedings of the International Conference on Software Engineering*, 2021.

[8] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang, "Deepgauge: multi-granularity testing criteria for deep learning systems," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, 2018, pp. 120–131. [Online]. Available: http://doi.acm.org/10.1145/3238147.3238202

[9] T. Byun and S. Rayadurgam, "Manifold for machine learning assurance," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ser. ICSE-NIER '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 97–100. [Online]. Available: https://doi.org/10.1145/3377816.3381734

[10] D. Shriver, S. G. Elbaum, and M. B. Dwyer, "Reducing DNN Properties to Enable Falsification with Adversarial Attacks," in *Proceedings of the International Conference on Software Engineering*, 2021.

[11] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski, "Simulation-based adversarial test generation for autonomous vehicles with machine learning components," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 1555–1562.

[12] S. R. Choudhary, A. Gorla, and A. Orso, "Automated test input generation for android: Are we there yet?(e)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 429–440.

[13] S. A. Khalek, G. Yang, L. Zhang, D. Marinov, and S. Khurshid, "Testera: A tool for testing java programs using alloy specifications," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 608–611.

[14] O. Tkachuk, M. B. Dwyer, and C. S. Pasareanu, "Automated environment generation for software model checking," in *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.* IEEE, 2003, pp. 116–127.

[15] D. Giannakopoulou, C. S. Pasareanu, and J. M. Cobleigh, "Assume-guarantee verification of source code with design-level assumptions," in *Proceedings. 26th International Conference on Software Engineering*. IEEE, 2004, pp. 211–220.

[16] C. Cadar, D. Dunbar, D. R. Engler *et al.*, "Klee: unassisted and automatic generation of high-coverage tests for complex systems programs." in *OSDI*, vol. 8, 2008, pp. 209–224.

[17] P. Dhaussy, J.-C. Roger, and F. Boniol, "Reducing state explosion with context modeling for model-checking," in *2011 IEEE 13th International Symposium on High-Assurance Systems Engineering*. IEEE, 2011, pp. 130–137.

[18] D. Kroening and M. Tautschnig, "Cbmc–c bounded model checker," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2014, pp. 389–391.

[19] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "Esbmc 5.0: an industrial-strength c model checker," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, 2018, pp. 888–891.

[20] M. Markthaler, S. Kriebel, K. S. Salman, T. Greifenberg, S. Hillemacher, B. Rumpe, C. Schulze, A. Wortmann, P. Orth, and J. Richenhagen, "Improving model-based testing in automotive software engineering," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*. IEEE, 2018, pp. 172–180.

[21] C. S. Păsăreanu, M. B. Dwyer, and W. Visser, "Finding feasible counter-examples when model checking abstracted java programs," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2001, pp. 284–298.

[22] A. Gurfinkel, T. Kahsai, A. Komuravelli, and J. A. Navas, "The seahorn verification framework," in *International Conference on Computer Aided Verification*. Springer, 2015, pp. 343–361.

[23] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 3–18.

[24] W. Ruan, X. Huang, and M. Kwiatkowska, "Reachability analysis of deep neural networks with provable guarantees," in *IJCAI*. ijcai.org, 2018, pp. 2651–2659.

[25] G. Singh, R. Ganvir, M. Püschel, and M. T. Vechev, "Beyond the single neuron convex barrier for neural network certification," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 15 072–15 083. [Online]. Available: http://papers.nips.cc/paper/9646-beyond-the-single-neuron-convex-barrier-for-neural-network-certification

[26] G. Singh, T. Gehr, M. Püschel, and M. T. Vechev, "An abstract domain for certifying neural networks," *PACMPL*, vol. 3, no. POPL, pp. 41:1–41:30, 2019.

[27] W. Xiang, H. Tran, and T. T. Johnson, "Output reachable set estimation and verification for multilayer neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 11, pp. 5777–5783, Nov 2018.

[28] O. Bastani, Y. Ioannou, L. Lampropoulos, D. Vytiniotis, A. V. Nori, and A. Criminisi, "Measuring neural net robustness with constraints," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, ser. NIPS'16. USA: Curran Associates Inc., 2016, pp. 2621–2629. [Online]. Available: http://dl.acm.org/citation.cfm?id=3157382.3157391

[29] K. Dvijotham, R. Stanforth, S. Gowal, T. Mann, and P. Kohli, "A dual approach to scalable verification of deep networks," in *Proceedings of the Thirty-Fourth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-18)*. Corvallis, Oregon: AUAI Press, 2018, pp. 162–171.

[30] A. Raghunathan, J. Steinhardt, and P. Liang, "Certified defenses against adversarial examples," in *ICLR*. OpenReview.net, 2018.

[31] V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HyGIdiRqtm

[32] E. Wong and J. Z. Kolter, "Provable defenses against adversarial examples via the convex outer adversarial polytope," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 5283–5292.

[33] T. Weng, H. Zhang, H. Chen, Z. Song, C. Hsieh, L. Daniel, D. S. Boning, and I. S. Dhillon, "Towards fast computation of certified robustness for relu networks," in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 5273–5282.

[34] X. Huang, M. Kwiatkowska, S. Wang, and M. Wu, "Safety verification of deep neural networks," in *CAV (1)*, ser. Lecture Notes in Computer Science, vol. 10426. Springer, 2017, pp. 3–29.

[35] G. Katz, C. W. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer, "Reluplex: An efficient SMT solver for verifying deep neural networks," in *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, 2017, pp. 97–117. [Online]. Available: https://doi.org/10.1007/978-3-319-63387-9_5

[36] R. Ehlers, "Formal verification of piece-wise linear feed-forward neural networks," in *Automated Technology for Verification and Analysis - 15th International Symposium, ATVA 2017, Pune, India, October 3-6, 2017, Proceedings*, 2017, pp. 269–286. [Online]. Available: https://doi.org/10.1007/978-3-319-68167-2_19

[37] R. R. Bunel, I. Turkaslan, P. H. S. Torr, P. Kohli, and P. K. Mudigonda, "A unified view of piecewise linear neural network verification," in *NeurIPS*, 2018, pp. 4795–4804.

[38] D. Shriver, D. Xu, S. G. Elbaum, and M. B. Dwyer, "DNNV," 2020, https://github.com/dlshriver/DNNV. [Online]. Available: https://github.com/dlshriver/DNNV

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

[40] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[41] 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 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680. [Online]. Available: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf

[42] M. F. Naeem, S. J. Oh, Y. Uh, Y. Choi, and J. Yoo, "Reliable fidelity and diversity metrics for generative models," in *International Conference on Machine Learning*. PMLR, 2020, pp. 7176–7185.

[43] A. M. Alaa, B. van Breugel, E. Saveliev, and M. van der Schaar, "How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models," *arXiv preprint arXiv:2102.08921*, 2021.

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

[45] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[46] L. Ruff, J. R. Kauffmann, R. A. Vandermeulen, G. Montavon, W. Samek, M. Kloft, T. G. Dietterich, and K.-R. Müller, "A unifying review of deep and shallow anomaly detection," *Proceedings of the IEEE*, 2021.

[47] 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.

[48] A. Vasilev, V. Golkov, M. Meissner, I. Lipp, E. Sgarlata, V. Tomassini, D. K. Jones, and D. Cremers, "q-space novelty detection with variational autoencoders," in *Computational Diffusion MRI*, E. Bonet-Carne, J. Hutter, M. Palombo, M. Pizzolato, F. Sepehrband, and F. Zhang, Eds. Cham: Springer International Publishing, 2020, pp. 113–124.

[49] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.

[50] B. Dai and D. P. Wipf, "Diagnosing and enhancing VAE models," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. [Online]. Available: https://arxiv.org/abs/1903.05789

[51] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. J. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2014.

[52] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 9, pp. 2805–2824, 2019.

[53] S. Wang, K. Pei, J. Whitehouse, J. Yang, and S. Jana, "Efficient formal safety analysis of neural networks," in *NeurIPS*, 2018, pp. 6369–6379.

[54] G. Katz, D. A. Huang, D. Ibeling, K. Julian, C. Lazarus, R. Lim, P. Shah, S. Thakoor, H. Wu, A. Zeljić *et al.*, "The Marabou framework for verification and analysis of deep neural networks," in *International Conference on Computer Aided Verification*, 2019, pp. 443–452.

[55] C. Liu, T. Arnon, C. Lazarus, C. Barrett, and M. J. Kochenderfer, "Algorithms for verifying deep neural networks," *CoRR*, vol. abs/1903.06758, 2019.

[56] D. Shriver, S. G. Elbaum, and M. B. Dwyer, "DNNF," 2021, https://github.com/dlshriver/DNNF. [Online]. Available: https://github.com/dlshriver/DNNF

[57] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 876–888. [Online]. Available: https://doi.org/10.1145/3368089.3409730

[58] T. Byun, A. Vijayakumar, S. Rayadurgam, and D. Cofer, "Manifold-based test generation for image classifiers," in *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. IEEE, 2020, pp. 15–22.

[59] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[60] A. Kurakin, I. J. Goodfellow, and S. Bengio, "Adversarial examples in the physical world," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings*. OpenReview.net, 2017.

[61] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: A simple and accurate method to fool deep neural networks," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 2574–2582.

[62] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[63] P. Henriksen and A. Lomuscio, "Efficient neural network verification via adaptive refinement and adversarial search," in *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, ser. Frontiers in Artificial Intelligence and Applications, G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, Eds., vol. 325. IOS Press, 2020, pp. 2513–2520. [Online]. Available: https://doi.org/10.3233/FAIA200385

[64] S. Bak, H.-D. Tran, K. Hobbs, and T. T. Johnson, "Improved geometric path enumeration for verifying relu neural networks," in *Computer Aided Verification*, S. K. Lahiri and C. Wang, Eds. Cham: Springer International Publishing, 2020, pp. 66–96.

[65] D. Xu, D. Shriver, M. B. Dwyer, and S. Elbaum, "Systematic generation of diverse benchmarks for dnn verification," in *Computer Aided Verification CAV*, 2020.

[66] C. Liu and T. T. Johnson, "Vnn-comp," https://sites.google.com/view/vnn20/vnncomp. [Online]. Available: https://sites.google.com/view/vnn20/vnncomp

[67] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2016. [Online]. Available: http://arxiv.org/abs/1511.06434