Targeted Deep Learning System Boundary Testing

OLIVER WEIßL, fortiss GmbH, Munich, Germany and Technische Universität München, Munich, Germany
AMR ABDELLATIF, Technische Universität München, Munich, Germany
XINGCHENG CHEN, fortiss GmbH, Munich, Germany and Technische Universität München, Munich, Germany
GIORGI MERABISHVILI, North Carolina State University at Raleigh, Raleigh, NC, USA
VINCENZO RICCIO, University of Udine, Udine, Italy
SEVERIN KACIANKA, fortiss GmbH, Munich, Germany
ANDREA STOCCO, School of Computation, Information and Technology, Technische Universität München, Munich, Germany and fortiss GmbH, Munich, Germany

Evaluating the behavioral boundaries of deep learning (DL) systems is crucial for understanding their reliability across diverse, unseen inputs. Existing solutions fall short as they rely on untargeted, random perturbations with limited controlled input variations. In this work, we introduce Mimicry, a novel black-box test generator for fine-grained, targeted exploration of DL system boundaries. Mimicry performs boundary testing by leveraging the probabilistic nature of DL outputs to identify promising directions for exploration. By using style-based GANs to disentangle inputs into content and style components, Mimicry generates boundary test inputs by mimicking features from both source and target classes. We evaluated Mimicry's effectiveness in generating boundary inputs for five DL image classification systems, comparing it to two baselines from the literature. Our results show that Mimicry consistently identifies inputs up to 25× closer to the theoretical decision boundary, outperforming the baselines with statistical significance. Moreover, it generates semantically meaningful boundary test cases that reveal new functional misbehaviors, while the baselines mostly produce corrupted or invalid inputs. Thanks to its enhanced control over latent space manipulations, Mimicry remains effective as dataset complexity grows, resulting in up to 36% higher validity rate and competitive diversity, as supported by a comprehensive human assessment.

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

This research was partially supported by the Bavarian Ministry of Economic Affairs, Regional Development and Energy, and the Practical Research Experience Program (PREP) of the Technical University of Munich. Vincenzo Riccio is supported by the Project SOP CUP N. H73C22000890001 PNRR M4 C2 I1.3 "SEcurity and RIghts in the CyberSpace (SERICS)" PE0000014 PE7 funded by Next-Generation EU.

Authors' Contact Information: Oliver Weißl (corresponding author), fortiss GmbH, Munich, Germany and Technische Universität München, Munich, Germany; e-mail: weissl@fortiss.org; Amr Abdellatif, Technische Universität München, Munich, Germany; e-mail: amr.abdellatif@tum.de; Xingcheng Chen, fortiss GmbH, Munich, Germany and Technische Universität München, Munich, Germany; e-mail: xchen@fortiss.org; Giorgi Merabishvili, North Carolina State University at Raleigh, Raleigh, NC, USA; e-mail: gmerabi@ncsu.edu; Vincenzo Riccio, University of Udine, Udine, Italy; e-mail: vincenzo.riccio@uniud.it; Severin Kacianka, fortiss GmbH, Munich, Germany; e-mail: kacianka@fortiss.org; Andrea Stocco, School of Computation, Information and Technology, Technische Universität München, Munich, Germany and fortiss GmbH, Munich, Germany; e-mail: andrea.stocco@tum.de.

ACM 1557-7392/2026/6-ART177
https://doi.org/10.1145/3771557

Additional Key Words and Phrases: DL testing, boundary testing, generative AI, search-based optimization

ACM Reference format:
Oliver Weißl, Amr Abdellatif, Xingcheng Chen, Giorgi Merabishvili, Vincenzo Riccio, Severin Kacianka, and Andrea Stocco. 2026. Targeted Deep Learning System Boundary Testing. *ACM Trans. Softw. Eng. Methodol.* 35, 7, Article 177 (June 2026), 31 pages.
https://doi.org/10.1145/3771557

1 Introduction

The increasing dependence on **Deep Learning (DL)** systems for both everyday tasks and critical sectors [60] makes rigorous testing for these systems a relevant topic [56, 73]. The concept of fault in DL systems is more complex than in traditional software [56]. Even if the code that builds the DL network is bug-free, the trained DL model may still deviate from the expected behavior due to faults introduced during the training phase, such as the misconfiguration of learning parameters or the use of an unbalanced or non-representative training set [25].

In data-intensive software systems, such as DL systems, faults often stem from the large, high-dimensional input space, which requires the generation of test data that accurately captures the complexity and diversity of the validity domain, i.e., the portion of the input space for which the system is expected to operate [56].

Test generation techniques have been developed to induce misbehaviors in DL systems [53, 56, 57, 63, 65, 73, 74]. However, the objectives of these techniques are often quite different. Some techniques focus on finding adversarial examples [9, 20, 36, 72], while other solutions aim to achieve high failure exposure and/or high values of DL-specific adequacy metrics, such as neuron [44] or surprise coverage [33], or explore the decision boundaries of the DL system [26, 57].

In particular, DL boundary testing targets regions of the input space where small input variations can lead to changes in behavior. These changes may reflect misbehavior—especially when the input remains semantically valid to a human observer—or they may simply indicate increased ambiguity near the boundary between valid and invalid inputs, where expected behavior becomes less clearly defined. Boundary inputs are crucial for evaluating the DL system's reliability, as they often expose how it handles edge cases, transitions between operational domains, and critical decision-making regions.

In traditional software systems, boundary testing is typically targeted. For example, consider a Java method `sum(x, y)` that adds two integers, where each parameter ranges from -2^{32} to 2^{31}. A boundary testing strategy for this method would include inputs such as the minimum allowed value (-2^{32}), its immediate successor ($-2^{32}+1$), an arbitrary in-range value (e.g., 100), the maximum allowed value (2^{31}), and its immediate predecessor ($2^{31}-1$). Since there are two parameters, this targeted approach yields only $5^2 = 25$ combinations, covering edge behaviors that are most likely to reveal bugs.

In contrast, boundary testing for DL systems is challenging due to high-dimensional, unconstrained input spaces (e.g., images) and unclear input space partitions. As such, existing solutions such as DeepJanus [57] and Sinvad [27] rely on untargeted boundary testing strategies. These are commonly driven by evolutionary algorithms that generate diverse inputs without any explicit consideration of specific source or target classes. While these methods can uncover unexpected behaviors, they tend to be inefficient and unfocused, as they treat the entire input space uniformly rather than concentrating on regions near critical decision boundaries. However, DL models inherently learn decision boundaries between classes. For instance, in a digit classification task, given an image of class 5, the DL model may assign high probabilities to both classes 5 and 6, reflecting the

probabilistic nature of the model's output rather than a definitive classification [67]. This suggests the model is uncertain between these two classes, making inputs from class 6 promising candidates for generating boundary cases. Thus, it is potentially more effective to focus testing on inputs near the classifier's decision boundary between classes that share some features, like 5 and 6, rather than sampling randomly across unrelated classes. Despite this potential, targeted boundary testing in DL systems remains largely unexplored.

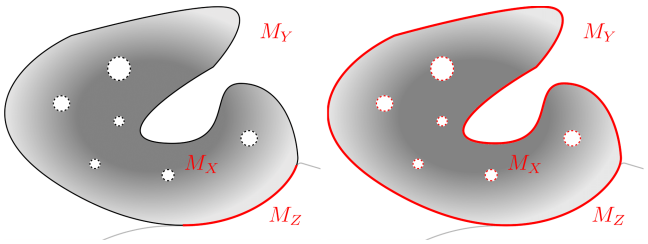
While researchers have explored various approaches, existing solutions have key limitations that hinder their effectiveness in boundary testing of complex DL systems for image recognition tasks. An example is DeepJanus [57], an input generation technique that relies on an abstract representation of the input domain (i.e., a model) to generate test cases. However, such domain models are typically unavailable for complex, feature-rich datasets such as ImageNet. Although recent advances in generative AI have addressed the lack of explicit input models, current techniques for generating inputs in the latent space of DL models [13–15, 26] either do not target boundary inputs, or they offer limited control over the generation process due to the use of a single, entangled latent vector perturbed by random noise [27], thereby severely constraining the ability to navigate the latent space.

In this article, we propose a technique to explore the boundary of image classification DL systems in the latent space of style-based **Generative Adversarial Networks (GANs)**. The key idea involves leveraging a style transfer architecture that automatically learns the separation of high-level features (e.g., shape) from lower-level ones (e.g., texture). While this architecture is primarily used for the generation of new, highly diverse datasets of complex inputs, in this work, we leverage the scale-specific control on the synthesis of disentangled latent factors for boundary testing of DL systems.

Our technique, Mimicry [1], uses style-specific interpolation operations to find boundary inputs. Mimicry uses a conditional StyleGAN [30] model trained to learn the class-wise visual characteristics of a given image dataset across all its inputs. StyleGAN maps latent vector inputs to an intermediate latent vector, which controls the image style at various granularity levels in the generative process.

The main idea of Mimicry involves the systematic mutation of the pre-defined set of latent vectors between source and target inputs using scale-specific interpolation and assessing the impact of these modifications in the image space. Moreover, Mimicry facilitates the targeted generation of boundary inputs by leveraging model confidence scores. Given a source input, Mimicry identifies the boundary target as the class with the second-highest predicted confidence from the DL system. It then establishes a closed feedback loop between the DL model under test and the StyleGAN network to guide input synthesis. Specifically, Mimicry employs StyleGAN to generate representative samples of the target class and manipulates the latent representation of the source input to incorporate features of these target samples, thereby adjusting its visual characteristics toward the decision boundary.

We have evaluated the effectiveness of Mimicry on five popular image classification datasets with increasing complexity (MNIST [39], FashionMNIST [69], SVHN [50], CIFAR-10 [35], and ImageNet [10]) to assess its robustness across a diverse range of visual patterns and challenges, using self- and pre-trained WideResNet [70] as DL systems under test. Additionally, we compare the effectiveness of Mimicry against the model-based DeepJanus [57] and the generative-based Sinvad [27]. Our experiments demonstrate that Mimicry consistently identifies inputs close to the decision boundary while maintaining a high validity rate and label preservation rate, as evaluated by human assessors. Moreover, Mimicry surpasses both DeepJanus and Sinvad in both quantitative and qualitative metrics, especially when increasing data complexity. Our paper makes the following contributions:



(a) Targeted Boundary Testing. (b) Untargeted Boundary Testing.

Fig. 1. Different types of boundary testing objectives for DL systems.

Technique: To the best of our knowledge, MIMICRY is the first targeted boundary testing technique for DL systems. MIMICRY is implemented in a publicly available replication package [1] and is based on a disentangled latent space representation that ensures high controllability.

Evaluation: An empirical study shows that MIMICRY is more effective than existing model-based and generative-based techniques in various quality metrics, including higher effectiveness, validity, and label-preservation rates.

2 Background

2.1 Testing Objectives for DL Systems

Testing methodologies to highlight behavior in DL models can have vastly different objectives. The distinctions are often unclear in the related literature. Therefore, we define key terms and specify the experimental domain. We illustrate these differences using a classifier manifold M. When processing inputs, the classifier maps all input elements onto M, resulting in the final predictions. Within this manifold, distinct regions $M_\bullet$ (sub-manifolds) exist, corresponding to the ground truth of each classifiable class. In Figure 1, the main sub-manifold for class X, M_X, is in focus, showing its "boundaries" to other regions on M and internal boundaries, which symbolize adversarial regions [64]. Specifically, Figure 1 illustrates the difference between targeted and untargeted boundary testing through the red highlights along the boundaries identified by each strategy. In the targeted approach, only a localized portion of the overall boundary is explored, as the search is conditioned on a specific target class. In contrast, untargeted testing is class-agnostic and aims to discover a broader range of decision boundaries across the manifold.

In this work, we explore boundary testing, a subset of functional testing, which targets the *generalizability* aspect of the SUT by generating functionally new inputs. Particularly, we aim to find boundary candidates, which are inputs that lie near decision boundaries, where small changes can cause prediction changes. These may occur either on true boundaries between classes or within adversarial regions, which are areas where small or semantically imperceptible perturbations cause misclassifications. While our focus is not on generating adversarial examples, some of our boundary candidates may fall within such regions. Approaches targeting adversarial input generation, such as DeepXplore [53], DLFuzz [20], and DeepTest [65], involve raw input manipulation techniques that modify/corrupt the original inputs (e.g., pixels). These techniques do not generate new functional inputs, as they produce changes in the original inputs and are therefore suitable to test the deficiencies in robustness of the DL system [46, 58], such as the discovery of adversarial regions. In contrast, the generation of functional tests focuses on creating new inputs that deviate from the original training distribution. These inputs target the long-tail problem of DL testing [71], testing the DNN's ability to generalize to novel, unseen scenarios. Instances of functional test generators

are the model-based approaches like DeepJanus [57], DeepHyperion [76], and DeepMetis [55] or latent space manipulation techniques like SINVAD [26, 27], CIT4DNN [13], and RBT4DNN [48].

2.2 Boundary Testing for DL Systems

Boundary testing identifies input samples near decision boundaries, where the classifier assigns equal, or near-equal, probabilities to multiple classes [26, 57]. The decision boundary of a classifier can be inferred from the predicted logits, where the theoretical boundary would be a perfect equilibrium in confidences between n classes ($n > 1$). In addition, boundary testing can be either targeted or untargeted. The goal of targeted boundary testing is to converge to the boundary between the origin class and a specified target class (e.g., M_Z in Figure 1(a)), while the goal of the untargeted case assumes no predetermined target class (Figure 1(b)).

While existing techniques such as DeepJanus [57] and Sinvad [26] focus on untargeted boundary testing, in our work, we focus on targeted boundary testing, with the goal of automatically retrieving inputs that are ambiguous in prediction, without restrictions on the input differences.

2.3 Style-Based GANs

GANs are DL models that learn the statistical distribution of training data, allowing the synthesis of new samples representative of the learned distribution [18].

GANs involve jointly training a pair of networks that compete with each other. This approach is based on game theory and is implemented using two neural networks. A first neural network, called the generator, aims to produce realistic images, while a second neural network, called the discriminator, acts as an expert that receives both fake and real (authentic) images and aims to distinguish between them. In this way, the generator improves its ability to produce realistic images to fool the discriminator, which can be leveraged for test generation [12, 15].

StyleGAN [29, 30, 32, 59] extends the GAN architecture to introduce new methods for controlling the image synthesis process. Unlike traditional GANs, StyleGAN enables style control at multiple levels within the network. The proposed changes to the generator model involve the use of a mapping network to map points in the initial latent space to an intermediate latent space. This intermediate latent space controls the intensity of the image features at various scales in the generator model, inspired by the style transfer literature [24]. This architectural change, combined with the noise injected directly into the network, enables the automatic, unsupervised separation of high-level attributes from stochastic variations in the generated images, which we exploit for boundary testing.

3 Methodology

Mimicry is a black-box approach[1] that leverages StyleGANs to generate boundary inputs through *targeted* optimization. Concretely, Mimicry uses four separate components:

— *SUT*: The system under test, for which the behavioral aspects should be explored.
— *Manipulator*: A component that adapts inputs to the SUT by generating new data points based on a given strategy.
— *Optimizer*: Responsible for providing strategies to the manipulator by evaluating the quality or fitness of previous strategies.
— *Objectives*: Metrics that quantify the quality of solutions, guiding the optimizer's search.

[1] A recent survey classifies methods that need access to both the training and test datasets of a learned component as *data-box* methods. However, to avoid confusion, we refer to these as black-box methods, since they do not utilize any internal information from the model itself [56].

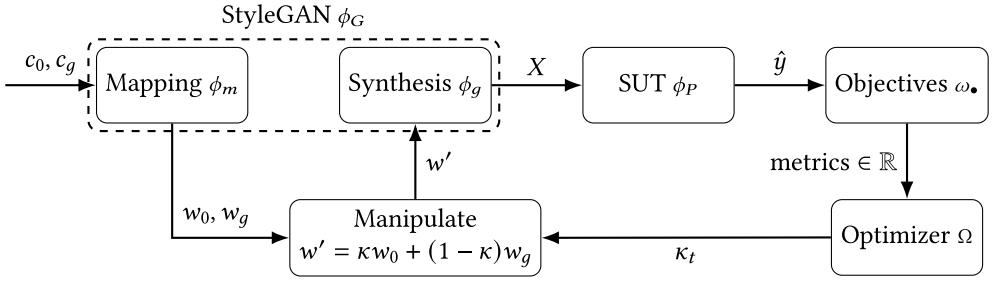


Fig. 2. Mimicry component interactions through optimization.

Algorithm 1: Mimicry

Input: $MaxGenerations$; # The number of optimization generations.
$c_0 \in C$; # The initial class.
$w_0,\ X,\ \hat{y} \leftarrow getValidSeed(c_0, \phi_G, \phi_P)$; # Generates seed and validates with SUT.
$c_g \leftarrow \hat{y}_2$; # Get second most likely class from prediction.
$w_g,\ \_,\ \_ \leftarrow getValidSeed(c_g, \phi_G, \phi_P)$; # Get target seed from secondary class.
$i \leftarrow 0$; # Set current generation to 0.
while $i < MaxGenerations$ **do**
  $w' \leftarrow \kappa w_0 + (1-\kappa) w_g$; # Generate manipulated seed using strategy.
  $X' \leftarrow \phi_g(w')$; # Generate new image using perturbed w-latent.
  $\hat{y}' \leftarrow \phi_P(X')$; # Predict class content of new image.
  $objective \leftarrow \omega(\kappa, \hat{y}', c_0, c_g)$; # Evaluate objective functions.
  $\kappa \leftarrow \Omega(\kappa, objective)$; # Adapt strategy based on objective values.
  $i \leftarrow i + 1$;
return X'_{best} ; *# Best image based on fitness.*

Mimicry operates by identifying boundary inputs based on feedback from the SUT's (e.g., a DL classifier) predictions. These boundary inputs are generated via latent space manipulations in a StyleGAN model trained on the same data as the classifier. These manipulations are driven by strategies optimized according to a set of objective functions (Figure 2).

The process is initialized by specifying the number of optimization generations and the initial class c_0 to test. An initial latent vector w_0 is sampled from the StyleGAN and used to generate the corresponding image X. If the predicted class of this image does not match the intended initial class, a new sample is drawn, as the current input is already failure-inducing. Once a valid initial image is obtained, its latent vector (seed) is iteratively optimized toward a boundary candidate by applying linear interpolations with another (target) latent vector.

Mimicry's targeted nature lies in its treatment of boundary discovery: instead of focusing solely on maximizing misclassifications [26, 27, 57], it identifies the second most probable class as the target. This initial bias toward a specific class allows Mimicry to exploit the proximity to a specific decision boundary. The target seed w_g, together with the original latent vector, is then manipulated according to a strategy κ, which is optimized by the optimizer using defined objectives $\omega_\bullet$. By leveraging style-based latent manipulations, our technique "mimics" characteristics of multiple classes to explore decision boundaries, revealing subtle misbehaviors in DL systems through controlled feature mixing.

An algorithmic description of Mimicry can be found in Algorithm 1. Here, $objective \in \mathbb{R}$ consists of values for each objective function in ω, and $getValidSeed(\cdot, \phi_G, \phi_P)$ repeatedly generates new seeds until the condition (Equation (1)) is satisfied. This condition ensures that images from the

generator ϕ_G are predicted as their intended class by the SUT ϕ_P. Here c denotes the identity of any class.

$$\text{argmax } (\phi_P \circ \phi_G)(c) \stackrel{?}{=} c. \tag{1}$$

In the following, we will describe in detail each component of Mimicry.

3.1 System under Test

The first component used by Mimicry is the SUT. In this work, we target DL classification systems, as they inherently involve the notion of classes—and consequently, boundaries between classes, which is a requirement for performing boundary testing.

We denote the classifier as ϕ_P, a trainable map from image input to the set of possible classes C. Specifically, the output is a vector of class probabilities $\mathbb{R}^i \xrightarrow{\phi_P} \mathbb{R}^{||C||}$. Here, the superscript i denotes the shape of the input i. We can think of the classification operation as positioning our input on the classifier manifold, with the location being the predicted class confidences. On this classifier manifold, Mimicry aims to find boundaries between regions of different classes. The boundaries are regions where the classifier's confidence $\hat{y} \in \mathbb{R}^{||C||}$ is equidistant between two or more classes in the set of targeted classes C_t (Equation (2)). For example, a boundary between class X and class Y has the target classes set to $C_t = \{c_X, c_Y\}$, where each c_i denotes the index of class i. Note here that $\sum \hat{y} = 1$.

$$\forall c \in C_t, \quad \hat{y}_c = \frac{1}{||C_t||}. \tag{2}$$

3.2 Manipulator

To find boundary cases, Mimicry manipulates latent vectors from a conditional StyleGAN model. StyleGANs were specifically chosen because their disentangled latent space offers greater control over manipulations. Unlike traditional GAN architectures, where a single (noise) latent vector is used to generate outputs [18], StyleGAN uses a latent vector that passes through an additional network called the mapping network ϕ_m, which consists of multiple fully connected layers. This mapping network "disentangles" the latent space by distributing learned image characteristics across the layers of an intermediate latent vector w. The number of layers in w depends on the specific StyleGAN architecture, with more layers enabling finer control over image manipulations. In contrast, traditional GANs and VAEs can be seen as having only a single such layer in w, which limits the degree of control. The StyleGAN model generates new images from class information (Equation (3)) and can be denoted as a composition of a mapping network ϕ_m and a synthesis network ϕ_g (Equation (4)). We refer to the combination of both as the generator network ϕ_G. We denote the mapping network as $\phi_m(\cdot)$, where the only explicitly given input is the class information, as z is sampled noise.

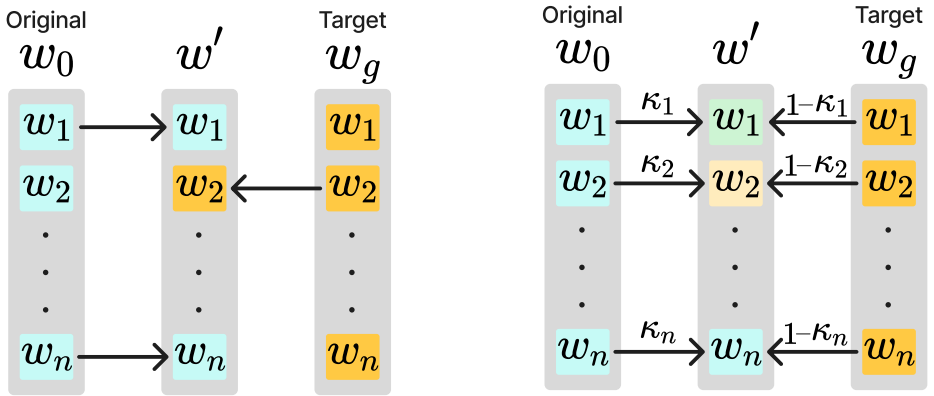
$$C \xrightarrow{\phi_G} \mathbb{R}^i. \tag{3}$$

$$\phi_G = \phi_g \circ \phi_m. \tag{4}$$

To get from a class to a generated image in StyleGAN, we sample a latent vector $z \sim \mathcal{N}$, which then is processed by the mapping network in combination with the class information to generate an intermediate latent vector w (Equation (5)). The latent vector w is then used for manipulation, as it can be separated into multiple independent layers, depending on the StyleGAN's architecture. The advantage of this conversion is that the manipulation of latent vectors z may produce erratic changes in the image, as observed in previous studies [13, 15, 26, 58].

$$C \times z \xrightarrow{\phi_m} w \xrightarrow{\phi_g} \mathbb{R}^i. \tag{5}$$

Fig. 3. Mixing features of an original image (blue car) with those of a target image (white truck) produces different outputs depending on the latent layers.



(a) Style Mixing. (b) Style Interpolation.

Fig. 4. Latent manipulation strategies.

Additionally, as noted by Karras et al. [32], the rank of manipulations in the intermediate latent space of StyleGANs allows for control at different levels of granularity in the generated images. Specifically, the initial layers of the intermediate latent vector w tend to control coarse features such as overall shape and perspective. The intermediate layers influence medium-scale attributes, including textures and finer structural details. Finally, the last layers typically affect only color schemes, making them responsible for the most fine-grained manipulations. An example of these effects can be seen in Figure 3, where layers of the original latent vector are mixed with elements of a differing target vector (here, car vs. truck in CIFAR-10). While the exact layer assignments are specific to each StyleGAN implementation, the general behavior is consistent across all StyleGAN architectures, as they share the same type of intermediate latent space w, albeit with varying numbers of layers [29, 30, 32, 59]. The intermediate latent vectors w can be subsequently synthesized into an image using the synthesis network ϕ_g.

3.3 Optimizer

We combine the latent vector of the initial class w_0 and the latent vector of a target class w_g into a new latent vector $w' = \kappa w_0 + (1 - \kappa) w_g$ (see Figure 4(b)), where κ is the manipulation strategy computed by the optimizer. To find adequate manipulation strategies, MIMICRY uses the AGE-MOEA-2 optimizer [51], known for its outstanding performance with one or multiple objectives. Contrary to the original style mixing approach of Karras et al. [30], the produced strategies κ do not swap individual layers (Figure 4(a)), but they are weights for linear interpolation between two layers of the same rank in two latent vectors (Figure 4(b)). This increases the controllability of feature mixing between the source and target seed, which is useful for precise DL boundary assessment.

The latent vectors for interpolation (seeds) in our case are selected as follows: The first seed is of the original class w_0, whereas the second seed w_g is dependent on the second most likely prediction of the primary seed by the SUT. This reuse of SUT behavior allows for a more targeted boundary search, as Mimicry incorporates knowledge of the decision space that is traversed. The rationale for selecting the second most likely class is that its decision boundary is likely to be closest to that of the predicted class, given that the classifier already assigns it the second highest probability [57]. At the start of optimization, the manipulation strategies κ are initialized at random to cover a wide range of manipulations. By *strategy*, we refer to the input provided to the manipulator. For Mimicry, these are interpolation weights, but other forms such as binary vectors are also possible (e.g., as shown in Figure 4(a)). Throughout optimization, this population of strategies is adapted by the optimizer, enabling convergence of the population toward some optimal strategy. In Figure 2, the optimizer is denoted as Ω, with its inputs being linked to the collection of objective functions used, described in Section 3.4.

3.4 Objectives for Boundary Testing

For boundary testing, we are interested in regions of the classifier manifold (decision space) where the predicted class probabilities are in equilibrium. When a manipulated input results in such an equilibrium, it indicates that the classifier is unable to decisively distinguish between two or more classes, revealing the presence of a decision boundary.

During classifier training, data points are mapped onto the classifier's manifold with the goal of separating instances from different classes while drawing instances of the same class closer together. This iterative process results in the formation of clusters within the decision space, with the boundaries between these clusters representing the decision boundaries.

As discussed in the previous section, Mimicry traverses this space by manipulating two or more latent vector seeds, allowing us to generate inputs that are approaching the decision boundary. Figure 5 shows different candidate boundary inputs generated by Mimicry for the classes "5" and "6" of the MNIST dataset. The small points are images from the original MNIST test set, whereas the larger points are the boundary images generated by Mimicry, with arrows connecting the source and target seeds. Since Mimicry uses the second most likely class in the SUTs' predictions, we have a (dynamic) target of boundary, allowing for more efficient search as additional knowledge of the decision surface is encoded into the procedure. In targeted boundary testing, the targets can either be rigid or dynamic, where the rigid approach would entail that once a target class is established, it cannot change. Mimicry uses the dynamic approach in targeted boundary testing, allowing for target change through optimization if the classifiers' behavior suggest a boundary to a different class might be closer. Allowing the target to change during optimization enables more flexible and efficient boundary search. It helps navigate intersections of multiple class boundaries and escape local minima by shifting focus to closer or more reachable decision regions. In our experiments, we use two objectives to optimize toward boundary candidates. The first objective function consists of a *dynamic confidence balance*, depicted in (Equation (6)). Here $\hat{y}'$ are the predicted class probabilities of the manipulated input X'. The subscript $1st$ denotes the index of the primary seed class, and the set $J = C_g$ is all the other elements that should be considered (in our case, we only consider the second most likely class from the SUT). This function essentially quantifies how similar multiple confidences are to each other and how much weight they have combined against the rest of the confidence values. This means that the more similar the targeted confidence values are and the more confidence they encapsulate in the prediction, the higher the dynamic confidence balance.

$$\omega_{dcb} : \frac{\sum_{j \in J} |\hat{y}'_1 - \hat{y}'_j|}{||J||(\hat{y}'_1 + \sum_{j \in J} \hat{y}'_j)}. \tag{6}$$

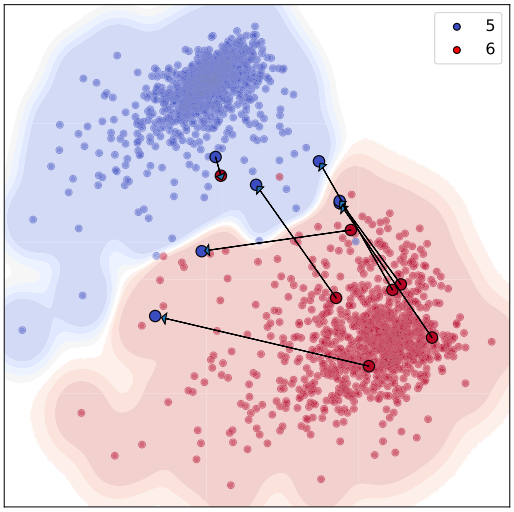


Fig. 5. Boundary discovery between classes "5" and "6" in the MNIST benchmark.

In addition to Equation (6), we also integrate a quality criterion, similar to DeepJanus [57]. This quality criterion measures the normalized Euclidean distance d_2 between strategies (genomes) of an archive κ_A and a population κ_P, to enforce a greater novelty of solutions in genomes. Here we want to maximize sparsity of new genomes, based on the parent population, which functions as the archive; therefore, we minimize (Equation (7)). The search for novelty is commonly used to limit the exploitation of local minima and to have a better traversal of the optimizer search space. This novelty measure does not concern the actual generated images, but rather their manipulation weights.

$$\omega_{d2} : 1 - min\left\{\frac{d_2(a,p)}{\sqrt{||a||}} \;\middle|\; (a,p) \in \kappa_A \times \kappa_P\right\}. \tag{7}$$

4 Empirical Study

4.1 Research Questions (RQs)

To evaluate the proposed tool, we consider the following research questions:

RQ₁ (effectiveness): How effective is Mimicry in finding boundary inputs?
RQ₂ (efficiency): How efficient is Mimicry in finding boundary inputs?
RQ₃ (quality): To what extent are the inputs generated by Mimicry valid and label-preserving?
RQ₄ (latent space usage): How do different objective configurations influence the use of StyleGAN's disentangled latent space when generating boundary inputs?

RQ_1 assesses whether Mimicry can find test cases close to the boundary and whether it can cover a wide range of different boundary cases with regard to the boundary target.

RQ_2 evaluates the efficiency in terms of runtime to investigate the potential cost of utilizing Mimicry.

RQ_3 studies the quality of the inputs produced by Mimicry, in terms of validity, as assessed by human evaluators.

RQ_4 involves an internal evaluation to determine how Mimicry's usage of the disentangled latent space affects the input's manipulations.

4.2 Objects of Study

4.2.1 Datasets. In our study, we used five image classification datasets, namely MNIST [39], FashionMNIST [69], SVHN [50], CIFAR-10 [35], and ImageNet-1k [10] (hereafter referred to as ImageNet for simplicity of exposition). We chose these five datasets because three (MNIST, FashionMNIST, and SVHN) are compatible with both our baselines, DeepJanus [57, 58] and Sinvad [26, 27]. This selection is also consistent with previous studies, such as Dola et al. [13]. However, Mimicry can be applied to any image dataset. CIFAR-10 and ImageNet are used to demonstrate the generalizability of our approach to data sets where a model input representation is not available for DeepJanus, and thus, we compare against the generative-AI-based approach Sinvad [27] as a baseline.

MNIST. Dataset of handwritten digits [39] consisting of grayscale images 28×28 labeled with the corresponding digit (the possible classes range from 0 to 9). MNIST has 60,000 training inputs and 10,000 test inputs. As StyleGAN only allows square images of size 2^n, we zero-pad the images to scale them to size and duplicate channels, resulting in an image of size $32 \times 32 \times 3$.

FashionMNIST. Another dataset consisting of 28×28 grayscale images of Zalando's articles belonging to 10 categories [69]. The dataset has more complex patterns and variations than MNIST and contains 60,000 images for training and 10,000 for testing. Similarly to MNIST, we again zero-pad the data to make it compatible with StyleGAN, having a shape of $32 \times 32 \times 3$.

SVHN. A more complex dataset contains $32 \times 32 \times 3$ color digits of house numbers cropped from Google Street View images [50]. As the data is already compatible with StyleGANs, no transformation was used. It has 73,257 training input and 26,032 test input. The classification task is particularly challenging due to variations in lighting, background clutter, and the presence of distracting digits adjacent to the digit of interest.

CIFAR-10. Another standard benchmark for image classification tasks is divided into 10 classes of different objects [35] and is divided into 50,000 training images and 10,000 testing images. Although the images are small ($32 \times 32 \times 3$), they contain visual complexities and variations of real-world objects, requiring models to extract meaningful features from low-resolution images. Again, the data is compatible with StyleGAN; as such, no transformations were performed.

ImageNet-1k. This dataset consists of over 14 million images spanning 1,000 classes. It has been widely recognized for its role in the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC) since 2010, with the 2012 version being a benchmark standard for image classification tasks. Compared to the other three datasets, ImageNet-1k includes high-resolution images and a significantly broader range of categories. Due to the large size and the number of classes in ImageNet, we focused on the first ten classes, namely *tench, goldfish, great white shark, tiger shark, hammerhead shark, electric ray, stingray, cock, hen, and ostrich.* To make the data compatible with the StyleGAN, all images were transformed to fit into $128 \times 128 \times 3$.

4.2.2 System under Test. We adopt a WideResNet-50-2 classifier [70] available in the PyTorch library [52] with pre-trained weights for ImageNet. The model achieves an accuracy of 0.816 on ImageNet1k. For the other datasets (MNIST, FashionMNIST, SVHN, CIFAR-10), we train the same WideResNet-50-2 architecture with the default train data splits given in Torchvision [45]. This architecture was selected for its strong performance on ImageNet, the most challenging task in our study. Additionally, its well-maintained codebase supports reproducibility, which is essential for our experiments. We prioritized consistency across SUTs by using the same model architecture for all datasets, rather than using state-of-the-art or literature-sourced models, to reduce variability in our results due to architectural peculiarities. The trained networks achieved an accuracy above 0.9 on the test split for all datasets, except for CIFAR-10, where the accuracy was 0.81.

For optimization in training, we use AdamW, which has demonstrated superior performance across multiple tasks due to its adaptive weight decay compared to traditional Adam [42]. Furthermore, we schedule our learning rates using the OneCycle strategy [61], which has been shown to improve convergence during training [61].

4.3 Metrics

A boundary input is defined as an input that is close to the theoretical decision boundary. That is an input in which two or more classes are predicted as equally likely as described in Equation (2). Note that a perfect equilibrium in prediction probabilities is unlikely due to the nature of floating-point operations. Therefore, the resulting input can either be failure-inducing, if it is misclassified, or it can be class-preserving, if the class is predicted correctly.

To address RQ_1, we evaluated the quality of the generated boundary inputs using several metrics, described next.

Boundary Distance (↓). To quantify whether a candidate is a good boundary input, we measure the Euclidean distance d_2 between the predicted classes $\hat{y}' \in \mathbb{R}^{||C||}$ to the theoretical boundary (Equation (8)). In this work, we specifically look at identifying boundary candidates between two classes rather than multiple; the theoretical boundary is assumed to be a vector $b \in \mathbb{R}^{||C||}$ where $\sum b = 1$, and all non-zero elements are equal to $\frac{1}{1+||C_g||}$, that is, in equilibrium between the classes used for boundary discovery C_g and the original class C_0. The lower this measure, the better the boundary input. As noted earlier, perfect equilibrium between class predictions is unlikely due to numerical imprecision, such as floating-point rounding errors. Consequently, we do not enforce exact equality between class scores. Instead, we treat the theoretical boundary as an ideal target and assess candidate quality based on how closely their predicted class distribution approaches this ideal. In practice, boundary inputs are ranked by their Euclidean distance (Equation (8)) to the theoretical boundary vector, where smaller distances indicate better boundary alignment.

$$m_1(\hat{y}', b) = d_2(\hat{y}', b). \tag{8}$$

Label Coverage (↑) and Escape Ratio (↓). Another important aspect of test case generation is the coverage of possible test outcomes. The label coverage indicates the distribution of the target labels, $\mathcal{Y}'_t = \{\hat{y}'_t \forall X'\}$, for the candidates of the boundaries, measured using the Kolmogorov–Smirnov distance (Equation (9)). Here, $\mathcal{U}_t = \mathcal{U}\{C \setminus \{c_0\}\}$ represents the uniform distribution of all possible target labels. A value of 1 indicates a uniform distribution in all possible classes, while a value of 0 indicates that all test cases share the same target label. Although achieving a perfect value of 1 is often infeasible due to the spatial separation of some classes on the decision surface, higher label coverage is generally preferred.

$$m_3(\mathcal{U}_t, \mathcal{Y}_t) = d_{KS}(\mathcal{U}_t, \mathcal{Y}_t). \tag{9}$$

In addition to label coverage, the escape ratio quantifies how often generated test cases deviate from the original class under test. Specifically, it measures the fraction of candidates whose predicted boundaries differ from those of their corresponding origin seed inputs, indicating that the test case has "escaped" the intended decision boundary. In Equation (10), $\mathcal{Y}$ represents the set of predictions on all initial seeds, $\mathcal{Y}'$ is the set of predictions for the candidates generated, and $[\cdot]$ is the Iverson bracket. The subscript on the predictions denotes the indices taken when an argmax is applied to the vector. This measure is critical because, when testing boundaries for a specific class, we are interested only in boundaries that relate to the original class. For example, if we are testing decision boundaries between class X and other classes, we want the generated test cases to remain relevant to class X. Test cases that explore boundaries between unrelated classes, where class X is no longer involved, are not useful for this purpose. Thus, the escape ratio should be small to ensure that the

generated candidates remain relevant to the objective.

$$m_4(\mathcal{Y}, \mathcal{Y}') = \frac{1}{||\mathcal{Y}'||} \sum_{\hat{y} \in \mathcal{Y}, \hat{y}' \in \mathcal{Y}'} [\hat{y}_{1st} \notin \{\hat{y}'_{1st}, \hat{y}'_{2nd}\}]. \tag{10}$$

Laplacian Variance of Image Differences (↑). This measure aims to quantify the change in information between the boundary input and the initial input (Equation (11)). Essentially, it indicates whether the method produces functionally different outputs or merely corrupts or blurs an image, a common phenomenon observed when applying simple perturbations to the latent space. Here, L is a 3×3 Laplacian kernel, applied using convolution on the differences of two images. The higher this measure, the better the boundary input.

$$m_2(X, X') = Var((X - X') * L). \tag{11}$$

About RQ_2, we evaluate the performance of Mimicry by computing the time required, in seconds, to generate a single candidate solution. For comparison, we aggregate the runtime across datasets for each method and report the mean runtime and its standard deviation.

Concerning RQ_3, we performed an evaluation study with human assessors to evaluate the quality of the generated inputs. Quality was assessed with several characteristics, described as follows. Label preservation describes whether the original class label is still assigned to the generated candidate by the evaluator. The inverse of this is target preservation, which quantifies whether the boundary target is visually depicted in the generated image. When combining these two, we get boundary preservation, as the generated image shows visual elements of both classes of the boundary. The latter is simply the sum of the first two. Finally, we measure the validity [58], i.e., whether any class within the considered domain was associated by the evaluator, meaning the image still has valid syntactic features to the human observer.

Answering RQ_4, we investigate how the latent space is used by Mimicry in the implementation used for the experiments. Additionally, we compare this usage to a configuration with a different selection of objective functions $\{\omega_{dcb}\}$, only evaluating dynamic confidence balance, and $\{\omega_{dcb}, \omega_{d2}\}$, which includes archive sparsity for novelty of solutions.

The evaluation is done by aggregating the genome weights, as they dictate the usage of the latent vectors. Specifically, we look at the distribution of those weights in combination with the general uniformity of the distributions to showcase differences in the extents of manipulation.

4.4 Baseline Approaches

To assess the relevance of our approach, we compare Mimicry against Sinvad and DeepJanus, two state-of-the-art test generators for the exploration of the frontier of behaviors of DL image classification systems.

Sinvad [27] is a latent manipulation-based tool that leverages **Variational Autoencoders (VAEs)** as its generative networks. Specifically, Sinvad encodes test inputs into latent vectors using a VAE trained on the corresponding training set. Since the approach relies on population-based optimization, the initial population is derived from the latent vector of the original images, perturbed with random noise to simulate a diverse set of slightly altered inputs. Sinvad then optimizes toward a fitness function through uniform crossover of latent vectors and mutation via noise. The mutation severity is dynamically adjusted: if progress toward the objective function stalls, the mutation magnitude decreases accordingly. In our study, we compare Sinvad against Mimicry on all datasets using the pre-trained VAEs available in the replication package [27] and VAEs for CIFAR-10 and ImageNet [46].

DeepJanus [57] is a representative of model-based approaches and uses a multi-objective search-based algorithm to mutate the control points of a model of the inputs to generate pairs of inputs that are close to each other yet produce different behaviors of the DL system [57].

The input model representation is obtained through a vectorization operation, which produces a sequence of control points that are iteratively displaced to achieve slight modifications. The input image can then be reconstructed through a rasterization operation.

Our comparison focused on the MNIST, FashionMNIST, and SVHN datasets, as DeepJanus's model representation supports these benchmarks. For CIFAR-10 and ImageNet, DeepJanus is not applicable since an appropriate input model is not available for such a feature-rich dataset and cannot be created with the adopted vectorization-rasterization approach, as noted by its authors [57, 58].

4.5 Procedure

Our approach requires generating seeds by sampling latent vectors using a conditional StyleGAN architecture. For MNIST, FashionMNIST, and SVHN, pretrained conditional StyleGAN2 networks are not available. Thus, we trained them on the training partition of each dataset, following the training configurations and guidelines of the original paper [31]. To monitor the model's performance during training, we used the **Fréchet Inception Distance (FID)** metric [22]. The final FID score obtained is 0.91 for MNIST, 2.34 for FashionMNIST, and 4.20 for SVHN, which is in line with the original paper [31]. For CIFAR-10, we used pre-trained StyleGAN2 networks available in the literature [28]. For ImageNet, we used a pre-trained StyleGAN-XL [59].

For RQ_1, for all applicable datasets, we execute all test generators (Mimicry, Sinvad, DeepJanus) using a budget of 15,000 predictions of the SUT per boundary candidate to be optimized for.

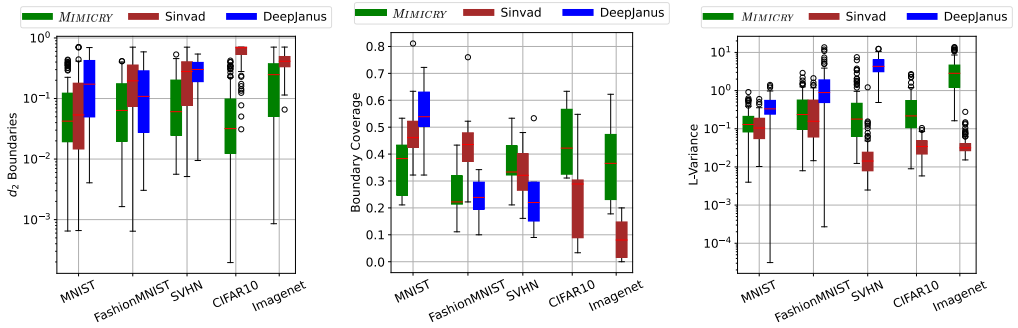
Given the varying approaches of the methods, we define the SUT as the determinant of the budget, ensuring consistency across experiments. While the budget may be reached, it does not need to be fully utilized, as some methods may terminate earlier. We selected 15,000 iterations as the baseline computational budget, which is lower than Sinvad's original budget of 25,000 SUT evaluations (500 generations × 50 population) [27] and DeepJanus's 20,000 evaluations (100 generations × 2 × 100 population) [57]. This reduced budget is justified because Sinvad often terminates early, rarely using its full 25,000 iterations, and DeepJanus typically converges before exhausting its budget.

We instruct the tools to search for 10 boundary candidates for each class, giving us a total budget of $1.5M$ predictions per test method and dataset. Overall, our study includes nearly $20M$ predictions (2 tools, Mimicry and Sinvad × $1.5M$ predictions × 5 datasets + $1.5M$ × 3 datasets for DeepJanus).

For RQ_2, in addition to the number of iterations used and the convergence of the boundary distance measure, we also record the runtime, acknowledging that implementation efficiency can influence performance.

For RQ_3, as image metrics often do not coincide with human perception [37], we use a human evaluation study to quantify the quality of generated cases. The evaluators are recruited on AWS Mechanical Turk, with an attention question incorporated to filter out inattentive or disengaged responses [62]. The attention question was given with the datasets' reference images, asking the participants to select a specific class.

Prior to the evaluation, images of the original datasets were shown to make the assessors familiar with the classes. For each dataset, we randomly selected 10 generated images from each method (Mimicry, Sinvad, DeepJanus), covering all classes. We ask the participants to select all classes they can recognize in the provided image, with a "Not applicable" option if they could not identify any of the provided classes. Each participant was only shown samples from a single dataset. We recruited 30 distinct evaluators for each dataset, with 23.4 ± 4.6 (mean and standard deviation) valid responses per dataset after filtering. Some responses were discarded due to incorrect answers



(a) Boundary Distance. (b) Label Coverage. (c) Laplacian Variance.

Fig. 6. Effectiveness results (RQ_1).

in the attention check, where participants were asked to select a specific image. This check was used to identify and exclude inattentive or disengaged respondents.

For RQ_4, we monitor the final latent interpolation weights, found in optimization. As the initial strategies are all initialized randomly, it is interesting to see whether certain types of genomes are more likely to appear at the end of optimization. For each found candidate solution, we therefore have a corresponding latent interpolation weight vector, which is then used for the analysis.

4.6 Results

4.6.1 RQ_1 (Effectiveness). Figure 6 reports two plots related to the considered effectiveness metrics. Each plot displays the distribution of the metrics as boxplots, aggregated across all datasets.

Considering boundary distance (Figure 6(a)), MIMICRY consistently generates boundary inputs characterized by lower boundary distances for all datasets, showing the competitiveness of our approach at generating inputs close to the equilibrium between the source and target class. Related to the baselines, DeepJanus's inputs exhibit higher distance, arguably due to the model-based transformation, which makes it impossible to perform fine-grained input manipulations. Concerning Sinvad, it exhibits competitive scores for simple datasets (MNIST, FashionMNIST, and SVHN), even though they are worse than those of MIMICRY. In contrast, for more complex datasets such as CIFAR-10 and ImageNet, the effectiveness of Sinvad is particularly low, especially for CIFAR-10.

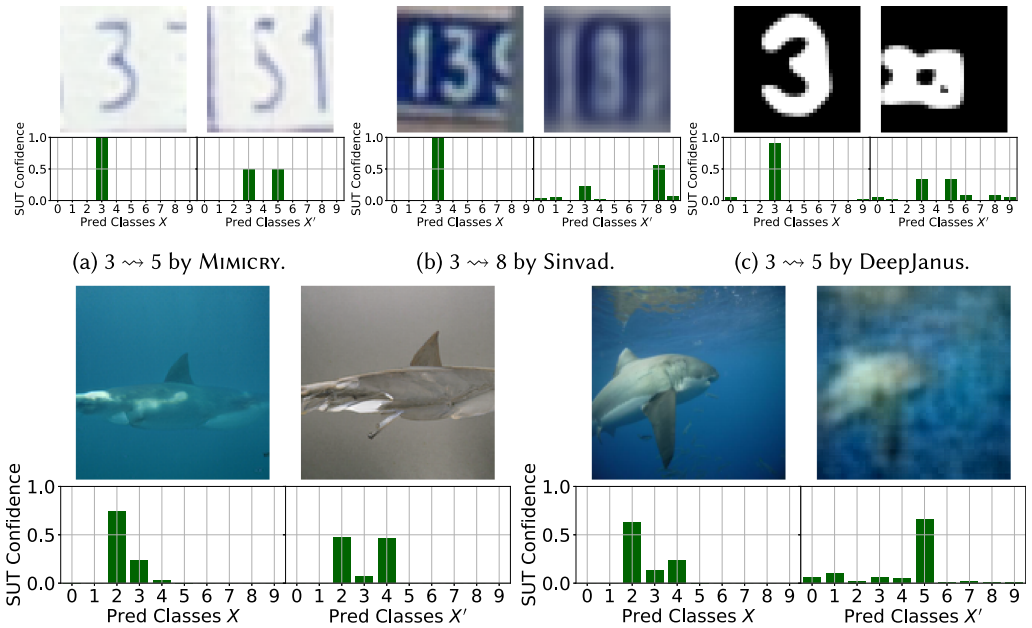
Regarding label coverage (Figure 6(b)), DeepJanus outperforms generative-based methods in label coverage for MNIST. However, this trend reverses for SVHN. With more complex datasets, MIMICRY outperforms Sinvad in terms of label coverage. It is important to note that label coverage alone does not provide a complete picture. For example, applying noise to images may result in high label coverage because of a wide variation in target labels. However, for a boundary candidate to be useful, it must remain relevant as a boundary candidate between the original class and others. This is exactly what the escape ratio quantifies. Table 1 reports the average escape ratio across all datasets. MIMICRY outperforms the competing methods across all datasets, indicating that the generated boundary inputs are more likely to be useful for testing specific boundaries.

Regarding Laplacian variance, Figure 6(c) highlights that DeepJanus performs better when a model is available. However, this metric is skewed due to the way DeepJanus generates solutions by modifying vector paths, which results in pixels being either black or white (see Figure 7(c)). This artificially increases variance, as the Laplacian filter responds strongly to sharp edges. In contrast, generative-AI-based solutions produce values across the entire spectrum, leading to smoother transitions and less pronounced edges. When comparing MIMICRY and Sinvad, an interesting trend

Table 1. RQ_1: Escape Ratio for All Approaches and Datasets

	MNIST	FashionMNIST	SVHN	CIFAR-10	ImageNet
Mimicry	**0**	**0**	**0.01**	**0**	**0.07**
Sinvad	0.01	0.05	0.23	0.30	0.59
DeepJanus	0.02	0.13	0.28	N/A	N/A

Bold values indicate the best escape ratio per dataset.



(a) 3 ⇝ 5 by Mimicry. (b) 3 ⇝ 8 by Sinvad. (c) 3 ⇝ 5 by DeepJanus.

(d) *white shark* ⇝ *hammerhead shark* by Mimicry. (e) *white shark* ⇝ *electric ray* by Sinvad.

Fig. 7. *SVHN and ImageNet*: Original image and corresponding boundary input with SUT confidence.

emerges, consistent with previous metrics. As data complexity increases, Sinvad's performance drops, producing images that appear blurred rather than functionally manipulated (see Figure 7(e)).

Aggregating these measures, we are interested in whether these differences have statistical significance. Therefore, we employ a one-tailed Mann–Whitney U test [68] (with $\alpha = 0.05$) between Mimicry and the baseline methods. Additionally, we calculate the Cohen's d effect size [8], whose magnitude is indicated in Table 2 by a colored symbol, where ★ = $d > 1$ (large), ♦ = $1 \geq d > 0.5$ (medium), ● = $d \leq 0.5$ (small). The statistical results confirm the trend observed in the figures, where Mimicry performs well on all datasets and outperforms the baselines, especially as their complexity increases.

RQ_1 (effectiveness): Mimicry significantly outperforms baseline methods in boundary distance and escape ratio, and shows stronger performance on complex datasets in terms of label coverage. As dataset complexity increases, it generates more relevant and effective candidates for boundary testing, demonstrating a clear advantage over existing approaches in manipulating image content for functional testing.

Table 2. RQ_1: Statistical Analysis

	MNIST		FashionMNIST		SVHN		CIFAR-10	ImageNet
	Sinvad	DeepJanus	Sinvad	DeepJanus	Sinvad	DeepJanus	Sinvad	Sinvad
Boundary distance	0.283	**1.19e-7** ♦	**9.51e-7** ♦	**0.005** ★	**5.21e-9** ♦	**7.7e-15** ★	**4.08e-31** ★	**1.16e-11** ★
Laplacian variance	**0.014** ●	~ 1	0.083	~ 1	**1.61e-27** ♦	~ 1	**1.03e-25** ★	**1.31e-34** ★
Label coverage	0.986	0.998	0.998	0.32	0.213	**0.006** ★	**0.001** ★	**1.41e-40** ★

Significant p-values are boldfaced.

Table 3. Mean Runtime and Standard Deviation per 15,000 Iterations (in Seconds) and Trainable Parameters

	MNIST	FashionMNIST	SVHN	CIFAR-10	ImageNet
MIMICRY	78.98 ± 2.03	76.56 ± 1.02	77.97 ± 1.41	81.04 ± 1.71	412.56 ± 14.37
ϕ_G *params*	*21M*	*21M*	*21M*	*20M*	*158M*
Sinvad	**3.34 ± 0.36**	**3.36 ± 0.45**	**3.54 ± 0.49**	**4.91 ± 1.46**	**7.77 ± 1.72**
ϕ_G *params*	*4M*	*4M*	*13M*	*83M*	*79M*
DeepJanus	3.90 ± 0.26	7.85 ± 22.45	105.32 ± 257.80	N/A	N/A

4.6.2 RQ_2 (Efficiency). Table 3 reports the runtime efficiency results, normalized to a budget of 15,000 iterations to ensure comparability across methods. Sinvad emerges as the fastest approach, outperforming both MIMICRY and DeepJanus in terms of raw execution time.

When examining wall-clock runtime, all StyleGAN2-based methods (targeting MNIST and CIFAR-10) retain relatively consistent performance. In contrast, the StyleGAN-XL-based ImageNet generator incurs significantly higher computational costs due to the larger model size. This is reflected in Table 3, where runtime differences correspond to the underlying generator architecture.

Sinvad's runtime shows higher variability due to its early termination mechanism, which adapts the mutation size when progress stalls. On more complex datasets like CIFAR-10 and ImageNet, this mechanism often causes Sinvad to terminate prematurely, before fully utilizing its computational budget. This indicates a loss of control over candidate generation in high-complexity settings.

For DeepJanus, both the average runtime and its variability increase substantially with dataset complexity. This trend stems from its reliance on two mutation operators that act on SVG path structures. When the necessary patterns are absent in the input, these operators become ineffective, resulting in extended computation times.

To evaluate convergence, we aggregate the minimum distance to decision boundaries across generations for Sinvad, DeepJanus, and MIMICRY. As shown in Figure 8, MIMICRY consistently outperforms both baselines—even when constrained to the budget consumed before Sinvad's early termination. The plot illustrates the average minimum distance per generation across all experiments, highlighting MIMICRY's superior convergence and stability. Since Sinvad terminates early in some runs, its curve reflects greater variance. The vertical black line at $BudgetUsed = 4{,}000$ marks the last iteration for which data is available from every run. On average, Sinvad terminates after 5,420 ± 1,278 iterations.

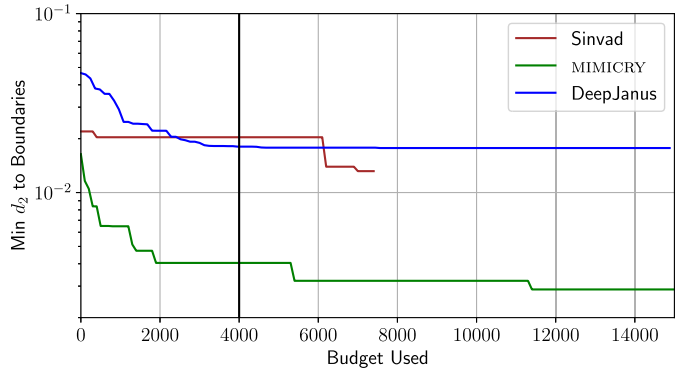


Fig. 8. Tools convergence in optimization.

Table 4. Human Image Evaluation Statistics

		Label Preservation	Target Preservation	Boundary Preservation	Validity
MNIST	MIMICRY	**0.460**	**0.422**	**0.882**	**0.965**
	Sinvad	0.360	0.268	0.628	0.790
	DeepJanus	0.690	0.090	0.780	0.830
FashionMNIST	MIMICRY	**0.417**	**0.280**	**0.697**	**0.944**
	Sinvad	0.272	0.210	0.481	0.757
	DeepJanus	0.266	0.207	0.474	0.816
SVHN	MIMICRY	0.240	**0.325**	**0.565**	**0.865**
	Sinvad	0.134	0.233	0.366	0.673
	DeepJanus	**0.328**	0.078	0.405	0.790
CIFAR-10	MIMICRY	**0.470**	0.089	**0.559**	**0.803**
	Sinvad	0.255	**0.120**	0.375	0.589
ImageNet	MIMICRY	0.188	**0.064**	0.252	**0.711**
	Sinvad	**0.243**	**0.064**	**0.307**	0.707

RQ_2 (efficiency): While MIMICRY is slower than Sinvad in terms of raw execution time, it makes more effective use of the available budget by achieving stronger and more consistent convergence. Sinvad often terminates early without significant progress on complex datasets, and DeepJanus slows down substantially as complexity increases. In contrast, MIMICRY delivers more meaningful results within the same budget, making the additional runtime a worthwhile tradeoff.

4.6.3 RQ_3 (Quality). Table 4 shows the results of the human study. The table reports, for each dataset and approach, the average label preservation and target preservation scores, as well as the boundary preservation and validity scores.

MIMICRY outperforms all baselines in terms of validity because the generated images are more likely to have a visibly recognizable class for the human observers. When looking at the boundary preservation, a similar trend emerges, except for ImageNet, in which Sinvad scores the best results.

For label and target preservation, MIMICRY outperforms the baselines in most datasets, with some exceptions being DeepJanus in SVHN and Sinvad in CIFAR-10 and ImageNet. The ImageNet results are especially interesting, as they have implications for human studies when doing boundary testing, which seems to be challenging in more complex and feature-rich datasets.

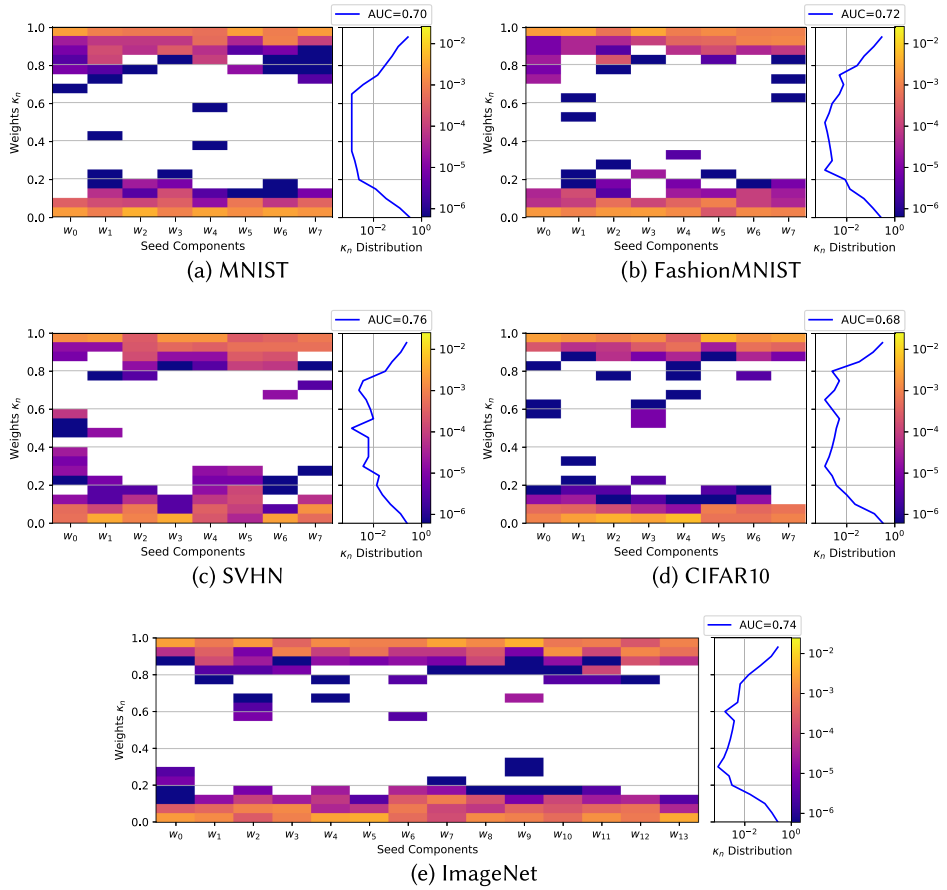
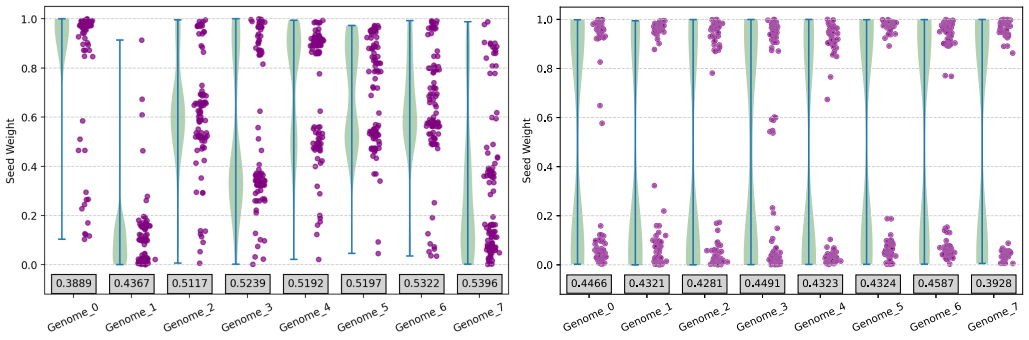


Fig. 9. Weights κ_n of seed components.

RQ_3 (quality): MIMICRY outperforms the baselines across datasets except for ImageNet, where Sinvad had higher preservation scores but lower validity. This suggests that while MIMICRY is generally effective in maintaining both validity and label preservation, its optimization for DL decision boundaries may reduce interpretability in complex datasets.

4.6.4 RQ_4 (Latent Space Usage). In Figure 9, we show the distribution of genome component values, aggregated across all candidates for each dataset. This aggregation is done for the experiments using $\{\omega_{dcb}, \omega_{d2}\}$. The figures show a histogram for each component w_n, with the frequency in each bin being color-coded on a log scale. The empty regions are zeros, i.e., no contribution. Additionally, we aggregate the usage across all seed components, giving us the κ_n-Distribution. With this distribution, the usage can be shown more effectively, where the **Area under the Curve (AUC)** acts as a proxy for latent manipulation complexity.

From Figure 9, we observe a clear trend towards extreme values, with component values increasingly concentrated around smaller differences. Additionally, as dataset complexity increases, the spread of these concentrations widens, which is also shown in the change of AUC. However, this trend does not hold for CIFAR-10, indicating that other factors beyond data complexity affect the manipulation.



(a) $\omega = \{\omega_{dcb}\}$ (b) $\omega = \{\omega_{dcb}, \omega_{d2}\}$

Fig. 10. Genome usage in CIFAR-10, without and with diversity constraints.

To investigate how the usage of the genome changes, we remove the genome diversity objective. As expected, changing the objectives leads to changes in the resulting latent space usage (see Figure 10). The plots in Figure 10 show the distribution of weights for each genome component in the CIFAR-10 case as a violin and scatter plot. The number below each plot quantifies the uniformity of the distribution, with 1 representing perfect uniformity and 0 indicating all weights being identical.

Looking at the baseline with all objectives in Figure 10(b), we observe a clear trend toward extreme values in the seed weights, as confirmed by Figure 9(d). Interestingly, some genome components (such as the "coarser" layers w_1 and w_2) show a preference for lower weight values.

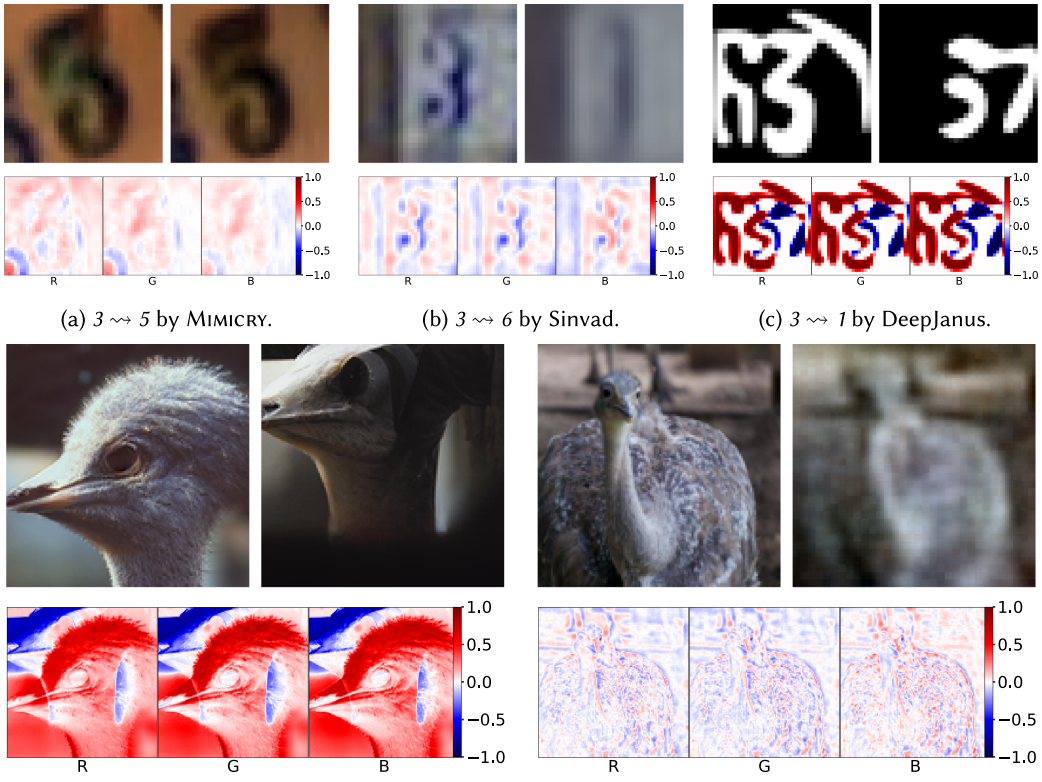
When comparing this to Figure 10(a), where the constraint on the genomes is removed, the distribution of weights changes noticeably. The uniformity measure is higher in this case, indicating a more distributed layer usage. In contrast to the baseline, we now see distinct preferences in some genome components, resulting in less divergent distributions.

RQ_4 (latent space usage): Mimicry *can use the latent space of its generator with great flexibility. Concerning datasets, the latent usage is more fine-grained as the dataset complexity increases, except for CIFAR-10. Additionally, the choice of optimization objectives has a significant impact on how the latent space is used. In our case, the latent manipulations resemble classical style-mixing operations with a novelty constraint used, whereas without diversity constraints, the manipulations have a more uniform distribution in interpolation weights.*

4.7 Qualitative Analysis

While metrics and quantitative analysis allow for comparability of results, they often do not communicate the full picture adequately, especially concerning image realism [37]. To showcase the performance differences between Mimicry and the baseline methods, we perform a qualitative analysis of the produced outputs by manually analyzing samples generated by each method.

The first interesting aspect relates to the Laplacian Variance of image differences seen in (Equation (11)). This numerical measure is not widely used and therefore needs more explanation and positioning. Especially in comparing Mimicry to Sinvad, it is useful, while it fails for the comparison with DeepJanus. As described earlier, with this measure, the type of change in the image across the optimization process is quantified. Low Laplacian variance in the image differences here means that the image gets gradually blurred, not producing functional differences in the output candidate Figures 11(b) and (e). This can easily be seen in the examples for Sinvad, where with more complex



(a) *3* ⇝ *5* by Mimicry. (b) *3* ⇝ *6* by Sinvad. (c) *3* ⇝ *1* by DeepJanus.

(d) *ostrich* ⇝ *electric ray* by Mimicry. (e) *ostrich* ⇝ *stringray* by Sinvad.

Fig. 11. *SVHN and ImageNet:* Initial to final candidate comparison with channel-wise differences.

data, this phenomenon gets more prominent. In contrast, Mimicry does not blur the original seed images but rather produces functionally different outputs, even if those outputs may no longer convey clear class information to the human observer. The blurring observed in Sinvad is a result of the underlying VAE's architecture. VAEs typically use smaller latent dimensions, which limits the complexity of the generated outputs. Additionally, the training objective focuses on minimizing reconstruction loss [34]. This loss does not necessarily correlate with visual realism. In contrast, GANs optimize separate loss functions for the Generator and Discriminator networks, encouraging the generation of images that are more difficult to distinguish from the underlying dataset [18].

4.8 Threats to Validity

4.8.1 Internal Validity. All experiments were conducted using the same computational budget based on SUT predictions. While Sinvad employs an early termination condition and DeepJanus does not, we preserved the original behavior of both baselines to maintain their methodological integrity. Additionally, regarding the StyleGAN models, we utilized pre-trained models available from the literature [32, 59]. When not available, we trained the StyleGAN models using the scripts available in the replication package of the original paper [32], as it is difficult to envision less threat-prone approaches. In the human evaluation study, we included attention questions to ensure that participants were engaging truthfully [62]. Since both DeepJanus and Sinvad require input

images, rather than generating inputs like Mimicry, the input selection process was randomized. This approach aims to reduce bias and ensure a fairer comparison with Mimicry.

4.8.2 External Validity. The limited number of DL systems included in our evaluation poses a threat to the generalizability of our results. We addressed this issue by incorporating a variety of datasets with increasing complexity and high-performing models from related literature. We demonstrated the usefulness of the StyleGAN architecture [32, 59]; however, other style-based architectures [29, 30] may also yield promising results.

5 Discussion

Latent Feature Mixing Enables Semantic Control during Test Generation. Mimicry proved effective for boundary identification in all considered benchmarks and SUTs. The effectiveness can be attributed to the disentangled latent space representation, which produces high-quality boundary inputs if explored effectively. On the contrary, entangled latent space representations often result in blurred effects applied to the original images, with no benefits in functional coverage. Interestingly, the boundary case quality seems to be independent of the SUT, where more ambiguity in the predicted class probabilities does not result in worse boundary candidates, as it is observable with the baseline methods (see Figure 7). However, our results show that both the SUT quality and benchmark complexity affect the manipulations in Mimicry's generator, as in the case of CIFAR-10 and ImageNet. Our experiments also show that imposing diversity constraints in the objective function affects the latent space manipulation, which in turn affects the final quality and validity of the generated boundary candidates. Concerning the relevance of generated boundary inputs, our results show that Mimicry outperforms the baselines in all datasets, as it produces inputs that represent the target class constraint (i.e., they do not "escape" the given boundary). Overall, Mimicry maintains competitive scores across datasets while staying within the distribution of the data domain under test (Figure A.1). In contrast, Sinvad and DeepJanus maintain reasonable effectiveness in smaller datasets such as MNIST and FashionMNIST but deteriorate with more complex data, where they either produce corrupted or out-of-distribution data (Figures A.2 and A.3).

Targeted Boundary Exploration Improves Functional Coverage. Concerning coverage, it is important to consider that for some classes, only a subset of all possible labels is meaningful, whereas high coverage would suggest poor control over test case generation. As an example, the digit 7 in MNIST has meaningful boundary candidates in the numbers 1 or 5, whereas an 8 probably is not bound to the decision region of 7 s. In this case, Mimicry's targeted exploration overcomes existing tools, and it proves to outperform the competitors, especially when dataset complexity increases (Figure 6(b)), thanks to its targeted exploration.

Generating Well-Defined, Unambiguous Boundary Inputs for High-Resolution Datasets Remains an Open Challenge. Mimicry produces inputs with high validity rates, provided the generated images depict a class that is perceptible to human observers. In terms of boundary preservation, Mimicry generally performs well except in the case of ImageNet. Unlike Sinvad, which mainly blurs existing dataset images, Mimicry generates functionally novel inputs that may sometimes appear ambiguous to humans. For ImageNet, the relatively high resolution (128×128) allows the blur introduced by Sinvad to retain enough visual cues for human evaluators to identify the original class. In contrast, Mimicry may generate objects that are less recognizable or entirely unfamiliar, given the feature mixing between two classes. This highlights an open challenge: as dataset complexity increases, assessing the validity of generated test cases becomes more difficult for human evaluators, especially for classes with low to no semantic affinity. In such contexts, alternative evaluation methods such as using large language models as judges [19] might be of interest to corroborate the human assessment. LLMs have shown high correlation with human

assessments in tasks involving image content understanding [43], and they can approximate human decisions with notable consistency [19, 66, 75]. While they should not replace human evaluation entirely, their use could enhance both the consistency and efficiency of the validation process, particularly in large-scale or ambiguous scenarios.

Balancing Speed and Realism Is Key to Practical DL System Testing. Mimicry presents a powerful technique for testing DL image classification systems, which are relevant across various domains. Use cases that particularly benefit from comprehensive testing in semantically rich image domains include autonomous driving systems, which rely on accurate interpretation of sensory inputs such as camera feeds. An example test case might involve modifying input images toward behavioral boundaries that lead to failures, such as crashes or hazardous situations. This enables assessment of the system's safety using realistic inputs that exhibit semantic changes, unlike pixel-level corruptions such as adversarial examples, which are unlikely to occur in real-world scenarios. Another important application domain is medical imaging, such as the classification of cancerous skin lesions. For such safety-critical domains, the runtime performance of Mimicry is less relevant, as testing is not performed in an online setting but rather for validation purposes. As such, the realism of generated test cases is far more important: verification is only meaningful if the input distribution closely resembles real-world data. While approaches like Sinvad offer faster execution, they often fall short in producing realistic examples, limiting their usefulness in ensuring the trustworthiness and verifiability of such systems.

6 Related Work

The methodologies concerning DL system testing can be grouped into three families. These families of DL test generation methodologies are model-based input manipulation, raw input manipulation, and latent space manipulation. We overview the main propositions next to clarify the positioning of Mimicry in the state of the art. We also discuss how boundary testing in classical software systems relates to DL boundary testing, highlighting both intersections and fundamental differences.

6.1 Boundary Testing in Classical Software Systems

Boundary testing, sometimes referred to as catalog-based testing, is a black-box technique that targets the edges of the input domain [54]. Inputs are partitioned into valid and invalid classes, and test cases are selected at and around each class boundary (e.g., minimum, maximum, and values just inside or outside) [23]. This approach exploits the empirical observation that faults such as off-by-one errors and missing checks are most likely to manifest near boundary values. It is extensively employed in both general-purpose and safety-critical systems to verify correct handling of extreme, out-of-range, or mistyped inputs [3, 11, 16, 21, 40]. While in both classical and DL systems boundary testing seeks to expose weaknesses at the edges of separated semantic regions of the input space, DL models operate over high-dimensional and weakly structured domains, such as images, where explicit numeric boundaries are absent. Consequently, traditional boundary testing methods are not directly applicable, motivating the use of generative approaches to probe and explore a system's decision boundaries.

6.2 Model-Based Input Manipulation

Model Input Manipulation (MIM) techniques leverage a model of the input domain to generate test inputs, similar to conventional model-driven engineering practices that uphold compliance with domain-specific constraints [2, 5, 6, 17, 49, 57].

The manipulation occurs on the model, which is subsequently reconverted to the original format [38]. MIM techniques operate within a restricted input space, specifically the control parameters

of the model representation. These techniques enhance the realism of the produced outputs by implementing appropriate model constraints.

Several search-based MIM approaches have been applied to DL-based image classifiers. DeepHyperion [76] uses the MAP-Elites Illumination Search algorithm [47] to explore the feature space of the input domain and identify misbehavior-inducing features. DeepMetis [55] is a MIM approach that generates inputs that behave correctly on original DL models and misbehave on mutants obtained through injection of realistic faults [25], which can be useful to enhance the mutation killing ability of a test set. DeepJanus [57] is the MIM approach most related to this work since it performs boundary testing of DL systems by leveraging SVG paths in datasets to synthesize new test cases. Therefore, we performed an explicit empirical comparison with the DeepJanus approach in this work.

A significant limitation of MIM approaches is their reliance on the availability of a high-quality model representation for the specific input domain, which is manually crafted [58]. Unlike MIM techniques, Mimicry leverages a generative network to learn the distribution of the input domain. This approach is largely automated and requires no labeling or other costs, except for hyperparameter tuning. These characteristics of Mimicry broaden its applicability across various domains.

6.3 Raw Input Manipulation

Raw Input Manipulation (RIM) techniques involve modifying an image's original pixel space to create a new input by perturbing the pixel values. RIM techniques aim to produce minimal, often imperceptible changes to the original to trigger misbehavior in the DL system [9, 36, 41, 72, 76]. These methods do not focus on boundary analysis and target different aspects of testing, such as data augmentation or adversarial attacks, which are not directly aligned with our goal. Our method is a *functional* test generator, differing from adversarial testing in both goals and techniques. Functional testing creates new, valid, in-distribution inputs to evaluate a DNN's generalization. In contrast, adversarial testing adds minor perturbations to original inputs to test *robustness* [9]. Given these distinct objectives and methods, direct comparisons are inappropriate. However, for completeness, we describe the main propositions next.

DeepXplore [53] employs various techniques, including occlusion, light manipulation, and blackout, to cause misbehavior. These perturbations are intended to improve neuron coverage within the DL system. DLFuzz [20] introduces noise to the seed image to increase the likelihood of system misbehavior. DLFuzz generates adversarial inputs for DL systems without relying on cross-referencing other similar DL systems or manual labeling. DeepTest [65] alters the images using synthetic affine transformation from the computer vision domain, such as blurring and brightness adjustments, to create simulated rain/fog effects.

RIM techniques are limited to modifying existing inputs, and they cannot thoroughly explore the input domain and its boundaries, while generative DL models can sample novel inputs from the data distribution.

Moreover, the manipulated images might not always represent real-world functional inputs, e.g., images with artificial artifacts at the corners or unnatural lighting conditions generated by DeepXplore.

Consequently, such techniques are more suitable for security and robustness testing rather than for functional testing [58].

Differently, our technique targets functional testing, specifically boundary value analysis of DL systems. We achieve this by manipulating the latent space of a StyleGAN to efficiently find test cases that expose behavioral changes in the SUT.

6.4 Latent Space Manipulation

Latent space manipulation techniques generate new inputs by learning and reconstructing the underlying distribution of the input data. The most commonly used techniques are **Variational Autoencoders (VAE)** [34], GANs [18], and Diffusion Models [4, 46, 48].

Sinvad [26, 27] constructs the input space using VAE and navigates the latent space by adding a random value sampled from a normal distribution to a single element of the latent vector. Sinvad aims to explore the latent space by maximizing either the probability of misbehaviors, estimated from the softmax layer output, or by surprise coverage [33]. Sinvad is our second baseline, as it was also used for testing decision boundaries in DL systems [26, 27].

The Feature Perturbations technique [14, 15] involves injecting perturbations into the output of the generative model's first layers, which represent high-level features of images. These perturbations can affect various characteristics of the image, such as shape, location, texture, or color. DeepRoad [74] generates driving images using GANs for image-to-image translation.

CIT4DNN [13] combines VAE and combinatorial testing [7]. This allows the systematic exploration and generation of diverse and infrequent input datasets. CIT4DNN partitions latent spaces to create test sets that contain a wide range of feature combinations and rare occurrences. A recently proposed technique, Instance Space Analysis, aims to pinpoint the critical features of test scenarios that impact the detection of unsafe behavior [49].

Unlike conventional latent space manipulation techniques, our approach leverages the richer and more complex latent space of StyleGAN for boundary testing of DL systems. While existing state-of-the-art methods are often constrained by limited data complexity, our framework addresses this limitation by incorporating more complex datasets, facilitating better transferability to real-world scenarios. Furthermore, we integrate feedback from the SUT in the form of model predictions to guide manipulations toward more promising regions of the decision space.

7 Conclusion and Future Work

In this work, we present Mimicry, a technique for targeted boundary testing of DL classifiers by identifying inputs near decision boundaries. Our empirical analysis demonstrates that Mimicry outperforms existing methods such as DeepJanus [57] and Sinvad [26], particularly in complex data domains, by leveraging latent space manipulations and incorporating SUT behavior into the search process. Unlike DeepJanus, which relies on model representations of inputs, and Sinvad, which suffers from limited control over generative manipulations, Mimicry effectively balances control, fidelity, and performance in generating meaningful boundary test cases.

Future work will investigate the interplay between classifier quality and latent space complexity. Especially the concept of the boundary to a validity domain is seldom talked about in related literature, as it has an especially hard oracle problem. Another direction for future work is the change of manipulator technologies to other generators, such as diffusion- or transformer-based generators. Additionally, unlike previous methods, using Mimicry on more complex datasets can be considered, making future work increasingly more relevant to real-world problems.

Data Availability

The experiment codebase, analysis scripts, and all artifacts generated for this work can be found in the replication package [1]. Artifacts that were generated by other works are linked accordingly.

References

[1] GitHub. 2025. Replication Package. Retrieved from https://github.com/oliverweissl//SMOO/tree/archive/mimicry

[2] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *Proceedings of the 33rd ACM/IEEE*

International Conference on Automated Software Engineering (ASE '18). ACM, New York, NY, 143–154. DOI: https://doi.org/10.1145/3238147.3238192

[3] Sabinakhon Akbarova, Felix Dobslaw, Francisco Gomes de Oliveira Neto, and Robert Feldt. 2025. SETBVE: Quality-diversity driven exploration of software boundary behaviors. arXiv:2505.19736. Retrieved from https://arxiv.org/abs/2505.19736

[4] Luciano Baresi, Davide Yi Xian Hu, Andrea Stocco, and Paolo Tonella. 2025. Efficient domain augmentation for autonomous driving testing using diffusion models. In *Proceedings of the 47th International Conference on Software Engineering (ICSE '25)*. IEEE.

[5] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE '16)*, 63–74.

[6] R. Ben Abdessalem, S. Nejati, L. C. Briand, and T. Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering (ICSE '18)*, 1016–1026. DOI: https://doi.org/10.1145/3180155.3180160

[7] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. 1997. The AETG system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering* 23, 7 (1997), 437–444. DOI: https://doi.org/10.1109/32.605761

[8] Jacob Cohen. 1988. *Statistical Power Analysis for the Behavioral Sciences*. L. Erlbaum Associates.

[9] Francesco Croce and Matthias Hein. 2020. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *Proceedings of the International Conference on Machine Learning*. PMLR, 2196–2205.

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

[11] Felix Dobslaw, Francisco Gomes de Oliveira Neto, and Robert Feldt. 2020. Boundary value exploration for software analysis. In *Proceedings of the IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW '20)*. IEEE, 346–353.

[12] Swaroopa Dola, Matthew B. Dwyer, and Mary Lou Soffa. 2023. Input distribution coverage: Measuring feature interaction adequacy in neural network testing. *ACM Transactions on Software Engineering and Methodology* 32, 3 (2023), 1–48.

[13] Swaroopa Dola, Rory McDaniel, Matthew B. Dwyer, and Mary Lou Soffa. 2024. CIT4DNN: Generating diverse and rare inputs for neural networks using latent space combinatorial testing. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, 1–13.

[14] Isaac Dunn, Tom Melham, and Daniel Kroening. 2020. Semantic adversarial perturbations using learnt representations. arXiv:2001.11055. Retrieved from https://arxiv.org/abs/2001.11055

[15] Isaac Dunn, Hadrien Pouget, Daniel Kroening, and Tom Melham. 2021. Exposing previously undetectable faults in deep neural networks. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 56–66.

[16] Jingzhou Fu, Jie Liang, Zhiyong Wu, Yanyang Zhao, Shanshan Li, and Yu Jiang. 2025. Understanding and detecting SQL function bugs: Using simple boundary arguments to trigger hundreds of DBMS bugs. In *Proceedings of the Twentieth European Conference on Computer Systems*, 1061–1076.

[17] Alessio Gambi, Marc Mueller, and Gordon Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '19)*. ACM, New York, NY, 318–328. DOI: https://doi.org/10.1145/3293882.3330566

[18] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Communications of the ACM* 63, 11 (2020), 139–144.

[19] Jiawei Gu, Xuhui Jiang, Zhichao Shi, Hexiang Tan, Xuehao Zhai, Chengjin Xu, Wei Li, Yinghan Shen, Shengjie Ma, Honghao Liu, et al. 2025. A survey on LLM-as-a-judge. arXiv:2411.15594. Retrieved from https://arxiv.org/abs/2411.15594

[20] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Differential fuzzing testing of deep learning systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 739–743.

[21] Xiujing Guo, Hiroyuki Okamura, and Tadashi Dohi. 2024. Optimal test case generation for boundary value analysis. *Software Quality Journal* 32, 2 (2024), 543–566.

[22] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, Günter Klambauer, and Sepp Hochreiter. 2017. GANs trained by a two time-scale update rule converge to a Nash equilibrium. arXiv:1706.08500. Retrieved from http://arxiv.org/abs/1706.08500

[23] Robert M. Hierons. 2006. Avoiding coincidental correctness in boundary value analysis. *ACM Transactions on Software Engineering and Methodology* 15, 3 (2006), 227–241.

[24] Xun Huang and Serge Belongie. 2017. Arbitrary style transfer in real-time with adaptive instance normalization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.
[25] Nargiz Humbatova, Gunel Jahangirova, Gabriele Bavota, Vincenzo Riccio, Andrea Stocco, and Paolo Tonella. 2020. Taxonomy of real faults in deep learning systems. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. ACM, New York, NY, 12 pages. DOI: https://doi.org/10.1145/3377811.3380395
[26] Sungmin Kang, Robert Feldt, and Shin Yoo. 2020. Sinvad: Search-based image space navigation for DNN image classifier test input generation. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops*. IEEE, 521–528.
[27] Sungmin Kang, Robert Feldt, and Shin Yoo. 2024. Deceiving humans and machines alike: Search-based test input generation for DNNs using variational autoencoders. *ACM Transactions on Software Engineering Methodologies* 33, 4 (2024), Article 103, 1–24. DOI: https://doi.org/10.1145/3635706
[28] Tero Karras, Miika Aittala, Janne Hellsten, Samuli Laine, Jaakko Lehtinen, and Timo Aila. 2020. Training generative adversarial networks with limited data. arXiv:2006.06676. Retrieved from https://arxiv.org/abs/2006.06676
[29] Tero Karras, Miika Aittala, Samuli Laine, Erik Härkönen, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2021. Alias-free generative adversarial networks. *Advances in Neural Information Processing Systems* 34 (2021), 852–863.
[30] Tero Karras, Samuli Laine, and Timo Aila. 2019. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 4401–4410.
[31] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2019. Analyzing and improving the image quality of StyleGAN. arXiv:1912.04958. Retrieved from http://arxiv.org/abs/1912.04958
[32] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. 2020. Analyzing and improving the image quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, 8110–8119.
[33] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE '19)*. IEEE, 1039–1049. DOI: https://doi.org/10.1109/ICSE.2019.00108
[34] Diederik P. Kingma. 2013. Auto-encoding variational bayes. arXiv:1312.6114. Retrieved from https://arxiv.org/abs/1312.6114
[35] Alex Krizhevsky. 2009. Learning multiple layers of features from tiny images. Retrieved from http://www.cs.utoronto.ca/ kriz/learning-features-2009-TR.pdf
[36] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. 2018. Adversarial examples in the physical world. *Artificial Intelligence Safety and Security*. Chapman and Hall/CRC, 99–112.
[37] Stefano Carlo Lambertenghi and Andrea Stocco. 2024. Assessing quality metrics for neural reality gap input mitigation in autonomous driving testing. In *Proceedings of the 2024 IEEE Conference on Software Testing, Verification and Validation (ICST '24)*. IEEE, 173–184. DOI: https://doi.org/10.1109/ICST60714.2024.00024
[38] Craig Larman, et al. 1998. *Applying UML and Patterns*. Vol. 2. Prentice-Hall.
[39] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86, 11 (1998), 2278–2324.
[40] Bruno Legeard, Fabien Peureux, and Mark Utting. 2002. Automated boundary testing from Z and B. In *Proceedings of the International Symposium of Formal Methods Europe*. Springer, 21–40.
[41] Yue Liu, Lichao Feng, Xingya Wang, and Shiyu Zhang. 2022. DeepBoundary: A coverage testing method of deep learning software based on decision boundary representation. In *Proceedings of the 2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*. IEEE, 166–172.
[42] Ilya Loshchilov and Frank Hutter. 2017. Decoupled weight decay regularization. arXiv:1711.05101. Retrieved from https://arxiv.org/abs/1711.05101.
[43] Yujie Lu, Xianjun Yang, Xiujun Li, Xin Eric Wang, and William Yang Wang. 2023. Llmscore: Unveiling the power of large language models in text-to-image synthesis evaluation. *Advances in Neural Information Processing Systems* 36, (2023), 23075–23093.
[44] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Chunyang Chen, Ting Su, Li Li, Yang Liu, et al. 2018. Deepgauge: Multi-granularity testing criteria for deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. ACM, New York, NY, 120–131. DOI: https://doi.org/10.1145/3238147.3238202
[45] TorchVision Maintainers and Contributors. 2016. *TorchVision: PyTorch's Computer Vision Library*.
[46] Maryam, Matteo Biagiola, Andrea Stocco, and Vincenzo Riccio. 2025. Benchmarking generative AI models for deep learning test input generation. In *Proceedings of 18th IEEE International Conference on Software Testing, Verification and Validation (ICST '25)*. IEEE, 12 pages.
[47] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. arXiv:1504.04909. Retrieved from http://arxiv.org/abs/1504.04909

[48] Nusrat Jahan Mozumder, Felipe Toledo, Swaroopa Dola, and Matthew B. Dwyer. 2025. RBT4DNN: Requirements-based testing of neural networks. arXiv250402737. Retrieved from https://arxiv.org/abs/2504.02737
[49] Neelofar Neelofar and Aldeida Aleti. 2024. Identifying and explaining safety-critical scenarios for autonomous vehicles via key features. *ACM Transactions on Software Engineering and Methodology* 33, 4 (2024), 1–32.
[50] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y. Ng, et al. 2011. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. Granada, Vol. 2011, 4.
[51] Annibale Panichella. 2022. An improved Pareto front modeling algorithm for large-scale many-objective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 565–573.
[52] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. Retrieved from https://openreview.net/forum?id=BJJsrmfCZ
[53] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated whitebox testing of deep learning systems. *Communications of ACM* 62 (2017), 1–18. DOI: https://doi.org/10.1145/3132747.3132785
[54] Mauro Pezzè and Michal Young. 2008. *Software Testing and Analysis: Process, Principles, and Techniques*. John Wiley & Sons.
[55] Vincenzo Riccio, Nargiz Humbatova, Gunel Jahangirova, and Paolo Tonella. 2021. DeepMetis: Augmenting a deep learning test set to increase its mutation score. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE '21)*. IEEE/ACM.
[56] Vincenzo Riccio, Gunel Jahangirova, Andrea Stocco, Nargiz Humbatova, Michael Weiss, and Paolo Tonella. 2020. Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering* 25 (2020), 5193–5254.
[57] Vincenzo Riccio and Paolo Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 876–888.
[58] Vincenzo Riccio and Paolo Tonella. 2023. When and why test generators for deep learning produce invalid inputs: An empirical study. In *Proceedings of 45th International Conference on Software Engineering (ICSE '23)*. ACM, 12 pages.
[59] Axel Sauer, Katja Schwarz, and Andreas Geiger. 2022. StyleGAN-XL: Scaling StyleGAN to large diverse datasets. In *ACM SIGGRAPH 2022 Conference Proceedings*, 1–10.
[60] Hinrich Schütze, Christopher D. Manning, and Prabhakar Raghavan. 2008. *Introduction to Information Retrieval*, Vol. 39. Cambridge University Press, Cambridge.
[61] Leslie N. Smith and Nicholay Topin. 2018. Super-convergence: Very fast training of neural networks using large learning rates. arXiv:1708.07120. Retrieved from https://arxiv.org/abs/1708.07120
[62] Alexander Sorokin and David Forsyth. 2008. Utility data annotation with Amazon Mechanical Turk. In *Proceedings of the 2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 1–8.
[63] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2018. Testing deep neural networks. arXiv:1803.04792. Retrieved from https://arxiv.org/abs/1803.04792
[64] Pedro Tabacof and Eduardo Valle. 2016. Exploring the space of adversarial images. In *Proceedings of 2016 International Joint Conference on Neural Networks (IJCNN)*, 426–433. DOI: https://doi.org/10.1109/IJCNN.2016.7727230
[65] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, 303–314. DOI: https://doi.org/10.1145/3180155.3180220
[66] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025. Can LLMS replace human evaluators? An empirical study of LLM-as-a-judge in software engineering. In *Proceedings of the ACM on Software Engineering*. ISSTA, Vol. 2, 1955–1977.
[67] Michael Weiss, André García Gómez, and Paolo Tonella. 2023. Generating and detecting true ambiguity: A forgotten danger in DNN supervision testing. *Empirical Software Engineering* 28, 6 (2023), 36 pages. DOI: https://doi.org/10.1007/s10664-023-10393-w
[68] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics Bulletin* 1, 6 (1945), 80. DOI: https://doi.org/10.2307/3001968
[69] Han Xiao, Kashif Rasul, and Roland Vollgraf. 2017. Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. arXiv:1708.07747. Retrieved from https://arxiv.org/abs/1708.07747
[70] Sergey Zagoruyko. 2016. Wide residual networks. arXiv:1605.07146. Retrieved from https://arxiv.org/abs/1605.07146
[71] Chongsheng Zhang, George Almpanidis, Gaojuan Fan, Binquan Deng, Yanbo Zhang, Ji Liu, Aouaidjia Kamel, Paolo Soda, and João Gama. 2024. A systematic review on long-tailed learning. arXiv:2408.00483. Retrieved from https://arxiv.org/abs/2408.00483

[72] Fuyuan Zhang, Sankalan Pal Chowdhury, and Maria Christakis. 2020. Deepsearch: A simple and effective blackbox attack for deep neural networks. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 800–812.

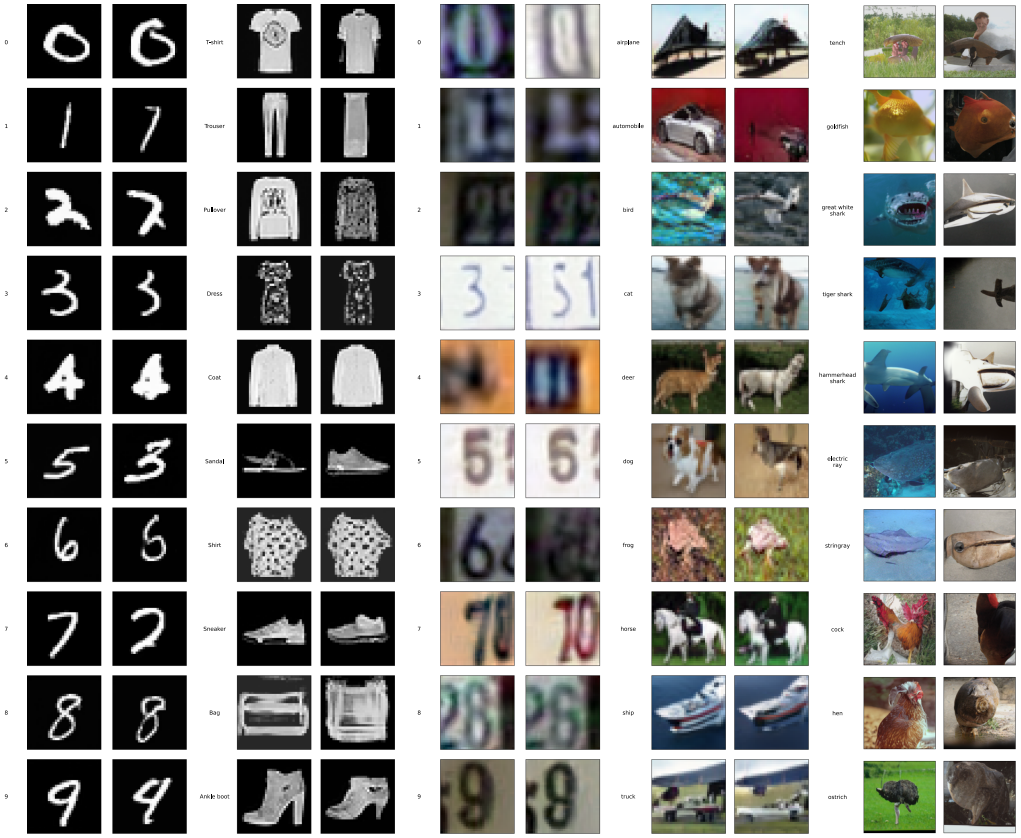
[73] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2020. Machine learning testing: Survey, landscapes and horizons. *IEEE Transactions on Software Engineering* 48, 1 (2020), 1–36. DOI: https://doi.org/10.1109/TSE.2019.2962027

[74] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE '18)*. ACM, New York, NY, 132–142. DOI: https://doi.org/10.1145/3238147.3238187

[75] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging LLM-as-a-judge with MT-bench and Chatbot arena. *Advances in Neural Information Processing Systems* 36, 2023, 46595–46623.

[76] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2021. Deephyperion: Exploring the feature space of deep learning-based systems through illumination search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA '21)*. ACM, 79–90.

Appendix

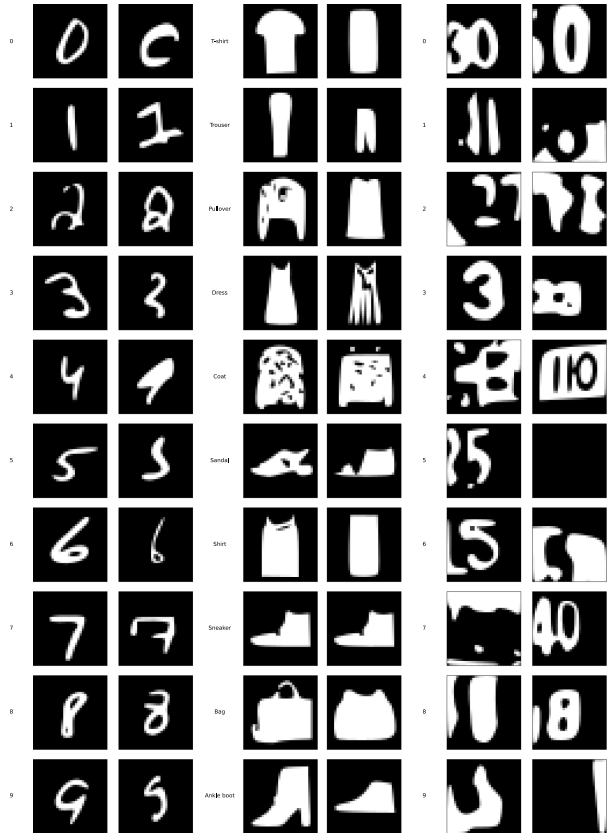


(a) MNIST (b) FashionMNIST (c) SVHN (d) CIFAR-10 (e) ImageNet

Fig. A.1. Mimicry original vs. final candidate examples.

(a) MNIST (b) FashionMNIST (c) SVHN (d) CIFAR-10 (e) ImageNet

Fig. A.2. Sinvad original vs. final candidate examples.



(a) MNIST (b) FashionMNIST (c) SVHN

Fig. A.3. DeepJanus original vs. final candidate examples.

Received 8 May 2025; revised 19 September 2025; accepted 28 September 2025