Distribution-aware Fairness Test Generation

Sai Sathiesh Rajan
Singapore University of Technology and Design
Singapore
sai_rajan@mymail.sutd.edu.sg

Ezekiel Soremekun
Royal Holloway, University of London
United Kingdom (UK)
ezekiel.soremekun@rhul.ac.uk

Yves Le Traon
SnT, University of Luxembourg
Luxembourg
yves.letraon@uni.lu

Sudipta Chattopadhyay
Singapore University of Technology and Design
Singapore
sudipta_chattopadhyay@sutd.edu.sg

ABSTRACT

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.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

1 INTRODUCTION

Image classification has several critical applications in autonomous driving, robotics and healthcare, among others. Image classification may involve several tasks [45]. 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) [45]. 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 [27] and women were usually associated with activities such as cooking, shopping etc [56]. 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.

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 [40]. 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

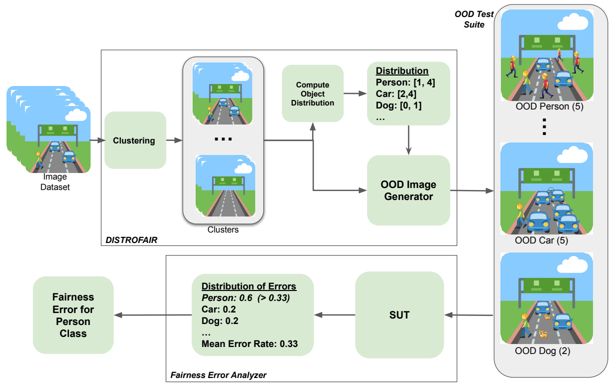


Figure 1: An illustration of our DistroFair approach

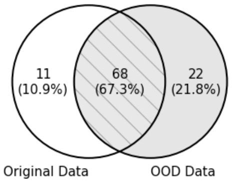


Figure 2: Classes with higher than the mean error rate in original sample vs. OOD sample generated from the original

number of objects detected for each class and their orientation. Subsequently, OOD images are generated by leveraging 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 [31, 39, 53] and functional testing [26, 35] 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 [36]. In particular, while the aforementioned work presented new metrics for confusion and bias detection for a class [36], 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 [44] or detecting functional errors without evaluating semantic validity [41] [50]. 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 and propose a novel OOD test generation approach (DistroFair) to discover such errors (section 3).
(2) We propose and implement three metamorphic OOD transformation 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 erroneous 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).

2 OVERVIEW

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

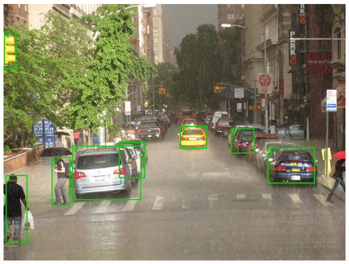
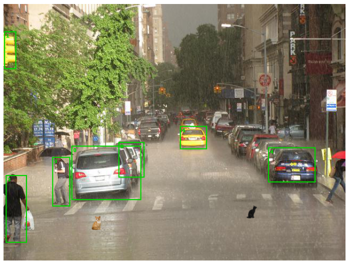
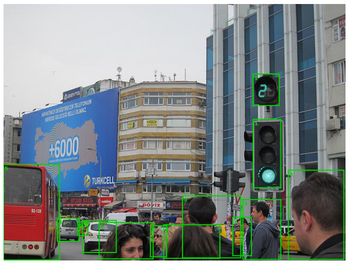
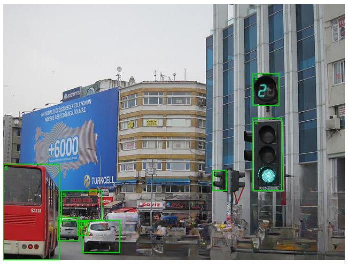
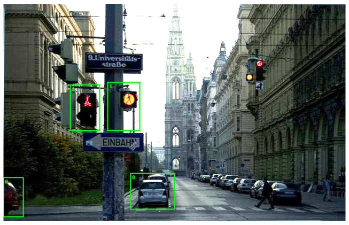
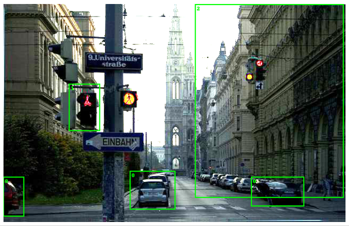
Class-level fairness: In this work, we investigate and discover class-level fairness errors in computer vision (CV) systems. Class level fairness is directly related to the concept of group fairness. Fundamentally, group fairness is concerned with ensuring that different groups exhibit similar statistical properties under similar stimuli [40]. 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 $\mathbb{C}$:

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

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 [34, 49]. In particular, significant manual effort has been put forward to create OOD benchmark [13]. Moreover, we have observed a line of research that focused on improving

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)

the accuracy of ML models on OOD benchmark [2]. 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:

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

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 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.

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 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 [46] 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 [22] 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* under test. In the beginning, DistroFair learns a distribution for the set of images Img_{List} under test, $Dist_{List}$. 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)

Algorithm 1 OOD Image Generation

```
procedure OOD_Image_Generation($Img_{List}, OP_{List}, LBL_{List}$)
    $OOD\_Set \leftarrow \varnothing$
    $MUT_{List} \leftarrow \{x, y\} : x \in (OP_{List}), y \in (LBL_{List})$
    ▷ $\mathcal{F}$ computes the distribution of the set of images in $Img_{List}$
    ▷ $SUT$ is the ML model under test
    $Dist_{List} \leftarrow \mathcal{F}(Img_{List}, SUT)$
    for $M \in MUT_{List}$ do
        for $Img \in Img_{List}$ do
            $Dist_{Img} \leftarrow F_{Img}(Img, SUT)$
            $Mut_{Num} \leftarrow MutGen(Dist_{Img}, Dist_{List})$
            $Gen_{Img} \leftarrow ImageGen(Img, M, Mut_{Num})$
            $OOD\_Set \cup= \{(Gen_{Img}, Img, M)\}$
        end for
    end for
    return $OOD\_Set$
end procedure
```

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* [44]. The goal is to preserve the perception of the original image, except for the perception (e.g., number or orientation) of 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*:** We leverage panoptic segmentation [19] [46] 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.

We first aim to determine the size of the object to be inserted based on the size of existing objects in the image. To this end, we use information on relative size between different classes if they are placed together in an image. For instance, consider the insertion of *cat* class objects in Table 1. Using the information on relative sizing, we check what would be the size of a *cat* if it was placed beside one of the other existing objects, e.g., a *car* or *Person* in the image. Then, using the different sizes of the *cat* at these locations, we extrapolate the size of a *cat* at the final chosen location for insertion. It is worthwhile to mention that we find the relative size between different classes via some initial experiments. We believe this is acceptable as it is a one time effort and determining the relative size information between our mutable classes only take a few minutes. Additionally, with this simple effort, we allow larger flexibility in choosing the location for insertion.

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
```

After determining the size of an inserted object, we need to determine the location of insertion. To this end, we implement an additional check to ensure that the object is placed in an appropriate location. For instance, we ensure that a car 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.

***Deletion*:** During deletion, we delete 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. We leverage the panoptic segmentation map to identify the objects before applying a mask. We then use inpainting [6] [33] to delete the masked objects.

***Rotation*:** For rotation, we first identify an object belonging to the target class, taking care to ensure that said object is not obstructed by another object. We then extract the image level information for the object being rotated in that location before deleting the object through inpainting [6] [33]. Finally, we rotate the extracted image (i.e., the target object) and insert 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.

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:

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

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:          $GT\_Pred \leftarrow SUT(TUP.Img)$
5:      else if $Case\_Type$ = "$GT$" then
6:          ▷ $GT$ is the union of results from all SUTs and the dataset ground truth
7:          $GT\_Pred \leftarrow GT(TUP.Img)$
8:      end if
9:      $Org\_Pred \leftarrow SUT(TUP.Img); Gen\_Pred \leftarrow SUT(TUP.Gen_{Img})$
10:     Let $TUP.M = (-, LBL)$
11:     $GT\_Pred[LBL] = Org\_Pred[LBL] = Gen\_Pred[LBL] = Diff\_Pred[LBL] \leftarrow \emptyset$;
12:     for $(-, Obj_L) \in GT\_Pred$ do
13:         $Org\_Err \leftarrow |Org\_Pred[Obj_L] - GT\_Pred[Obj_L]|$
14:         $New\_Err \leftarrow Gen\_Pred[Obj_L] - GT\_Pred[Obj_L]$
15:         if $Case\_Type$ = "$GT$" then
16:             $Diff\_Pred[Obj_L] = |New\_Err| - Org\_Err$
17:         else if $Case\_Type$ = "$SUT$" then
18:             if $Err\_Type$ = "$INC$" then
19:                 if $Change\_Error > 0$ then
20:                     $Diff\_Pred[Obj_L] = New\_Err$
21:                 end if
22:             else if $Err\_Type$ = "$EXC$" then
23:                 if $Change\_Error < 0$ then
24:                     $Diff\_Pred[Obj_L] = -1 \cdot New\_Err$
25:                 end if
26:             end if
27:         end if
28:     end for
29:     $Oracle\_Ret = \{GT\_Pred, Diff\_Pred\}$
30:     return $Oracle\_Ret$
31: end procedure
```

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 10-Line 11). 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 12-Line 14). This is used to compute the errors (Line 15-Line 24). 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, we highlight a class c facing 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.

4 EVALUATION SETUP

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

Table 2: Details of Experimental Datasets

Dataset	Description	#Images	#Classes	First Published
MS-COCO [21]	Microsoft Common Objects images	300	183	2014
CityScapes [10]	TU Darmstadt's Urban Street Scenes images	315	30	2015

Table 3: Details of Subject Programs

Subject Programs	Description (No. of labels supported)	#Classes (Our Experiments)	Availability Date
GCP [9]	9000	89	2017
AWS [29]	2000+	76	2016
MS [3]	10000	58	2016

- **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?

Datasets and Subject Programs: We selected MS-COCO and CityScapes (see Table 2) 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) and the research community [15] [47] [20] [38] [23]. Additionally, our chosen evaluation subjects (see Table 3) 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 [46] to detect objects, whereas K-Means algorithm from SciKitLearn [28] 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 [22].
Image Generation: For all mutations, DistroFair attempts to generate one image for each selected class for each given image. All

Table 4: 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)

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.

Mutated Objects: 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.

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. In addition, we also used the Detectron2 [46] and LaMa [33] to aid in the image generation. For evaluation, we use APIs (with default settings) for each of our subject programs (Table 3). 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 2). We also ensure that most of the selected images (24 out of 30) induce errors for at least one subject program (*see* Table 4). 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) [25]. 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 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).

Table 5: Effectiveness of DistroFair using GT oracle (maximum fairness error rate and violation rate for each (sub)category are in bold). Detailed results w.r.t. each mutation operator is provided in the supplement.

				#Class Violations	Violative Rate	Error Inputs		Fairness Error Rate
Subject	**Datasets**	**Mutation Ops**	**#Classes**	**Err Classes**	**Violative Rate**	**Inputs**	**#Gen Inputs**	**Error Rate**
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

5 EVALUATION RESULTS

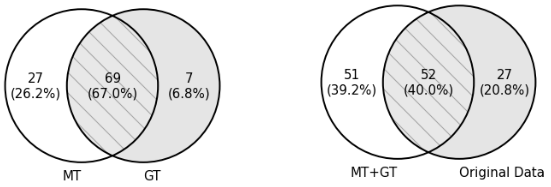
RQ1 Effectiveness: We evaluate the effectiveness of DistroFair using both the GT and MT oracle (Table 5 and Table 6), as discussed in section 2. The choice of GT test oracle is useful when practitioners have access to ground truth information on the dataset, whereas the MT oracle is useful when practitioners lack access to detailed information on the dataset or other subject programs.

Using Ground Truth Oracle: Table 5 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 5). 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 6 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 6). 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.



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

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

Figure 3: Illustration of DistroFair effectiveness

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 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.

Class-level fairness analysis with DistroFair is more effective than using only the original dataset. DistroFair exposes (30%) more

Table 6: Effectiveness of DistroFair using MT oracle (maximum fairness error rate and violation rate for each (sub)category are in bold). Ex.: Exclusion, Inc.: Inclusion. Detailed results w.r.t. each mutation operator is provided in supplement.

				#ClassViolations		Violative Rate		#Error-inducing inputs			Fairness Error Rate	
Subject	**Datasets**	**Mutation Ops**	**#Class**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**Ex.**	**Inc.**	**#gen-Inputs**	**Ex.**	**Inc.**
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

Table 7: 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. Detailed results w.r.t. each subject is provided in supplement.

				#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

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 operation for this comparison. Our evaluation (see Table 7) results show that *OOD style mutation outperforms the ID-based mutation approach in revealing class-level group fairness violations.* Specifically, DistroFair reveals up to 74% more class-level fairness violations than the baseline for exclusion errors (*see* Table 7). 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.

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

RQ3 Efficiency: Table 8 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 6). Further inspection shows that these results hold across 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.

DistroFair is fast in generating test suites for class-level fairness. It takes ≈ 7.7 sec on average to generate a test.

RQ4 Semantic Validity: We have conducted a *user study* to evaluate the *semantic validity* of the images generated by DistroFair. We conducted the study with 105 participants and 60 images (see subsection 4.2). Our user study results show that *images generated by our test generator (DistroFair) are semantically valid, when*

Table 8: 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)

compared to real-world images. Table 9 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 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.

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.

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 [8, 16, 30].
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 [30] (*see* Table 2). In addition, our subjects are off-the-shelf, mature, commercial image classifiers provided by software companies such as Google, Amazon and Microsoft (*see* Table 3).

7 RELATED WORK

Fairness Test Generation: Recent surveys [8, 16, 30] 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 [53, 55] and EIDIG [52]). 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 [31, 32, 39, 48]. Grey-box fairness testing approaches [37] employ both input space exploration and model analysis for test generation. Besides, some methods employ program analysis techniques, e.g., symbolic execution [1] and combinatorial testing [24] 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.
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. [5] found that data distribution awareness in both testing and enhancement phases outperforms distribution unaware retraining. Likewise, Zhou et al. [57] 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. [4] 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. [17] 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.
Fairness Analysis of Image Recognition Systems: Several works have studied and analysed bias in image recognition systems [7, 11, 12, 18, 42, 43, 51]. For instance, DeepFAIT [54] 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. [14] 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 [36] 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

Table 9: 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)

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.

REFERENCES

[1] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. 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*. 625–635.

[2] Aishwarya Agrawal, Dhruv Batra, Devi Parikh, and Aniruddha Kembhavi. 2018. 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*. Computer Vision Foundation / IEEE Computer Society, 4971–4980.

[3] Microsoft Azure. [n. d.]. Azure Computer Vision API. https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/

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

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

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

[7] Martim Brandao. 2019. Age and gender bias in pedestrian detection algorithms. *arXiv preprint arXiv:1906.10490* (2019).

[8] Zhenpeng Chen, Jie M Zhang, Max Hort, Federica Sarro, and Mark Harman. 2022. Fairness Testing: A Comprehensive Survey and Analysis of Trends. *arXiv preprint arXiv:2207.10223* (2022).

[9] Google Cloud. [n. d.]. Google Cloud Vision API. https://cloud.google.com/vision

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

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

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

[13] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. 2017. 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*. IEEE Computer Society, 6325–6334.

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

[15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[16] Max Hort, Zhenpeng Chen, Jie M Zhang, Federica Sarro, and Mark Harman. 2022. Bias Mitigation for Machine Learning Classifiers: A Comprehensive Survey. *arXiv preprint arXiv:2207.07068* (2022).

[17] Wei Huang, Xingyu Zhao, Alec Banks, Victoria Cox, and Xiaowei Huang. 2022. Hierarchical Distribution-Aware Testing of Deep Learning. *arXiv preprint arXiv:2205.08589* (2022).

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

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

[20] Cui-jin Li, Zhong Qu, Sheng-ye Wang, and Ling Liu. 2021. 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 (2021), 127–134.

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

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

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

[24] Daniel Perez Morales, Takashi Kitamura, and Shingo Takada. 2021. Coverage-Guided Fairness Testing. In *International Conference on Intelligence Science*. Springer, 183–199.

[25] Amazon MTurk. [n. d.]. Amazon Mechanical Turk. https://www.mturk.com/

[26] Kexin Pei, Yinzhi Cao, Junfeng Yang, and Suman Jana. 2017. DeepXplore: Automated Whitebox Testing of Deep Learning Systems. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*. ACM, 1–18.

[27] Adam Rose. 2010. Are Face-Detection Cameras Racist? http://content.time.com/time/business/article/0,8599,1954643,00.html.

[28] scikit learn:. [n. d.]. scikit-learn: Machine Learning in Python. https://scikit-learn.org/stable/

[29] Amazon Web Services. [n. d.]. Amazon Rekognition API. https://aws.amazon.com/rekognition/

[30] Ezekiel Soremekun, Mike Papadakis, Maxime Cordy, and Yves Le Traon. 2022. Software Fairness: An Analysis and Survey. *arXiv preprint arXiv:2205.08809* (2022).

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

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

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

[34] Damien Teney, Ehsan Abbasnejad, Kushal Kafle, Robik Shrestha, Christopher Kanan, and Anton van den Hengel. 2020. 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*.

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

[36] Yuchi Tian, Ziyuan Zhong, Vicente Ordonez, Gail E. Kaiser, and Baishakhi Ray. 2020. 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*. ACM, 1122–1134.

[37] Saeid Tizpaz-Niari, Ashish Kumar, Gang Tan, and Ashutosh Trivedi. 2022. Fairness-aware Configuration of Machine Learning Libraries. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. IEEE.

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

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

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

[41] Shuai Wang and Zhendong Su. 2020. 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*. IEEE, 1053–1065.

[42] Tianlu Wang, Jieyu Zhao, Mark Yatskar, Kai-Wei Chang, and Vicente Ordonez. 2019. 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*. 5310–5319.

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

[44] Trey Woodlief, Sebastian G. Elbaum, and Kevin Sullivan. 2022. 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*. ACM, 1958–1969.

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

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

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

[48] Zhou Yang, Muhammad Hilmi Asyrofi, and David Lo. 2021. 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*. 1540–1544.

[49] Nanyang Ye, Kaican Li, Haoyue Bai, Runpeng Yu, Lanqing Hong, Fengwei Zhou, Zhenguo Li, and Jun Zhu. 2022. OoD-Bench: Quantifying and Understanding Two Dimensions of Out-of-Distribution Generalization. In *CVPR*. IEEE, 7937–7948.

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

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

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

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

[54] Peixin Zhang, Jingyi Wang, Jun Sun, and Xinyu Wang. 2021. Fairness Testing of Deep Image Classification with Adequacy Metrics. *arXiv preprint arXiv:2111.08856* (2021).

[55] Peixin Zhang, Jingyi Wang, Jun Sun, Xinyu Wang, Guoliang Dong, Xingen Wang, Ting Dai, and Jin Song Dong. 2021. Automatic Fairness Testing of Neural Classifiers through Adversarial Sampling. *IEEE Transactions on Software Engineering* (2021).

[56] Jieyu Zhao, Tianlu Wang, Mark Yatskar, Vicente Ordonez, and Kai-Wei Chang. 2017. Men Also Like Shopping: Reducing Gender Bias Amplification using Corpus-level Constraints. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, EMNLP 2017, Copenhagen, Denmark, September 9-11, 2017*, Martha Palmer, Rebecca Hwa, and Sebastian Riedel (Eds.). Association for Computational Linguistics, 2979–2989.

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