Distribution-aware Fairness Test Generation

Sai Sathiesh Rajan, Ezekiel Soremekun, Yves Le Traon, Sudipta Chattopadhyay

Abstract—Ensuring that all classes of objects are detected with equal accuracy is essential in AI systems. For instance, being unable to identify any one class of objects could have fatal consequences in autonomous driving systems. Hence, ensuring the reliability of image recognition systems is crucial. This work addresses *how to validate group fairness in image recognition software*. We propose a *distribution-aware fairness testing* approach (called DISTROFAIR) that systematically exposes class-level fairness violations in image classifiers via a synergistic combination of *out-of-distribution (OOD) testing* and *semantic-preserving image mutation*. DISTROFAIR automatically *learns the distribution* (e.g., number/orientation) of objects in a set of images. Then it *systematically mutates objects in the images* to become OOD using three *semantic-preserving image mutations – object deletion*, *object insertion* and *object rotation*. We evaluate DISTROFAIR using two well-known datasets (CityScapes and MS-COCO) and three major, commercial image recognition software (namely, Amazon Rekognition, Google Cloud Vision and Azure Computer Vision). Results show that about 21% of images generated by DISTROFAIR reveal class-level fairness violations using either ground truth or metamorphic oracles. DISTROFAIR is up to 2.3x more effective than two main *baselines*, i.e., (a) an approach which focuses on generating images only *within the distribution* (ID) and (b) fairness analysis using only the original image dataset. We further observed that DISTROFAIR is efficient, it generates 460 images per hour, on average. Finally, we evaluate the semantic validity of our approach via a user study with 81 participants, using 30 real images and 30 corresponding mutated images generated by DISTROFAIR. We found that images generated by DISTROFAIR are 80% as realistic as real-world images.

1 INTRODUCTION

Image classification has several critical applications in autonomous driving, robotics and healthcare, among others. Image classification may involve several tasks [57]. For instance, given an image, one of the crucial tasks for several autonomous applications is to recognize the different objects in the image i.e., multi-label object classification (MLC) [57]. Consider the MLC system used in autonomous driving, it is pertinent for the classifier to detect the objects on roads, including vehicles, pedestrians and animals; all with *fairly* high accuracy. Failure to do so may lead to severe consequences, resulting in accidents. Indeed, image classification software have shown significant biases towards certain *class(es)*, e.g., dark-skinned people were more likely to be misclassified [37] and women were usually associated with activities such as cooking, shopping etc [68]. Disparities between class-level accuracy of a given image classification task may have several societal, legal and safety concerns. Therefore, systematic testing of image classification task, to detect potential bias against certain classes, is of critical importance.

In this paper, we study the fairness of class-level accuracy in image classification tasks, specifically in MLC tasks. We choose MLC due to its applicability in several safety critical, autonomous applications e.g., driving and robotics. Given an arbitrary MLC model (*system under test* (SUT)) and a set of initial images, our fairness test generation approach (called DISTROFAIR) highlights the classes that face *unusually high error rates* for the SUT to reveal an unfair treatment of one class as compared to others. Additionally, each error is associated with concrete test images that can be used by the developer to further investigate the errors.

• *S.S. Rajan, and S. Chattopadhyay are with Singapore University of Technology and Design. E. Soremekun is with Royal Holloway University of London. Y.L. Traon is with SnT, University of Luxembourg.*

Our approach employs *out-of-distribution* (OOD) testing. By learning the distribution of objects detected in an initial set of images, DISTROFAIR systematically generates a set of images that portrays a *distributional shift* in the image dataset, such that the generated images are "outside" the learned distribution of objects in the initial sample. The generated images are called *OOD images*. The *key insight* behind our approach is to *ensure that the fairness properties of an MLC system generalize to unlikely, yet possible scenarios via OOD images*. We hypothesize that developers may ascertain fairness properties on likely scenarios (aka in-distribution) but ignore the unlikely scenarios, i.e., OOD. For instance, consider a scenario where we generate a crowded road scene e.g., by inserting many pedestrians in an image that contained only a few pedestrian objects. Suppose we find that the accuracy of the "traffic light" class in such an OOD image is significantly lower than the accuracy of the "car" class. Then, this implies that the prediction of the "traffic light" class is *unfair* in comparison to the "car" class. Such a different treatment for the two classes violates statistical parity [49]. DISTROFAIR works both in the presence and absence of ground truth, making it general and applicable also to unlabelled/partially labeled datasets. *To the best of our knowledge, we present the first OOD testing approach to discover and analyze the class-level fairness errors in image classification tasks.*

Figure 1 illustrates the different steps of our DISTROFAIR approach. DISTROFAIR starts with randomly sampled images from a dataset. This sample set of images are then clustered into different similar sub-groups to take into account the diversity of images in the initial sample. For each sub-group/cluster, DISTROFAIR then computes a distribution of objects detected by the SUT. Such distribution includes information about the minimum and maximum number of objects detected for each class and their orientation. Subsequently, OOD images are generated by leveraging

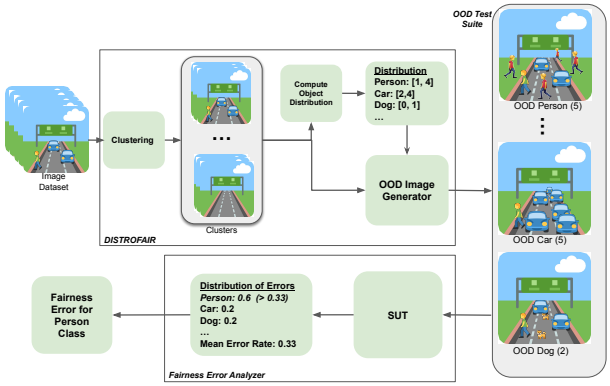


Fig. 1: An illustration of our DISTROFAIR approach.

this information and using semantic-preserving mutation operators (e.g., insertion, deletion and rotation of objects). For instance, three OOD images are shown in Figure 1, each one exceeds the maximum number of "*Person*", "*Car*" or "*Dog*" objects detected by the SUT in the respective cluster. Finally, the SUT is subject to analysis on the generated OOD images. As observed in Figure 1, if a class (e.g., "*Person*") is detected with an error rate (i.e., 0.6) more than the mean error rate across all classes (i.e., 0.33), then DISTROFAIR highlights the class (i.e., *"person"*) as facing a fairness error. Although we target our evaluation for MLC tasks, our OOD testing approach is general and can be applied to other multi-label image classification tasks.

Despite several approaches on fairness testing [48, 65, 40] and functional testing [36, 44] of machine-learning based systems, systematic fairness testing of class-level errors is relatively less explored. Our approach is complementary to recent effort in detecting class-level confusion and bias errors in deep learning models [45]. In particular, while the aforementioned work presented new metrics for confusion and bias detection for a class [45], we propose an OOD test generation approach to complement the detection of class-level fairness errors. Recent works on image fuzzing are focused on generating semantically valid images [55] or detecting functional errors without evaluating semantic validity [52] [62]. In contrast, we propose a novel OOD test generation method for systematically discovering class-level fairness errors. We also evaluate the semantic validity of generated OOD images via a user study.

This paper makes the following contributions:

1) We formalize how to measure class-level fairness errors in image recognition software and propose a novel OOD test generation approach (DISTROFAIR) to discover such errors (section 3).
2) We propose and implement three metamorphic OOD transformations such that the resulting images are semantically valid with high likelihood (section 3).
3) Based on the OOD images, we propose an automated approach to detect the class-level fairness errors in image classification tasks (section 3).
4) We implement our DISTROFAIR approach and evaluate it with three image classification systems from major vendors (Google, Amazon and Microsoft) using two datasets (MS-COCO and CityScapes). Our evaluation generates ≈24K error-inducing OOD images (out of a total ≈ 112K OOD images), finding nearly 368 classes (out of a total 879 classes) facing fairness errors across different models, datasets, OOD style mutations and fairness test oracles (section 5).
5) We compare our OOD test generation approach with two main baselines, namely (a) fairness analysis using *only* the original dataset, and (b) a test generation approach tailored to generating inputs *within distribution* (ID). We show that our OOD test generation approach improves the discovery of fairness error rate by up to 131.48% (section 5).
6) We conduct a user study to evaluate the semantic validity of our OOD images. Our study reveals that our generated OOD images are about 80% as realistic as original, real-world images, on average (section 5).

We discuss threats to validity (section 6). We then describe closely related work (section 7) before concluding (section 8).

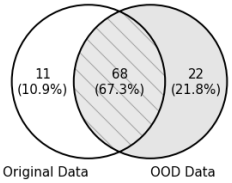


Fig. 2: Classes with higher than the mean error rate in original sample vs. OOD sample generated from the original.

2 OVERVIEW

In this section, we outline the motivation behind our approach and illustrate it with an example.

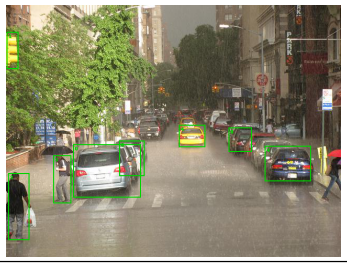
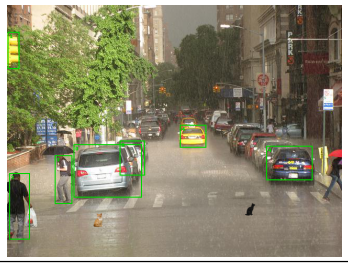
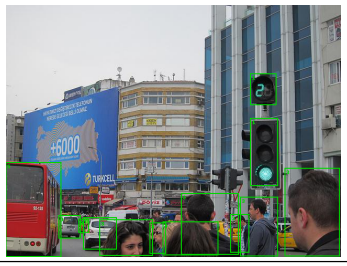
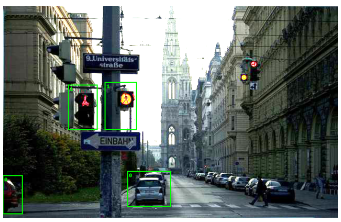
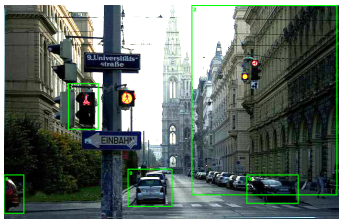
Class-level fairness: In this work, we investigate and discover class-level fairness errors in computer vision (CV) systems. Class level fairness is directly related to the concept of group fairness. Fundamentally, group fairness is concerned with ensuring that different groups exhibit similar statistical properties under similar stimuli [49]. For instance, a CV system that recognizes different classes of objects (e.g., cars, people) with a similar degree of precision and recall is said to be fair. This formulation is appropriate in safety-critical situations such as autonomous driving, where accurately identifying all objects on the road is desirable. More concretely, an autonomous car with a CV system that preferentially detects vehicles when compared to pets or animals is unfair. We note that such a formulation does not make any assumptions on the protected group(s). Concretely, for two arbitrary class labels a and b, we expect that class-level fairness is satisfied if and only if the following holds for a model f with a set of classes $\mathbb{C}$:

$$Pr(f(a)) \approxeq Pr(f(b)) \quad \forall a, b \in \mathbb{C} \tag{1}$$

where $Pr(f(a))$ and $Pr(f(b))$ capture the probability that class a and class b are correctly classified by f, respectively.

Key Insight (Why OOD samples?): OOD testing is increasingly becoming popular to evaluate the capability of an ML-based system beyond the training set [43, 61]. In particular, significant manual effort has been put forward to create OOD benchmark [17]. Moreover, we have observed

TABLE 1: Outline of DISTROFAIR: Inclusion errors [Inc.] are highlighted in blue, exclusion errors [Ex.] are highlighted in red, and GT errors are underlined. The numbers within (parenthesis) in column 3 and column 5 capture respective ground truths.

Subject/ Mutation	Original Image	Detected Objects	Mutated Image	Detected Objects
MS (Insertion) Cat		Car: 3 (15) Person: 2 (7) Taxi: 2 (2) Traffic Light: 1 (5)		[Ex.] Car: 2 (15) Person: 2 (7) Taxi: 2 (2) Traffic Light: 1 (5)
AWS (Deletion) Person		Car: 3 (8) Person: 7 (12) Traffic Light: 2 (3) Bus: 1(1)		Car: 3 (8) [Inc.] Traffic Light: 3 (3) [Inc.] Bus: 2 (1)
GCP (Rotation) Person		Car: 2 (12) Traffic Light: 2 (6)		[Inc.]Car: 3 (12) [Ex.]Traffic Light: 1 (6) [Inc.]Building: 1 (1)

a line of research that focused on improving the accuracy of ML models on OOD benchmark [3]. This is particularly important in autonomous driving systems since the training data is unlikely to capture the full set of scenarios that can occur in the real world. In addition, certain scenarios might not be easily obtained from real world testing on account of it being dangerous or time consuming. For instance, the likelihood of encountering a car that is being driven on the wrong side of the road is comparatively low and the number of such scenarios captured in the training set is likely to be small. As such, we postulate that OOD generation approach that seeks to generate these unseen scenarios is likely to be effective at exposing weaknesses in image recognition systems, including fairness issues.

In this paper, we propose a methodology to automatically generate OOD images from arbitrary image samples to validate class-level fairness of a target ML model. Our *key insight* is driven by the observation of *distributional shift* in class-level accuracy between an original dataset and their corresponding OOD images. Figure 2 illustrates the set of classes that have higher than the mean accuracy across three widely used object recognition models from Microsoft, Amazon (AWS), and Google (GCP). This is shown both for a sample of original data (taken from an existing dataset) and the OOD images created from this sample using DISTROFAIR. Concretely, we observe that the accuracy of 21.8% of classes drops below the mean accuracy *only when considering the OOD images*. From this observation, *we posit that inducing distributional shifts* (such as those illustrated in Figure 2) *may unmask hidden biases*. Therefore, it is desirable to investigate class-level biases in the OOD dataset w.r.t. to its distributional shift from the original dataset. Our generation of OOD images considers scenarios that may occur in real world. Thus, the class-level accuracy on the OOD images provides the model developers useful debugging information. For instance, such information may highlight the specific classes where the model performs poorly when stressed with generated OOD images.

An illustrative example: Table 1 shows an example illustrating our OOD-image generation and the class-level error detection. All the illustrated errors are taken from our evaluation on real-world system from Microsoft (MS), Amazon (AWS) and Google (GCP). The first column shows the targeted subject (MS/AWS/GCP) and the mutation operation (e.g., insertion, deletion, rotation of object). The second column captures the original image and the third column highlights the class-level detection on the original image by the respective subject. The fourth column captures the OOD image based on the mutation shown in the first column and the rightmost column captures the subject output on the mutated images.

Intuitively, given a dataset S and a model M, we capture the distribution of any class $c \in \mathbb{C}$ ($\mathbb{C}$ being the set of all classes) as follows: we record the minimum and maximum occurrences of class c detected by M for any image $s \in \mathbb{S}$. Additionally, we also record the orientation (angle) in a similar fashion for all classes. The generation of OOD images for model M thus focuses on creating an image that deviates

from the captured distribution. For instance, consider the insertion operation in Table 1 for MS. In our evaluation, we observed that MS did not detect any *cat* class for our original sample set. Thus, we consider the insertion of even a single *cat* object will result in an OOD image. In the example shown in Table 1, we insert two *cat* objects as shown in the mutated image. As a consequence, MS fails to detect one of the *car* objects that was detected in the original image. In general, we consider two different test oracles as follows to detect errors in the generated OOD images:

1) *Ground Truth* (**GT**) based Oracle: A class c in an OOD image faces error if and only if the detection accuracy of c with respect to the ground truth drops below the detection accuracy of c in the corresponding original image. For example, the insertion operation shown in Table 1 drops the detection accuracy of the *Car* class in the OOD image (from $\frac{3}{15}$ to $\frac{2}{15}$). Hence, one error is accounted for the *Car* class. In contrast, the detection accuracy of *Traffic Light* class improves with the deletion operation for AWS. As the detection accuracy improves with respect to ground truth, we do not count such phenomenon as an error. Nonetheless, we also account for such improvement in accuracy, as our approach is targeted to compute fairness metrics across classes. Hence, our approach allows for negative errors to consider cases where the detection of a class improves with mutation. Formally, the number of errors for an unmodified class c (via the mutation operation) is accounted as follows:

$$Err_c = |num_{ood}(c) - GT_c| - |num_{orig}(c) - GT_c| \quad (2)$$

where $num_{ood}(c)$, $num_{orig}(c)$ and GT_c capture the number of class c objects detected in the OOD image, in the corresponding original image and the ground truth for class c in the original image, respectively.

2) *Metamorphic* (**MT**) Oracle: It is often infeasible in practice to use the ground truth data due to the unavailability of perfectly labeled data. Moreover, class detection varies across subjects, tasks and contexts. For example, a speed camera detects only license plates, whereas surveillance systems track multiple objects. Similarly, even for the same class, models might prioritize foreground objects over background objects. Consequently, a universal ground truth may not capture the intent of the model under test. To address this, we also design a metamorphic (MT) oracle that considers changes in detection accuracy with respect to the detection accuracy in the original image. In other words, we capture the intent of the targeted model in line with its accuracy in the original, unmodified image. Then, we investigate whether the prediction of different classes are consistent with respect to OOD style mutations.

   Concretely, we consider errors in two categories: *(i) Inclusion error* means that some object from a given class was *not detected* in the original image, but it is detected in the corresponding OOD image. *(i) Exclusion error* means that some object from a given class was detected in the original image, but it is *not detected* in the corresponding OOD image. As illustrated in Table 1, the deletion operation leads to one inclusion error for the classes *Traffic Light* and *Bus* in AWS. On the contrary, the insertion operation results in one exclusion error in MS for the *Car* class, whereas the rotation operation leads to an exclusion error in GCP for the *Traffic Light* class. We compare the effectiveness of both the GT and MT oracles in **RQ1**.

We exclude any errors due to the mutated class (i.e., the *cat* class for insertion operation). This is to eliminate the potential impact of bias in our experiments, as the mutated class is often likely to have more errors than the unmodified classes.

Our OOD image mutation is carefully engineered to generate semantically valid images. For example, while inserting an object, DISTROFAIR tries to compute the appropriate size of the respective object in the image. This is accomplished by heuristically estimating the size of the inserted object with respect to the size of existing objects in the image. For example, as observed from Table 1, our mutation inserts appropriately sized *cat* objects. Likewise, the other mutations keep the classes in the OOD image recognizable.

Computing fairness errors: Starting with a dataset S, we apply all the operations (insertion/deletion/rotation) to get the set of OOD images S'. For a given model M, we then compute the number of exclusion and inclusion errors for each class $c \in \mathbb{C}$ over the dataset S'. Such errors provide an overall distribution of errors across all classes in the OOD image set. We consider that a class $c \in \mathbb{C}$ exhibits fairness errors when its error rate exceeds the mean error rate across all classes. For example, if Err_c captures the error rate for class c, then a class c' exhibits fairness error if and only if $Err_{c'} > \frac{\sum_{c \in \mathbb{C}} Err_c}{|\mathbb{C}|}$. We note that Err_c is computed as the ratio between the total number of errors faced by class c in S' and the total number of objects of class c, in dataset S. In Table 1, using the MT oracle, the *car* class has an error rate of 33% (=1/3) for MS considering just one image in S'. Likewise for AWS, the classes *Traffic Light* and *Bus* have error rates of 50% and 100%, respectively.

3 METHODOLOGY

In this section, we first formally define the notion of OOD images considered within DISTROFAIR. Then we discuss DISTROFAIR in detail. DISTROFAIR can broadly be considered to have three components, namely, clustering, an OOD image generator and a fairness error analyzer. In the following, we elaborate each of the three components.

Definition 1. *(OOD Image)* Let us assume an initial set of images Img_{list} where the number of objects of a given class c is bounded by $\langle min_c, max_c \rangle$. Additionally, the set of orientations (angles) for any object of class c within Img_{list} is captured by Θ_c. We call an image $\mathcal{I}'$ an OOD image with respect to the initial set of images Img_{list} and the given class c if and only if one of the following conditions hold: *(i)* $\mathcal{I}'_c < min_c$ *(ii)* $\mathcal{I}'_c > max_c$ *(iii)* $\Theta(\mathcal{I}'_c) \notin \Theta_c$. $\mathcal{I}'_c$ captures the number of objects of class c in image $\mathcal{I}'$ whereas $\Theta(\mathcal{I}'_c)$ captures the set of orientations of class c objects in $\mathcal{I}'$.

3.1 Clustering

Our DISTROFAIR approach starts with an arbitrary sample of images. We first employ clustering on the initial sample to create smaller groups of images. This grouping is performed

Algorithm 1 OOD Image Generation.

```
1: procedure OOD_IMAGE_GENERATION(Img_List, OP_List, LBL_List)
2:     OOD_Set ← ∅
3:     MUT_List ← {x, y}: x ∈ (OP_List), y ∈ (LBL_List)
4:     ▷ F computes the distribution of the set of images in Img_List
5:     ▷ SUT is the ML model under test
6:     Dist_List ← F(Img_List, SUT)
7:     for M ∈ MUT_List do
8:         for Img ∈ Img_List do
9:             Dist_Img ← F_Img(Img, SUT)
10:            Mut_Num ← MutGen(Dist_Img, Dist_List)
11:            Gen_Img ← ImageGen(Img, M, Mut_Num)
12:            OOD_Set ∪= {(Gen_Img, Img, M)}
13:        end for
14:    end for
15:    return OOD_Set
16: end procedure
```

for images with similar objects and scenery. Additionally, the clustering handles variance of images/objects and the mixture of distributions in our initial sample. Specifically, our DISTROFAIR approach determines, for each image in the initial sample, the number of objects for each class. We leverage a state-of-the-art detection and segmentation library i.e., Detectron2 [58] for this purpose. The class-level information for all images is then fed to a clustering algorithm to divide the initial sample into similar subgroups. In general, our approach can leverage any clustering algorithm. We use K-Means clustering algorithm [31] within DISTROFAIR. Once the clusters of images are computed, OOD image generation is employed on each cluster of images independently. In the following, we discuss OOD image generation for an arbitrary cluster of images.

3.2 OOD Image Generation

Algorithm 1 outlines our OOD image generation process for a target ML model SUT. In the beginning, DISTROFAIR learns a distribution, $Dist_{List}$, for the set of images Img_{List} under test. The knowledge of this distribution is leveraged for OOD image generation process. Concretely, for each class $c \in \mathbb{C}$, the distribution captures a triplet $\langle \Theta_c, min_c, max_c \rangle$. Θ_c captures the set of orientations (angles) for objects in class c and min_c (respectively, max_c) captures the minimum (respectively, maximum) number of objects of class c detected by the SUT in Img_{List}. After computing the distribution $Dist_{List}$, DISTROFAIR aims to generate OOD images for each image in the Img_{List}. To this end, we consider a list of mutation operators MUT_{List} where each $M \in MUT_{List}$ is a pair, containing the operation (insertion/deletion/rotation) and the target class for mutation. For generating an OOD image, DISTROFAIR identifies the distribution for a single image, $Dist_{Img}$. $Dist_{Img}$ is used to compute the exact characteristics of the mutation for an OOD transformation. For example, given $Dist_{Img}$ and $Dist_{List}$, we compute the possible number insertions (e.g., MUT_{num} in Algorithm 1) of class c objects such that the total number of class c objects exceeds max_c. This is then used to produce the OOD image Gen_{Img} via the procedure $ImageGen$. All successfully generated OOD images are stored for subsequent analysis of class-level fairness errors.

3.3 Mutation Operators

Semantic-preserving Mutations: In this work, mutation operators are designed to preserve the image semantics i.e., *the meaning of the image in the real world* [55]. The goal is to preserve the perception of the original image, except for the mutated object(s). Our mutation operators rely on state-of-the-art tools for fine-grained image modifications. However, due to the current limitations of these tools, there is no guarantee that the semantics are always preserved in the OOD images. To mitigate this, we conducted a user study (**RQ4**) to check the semantic validity of generated images. In the following, we discuss the design details of the three mutation operators (*see* Table 1).

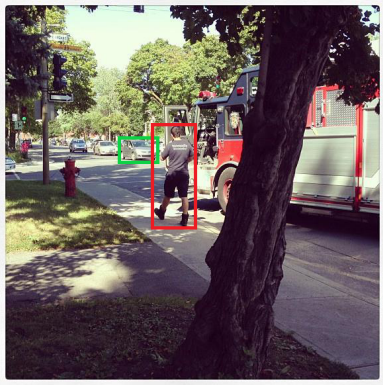
***Insertion*:** The insertion operation of DISTROFAIR employs several heuristics to ensure semantic correctness. We describe and illustrate this operation using Table 2.

We first determine the relative size of each object to be inserted by comparing it to a reference object. As an example, let us consider a "car" to be the reference object. We first find relative heights, w.r.t. the number of pixels, for all the other objects (e.g., "person" or "bicycles") to be inserted by determining how much larger or smaller they are in comparison to the reference object ("car"). We determine these relative sizes via initial experimentation. This is a one time effort that is leveraged for all insertion operations going forward.

DISTROFAIR then leverages a technique named panoptic segmentation [26] [58] on the original image to find the class label of each pixel. This is used to determine the size and location of the object to be inserted. In particular, we aim to determine what size the reference object would be if it were to be inserted in the middle of the image. In addition, we also determine how the size of the reference object would vary if the object were moved one pixel up or down. For instance, consider the image in Table 2. We observe that there are multiple cars and a person present in the image. Let us consider the person in the red box (top right image in Table 2 (Resizing Process)). DISTROFAIR uses our segmentation map to determine the height of the person in terms of the number of pixels. It then leverages our pre-computed relative reference sizes to determine the height of a hypothetical car in the same position. Similarly, it measures the height of the car in the green box. Next, it combines these information to compute the scaling factor, i.e., how the height of the car changes with respect to the changes in y-coordinate of a two-dimensional image. The scaling factor is employed to find the size of the reference "car" if it were to be situated in the center of the image. By passing this information along with the scaling factor, DISTROFAIR is able to easily determine the size of any object to be inserted in the original image, irrespective of its type or location with the aid of our knowledge of relative sizes. We further note that computing the scaling factor for each original image is a one-time effort, i.e., it is not recomputed for future insertion operations.

To identify appropriate locations for each insertion operation, DISTROFAIR implements checks to ensure that the object is placed at an appropriate location. For instance, it ensures that a person is placed on the ground (road, pavement, or dirt) by checking whether the pixels that will be occupied by the bottom portion of the object are classified as belonging to the ground. Table 2 (bottom left image) illustrates appropriate (✓) and inappropriate (×) locations for the insertion

TABLE 2: Table illustrating the different steps present in the insertion operation. First, the heights of the objects in the image are obtained and rate at which the size changes as the position changes is noted. We then determine whether the object can be safely placed on the ground in the chosen location and reject it in cases where it would not be on the ground. We then show the final result of the insertion operation.

Original Image	Resizing Process
	
Insertion Locations	**Final Image**
	

operation. In addition, it ensures that objects that are further away from the perspective of the observer are placed before objects that are nearer to the observer. This is to avoid objects in the background inadvertently obscuring objects in the foreground. Furthermore, DISTROFAIR determines the feasibility of placing the requisite number of objects into the original image. In the event that DISTROFAIR is unable to place enough objects to generate an OOD image, the insertion operation is not performed on the image. DISTROFAIR skips the image and attempts mutating the next image in the dataset. This prevents us from forcibly inserting objects into a crowded image that might be unable to accommodate the additional objects.

***Deletion*:** During deletion, DISTROFAIR deletes all object instances that belong to the class being mutated. We note that such deletion operation is an extreme case of OOD mutation when object deletion is considered for a given class. We choose this option to keep our test generation simple. DISTROFAIR leverages the panoptic segmentation map to identify the objects before applying a mask. It then uses inpainting [7] [42] to delete the masked objects.

***Rotation*:** For rotation, DISTROFAIR first identifies an object belonging to the target class, taking care to ensure that said object is not obstructed by another object. It then extracts the image level information for the object being rotated in that location before deleting the object through inpainting [7] [42]. Finally, it rotates the extracted image (i.e., the target object) and inserts it back into the original image. During insertion, we ensure that the physical dimensions such as height and width remain unchanged for the rotated object.

Algorithm 2 Fairness Error Analysis.

procedure FAIRNESS_ERROR_COUNTER($OOD\_Set, Case\_Type, Err\_Type$)
  $Tot\_Count \leftarrow \varnothing$
  $Err\_Count \leftarrow \varnothing$
  for $TUP \in OOD\_Set$ **do**
    ▷ number and type of objects in the image as found by $ORACLE$ and SUT
    $Oracle \leftarrow ORACLE(TUP, Case\_Type, Error_Type)$
    Let $Oracle = (GT, Diff)$
    for do$(-, Obj_L) \in Diff$
      ▷ Accumulate the total number of objects and errors for each class
      $Tot\_Count[Obj_L] += GT[Obj_L]$
      $Err\_Count[Obj_L] += Diff[Obj_L]$
      ▷ Increment error count for class Obj_L
      if $Diff[Obj_L] > 0$ **then**
        $Err\_ImgDict[Obj_L] = Err\_ImgDict[Obj_L] + 1$
      end if
    end for
  end for
end procedure

Algorithm 3 Fairness Error Oracle.

```
1: procedure ORACLE(TUP, Case_Type, Err_Type)
2:     if Case_Type = "SUT" then
3:         ▷ number and type of objects in the image as found by SUT
4:         GT_Pred ← SUT(TUP.Img)
5:     else if Case_Type = "GT" then
6:         ▷ GT is the union of all SUT results and the dataset ground truth
7:         GT_Pred ← GT(TUP.Img)
8:     end if
9:     Org_Pred ← SUT(TUP.Img)
10:    Gen_Pred ← SUT(TUP.Gen_Img)
11:    Let TUP.M = (−, LBL)
12:    Diff_Pred[LBL] ← ∅
13:    GT_Pred[LBL] ← ∅
14:    Org_Pred[LBL] ← ∅
15:    Gen_Pred[LBL] ← ∅
16:    for (−, Obj_L) ∈ GT_Pred do
17:        Org_Err ← |Org_Pred[Obj_L] − GT_Pred[Obj_L]|
18:        New_Err ← Gen_Pred[Obj_L] − GT_Pred[Obj_L]
19:        if Case_Type = "GT" then
20:            Diff_Pred[Obj_L] = |New_Err| − Org_Err
21:        else if Case_Type = "SUT" then
22:            if Err_Type = "INC" then
23:                if Change_Error > 0 then
24:                    Diff_Pred[Obj_L] = New_Err
25:                end if
26:            else if Err_Type = "EXC" then
27:                if Change_Error < 0 then
28:                    Diff_Pred[Obj_L] = −1 · New_Err
29:                end if
30:            end if
31:        end if
32:    end for
33:    Oracle_Ret = {GT_Pred, Diff_Pred}
34:    return Oracle_Ret
35: end procedure
```

3.4 Fairness Error Analysis

Algorithm 2 outlines our fairness error analysis. Given a set of OOD images (computed via Algorithm 1), Algorithm 2 computes, for each class, the total number of detected objects (Tot_Count) and the number of errors in the detection (Err_Count). To compute the number of errors, it relies on Algorithm 3 to find the expected number of objects in each class. Algorithm 3 takes the type of error being computed, and returns the appropriate number of errors for each class along with the initial reference.

To obtain the ground truth reference i.e., GT_{Ref} for each image, we use the following equation:

$$GT_{Ref}(Img) = GT_{Data}(Img) \cup \bigcup_{i \in All\_SUT} SUT_{(i)}(Img) \tag{3}$$

In essence, we take the reference to be the multiset union of the results from each subject under test and the ground truth from the provided data (GT_{Data}). We then set the expected count for the class of object being mutated to be zero, both in the original image and the corresponding OOD image (Line 11-Line 15). This prevents us from inadvertently including errors that were directly introduced by the mutated objects themselves. Intuitively, in the absence of errors, we expect the original and corresponding OOD image to detect the same number of objects for each class, except the mutated class. We then find the degree to which the detected output for the mutated images has changed from the original output. (Line 16-Line 18). This is used to compute the errors (Line 19-Line 28). Algorithm 2 then accumulates the error counts for all the images in the set of OOD images (Line 9-Line 11). It also calculates the number of images in which a particular class is exhibiting errors.

Once the errors for each class is computed in Err_Count, we can compute the error rate for each class as follows:

$$Err_c = \frac{Err\_Count[c]}{Tot\_Count[c]}, \quad \forall c \in \mathbb{C} \tag{4}$$

Finally, a class c faces a fairness error when its detection error rate exceeds the mean error rate across all classes:

$$Err_c > \frac{\sum_{i \in \mathbb{C}} Err_i}{|\mathbb{C}|} \tag{5}$$

In summary, the developer can use our framework to investigate the distribution of errors faced by each class and observe the classes exhibiting unusually high error rates. Additionally, each error is associated with a test case that allows the developer to investigate and reproduce the error.

3.4.0.1 ***Usage of the OOD Tests::*** Given the set of classes that induce fairness errors, developers can direct their efforts towards improving model performance for unfair classes. This could be achieved in several ways. For instance, developers could direct their data collection teams to obtain more instances of the unfair classes to augment their training data. Developers could also augment the training set with the error-inducing images generated by DISTROFAIR. Recent research [36, 44] has shown that the addition of error-inducing inputs to the training data improves the accuracy of computer vision models. We further note that our technique inherently generates scenarios that are previously unseen in the initial dataset. As such, the addition of the error-inducing inputs to the training data could conceivably improve the epistemic uncertainty [15, 22] of the models since they represent rare scenarios. This is particularly important in the case of autonomous driving where replicating these scenarios in the real world might be too dangerous or time consuming.

4 EVALUATION SETUP

We evaluate the following *research questions* (RQs):

- **RQ1 Effectiveness:** How *effective* is DISTROFAIR in generating error-inducing inputs that induce class-level fairness violations in image recognition software?
- **RQ2 Baseline Comparison:** How effective is the OOD mutation in comparison to the *baselines*?
- **RQ3 Efficiency:** What is the efficiency (time performance) of DISTROFAIR to generate fairness test cases?
- **RQ4 Semantic Validity:** Are the images generated by DISTROFAIR *semantically valid*, in terms of realism and likelihood of the depicted scenario occurring in real life? Are they comparable to real-world images?
- **RQ5 Generated images vs. Real-World OOD images:** What is the model accuracy of our SUT (image classifiers) on images generated by DISTROFAIR versus real-world OOD images?
- **RQ6 Original images vs. Error-inducing OOD images:** What is the accuracy of our SUT (image classifiers) on the error-inducing images generated by DISTROFAIR versus the corresponding original images from the dataset?

Datasets and Subject Programs: We selected MS-COCO and CityScapes (see Table 3) due to the large number of classes (thus, appropriate for testing class-level fairness) present, and their high prevalence in practice (e.g., autonomous cars)

TABLE 3: Details of Experimental Datasets.

Dataset	Description	#Images	#Classes	First Published
MS-COCO [29]	Microsoft Common Objects images	300	183	2014
CityScapes [10]	TU Darmstadt's Urban Street Scenes images	315	30	2015

TABLE 4: Details of Subject Programs.

Subject Programs	Description (No. of labels supported)	#Classes (Our Experiments)	Availability Date
GCP [16]	9000	89	2017
AWS [4]	2000+	76	2016
MS [33]	10000	58	2016

and the research community [19, 59, 27, 47, 32]. In particular, we randomly select 300 images containing traffic lights from MS-COCO. For CityScapes, we select 315 road scene images taken in Bremen. Restricting the choice of images in this manner allows us to limit the classes being considered for mutation to the set of objects that are most relevant in road scenes. In the absence of such filtering, we could conceivably insert a car into an image of the sky. Such an insertion would be inappropriate due to the low likelihood of such an situation occurring. Additionally, our chosen evaluation subjects (see Table 4) are the most prominent cloud-based image recognition systems supporting thousands of objects and scenes.

Metrics and Measures: These are defined as follows:

- **Class-level Violations & Violation Rate:** We detect *biased* classes via Equation 5. The *class-level violation rate* is the proportion of biased classes out of all considered classes (*see* **RQ1**/**RQ2**).
- **Error-inducing Inputs & Fairness Error Rate:** We consider a generated input (image) to be *error-inducing* if (1) it leads to an error for a subject and (2) it contributes to the number of errors for a class-level violation. The *fairness error rate* is the proportion of error-inducing inputs out of all generated inputs (section 5).
- **Test Generation Time:** This refers to the time-taken to generate a test suite for class-level group fairness (*see* **RQ3** section 5).

4.1 Research Protocol

We describe the experimental protocol for 18 different settings (two datasets, three subjects, and three mutations) in our experiments.

Clustering and Distribution of Objects: For finding the distribution of objects in our initial dataset, we use state-of-the-art library Detectron2 [58] to detect objects, whereas K-Means algorithm from SciKitLearn [38] was used for clustering. We have selected K-Means since it scales to large data sets, guarantees convergence and generalizes to clusters of different shapes and sizes [31].

Image Generation: For all mutations, DISTROFAIR attempts to generate one image for each selected class for each given image. All experiments except deletion were conducted five times to account for randomness in the position, orientation and type of mutated objects. Experiments for deletion were performed once since deletion is deterministic: We delete all objects of a class for all images.

TABLE 5: Details of Images in the User Study Dataset.

	# Images (# Error-inducing images)				
Dataset	**Real**	**Mutated**	**Insertion**	**Deletion**	**Rotation**
MS-COCO	20	20 (17)	6 (6)	9 (7)	5 (4)
CityScapes	10	10 (7)	4 (4)	1 (0)	5 (3)
Total	30	30 (24)	10 (10)	10 (7)	10 (7)

Mutated Objects: Due to time constraints, in our experiments, DISTROFAIR applies the mutations to a subset of all the classes present in the dataset. Developers can extend the technique to more classes by obtaining suitable images belonging to each class in question and expending more time to generate OOD images for each class in question. We do, however, consider all the classes present in $\mathbb{C}$ in our fairness error analysis. In particular, DISTROFAIR attempts to delete or rotate objects belonging to four class labels (namely people, cars, motorcycles, and trucks). These classes were selected due to their prevalence in the datasets. For insertion operation, apart from the four aforementioned classes, we also insert three additional classes (namely birds, cats and dogs), as these three classes are common in road scenes.

Image Caching: We cache the generated images for test efficiency. For a fair evaluation (**RQ3**), we only report the results for the initial runs for each subject, i.e., MS-COCO using Amazon Rekognition and CityScapes with Google Vision.

Baseline: We use two baselines to compare the effectiveness of our OOD mutations: 1) Original data, and 2) ID Mutations. In the first case, a developer aims to find class-level fairness violations *only* using the original data and not having access to DISTROFAIR. Meanwhile, ID mutations aim to transform an image in such a fashion that the distribution of objects (i.e., maximum and minimum number of occurrences and the angle of orientations) in the transformed image remains within the learned distribution in the respective cluster (i.e., $Dist_{List}$ in Algorithm 1).

Baseline Comparison: For the baseline only using original data, we use the ground truth information (Equation 3) to compute the accuracy of each class. Then, the unfair/biased classes are detected as the set of classes whose accuracy is below the mean accuracy across all classes (in line with Equation 5). For ID mutations, we compare it with the OOD mutation in DISTROFAIR for insertion operations. This is because most mutated classes in our experiments have a minimum object count of zero, thus deleting all objects of a class may often generate an image that is ID. Thus, there is no clear boundary between our OOD and an ID style deletion operation. Additionally, classes in our dataset have such orientation that any rotation of an object will result in an OOD image. Thus, for rotation, the only ID equivalent image is the original image. In the insertion experiment, we generate ID images by replacing the OOD constraints in DISTROFAIR with ID constraints, such that object insertions are only performed within the range of the distribution of the object in each cluster. However, due to the small range of ID vs. OOD, the generated ID inputs beyond first iterations are significantly smaller or already seen in the first. Hence, for a balanced evaluation, we compare only the first iteration

of DISTROFAIR with OOD to the single run of ID. For both OOD and ID, we use the same set of images in the initial datasets with an unlimited time budget.

OOD Image Accuracy: We investigate the impact of our OOD style mutations on model accuracy. To this end, we compare the accuracy of each SUT on our generated OOD images versus their model accuracy on corresponding real OOD images. In particular, for a given class c, we design an experiment to first compute the accuracy on an OOD image Img generated for class c. This accuracy is then compared with the accuracy on a real image Img' that contains the same number of objects for all classes being considered as in the OOD image Img. Similar to the baseline comparisons, we also use the insertion mutation operation in this experiment. *RQ5* presents the results of this experiment.

Specifically, we implement this experiment as follows: Using the dataset ground truth, ($GT_{Data}(Img)$ from Equation 3), we determine the number of objects for each considered class present in an original image, i.e., the classes considered for the insertion operation. We then use the dataset ground truth as a common oracle across all subject programs and for the original images. Additionally, the oracle is modified accordingly for generated OOD images, specifically, taking into account the number of inserted objects. Algorithm 4 details the procedures involved in implementing this oracle. Given the OOD image, Img, the list of classes to be considered, LBL_{List}, the class of object to be inserted, $Class_{Name}$, and the number of objects of said class inserted, $Insert_{Count}$, Algorithm 4 returns the oracle for the number of objects present in the corresponding OOD image (OOD_Oracle). We restrict the set of objects present to the list of classes being considered (Line 5-Line 7) for insertion. We then find the expected number of objects in the OOD image for each such class (Line 10-Line 14).

For each OOD image with respect to a class c ($\in LBL_{List}$), we identify original images from our dataset that have identical number of objects for all classes in LBL_{List} (according to ground truth GT_{Data}). We note that it is possible for multiple images in the dataset to satisfy such a criteria. Thus, we ensure that each generated OOD image is paired with exactly one such image (randomly taken) from the dataset. We then compare the accuracy with which the classes being considered, LBL_{List}, are detected in the OOD image (using the oracle OOD_Oracle) and the accuracy with which the classes being considered, LBL_{List}, are detected in the corresponding original image (using the GT_{Data} oracle). We also note that the total number of objects present in both OOD_Oracle and GT_{Data} oracle is necessarily equivalent since we specifically find images that contain the same number of objects.

Fairness Error Analysis: To determine class-level fairness violations, we first filter our classes that are not prominent (occurs ¡10 times) in our (sub-)datasets to avoid skewness. We perform filtering for all experiments, except for the baseline comparison (*see* **RQ2** section 5). This is due to the relatively smaller set of images involved in baseline (original data and ID generation).

Implementation Details and Platforms: DISTROFAIR contains 5K lines of Python code using Python 3.7. It uses (machine learning and image processing) packages such as PyTorch 1.9, CUDA 110, scikit-learn, numpy and Pillow.

Algorithm 4 OOD oracle for comparison.

```
1: procedure OOD_ORACLE(Img, LBL_List, Class_Name, Insert_Count)
2:     ▷ F returns the dataset ground truth for the image, Img
3:     GT_Data ← F(Img)
4:     OOD_Oracle ← ∅
5:     for (Obj_L, −) ∈ GT_Data do
6:         if Obj_L ∈ LBL_List then
7:             OOD_Oracle[Obj_L] ← GT_Data[Obj_L]
8:         end if
9:     end for
10:    for (Obj_L, −) ∈ OOD_Oracle do
11:        if Obj_L = Class_Name then
12:            OOD_Oracle[Obj_L] ← GT_Data[Obj_L] + Insert_Count
13:        else
14:            OOD_Oracle[Obj_L] ← GT_Data[Obj_L]
15:        end if
16:    end for
17:    return OOD_Oracle
18: end procedure
```

In addition, we also used the Detectron2 [58] and LaMa [42] to aid in the image generation. For evaluation, we use APIs (with default settings) for each of our subject programs (Table 4). All experiments were conducted on a Google Cloud Platform VM using an N1 series machine with one vCPU, 20 GB of memory and one attached Nvidia Tesla K80 GPU.

4.2 User Study Design

Our study had 105 users and 60 images to examine if the generated images are *realistic* to humans and *likely to occur in the real world*.

Study Dataset: We first randomly selected 30 mutated images from DISTROFAIR such that all three mutation operators were equally represented. We also took additional care to ensure that images from both datasets were included. In particular, we selected 20 images from MS-COCO (vs 10 from CityScapes) due its significantly larger number of class labels in comparison to CityScapes (183 vs. 20, *see* Table 3). We also ensure that most of the selected images (24 out of 30) induce errors for at least one subject program (*see* Table 5). We then select the corresponding 30 real images for comparison.

Survey Questionnaire: We provide participants a randomly ordered set of 60 images in our study dataset. To avoid bias, we ensure that all consecutive images do not have the same mutation operation, and a mutated image is not next to its corresponding original image. To validate the soundness of participant responses, we ask participants to also provide the number of vehicles in each image. Specifically, the following questions were posed:

- **Image Realism:** "On a scale of 0 to 10, how realistic is the image? "Realistic" means the image depicts or seems to depict real people, objects or scenarios."
- **Scene Likelihood:** "On a scale of 0 to 10, how likely is the scenario depicted in the image to occur in real life?"
- **Validation:** "How many vehicles (e.g., cars) are in this image?"

The questionnaire is available here: https://bit.ly/3B1Qc12

Participants: We conducted this study on Amazon Mechanical Turk (MTurk) [35]. We received 105 responses in 11.25 hours. Each participant took about 66 minutes to complete the study, on average.

Response Data Validation: We validated 81 responses by checking the answers for the number of vehicles in the

TABLE 6: Effectiveness of DISTROFAIR using GT oracle (maximum fairness error rate and violation rate for each (sub)category are in **bold**).

				#Class Violations	Violative Rate	Error Inputs		Fairness Error Rate
Subject	**Datasets**	**Mutation Ops**	**#Classes**	**Err Classes**	**Violative Rate**	**Inputs**	**#Gen Inputs**	**Error Rate**
GCP	MS-COCO	Insert	61	16	0.26	1233	6027	0.205
		Deletion	28	15	**0.54**	230	572	**0.402**
		Rotation	49	13	0.27	317	1425	0.222
	CityScapes	Insert	22	2	0.09	377	8486	0.044
		Deletion	14	4	0.29	99	620	**0.160**
		Rotation	19	8	**0.42**	297	2500	0.119
MS	MS-COCO	Insert	46	16	0.35	2895	5908	0.490
		Deletion	19	7	0.37	253	572	0.442
		Rotation	31	14	**0.45**	762	1425	**0.535**
	CityScapes	Insert	17	7	**0.41**	3305	9085	**0.364**
		Deletion	10	4	0.40	182	620	0.294
		Rotation	17	5	0.29	493	2500	0.197
AWS	MS-COCO	Insert	35	6	0.17	716	4583	0.156
		Deletion	17	8	**0.47**	96	572	**0.168**
		Rotation	27	10	0.37	124	1425	0.087
	CityScapes	Insert	15	2	0.13	139	7104	0.020
		Deletion	11	5	**0.45**	293	620	**0.473**
		Rotation	13	1	0.08	18	2500	0.007
Dataset	MS-COCO	All	313	105	**0.34**	6626	22509	**0.294**
	CityScapes	All	138	38	0.28	5203	34035	0.153
Subject	GCP	All	193	58	0.30	2553	19630	0.130
	MS	All	140	53	**0.38**	7890	20110	**0.392**
	AWS	All	118	32	0.27	1386	16804	0.082
Total	All	All	451	143	0.32	11829	56544	0.209

images. We randomly chose five unambiguous images (with few and clear number of vehicles) for validation. We also ensured that the user agreement for the number of vehicles in the images is high (above 75%). Then we set a 60% (3 out 5) correctness threshold for these five images.

Response Data Analysis: To determine semantic validity of our images, we collated the Likert scale scores for the 81 valid responses using the two questions on the realism of the images and the likelihood of the depicted scenarios. We analyse semantic validity using both scores for original versus mutated images across different mutations, datasets and error-inducing images (see **RQ4** in section 5).

5 EVALUATION RESULTS

RQ1 Effectiveness: We evaluate the effectiveness of DISTROFAIR using both the GT and MT oracle (Table 6 and Table 7), as discussed in section 2. The choice of GT test oracle is useful when practitioners have access to ground truth information on the dataset, whereas the MT oracle is useful when practitioners lack access to detailed information on the dataset or other subject programs.

Using Ground Truth Oracle: Table 6 shows that DISTROFAIR *is effective in exposing class-level fairness violations using ground truth information.* In particular, DISTROFAIR reveals 32% class-level fairness violations w.r.t. ground truth information. About *one in five inputs* (21%) generated by DISTROFAIR exposed a class-level fairness violation. We observed that MS-COCO dataset and the Microsoft Vision subject program are more error-prone than other datasets (i.e., CityScapes) and other subject programs (GCP and AWS). For instance, DISTROFAIR exposed more fairness violations (0.34 vs. 0.28) and generated more error-inducing inputs (0.294 vs. 0.153) for MS-COCO than CityScapes (*see* Table 6). Overall, DISTROFAIR effectively exposes class-level fairness violations with GT oracle.

21% of the OOD images generated by DISTROFAIR *reveal class-level fairness violations in 32% of classes, using ground truth oracle.*

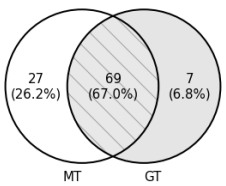
Using Metamorphic Oracle: Table 7 shows that DISTROFAIR revealed 32% class-level fairness violations relating to exclusion errors and 21% class-level fairness violations for inclusion errors. In addition, we observed that up to one in five inputs generated by our approach reveals a class-level fairness violation. For instance, 21% of the generated inputs exposed class-level fairness violations relating to inclusion errors across all settings (*see* Table 7). Although DISTROFAIR is effective across all settings, we found that it finds more errors using CityScapes dataset than using MS-COCO. We attribute the effectiveness to the use of distribution-aware mutations, which drive the input generation to induce class-level fairness violations.

Using the metamorphic oracle, one-fifth of the inputs generated by DISTROFAIR *revealed fairness errors in one-third of classes.*

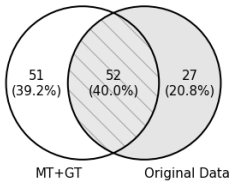
Test Oracle Comparisons: Figure 3a illustrates that the MT oracle exposed most (91% =69/76) of the fairness violations found by the GT oracle. Besides, *two-third (67% = 69/103) of all violated classes are found by both oracles*. We also observed that *almost 7% the violated classes found by the GT oracle are missed by the MT oracle*. This is due to the difference in the number of classes identified by both oracles. In our setting, the GT oracle identifies more classes than the MT oracle, since it obtains image recognition data from multiple sources (i.e., all

TABLE 7: Effectiveness of DISTROFAIR using MT oracle (maximum fairness error rate and violation rate for each (sub)category are in **bold**). **Ex.:** Exclusion, **Inc.:** Inclusion.

				#ClassViolations		Violative Rate		#Error-inducing inputs			Fairness Error Rate	
Subject	**Datasets**	**Mutation Ops**	**#Class**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**#gen-Inputs**	**Ex.**	**Inc.**
GCP	MS-COCO	Insertion	61	13	12	0.21	0.2	354	1072	6027	0.059	0.178
		Deletion	23	9	7	**0.39**	**0.3**	124	193	572	**0.217**	**0.337**
		Rotation	49	12	8	0.24	0.16	144	240	1425	0.101	0.168
	CityScapes	Insertion	22	11	2	0.5	0.09	2354	418	8486	**0.277**	0.049
		Deletion	11	4	4	0.36	**0.36**	159	195	620	0.256	**0.315**
		Rotation	18	7	4	**0.39**	0.22	269	50	2500	0.108	0.020
MS	MS-COCO	Insertion	43	16	5	0.37	0.12	1731	1640	5908	0.293	0.278
		Deletion	14	6	6	**0.43**	**0.43**	221	192	572	**0.386**	0.336
		Rotation	29	10	8	0.34	0.28	348	672	1425	0.244	**0.472**
	CityScapes	Insertion	17	7	5	**0.41**	0.29	416	2712	9085	0.046	**0.299**
		Deletion	9	3	3	0.33	**0.33**	129	70	620	**0.208**	0.113
		Rotation	17	3	5	0.18	0.29	213	291	2500	0.085	0.116
AWS	MS-COCO	Insertion	34	9	1	0.26	0.03	245	156	4583	0.053	0.034
		Deletion	16	5	6	**0.31**	**0.38**	74	49	572	**0.129**	0.086
		Rotation	26	7	6	0.27	0.23	100	223	1425	0.070	**0.156**
	CityScapes	Insertion	15	7	2	**0.47**	0.13	842	3886	7104	0.119	**0.547**
		Deletion	11	4	4	0.36	**0.36**	300	27	620	**0.484**	0.044
		Rotation	13	3	1	0.23	0.08	566	18	2500	0.226	0.007
Dataset	MS-COCO	All	295	87	59	0.29	0.20	3341	4437	22509	0.148	0.197
	CityScapes	All	133	49	30	**0.37**	**0.23**	5248	7667	34035	**0.154**	**0.225**
Subject	GCP	All	184	56	37	0.30	0.20	3404	2168	19630	**0.173**	0.110
	MS	All	129	45	32	**0.35**	**0.25**	3058	5577	20110	0.152	**0.277**
	AWS	All	115	35	20	0.30	0.17	2127	4359	16804	0.127	0.259
Total	All	All	428	136	89	0.32	0.21	8589	12104	56544	0.152	0.214



(a) Unfair Classes found by GT vs. MT oracles.



(b) Unfair classes (GT & MT Oracles vs. Original Data).

Fig. 3: Illustration of DISTROFAIR effectiveness.

subjects and dataset labels) in comparison to the MT oracle (a single subject). This directly influences the mean error rate and the found violated classes. Finally, we observed that the *MT oracle exposed 26% of fairness violations that are missed by the GT oracle*. Unlike the GT oracle, the MT oracle accounts for errors where the subject performs better on the mutated image (e.g., AWS in Table 1). This is useful to expose weaknesses in a subject.

The MT oracle is a good (proxy) estimator of the GT oracle. MT revealed most (69/76 ≈ 91%) of the fairness violations found by GT.

RQ2 Baseline Comparison: We compare DISTROFAIR to fairness analysis with (a) only *original data* (DISTROFAIR vs. Original Data) and (b) only *in-distribution* (ID) mutation (DISTROFAIR vs. ID).

DISTROFAIR *vs. Original Data:* In this experiment, we consider an approach with developers inspecting fairness violations *only* in the original dataset and without access to OOD test suite. Figure 3b highlights the similarity and differences in the class-level fairness violations exposed by such an approach with respect to DISTROFAIR.

We found that DISTROFAIR *exposes (30%) more class-level fairness violations than the original data (103 vs. 79)* (*see* Figure 3b). More importantly, a developer using only the original dataset will miss 39.2% (51 out of 130) of the class-level fairness violations exposed. In addition, DISTROFAIR is a good proxy for determining the class-level fairness violations found in the original dataset, since it exposes 66% (52 out of 79) of the class-level fairness violations exposed by the original dataset. These results highlight the need for generating OOD data, as they demonstrate that DISTROFAIR is effective in exposing fairness violations missed by the original dataset.

In addition, we performed a statistical analysis on both sets of images. We calculate the error rates for each image based on our ground truth reference, GT_{Ref}, taking care to exclude the class being mutated from the error calculation. More specifically, we conducted a Mann–Whitney U-test to determine whether the original images from the dataset were distinguishable from generated OOD images using their error rates. The Mann-Whitney U test shows that there is a significant difference between the two sets of images; it yields a test statistic of 41951407.5 and a p-value of $\approx 1 \cdot 10^{-46}$

Class-level fairness analysis with DISTROFAIR *is more effective than using only the original dataset.* DISTROFAIR *exposes (30%) more class-level fairness violations than the original dataset and 39.2% of all found violations were exposed by* DISTROFAIR *only.*

DISTROFAIR *vs. ID:* In this experiment, we compare the OOD style mutation of DISTROFAIR with the alternative *in-distribution* (ID) mutation. As discussed in subsection 4.1 (Baseline Comparison), we employ only the insertion opera-

TABLE 8: Comparison of DISTROFAIR, i.e., *out-of-distibution* (OOD) mutation-based fairness test generation approach to the baseline, i.e., *in-distribution* (ID) mutation-based fairness test generation. **Ex.:** Exclusion, **Inc.:** Inclusion.

				#ClassViolations		Violative Rate		#Error-inducing inputs			Fairness Error Rate	
Distribution	**Subject**	**Datasets**	**#Class**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**#gen-Inputs**	**Ex.**	**Inc.**
ID	GCP	All	81	13	14	0.16	0.17	24	121	1100	0.022	0.11
	MS	All	58	17	11	0.29	0.19	144	222	917	0.157	0.242
	AWS	All	54	7	4	0.13	0.07	14	161	1355	0.01	0.119
DISTROFAIR	GCP	All	78	25	11	0.32	0.14	503	466	2896	0.174	0.161
	MS	All	56	23	11	0.41	0.2	453	834	3005	0.151	0.278
	AWS	All	37	8	4	0.22	0.11	73	92	2343	0.031	0.039
ID	All	All	193	37	29	0.19	0.15	182	504	3372	0.054	0.149
DISTROFAIR	All	All	171	56	26	0.33	0.15	1029	1392	8244	0.125	0.169
Improvement (%)			NA	NA	NA	73.68	0	NA	NA	NA	131.48	13.42

tion for this comparison.

Our evaluation results show that *OOD style mutation outperforms the ID-based mutation approach in revealing class-level group fairness violations* (see Table 8). Specifically, DISTROFAIR reveals up to 74% more class-level fairness violations than the baseline for exclusion errors (*see* Table 8). In addition, we found that a developer is more than two times likely (up to 131%) to find class-level fairness errors with OOD than ID. Furthermore, OOD generates over 8K inputs and 1029 error-inducing inputs for exclusion errors, while ID generates only 182 error-inducing inputs and over 3K total inputs. This is particularly due to the fact that the input space for OOD is typically much larger than ID, since ID mutations are constrained within a static range. These results suggest that our use of OOD-based mutation contributes significantly to the effectiveness of DISTROFAIR.

Similarly, we conducted a Mann–Whitney U-test to determine whether the generated in-distribution images were distinguishable from generated OOD images when using their error rates. We found that there is a significant difference between the two sets of images; it yields a test statistic of 86298859.0 and a p-value of $\approx 2 \cdot 10^{-20}$.

OOD mutation significantly contributes to DISTROFAIR*'s effectiveness. It is up to 2.3X as effective as ID mutation.*

RQ3 Efficiency: This RQ examines the efficiency (time performance) of our approach (DISTROFAIR) in generating fairness test suites. For a fair and balanced evaluation, we only report the time-taken for DISTROFAIR during initial execution without *caching the generated images* for each dataset. Hence, we report the time-taken for the two initial experimental settings with MSCOCO using Amazon Rekognition (aka AWS) and the CityScapes dataset with Google Vision API (aka GCP).

Table 9 reports the test generation time of DISTROFAIR. It highlights that the two initial experimental setups took about 39 hours to complete the generation of 18K inputs. This implies that DISTROFAIR generates a fairness test case in about 7.7 seconds, on average. Moreover, the number of exposed fairness violations and generated error-inducing inputs within the test generation time is reasonable for a developer. For instance, DISTROFAIR generated hundreds (847) of error-inducing inputs and exposed 34 class-level fairness violations within 15 hours of fairness test generation, when testing AWS using the MS-COCO dataset (*see* Table 7). Further inspection shows that these results hold across

TABLE 9: Test Generation Efficiency of DISTROFAIR.

	Time Taken in seconds (#Images Generated)			
Dataset (subject)	**Insertion**	**Deletion**	**Rotation**	**Total**
MS-COCO (AWS)	38112 (4583)	2698 (572)	12502 (1425)	53312 (6580)
CityScapes (GCP)	54518 (8486)	5257 (620)	27118 (2500)	86893 (11606)
Total	92630 (13069)	7955 (1192)	39620 (3925)	140205 (18186)

mutation operations. In particular, the deletion operation is the fastest mutation operation (about 6.7 seconds) and the rotation operation is the most expensive operation (10 seconds), on average. Deletion operation is cheaper due to the single deterministic attempt at deleting all objects of the class in the image. In contrast, rotation is more expensive since it requires inpainting and insertion. The performance of DISTROFAIR across the datasets is similar. Specifically, DISTROFAIR took about 7.5-8 seconds to generate an input across both datasets. We attribute this efficiency to the lightweight and inexpensive nature of our distribution-aware mutation operations.

On average, DISTROFAIR *takes* $\approx$ *7.7 sec to generate a test.*

RQ4 Semantic Validity: To measure validity, we conduct two experiments, namely (1) a qualitative user study, and (2) a quantitative image quality experiment. Both of which are accompanied with statistical analysis. In the following, we report the settings and results of each experiment.

User Study: Firstly, we conducted a *user study* to evaluate the *semantic validity* of the images generated by DISTROFAIR. Our study involves 105 participants and 60 images (see subsection 4.2). This qualitative user study allows to accurately capture the realism of our images, especially from the human perspective. We note that a user study continues to be common practice in assessing the effectiveness of the generated images [14, 28, 13]. In addition, most methods used to assess image quality are based on learning-based models where a quality prediction model is learned from data that is labeled by humans [30, 23].

Our user study results show that *images generated by our test generator (*DISTROFAIR*) are semantically valid, when compared to real-world images*. Table 10 shows that our mutation operations are (up to 91%) as realistic as real-world images and (up to 92%) likely to occur in real life (*see "Real vs. Mut" deletion operation*). We observed that the deletion operation produces the most (up to 92%) semantically valid images. Meanwhile, the insertion operation produces the

TABLE 10: Semantic validity (realism and likelihood) of real images versus DISTROFAIR's generated images.

	Semantic Validity of All Images (only Error-inducing images)									
	Realism of Images					Likelihood of Scenarios				
Dataset	**Real**	**Mutated**	**Insertion**	**Deletion**	**Rotation**	**Real**	**Mutated**	**Insertion**	**Deletion**	**Rotation**
MS-COCO	7.83	6.56 (6.52)	5.66 (5.66)	7.14 (6.99)	6.59 (6.99)	8.08	6.89 (6.85)	6.12 (6.12)	7.44 (7.31)	6.81 (7.16)
CityScapes	8.02	5.93 (5.53)	4.67 (4.67)	7.84 (NA)	6.56 (6.67)	8.12	6.36 (6.03)	5.28 (5.28)	7.79 (NA)	6.95 (7.04)
Total	7.89	6.35 (6.23)	5.26 (5.26)	7.21 (6.99)	6.57 (6.85)	8.11	6.71 (6.61)	5.78 (5.78)	7.48 (7.31)	6.88 (7.11)
Real vs. Mut (%)	NA	80.4 (78.9)	66.7 (66.7)	91.4 (88.6)	83.3 (86.7)	NA	82.8 (81.6)	71.3 (71.3)	92.2 (90.1)	84.9 (87.7)

least realistic images, yet images resulting from the insertion operation are (up to 71%) likely to occur in real life. We also observed that these results are similar for the error-inducing images, i.e., images that cause an error in at least one subject program. Furthermore, we found that both benign and error-inducing images were seen as being similarly valid, realistic and likely to occur. Overall results show that all tested images generated by DISTROFAIR are 80% as realistic as real-world images. Participants also report that generated images depict scenarios that are 83% as likely to occur in real life when compared to the original images. This suggests that the OOD images generated by DISTROFAIR do not deviate significantly from real-world expectations of humans. Additionally, such results hold regardless of the error-inducing ability of the images and type of mutation operators.

Statistical Analysis (User Study): In addition, we performed a statistical analysis of our user study results. Specifically, we conducted a Mann–Whitney U-test to determine whether the original images from the dataset were indistinguishable from their corresponding OOD images using the reported realism scores of participants. The Mann-Whitney U test shows that there is a clear difference in the realism of the two sets of images, it yields a test statistic of 3791661.5 and a p-value of zero. Figure 4 also provides an overlapping frequency graph of the two sets of scores. It shows that the scores for both real and OOD images mostly overlap. However, we also observe that a portion of our OOD images have scores that are very low (between one and three inclusive) indicating that some of our generated images might be unrealistic. This is primarily due to the current limitations of the state-of-the-art software tools for image mutations (e.g., current object insertions techniques are unable to perfectly blend inserted objects into the original image). For instance, inserted objects might not perfectly match the lighting conditions present in the original image. As such tools become more mature in the future, we expect to obtain more realistic images using DISTROFAIR and our mutation operations.

Image Quality Analysis: Finally, we quantitatively evaluate the overall quality of the generated images using the PyTorch Image Quality (PIQ) library [24]. In particular, we evaluate the image quality of the set of original images versus OOD images (generated by DISTROFAIR) in our user study using CLIP-IQA [51]. CLIP-IQA evaluates the quality of an image using two antonym prompts. Antonym prompts allows it to to accurately determine where on the spectrum a particular image falls. In our experiments, we use CLIP-IQA to evaluate the quality of the image by providing it with the custom prompts, "Realistic photo" and "Unrealistic photo". This allows us to evaluate the realism of the image.

Our evaluation results show that the quality of the set of OOD images, generated by DISTROFAIR, are similar to

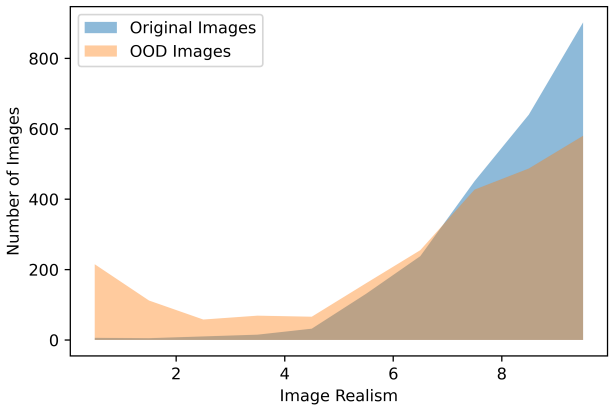


Fig. 4: Distribution of the number of images with a given realism score, where 1 is the lowest possible score and 10 is the highest possible score.

that of the original images. The CLIP-IQA score for the original images is 0.695, while the CLIP-IQA score for the OOD images, generated by DISTROFAIR, is 0.696, on average. Using the Mann-Whitney U test, we also perform statistical analysis to determine whether the two sets of images can be differentiated on the basis of the CLIP-IQA score. We found that the quality of the set of OOD images, generated by DISTROFAIR, are statistically indistinguishable from the original images. In particular, the Mann-Whitney U test yields a test statistic of 448.5 and a p-value of 0.99. These results show that DISTROFAIR and its mutation preserves the image quality in the original images.

Generated images are (up to 91%) as realistic as real images, and the depicted scenes are (up to 92%) likely to occur in real life.

RQ5 Generated images vs. Real-World OOD image: In this experiment, we compare the accuracy of our SUTs on 1) OOD images generated by the insertion operation in DISTROFAIR, versus 2) images from the dataset that contain equivalent number of objects present. Table 11 highlights our results.

We found that the SUT accuracy on OOD images generated by DISTROFAIR is better than the accuracy on real images containing similar number of objects (*see* Table 11). In particular, we note that AWS achieves similar accuracy for both generated images and real-world OOD images from the CityScapes dataset: In this case, the model accuracy on the images generated by DISTROFAIR are only 5.4% more than that of real-world OOD images. In addition, the OOD images produced by DISTROFAIR had a consistently higher accuracy when compared to the real images from the dataset. The difference in the accuracy can be attributed to our inserted objects being placed in focus, e.g., because our insertion

TABLE 11: Accuracy of our SUTs on OOD images generated by DISTROFAIR and similar images from the dataset.

	MSCOCO			CityScapes		
	Dataset Images	OOD Images	OOD Acc. Imp. (%)	Dataset Images	OOD Images	OOD Acc. Imp. (%)
GCP	0.180	0.290	60.9	0.132	0.358	170.6
MS	0.182	0.284	56.2	0.138	0.275	99.3
AWS	0.584	0.736	26.0	0.602	0.635	5.4

operation does not place objects behind already existing objects. Notably, images with object occlusion, which may be present in the images in the dataset, are more difficult to detect. However, for our approach, the increased likelihood of generating images without occlusion leads to much higher detection rates. Overall, these results indicate that (a) our image mutations do not adversely impact the accuracy of image recognition software, and (b) the errors exposed by DISTROFAIR are not due to performance degradation in the recognition of the image classifiers. This is evident for GCP where SUT accuracy on generated images is 2.7x as much as on real OOD images in the dataset.

SUT accuracy on OOD images generated by DISTROFAIR *is (up to 170%) more than real-world OOD images.*

RQ6 Original Images vs. Error-inducing OOD images: In this experiment, we compare the accuracy of our SUTs (image classifiers) on the error-inducing images generated by DISTROFAIR versus the corresponding original images from the dataset. In particular, we examine whether the SUTs correctly identify the objects from the non-mutated classes that are present in the image. We then compare the relative performance of each SUT across all non-mutated classes.

For non-mutated classes, we found that DISTROFAIR's mutation impairs SUT accuracy on the original images by up to 24.2%, on average. Table 12 shows that the accuracy of the SUT on error-inducing images is broadly lower than their accuracy on the corresponding original images. Furthermore, we find that the performance of our SUTs on CityScapes is worse than the performance on MSCOCO. We attribute this to the smaller set of classes found in CityScapes (30 as opposed to 183 in MSCOCO). For instance, certain objects (not present in the ground truth) recognised by the SUTs are excluded from the calculation of accuracy in Table 12. Such objects are present with higher frequency in CityScapes due to the relatively smaller set of classes in the ground truth. As such, the accuracy of our SUTs is lower for CityScapes as compared to MSCOCO.

We also find instances where the accuracy of the error-inducing images is similar to the accuracy on the original images. For instance, we see that AWS actually performs slightly better (1.4%) on the error-inducing images generated by DISTROFAIR on the MSCOCO dataset. The difference in accuracy can be attributed to the fact that these images are error-inducing and as such inherently contain classes that the SUTs struggle to classify. In addition, our mutation operators can introduce artifacts that affect the realism of the generated images. In conjunction with these artifacts, our insertion operator might also inadvertently cause occlusion that prevent SUTs from performing to their fullest abilities.

TABLE 12: Accuracy of our SUTs on non-mutated classes on error-inducing OOD images generated by DISTROFAIR and the corresponding original images from the dataset.

	MSCOCO			CityScapes		
	Original Images	OOD Images	Relative Acc. (%)	Original Images	OOD Images	Relative Acc. (%)
GCP	0.469	0.434	92.5	0.345	0.262	75.8
MS	0.344	0.285	82.9	0.292	0.237	81.1
AWS	0.649	0.658	101.4	0.656	0.463	70.5
Average	0.487	0.459	92.3	0.431	0.321	75.8

However, considering sufficiently large samples of original images, we note that these artifacts and occlusions ought to affect the different classes to a similar extent. As such, we should expect the performance of any particular class to drop by a similar amount when subject to these mutations.

On average, DISTROFAIR*'s mutation reduces the accuracy of the SUT on the OOD generated images versus the original images by up to 24.2% for non-mutated classes.*

6 LIMITATIONS AND THREATS TO VALIDITY

Internal Validity: The main threat to internal validity is whether our implementation indeed performs OOD-based test generation. We mitigate this threat by conducting typical software quality controls such as testing and code review. For instance, we ran several tests to ensure our implementation produced the expected outcome for each mutation, dataset and subject program. We also manually inspected random samples of generated images and compare them to the original image to ensure our mutation operations are indeed OOD and related to class-level fairness. Finally, we conducted a user study to examine the semantic validity of OOD images (**RQ4**).

Construct Validity: This relates to the metrics and measures employed in our experimental analysis. We mitigate this by employing standard measures of test generation effectiveness such as the number/rate of generated inputs, error-inducing inputs and fairness errors (or violations). Such measures are employed in the literature to evaluate fairness testing and test generation methods [9, 20, 39]. Additionally, our ground truth (GT) oracle implicitly relies on the accuracy of the labels in the dataset. These labels are typically labelled by humans. We mitigate this threat by having an alternative metamorphic (MT) oracle. We find that there is a substantial (67%) overlap between the unfair classes found by the two oracles.

External Validity: We acknowledge that DISTROFAIR may not generalize to all image datasets and image classifiers. However, we have evaluated our approach with well-known, commonly used datasets [39] (*see* Table 3). In addition, our subjects are off-the-shelf, mature, commercial image classifiers provided by software companies such as Google, Amazon and Microsoft (*see* Table 4).

Realism of Mutation Operators: The images generated by DISTROFAIR can contain elements that are inconsistent with the unmodified portions of the image. For instance, the objects introduced by the insertion operator could have been exposed to different lighting conditions than the objects

already in the image. The lack of appropriate shadow detail could conceivably lead to more errors. We also note that in some cases DISTROFAIR could conceivably attempt to remove a large portion of the original image due to the objects belonging to the mutating class making up most of the image. In such cases, the images generated by DISTROFAIR could be more unrealistic. We control for this by evaluating against an in-distribution baseline in RQ2. We also evaluate DISTROFAIR against OOD images present in the dataset in RQ5 and find that our mutation does not adversely affect the detection accuracy of the SUTs. This allows us to compare whether the OOD nature of the images is indeed causing the errors detected. Similarly, both the deletion and rotation can introduce artifacts that affect the realism of the generated images. Finally, our evaluation in RQ4 both subjectively (via user study) and objectively (via CLIP-IQA [51]) shows that our images are largely realistic, but we acknowledge that the mutation operators could benefit from better techniques that generate more realistic images.

7 RELATED WORK

Fairness Test Generation: Recent surveys [39, 20, 9] on software fairness show that researchers employ different software analysis and model analysis methods to expose bias in ML systems. On one hand, white box fairness testing approaches employ ML techniques (e.g., gradient computation, and clustering) to generate discriminatory test cases (e.g., ADF [65, 67] and EIDIG [64]). On the other hand, black-box approaches leverage the input space and search algorithms to generate discriminatory inputs, e.g., using schemas, grammar, mutation or search algorithms to drive fairness test generation [48, 60, 40, 41]. Grey-box fairness testing approaches [46] employ both input space exploration and model analysis for test generation. Besides, some methods employ program analysis techniques, e.g., symbolic execution [2] and combinatorial testing [34] to expose bias in ML systems. Likewise, we propose a black-box fairness test generation approach. Albeit, unlike prior works, we focus on fairness test generation for image recognition systems using distribution-aware and semantic-preserving mutations.

OOD Sampling, Distribution-aware & OOD Testing: Empirical studies on OOD testing have shown that it is important for test generation and revealing faults in ML systems. For instance, Berend et al. [6] found that data distribution awareness in both testing and enhancement phases outperforms distribution unaware retraining. Likewise, Zhou et al. [69] showed that OOD-aware detection modules have better performance and are more robust against random noises. Similar to these works, we show that OOD testing is important for automatically revealing faults in ML systems. Berend et al. [5] proposed a distribution aware robustness testing tool to generate unseen test cases for ML task and recommends that ML testing tools should be aware of distribution. Besides, Huang et al. [21] proposed a distribution-aware robustness testing approach for detecting adversarial examples using the input distribution and the perceptual quality of inputs. This work, unlike DISTROFAIR, focused on adversarial testing of ML, and not fairness testing.

Besides, Ackerman et al. [1] proposed an approach to find explainable data slices where a model underperforms. In contrast to this work, the objective of DISTROFAIR is to generate test inputs (images) for finding fairness errors. Additionally, the work by Ackerman et al. [1] neither generates tests beyond the dataset nor does it focus on fairness. Finally, Vernekar et al. [50] proposes an approach to generate OOD samples with the objective of improving the accuracy of classifiers on MNIST and Fashion-MNIST datasets. Our objective is orthogonal to this work. Specifically, we aim to generate tests that uncover fairness violations in commercial image recognition software. As such, DISTROFAIR proposes an efficient algorithm to generate class-level OOD images based on the occurrences and orientations of the class in an existing dataset. Moreover, the work proposed by Vernekar et al. [50] does not target fairness and only involves simple background and object pixel manipulations. In contrast, we employ insertion, deletion and rotation of arbitrary objects in an image, as such is crucial to detect the statistical disparity among class-level accuracy (i.e., fairness).

Testing of Image Recognition Systems: Researchers have leveraged traditional software testing approaches to test image recognition systems in recent years. For instance, MetaOD leverages an insertion operation to surface errors in object detection systems [52]. Studies have also demonstrated the benefits of applying image modification techniques to computer vision systems [55, 56, 62]. Similarly, we propose an automated testing system focused on object recognition. However, we seek to uncover fairness errors as opposed to the functional errors uncovered by previous works.

Fairness Analysis of Image Recognition Systems: Several works have studied and analysed bias in image recognition systems [63, 54, 25, 8, 11, 12, 53]. For instance, DeepFAIT [66] is a white-box fairness testing approach that requires access to the software at hand, which is not applicable for real-world commercial software systems such as our subject programs. Similar to our work, Guehairia et al. [18] also proposed an OOD detection approach for fairness analysis of facial recognition systems. The focus of this work is to enable fair dataset curation and data augmentation rather than test generation. In addition, DeepInspect [45] exposes class-level confusion and bias errors in image classifiers. Unlike DISTROFAIR, DeepInspect is a white-box approach that does not generate a new test suite for image classifiers. Instead, it analyzes image classifiers using *only* an existing dataset to determine class-level violations.

8 CONCLUSION

In this paper, we propose DISTROFAIR, a systematic approach to discover class-level fairness violations in image classification tasks. The crux of DISTROFAIR is OOD test generation, which is synergistically combined with semantic preserving mutation operations. We show that such an approach is highly effective in revealing class-level fairness violations (at least 21% of generated tests reveal fairness errors) and it significantly outperforms test generation within the distribution (2.3x more effective). Additionally, we show that our generated tests (OOD images) are 80% as realistic as real world images. Even though we apply our approach for image classification tasks, we believe that our approach

is generally applicable for validating multi-label object classification tasks in other domains. We hope that our open source OOD testing platform unfolds new opportunities for simple, yet effective class-level fairness testing for a variety of ML software systems.

9 Data Availability

We will make the experimental data and source code publicly available on acceptance. In line with that, we provide DistroFair and our experimental data for easy reproducibility, reuse and scrutiny:

https://github.com/sparkssss/DistroFair

References

[1] Samuel Ackerman, Orna Raz, and Marcel Zalmanovici. Freaai: Automated extraction of data slices to test machine learning models. In *Engineering Dependable and Secure Machine Learning Systems: Third International Workshop, EDSMLS 2020, New York City, NY, USA, February 7, 2020, Revised Selected Papers*, pages 67–83. Springer, 2020.

[2] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. Black box fairness testing of machine learning models. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 625–635, 2019.

[3] Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. Don't just assume; look and answer: Overcoming priors for visual question answering. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4971–4980. Computer Vision Foundation / IEEE Computer Society, 2018.

[4] Amazon. Amazon rekognition api, 2023. URL: https://aws.amazon.com/rekognition/.

[5] David Berend. Distribution awareness for ai system testing. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 96–98. IEEE, 2021.

[6] David Berend, Xiaofei Xie, Lei Ma, Lingjun Zhou, Yang Liu, Chi Xu, and Jianjun Zhao. Cats are not fish: Deep learning testing calls for out-of-distribution awareness. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering*, pages 1041–1052, 2020.

[7] Marcelo Bertalmio, Guillermo Sapiro, Vincent Caselles, and Coloma Ballester. Image inpainting. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 417–424, 2000.

[8] Martim Brandao. Age and gender bias in pedestrian detection algorithms. *arXiv preprint arXiv:1906.10490*, 2019.

[9] Zhenpeng Chen, Jie M Zhang, Max Hort, Federica Sarro, and Mark Harman. Fairness testing: A comprehensive survey and analysis of trends. *arXiv preprint arXiv:2207.10223*, 2022.

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] Terrance De Vries, Ishan Misra, Changhan Wang, and Laurens Van der Maaten. Does object recognition work for everyone? In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 52–59, 2019.

[12] Emily Denton, Ben Hutchinson, Margaret Mitchell, and Timnit Gebru. Detecting bias with generative counterfactual face attribute augmentation. 2019.

[13] Shaojing Fan, Tian-Tsong Ng, Jonathan S Herberg, Bryan L Koenig, Cheston Y-C Tan, and Rangding Wang. An automated estimator of image visual realism based on human cognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4201–4208, 2014.

[14] Raghudeep Gadde, Qianli Feng, and Aleix M Martinez. Detail me more: Improving gan's photo-realism of complex scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 13950–13959, 2021.

[15] Yarin Gal. *Uncertainty in Deep Learning*. PhD thesis, University of Cambridge, 2016.

[16] Google. Google cloud vision api, 2023. URL: https://cloud.google.com/vision.

[17] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the V in VQA matter: Elevating the role of image understanding in visual question answering. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 6325–6334. IEEE Computer Society, 2017.

[18] O Guehairia, F Dornaika, A Ouamane, and Abdelmalik Taleb-Ahmed. Facial age estimation using tensor based subspace learning and deep random forests. *Information Sciences*, 2022.

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[20] Max Hort, Zhenpeng Chen, Jie M Zhang, Federica Sarro, and Mark Harman. Bias mitigation for machine learning classifiers: A comprehensive survey. *arXiv preprint arXiv:2207.07068*, 2022.

[21] Wei Huang, Xingyu Zhao, Alec Banks, Victoria Cox, and Xiaowei Huang. Hierarchical distribution-aware testing of deep learning. *arXiv preprint arXiv:2205.08589*, 2022.

[22] Chiyu Max Jiang, Mahyar Najibi, Charles R Qi, Yin Zhou, and Dragomir Anguelov. Improving the intra-class long-tail in 3d detection via rare example mining. In *European Conference on Computer Vision*, pages 158–175. Springer, 2022.

[23] Xin Jin, Hao Lou, Heng Huang, Xinning Li, Xiaodong Li, Shuai Cui, Xiaokun Zhang, and Xiqiao Li. Pseudo-labeling and meta reweighting learning for image aesthetic quality assessment. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):25226–25235, 2022.

[24] Sergey Kastryulin, Jamil Zakirov, Denis Prokopenko, and Dmitry V. Dylov. Pytorch image quality: Metrics for image quality assessment, 2022. URL: https:

//arxiv.org/abs/2208.14818, doi:10.48550/ARXIV.2208.14818.

[25] Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9012–9020, 2019.

[26] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother, and Piotr Dollar. Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[27] Cui-jin Li, Zhong Qu, Sheng-ye Wang, and Ling Liu. A method of cross-layer fusion multi-object detection and recognition based on improved faster r-cnn model in complex traffic environment. *Pattern Recognition Letters*, 145:127–134, 2021.

[28] Chen-Hsuan Lin, Ersin Yumer, Oliver Wang, Eli Shechtman, and Simon Lucey. St-gan: Spatial transformer generative adversarial networks for image compositing. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9455–9464, 2018.

[29] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[30] Kede Ma, Wentao Liu, Kai Zhang, Zhengfang Duanmu, Zhou Wang, and Wangmeng Zuo. End-to-end blind image quality assessment using deep neural networks. *IEEE Transactions on Image Processing*, 27(3):1202–1213, 2017.

[31] J MacQueen. Classification and analysis of multivariate observations. In *5th Berkeley Symp. Math. Statist. Probability*, pages 281–297, 1967.

[32] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S Ecker, Matthias Bethge, and Wieland Brendel. Benchmarking robustness in object detection: Autonomous driving when winter is coming. *arXiv preprint arXiv:1907.07484*, 2019.

[33] Microsoft. Azure computer vision api, 2023. URL: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/.

[34] Daniel Perez Morales, Takashi Kitamura, and Shingo Takada. Coverage-guided fairness testing. In *International Conference on Intelligence Science*, pages 183–199. Springer, 2021.

[35] Amazon MTurk. Amazon mechanical turk, 2023. URL: https://www.mturk.com/.

[36] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. Deepxplore: Automated whitebox testing of deep learning systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 1–18. ACM, 2017.

[37] Adam Rose. Are face-detection cameras racist? http://content.time.com/time/business/article/0,8599,1954643,00.html, 2010.

[38] scikit learn:. scikit-learn: Machine learning in python, 2023. URL: https://scikit-learn.org/stable/.

[39] Ezekiel Soremekun, Mike Papadakis, Maxime Cordy, and Yves Le Traon. Software fairness: An analysis and survey. *arXiv preprint arXiv:2205.08809*, 2022.

[40] Ezekiel Soremekun, Sakshi Sunil Udeshi, and Sudipta Chattopadhyay. Astraea: Grammar-based fairness testing. *IEEE Transactions on Software Engineering*, 2022.

[41] Zeyu Sun, Jie M Zhang, Mark Harman, Mike Papadakis, and Lu Zhang. Automatic testing and improvement of machine translation. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pages 974–985, 2020.

[42] Roman Suvorov, Elizaveta Logacheva, Anton Mashikhin, Anastasia Remizova, Arsenii Ashukha, Aleksei Silvestrov, Naejin Kong, Harshith Goka, Kiwoong Park, and Victor Lempitsky. Resolution-robust large mask inpainting with fourier convolutions. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2149–2159, 2022.

[43] Damien Teney, Ehsan Abbasnejad, Kushal Kafle, Robik Shrestha, Christopher Kanan, and Anton van den Hengel. On the value of out-of-distribution testing: An example of goodhart's law. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020*, 2020.

[44] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. Deeptest: automated testing of deep-neural-network-driven autonomous cars. In Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman, editors, *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, pages 303–314. ACM, 2018.

[45] Yuchi Tian, Ziyuan Zhong, Vicente Ordonez, Gail E. Kaiser, and Baishakhi Ray. Testing DNN image classifiers for confusion & bias errors. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 1122–1134. ACM, 2020.

[46] Saeid Tizpaz-Niari, Ashish Kumar, Gang Tan, and Ashutosh Trivedi. Fairness-aware configuration of machine learning libraries. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE, 2022.

[47] Michael Treml, José Arjona-Medina, Thomas Unterthiner, Rupesh Durgesh, Felix Friedmann, Peter Schuberth, Andreas Mayr, Martin Heusel, Markus Hofmarcher, Michael Widrich, et al. Speeding up semantic segmentation for autonomous driving. 2016.

[48] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. Automated directed fairness testing. In Marianne Huchard, Christian Kästner, and Gordon Fraser, editors, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ASE 2018, Montpellier, France, September 3-7, 2018*, pages 98–108. ACM, 2018.

[49] Sahil Verma and Julia Rubin. Fairness definitions explained. In *2018 ieee/acm international workshop on software fairness (fairware)*, pages 1–7. IEEE, 2018.

[50] Sachin Vernekar, Ashish Gaurav, Vahdat Abdelzad, Taylor Denouden, Rick Salay, and Krzysztof Czarnecki. Out-of-distribution detection in classifiers via generation. 2019.

[51] Jianyi Wang, Kelvin CK Chan, and Chen Change Loy. Exploring clip for assessing the look and feel of images. In *Proceedings of the AAAI Conference on Artificial*

Intelligence, volume 37, pages 2555–2563, 2023.

[52] Shuai Wang and Zhendong Su. Metamorphic object insertion for testing object detection systems. In *35th IEEE/ACM International Conference on Automated Software Engineering, ASE 2020, Melbourne, Australia, September 21-25, 2020*, pages 1053–1065. IEEE, 2020.

[53] Tianlu Wang, Jieyu Zhao, Mark Yatskar, Kai-Wei Chang, and Vicente Ordonez. Balanced datasets are not enough: Estimating and mitigating gender bias in deep image representations. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5310–5319, 2019.

[54] Zeyu Wang, Klint Qinami, Ioannis Christos Karakozis, Kyle Genova, Prem Nair, Kenji Hata, and Olga Russakovsky. Towards fairness in visual recognition: Effective strategies for bias mitigation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8919–8928, 2020.

[55] Trey Woodlief, Sebastian G. Elbaum, and Kevin Sullivan. Semantic image fuzzing of AI perception systems. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1958–1969. ACM, 2022.

[56] Franz Wotawa, Lorenz Klampfl, and Ledio Jahaj. A framework for the automation of testing computer vision systems. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, pages 121–124. IEEE, 2021.

[57] Jian Wu, Victor S Sheng, Jing Zhang, Hua Li, Tetiana Dadakova, Christine Leon Swisher, Zhiming Cui, and Pengpeng Zhao. Multi-label active learning algorithms for image classification: Overview and future promise. *ACM Computing Surveys (CSUR)*, 53(2):1–35, 2020.

[58] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[59] Yuwen Xiong, Renjie Liao, Hengshuang Zhao, Rui Hu, Min Bai, Ersin Yumer, and Raquel Urtasun. Upsnet: A unified panoptic segmentation network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8818–8826, 2019.

[60] Zhou Yang, Muhammad Hilmi Asyrofi, and David Lo. Biasrv: Uncovering biased sentiment predictions at runtime. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 1540–1544, 2021.

[61] Nanyang Ye, Kaican Li, Haoyue Bai, Runpeng Yu, Lanqing Hong, Fengwei Zhou, Zhenguo Li, and Jun Zhu. Ood-bench: Quantifying and understanding two dimensions of out-of-distribution generalization. In *CVPR*, pages 7937–7948. IEEE, 2022.

[62] Boxi Yu, Zhiqing Zhong, Xinran Qin, Jiayi Yao, Yuancheng Wang, and Pinjia He. Automated testing of image captioning systems. In *ISSTA*, pages 467–479. ACM, 2022.

[63] Jun Yu, Xinlong Hao, Haonian Xie, and Ye Yu. Fair face recognition using data balancing, enhancement and fusion. In *European Conference on Computer Vision*, pages 492–505. Springer, 2020.

[64] Lingfeng Zhang, Yueling Zhang, and Min Zhang. Efficient white-box fairness testing through gradient search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 103–114, 2021.

[65] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. White-box fairness testing through adversarial sampling. In *ICSE '20: 42nd International Conference on Software Engineering, Seoul, South Korea, 27 June - 19 July, 2020*, pages 949–960. ACM, 2020.

[66] Peixin Zhang, Jingyi Wang, Jun Sun, and Xinyu Wang. Fairness testing of deep image classification with adequacy metrics. *arXiv preprint arXiv:2111.08856*, 2021.

[67] Peixin Zhang, Jingyi Wang, Jun Sun, Xinyu Wang, Guoliang Dong, Xingen Wang, Ting Dai, and Jin Song Dong. Automatic fairness testing of neural classifiers through adversarial sampling. *IEEE Transactions on Software Engineering*, 2021.

[68] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. Men also like shopping: Reducing gender bias amplification using corpus-level constraints. In Martha Palmer, Rebecca Hwa, and Sebastian Riedel, editors, *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, pages 2979–2989. Association for Computational Linguistics, 2017.

[69] Lingjun Zhou, Bing Yu, David Berend, Xiaofei Xie, Xiaohong Li, Jianjun Zhao, and Xusheng Liu. An empirical study on robustness of dnns with out-of-distribution awareness. In *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, pages 266–275. IEEE, 2020.