Metamorphic Object Insertion for Testing Object Detection Systems

Shuai Wang*
The Hong Kong University of Science and Technology
shuaiw@cse.ust.hk

Zhendong Su
ETH Zurich
zhendong.su@inf.ethz.ch

Abstract

Recent advances in deep neural networks (DNNs) have led to object detectors (ODs) that can rapidly process pictures or videos, and recognize the objects that they contain. Despite the promising progress by industrial manufacturers such as Amazon and Google in commercializing deep learning-based ODs as a standard computer vision service, ODs — similar to traditional software — may still produce incorrect results. These errors, in turn, can lead to severe negative outcomes for the users. For instance, an autonomous driving system that fails to detect pedestrians can cause accidents or even fatalities. However, despite their importance, principled, systematic methods for testing ODs do not yet exist.

To fill this critical gap, we introduce the design and realization of MetaOD, a metamorphic testing system specifically designed for ODs to effectively uncover erroneous detection results. To this end, we (1) synthesize natural-looking images by inserting extra object instances into background images, and (2) design metamorphic conditions asserting the equivalence of OD results between the original and synthetic images after excluding the prediction results on the inserted objects. MetaOD is designed as a streamlined workflow that performs object extraction, selection, and insertion. We develop a set of practical techniques to realize an effective workflow, and generate diverse, natural-looking images for testing. Evaluated on four commercial OD services and four pretrained models provided by the TensorFlow API, MetaOD found tens of thousands of detection failures. To further demonstrate the practical usage of MetaOD, we use the synthetic images that cause erroneous detection results to retrain the model. Our results show that the model performance is significantly increased, from an mAP score of 9.3 to an mAP score of 10.5.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; • **Security and privacy** → *Software and application security*; • **Computing methodologies** → Neural networks;

Keywords

testing, computer vision, object detection, deep neural networks

ACM Reference Format:
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 '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3324884.3416584

1 Introduction

Deep learning-based object detectors (ODs) identify objects in a given image using convolutional neural networks. Currently, several major industrial manufacturers, including Google, Amazon, Microsoft, and Lockheed Martin, are building and improving ODs to serve as the basis for various computer vision tasks. These models are widely-used in real-world applications, such as optical character recognition (OCR), robotics, machine inspection, and pedestrian detection in autonomous cars. They are also used as an initial step in surveillance and medical image analysis applications, which often require highly precise and reliable detection results.

Despite this spectacular progress, however, deep learning-based ODs — similar to traditional software — can yield erroneous prediction results that are potentially disastrous. In particular, given the widespread adoption of ODs in critical applications in the security, medical, and autonomous driving fields, incorrect or unexpected edge-case behaviors have caused severe threats to public safety or financial loss [11, 43, 73]. For instance, in one infamous case in 2016, Tesla's autopilot mode caused a fatal crash when the autonomous driving system failed to recognize a white truck against a bright sky [15]. More recently, an Uber autonomous car killed a pedestrian crossing the road, which is believed to have been due to its failure in recognizing a pedestrian in dark clothing [32].

In recent years, a number of techniques have been designed to test deep learning systems, including both convolutional neural networks (CNN) and recurrent neural networks (RNN) models [20, 57]. The techniques have also been applied to test domain-specific applications such as autodriving systems [11, 73, 84] and to test the underlying infrastructure of deep learning libraries [21, 39, 61]. However, the principles specific for testing ODs have not been investigated systematically by existing research, which, thus, unlikely results in comprehensive, systematic testing of ODs.

This paper tackles this important problem by introducing a metamorphic testing [16, 17] framework, MetaOD, to specifically and effectively expose erroneous predictions of ODs. Given a real image as the "background", MetaOD inserts an object instance into the background, generates a synthetic image, and then employs a metamorphic condition to check the consistency of OD results between the synthetic image and the corresponding background. To

*Most of the work was done when Shuai Wang was working at ETH Zurich.

ASE '20, September 21–25, 2020, Virtual Event, Australia

ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3416584

effectively generate *diverse* and *natural-looking* images that trigger *practical* prediction errors, MetaOD is designed as a three-step approach, performing object extraction, object refinement/selection, and object insertion. The object extraction module extracts object instance images from a large set of pictures using advanced instance segmentation techniques [14], thus aggregating many object sets distinguished by category. Then, given a background image, the object refinement/selection module implements a set of lightweight, albeit effective, criteria for selecting certain objects from object sets that are closely related to the background. To determine insertion locations, the object insertion module uses domain-specific criteria and techniques inspired by delta debugging [83] to find locations that presumably trigger prediction errors, while retaining realism and diversity of the synthetic images.

The proposed workflow is effective and shows promising results when evaluated on four commercial ODs provided by Amazon, Google, IBM, and Microsoft [1, 2, 4, 6] and four pretrained OD models provided by the TensorFlow API [3]. Our testing revealed tens of thousands of erroneous OD outputs from these popular (commercial) services. In addition, we retrained an OD model using synthetic images that cause this model to output erroneous outputs. The evaluation results show that the model performance improved substantially after retraining. In summary, this work makes the following main contributions:

- We introduce a novel metamorphic testing approach for ODs, vital components in various computer vision applications (e.g., self-driving cars). Our technique treats ODs as "black-boxes". Thus, it is highly generalizable for testing real-world ODs, such as remote services on the cloud.
- To generate diverse and natural-looking sets of images as the test inputs, we design and realize MetaOD, a streamlined workflow that performs object extraction, object refinement/selection, and object insertion to synthesize input images in an efficient and adaptive manner.
- Our approach tests ODs in a realistic setting and delineates the capabilities of state-of-the-art commercial ODs. From a total of 292,206 input images, MetaOD found 38,345 erroneous detection results in eight popular (commercial) ODs. By leveraging synthetic images that trigger erroneous OD outputs for retraining, we show that the performance of OD models can be substantially improved.

We have released MetaOD on Github [9]. All of our erroneous detection results can be found at [8].

2 Background of Object Detection Techniques

Object detection is conventionally addressed using handcrafted features and selective region searches [60, 77, 79]. The input images are dissected into small regions (each region is called a "region proposal" and is likely to contain an object) via heuristics [77]. Then, features are extracted from each region proposal for object classification. To date, two major lines of research (popular models proposed in both line of research are tested in this work; see Sec. 6) exist that have drastically improved OD techniques with deep learning, both of which are briefly introduced below.

Two-Stage Region-Based ODs. Motivated by the primary success in applying DNNs for image classification [41], RCNN [30] was

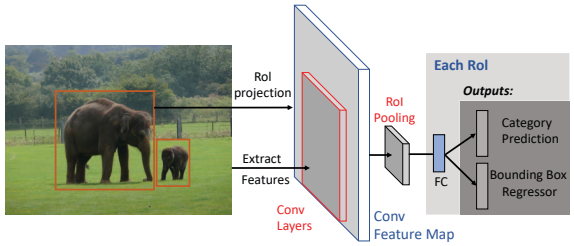

Figure 1: Typical two-stage OD architectures illustrated by the simplified Fast RCNN workflow.

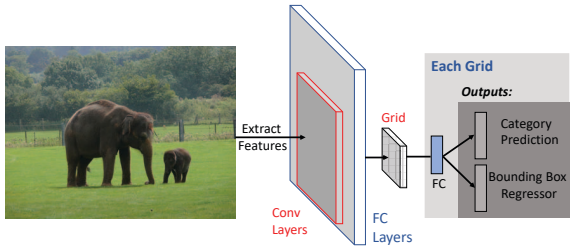

Figure 2: Typical single-stage OD architectures illustrated by the simplified YOLO workflow.

among the first to apply CNN for building ODs. The proposed technique forms a two-stage pipeline in which each region proposal extracted from the input image is an input to CNN for feature extraction. Then, the extracted features are forwarded to an SVM classifier and a bounding box regressor to determine the object category and bounding box offsets, respectively.

Since then, OD research has focused on rapidly evolving the RCNN architecture [18, 34, 67] and removing explicit dependence on region proposals to improve speed. Fast-RCNN [29] introduced a modern *end-to-end* prediction pipeline. As shown in Fig. 1, instead of region proposals, the entire image is forwarded to the CNN to generate a convolutional feature map and region proposals are extracted from the feature map (*first stage*). A Region of Interest (RoI) pooling layer is placed before the fully connected layer (FC) to reshape each proposal into a fixed size, and FC layer's outputs are fed to softmax and bbox regressor layers for object classification and for determining bounding box offsets, respectively (*second stage*).

Single-Stage ODs. Two-stage ODs use regions, explicitly or implicitly, for object localization. Another line of research aims to propose a cost-effective solution without region proposals by designing a single-stage feed-forward CNN network in a monolithic setting. Such networks are usually less computationally intensive by trading precision for speed, and are usually more suitable for real-time tasks or for use in mobile devices.

The YOLO (You Only Look Once) [64–66] and SSD (Single Shot Detector) [47] models are de facto ODs that feature single-stage architectures. Fig. 2 depicts the YOLO workflow, in which input images are first divided into an $S \times S$ grid; then, a fixed number of bounding boxes are predicted within each grid. For each bounding box, the network outputs a class probability and the bounding

box offsets. A bounding box is deemed to contain objects when its class probability exceeds a threshold value. The entire pipeline is typically orders of magnitude faster than region-based techniques. Indeed, the object extraction module of MetaOD (see Sec. 4.1) is built on top of YOLACT [14], a real-time instance segmentation model that was inspired by YOLO.

In general, given a well-trained model, region proposal-based object detection techniques can usually achieve high performances; in contrast, a "unified" pipelines such as YOLO and SSD achieve high detection efficiency by trading precision for speed. While precision may often be the highest priority, unified models have also been commonly adopted, particularly for mobile/wearable devices and analyzing videos, where speed is critical.

3 Approach Overview

Metamorphic testing (MT) has been widely used to automatically generate tests to detect conventional software faults [16, 17] and DNN prediction errors [22, 73, 84]. The strength of MT lies in its capability to alleviate the test oracle problem via metamorphic relations (MRs). Each MR depicts necessary properties of the target software in terms of inputs and their expected outputs. Even if the correctness of actual outputs are difficult to determine, it is possible to construct and check proper MRs among the expected outputs of the given inputs to detect software detects. In this research, we apply metamorphic testing to OD. To provide an approach overview, we start by formulating relevant notations.

By feeding a test image i to an OD d, the prediction output is denoted as $d[[i]]$, which consists of N three-element tuples (b_k, l_k, c_k), where N denotes the number of objects recognized in i, b_k the location of the kth object recognized in i, l_k the category label, and c_k the confidence score of the prediction. Then, given a set of object instance images $\mathbb{O}$ extracted from a large number of real images (see Fig. 4.1), and a set $\mathbb{C}$ where each $\mathcal{C} \in \mathbb{C}$ is a 2-D coordinate (x, y) in i, a synthetic image i' can be represented as:

$$i' = \mathcal{C}(o, i), o \in \mathbb{O} \text{ and } \mathcal{C} \in \mathbb{C}$$

where o is placed such that its cendroid is at the 2-D coordinate specified by $\mathcal{C}$. Note that in this research, we do **not** apply any transformation rules (rotation, blurring, etc.) on the inserted objects to preserve realism at our best effort, and $\mathcal{C} \in \mathbb{C}$ is deliberately constructed such that the inserted object o does **not** overlap with preexisting objects in the "background" image i. Therefore, the MR adopted in this research can be formalized as follows:

$$\forall \mathcal{C} \in \mathbb{C}\ \forall o \in \mathbb{O}.\ \mathcal{E}(d[[i]], d[[i']] \setminus \{(b_o, l_o, c_o)\})$$

where $i' = \mathcal{C}(o, i)$. Here, we **exclude** the prediction result on o (i.e., tuple (b_o, l_o, c_o)) from $d[[i']]$, and $\mathcal{E}$ is a criterion asserting the equality of OD results (see Sec. 3.1). The given MR is defined such that no matter how the image i' is synthesized by inserting an additional object o on i, the OD results are expected to be consistent with those in the original image. Consequently, erroneous predictions can be revealed by checking the failure of this MR.

While the given MR holds for any synthetic image $i' = \mathcal{C}(o, i)$, one practical challenge is that not all the synthetic images represent real-world scenarios. Indeed, there exists research in the CV community where unrealistic images are synthesized to train image analysis models, for example, by placing a car on the table [74]. While the synthetic "unrealistic" images may fulfill the requirement of "model training", we aim to also augment the realism of the synthetic images such that flagged erroneous behaviors unveil practical defects that can cause confusion during daily usage of ODs. Additionally, as we will explain in Sec. 4.3, randomly deciding a position for insertion without considering preexisting objects' positions in the background would undermine the effectiveness of the proposed technique.

Therefore, in this research, we gather $\mathbb{O}' \subset \mathbb{O}$ such that $\mathbb{O}'$ contains object instance images that are closely related to the background image i (see Sec. 4.2). We also form favorable insertion locations $\mathbb{C}' \subset \mathbb{C}$ likely to trigger prediction errors by leveraging empirical evidence and strategies inspired by delta debugging (see Sec. 4.3). Hence, the MR is modified as follows:

$$\forall \mathcal{C} \in \mathbb{C}'\ \forall o \in \mathbb{O}'.\ \mathcal{E}(d[[i]], d[[i']] \setminus \{(b_o, l_o, c_o)\})$$

In all, an OD system "failure" indicates a wrong prediction over synthesized image i' w.r.t. its reference image i (i.e., the ground truth), consistent with existing SE research on testing deep image classifiers (e.g., [22, 84]).

3.1 Equality Criteria

Asserting the *equality* of OD outputs (i.e., N three-element tuples (b_k, l_k, c_k)) is too strict because bounding boxes of certain objects could slightly drift within each round of prediction. The CV community instead uses a standard metric, Average Precision (AP) [24], to compensate small localization drifting when measuring ODs. The AP score is computed by taking both "precision" and "recall" values into account, as we will explain soon.

To compute AP, Intersection over Union (IoU) is used to measure each object detection boundary w.r.t. the ground truth. As shown in Fig. 4, IoU measures the overlap between two bounding boxes with the same prediction label (i.e., "elephant"), and denotes how much the predicted boundary overlaps with the ground truth. In case IoU is greater than a threshold ϵ (e.g., 0.5), the prediction is a *true positive*. The precision and recall scores are then computed by taking all the prediction results into account, and the AP score can be derived by computing the area under the precision-recall curve [42, 69]. For an image with objects of different categories, mean AP (mAP) is further computed by averaging all AP scores.

Our equality criteria $\mathcal{E}$ is derived from mAP. In our setting, the prediction results of the "background" image i entails the ground truth, and are compared with detection results of the synthetic image $i' = \mathcal{C}(o, i)$. Since o does not overlap with existing objects on i, and, therefore, does not interfere with relevant predictions, the mAP score is expected to be 100%. Thus, $\mathcal{E}$ is defined as follows:

$$\mathcal{E}(d[[i]], d[[i']] - (b_o, l_o, c_o)) \doteq \\ mAP(d[[i]], d[[i']] - (b_o, l_o, c_o)) = 100\%$$

To date, multiple variants of the standard mAP definition exist. We adopt one of the most popular mAP calculation methods, the PASCAL VOC metric [42], in our implementation.

3.2 Case Study of OD Failures

$\mathcal{E}$ defined in Sec. 3.1 enables a unified approach to check OD failures — it is image content agnostic and therefore can be automatically

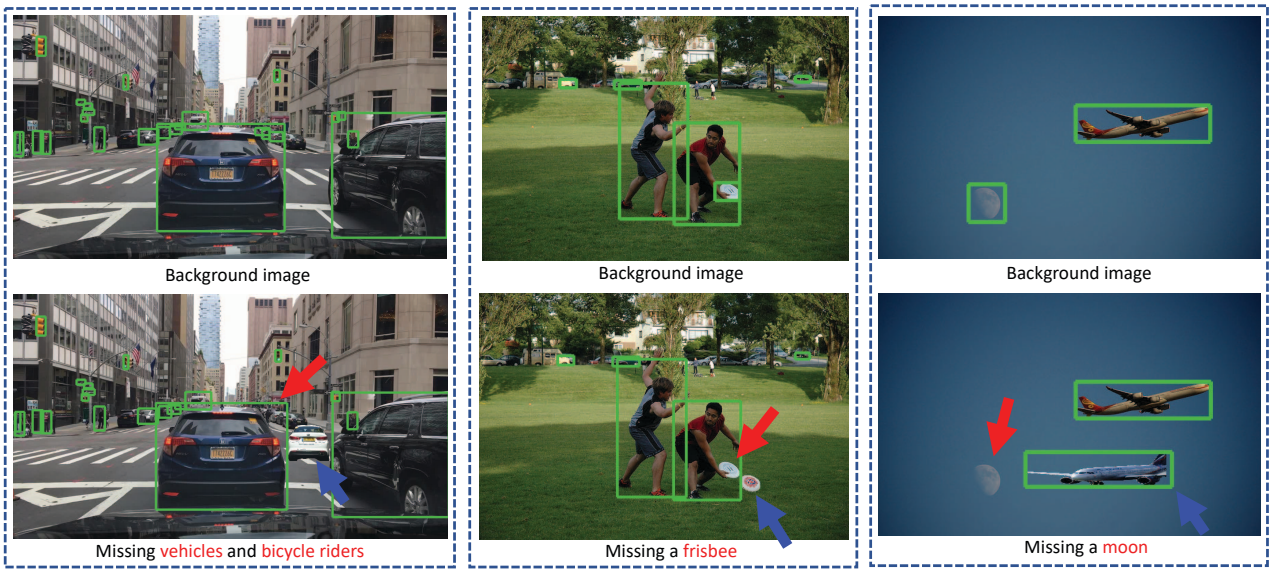

Figure 3: OD errors found by MetaOD. We slightly cherry picked images in favor of readability. Browse the full results at [8]. The inserted objects are pointed by blue arrows. To preserve realism of the synthetic images at our best effort, the inserted objects are resized to the average size of existing objects of the same category. See our discussions in Sec. 4.2.

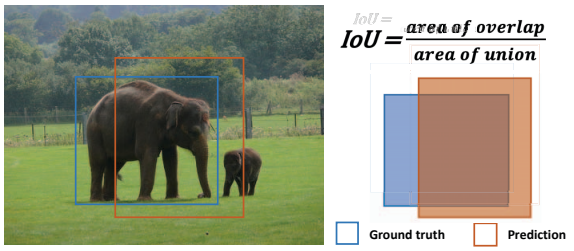

Figure 4: Intersection over Union (IoU).

conducted. From a holistic perspective, the following categories of OD defects can be found by asserting $\mathcal{E}$:

- **Recognition failures** represent errors which treat an arbitrary region on the image containing no object as an "object" or fail to recognize an existing object.
- **Classification failures** represent labeling errors, for instance labeling a human being as a "bird."
- **Localization failures** represent the failure where the OD uses a too large or too small bounding box to localize objects. As illustrated in Fig. 4, too large driftings ($IoU < \epsilon$) on the bounding box are not allowed.

Nevertheless, after manually checking OD failures found by MetaOD, we only find *recognition failures*.[1] MetaOD has successfully found a large number of OD failures by eight popular ODs (see Table 2). Fig. 3 reports three cases, where the "background" images on the first row are from the Berkeley DeepDrive dataset [5, 82] and the COCO dataset [45]. Images on the second row are generated by inserting one extra object on their corresponding reference "background."

By inserting extra objects (indicated by the blue arrows) into the background and checking the equality criteria $\mathcal{E}$ (Sec. 3.1), we were able to uncover many detection defects. The first column in Fig. 3 illustrates a recognition failure (indicated by the red arrows), where a bike rider and several cars in the traffic scene image could not be recognized after a new vehicle was inserted. Similarly, the synthetic images in the second and third columns unveil detection failures, where after inserting one extra object into images of real-world scenes, existing objects (moon and frisbee) cannot be recognized. We note that Fig. 3 demonstrates the *diversity* in the issues we found; MetaOD synthesizes test images of different scenes and therefore can find a broad set of defects. In contrast, existing relevant efforts [73, 84] primarily transform or synthesize images of only driving scenes (see Sec. 3.3). Also, we note that while the synthetic images reasonably represent real-life scenes, making inserted objects *fully consistent with arbitrary backgrounds* could be extremely challenging, if at all possible. See our further evaluation and discussion on this matter in Sec. 6.3 and Sec. 7.

3.3 Application Scope

It is worth noting that we are *not* testing extreme cases to stress ODs [12]. Apparently, we can synthesize images that are highly challenging to human beings and therefore challenging to ODs as well, for instance, by tweaking the contrast of objects and its background [23]. Therefore, while in this research we propose a set of techniques to select "realistic" objects for insertion (see Sec. 4.2 and Sec. 4.3), we still define a conservative test oracle such that we *exclude* the prediction over the newly inserted object, and check only the consistency of the remaining predictions.

[1] We found over 28K images triggering OD errors (reported in Sec. 6). We manually checked about 800 images by re-querying the remote services with the error triggering images and screened the detection outputs.

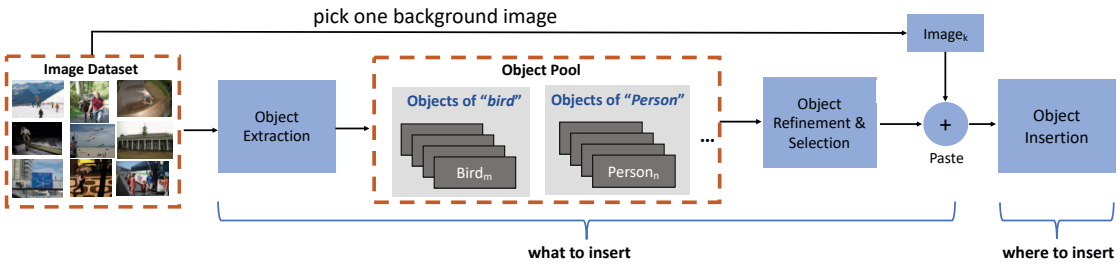

Figure 5: Workflow of MetaOD.

MetaOD is designed as a general framework to treat ODs as *black-boxes*. While de facto implementations of ODs are currently all on the basis of DNN, MetaOD could also be used to assess the performance of ODs based on other techniques. In addition, existing approaches on testing computer vision models [73, 84] apply predefined "severe weather conditions" (e.g., foggy and rainy) to transform or directly synthesize *entire images*. They are not tailored to pinpointing OD failures, and are conceptually *orthogonal* to the object-level mutations proposed in this work. Also, their transformations may be inapplicable to mutate arbitrary images while preserving realism. For instance, applying severe weather conditions toward images of indoor scenes is likely unreasonable.

4 Design

Fig. 5 depicts a holistic view of the proposed technique. To generate image $i' = \mathcal{C}(o, i)$ for testing, MetaOD is constructed as a streamlined workflow that includes object extraction, object selection/refinement, and object insertion modules. By providing MetaOD with a set of images (e.g., images from the COCO dataset [45]), its object extraction module performs advanced object instance segmentation techniques to identify and extract object instances (Sec. 4.1). Then, given an image as the "background", the object selection module determines an appropriate object to be inserted in the background (Sec. 4.2), using a set of criteria to find similar objects, rule out low-quality objects and adjust the object size. While the first two steps address the challenge of "what to insert", for a particular background image, we need to further answer the question of "where to insert." We aggregate empirical evidence and derive heuristics to select insertion positions. Furthermore, motivated by how delta debugging [83] is applied to test conventional software, we propose techniques to identify more locations for object insertion and therefore synthesize more images (Sec. 4.3).

4.1 Object Extraction

We start by performing object extraction to identify and extract a pool of object instances from input images. While object extraction is generally considered difficult, deep learning-based *instance segmentation* models have been shown to work well to extract objects from images [14, 33, 44]. For example, given the "two elephants" image in Fig. 1, instance segmentation models can recognize both elephants and put two masks over both "elephant" objects.

Similar to OD, instance segmentation model design also has two primary focuses, accuracy [33, 44] and speed [14, 76], which usually cannot coexist. In this work, we concentrate on models that emphasize speed over accuracy. The object extraction module is designed to swiftly extract objects from large sets of diverse images. Therefore, speed takes priority over accuracy (although in practice our adopted instance segmentation model has a good accuracy as well). In Sec. 4.2, we compensate for the "accuracy" of extracted objects by proposing techniques to rule out low-quality object images. By orchestrating object extraction and refinement modules in pipeline, we output sets of high-quality labeled object images with a modest cost and high speed.

To this end, we reuse YOLACT [7], a recently developed real-time instance segmentation tool, to build the object extraction module. Our empirical evidence (also reported in its paper [14]) shows that YOLACT has impressive speed and quite good accuracy in practice when processing real-world images. YOLACT outputs a mask over each recognized object instance. We extend YOLACT by reusing the object masks to extract each object from the background image and save each object into an individual image.

4.2 Object Refinement and Selection

Despite significant progress, instance segmentation remains a difficult problem, and we have observed that some of its outputs are of low quality. According to our observations, these "low-quality" object images occur for two main reasons: (1) some objects in the input image are too small, and (2) some objects overlap and therefore fragmentary object images are extracted.[2] We acknowledge the general difficulty of outputting high-quality object images. Instead, our object extraction module processes large sets of images at high speed, and we further prune low-quality objects and select appropriate objects closely related to a "background" image.

Small Object Image Pruning. As shown in Fig. 5, the output of object extraction consists of multiple sets of images, where each set contains object instances with the same label. During this step, we first prune small object images within each object set, which presumably include low-resolution or fragmentary images unsuitable for use. To perform pruning, we sort the object instance images within each set by image size and remove the majority of object images (in our implementation, we empirically decided to remove 90% of the object images).

Object Image Similarity Analysis. For a particular "background" image with several preexisting objects, we aim to find object instance images from the pool that are closely related to the background to fulfill the requirement of testing OD while also preserving the realism of the synthetic images as much as possible. To this

[2]Such general challenges still exist even if we tentatively tried more "heavy-weight" instance segmentation models like the Tensorflow implementation of Mask RCNN [33].

end, we perform an image similarity analysis using image hashing techniques. Image hashing is a standard technique for pixel-level image similarity analysis. The process creates similar hashes for similar images. In contrast, when using a crypto hash algorithm such as MD5, one byte of difference can lead to drastic hash value changes due to the avalanche effect [56].

Given an image i with three "birds", we start by computing the average image hash value of these "bird" object images. Then we iterate over all the "bird" images in the pool (see Fig. 5) and identify a "bird" whose image hash value has the shortest Hamming distance with the average hash value. This "bird" will be used for insertion. If image i contains objects with N different labels, we repeat the procedure N times. Therefore, N objects of different categories will be selected for insertion. In this way, we ensure the "realism" of the synthetic images as much as possible. Our observations show that the selected "similar" object images can usually exhibit texture and resolution that are close to those of the background image.

For the implementation, we use the average hash [40], which is a standard implementation for image hashing. Our tentative tests showed that this method helps find similar objects to a good extent at modest cost. Nevertheless, we acknowledge the difficulty, if it is at all possible, of finding *semantically similar* objects through a unified and cost-efficient approach. Indeed, image hashing uses *pixel-level* similarity instead of reflecting on the *meaning* of each object instance. We leave for future work to explore practical techniques to comprehend the semantic information of each object instance and refine object selection at this step.

Object Image Resizing. There is a relationship between object size and its "distance" in the image. A tiny object can be "further away" and challenging to identify for even human eyes, and is thus not considered in this work. Hence, before inserting a selected object into a background image i, we adjust the object size to match that of the existing objects in image i. We resize the object image to the average size of objects in i that belong to the same category. Also, as notated in Sec. 3, besides resizing, we do not "transform" objects (rotation, blurring, etc.) to preserve realism at our best effort.

4.3 Object Insertion

After selecting proper objects, we then seek proper locations on the background image for insertion. As discussed in Sec. 3.3, the software engineering (SE) community transforms entire images for testing, while the computer vision (CV) community primarily concerns with the visual appearance of the inserted object, rather than the "background" into which the object is placed [23, 36, 74]. Several studies have attempted to infer reasonable insertion locations using statistical methods such as probabilistic grammar models and have only applied them to images of indoor scenes [63]. However, building a generalized model for arbitrary scenes, if at all possible, is highly challenging in this research, where large-scale synthetic images are required to reveal erroneous OD results.

Given the general difficulty of leveraging heavy-weight statistical methods to infer "optimal" insertion locations, we propose lightweight strategies. In this section, we start by conducting empirical studies on locations where insertion can presumably trigger OD defects. Then, motivated by delta debugging used in testing

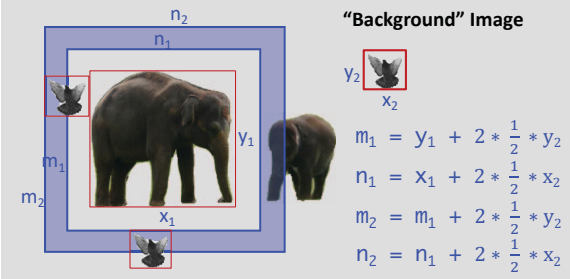

Figure 6: The "guided insertion" strategy to insert a "bird." The blue region is symmetrical and centered on the larger "elephant."

traditional software [83], we generate more synthetic images by progressively relocating inserted objects on background images.

Determining Object Insertion Locations. Our preliminary studies show that inserting objects *close to* existing objects in an image (referred to as *guided insertion* later in this paper) is likely to trigger erroneous predictions. This section presents empirical results to support our observation. To set up the study, we randomly selected 50 images from the COCO image set [45] and tentatively inserted a "bird" image. We tested eight popular ODs and show the evaluation results (descriptions of these ODs can be found in Table 2).

We adopt two insertion schemes: *random insertion* and *guided insertion*. Guided insertion works by randomly selecting one existing object from the background and inserting extra objects *close* to it. As shown in Fig. 6, after randomly selecting one elephant (e.g., the larger one) on the background and deciding to use a "bird" object, we create a blue region that is *symmetrical* and *centered* on the selected elephant.[3] We randomly select locations within the blue region as the centroid of the "bird". Our sampling guarantees that the "bird" will not overlap with the larger "elephant." Moreover, overlapping with any other objects is *not* allowed either; whenever the "bird" is sampled over existing objects in i we discard the synthetic image and resample.

In contrast, the random insertion scheme implements a simple strategy in which object o is placed randomly on the background. Again, we disallow overlapping of o with existing objects and resample whenever overlapping occurs. Additionally, for each background image i with N existing objects, we perform $10 \times N$ guided or random insertions. We report the erroneous OD outputs found w.r.t. these two setups as follows:

OD (See Table 2 for Descriptions)	#Errors Found By Random Insertion	#Errors Found By Guided Insertion	#Synthetic Images
Amazon Rekognintion	232	432	2,270
Microsoft Azure Vision	76	167	1,190
IBM Vision	39	86	850
Google AutoML Vision	365	461	1,570
SSD Mobilenet	45	112	1,290
SSD Inception	185	310	1,510
RCNN Resnet	218	381	2,980
RCNN Inception	294	580	3,190
Total	1,454	2,529	14,850

[3]When deciding the size of the blue region, we find that if it is too narrow, the search space of "object insertion locations" are too small and undesired. Similarly, if the blue region is too wide, it may overlap with other existing objects. Since we disallow the inserted object to overlap with existing objects, higher chance of overlapping leads to lower chance of finding proper insertion locations, and is thus undesired as well.

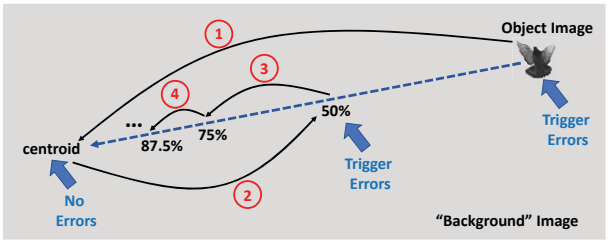

Figure 7: Move object instance image o toward the centroid.

Since ODs (introduced soon in Table 2) identify different numbers of objects from each image, we synthesize different amount of images for testing. The results show that the guided setting notably outperforms the first setting. This is consistent with our intuition; by inserting images near the local region of existing objects, the inserted images may disturb the regions or grids used for object recognition and thus cause failures in ODs.[4] As a result, MetaOD is configured with the guided insertion strategy.

Object Relocation. While the proposed methods provide practical guidelines on object insertion, "guided insertion" primarily focuses on locations *close* to existing objects and may miss opportunities for object insertion in other image regions. We further propose techniques to identify additional locations for object insertion, while still maintaining the "realism" of generated images insofar as possible. To accomplish this, we first compute the *centroid* of objects in the background image; then, motivated by the use of delta debugging for conventional software [83], the inserted object is progressively relocated toward the centroid to generate visually more diverse images while retaining to cause prediction errors.

The procedure is illustrated in Fig. 7, where our relocation scheme is implemented to explore locations closest to the centroid. Starting from an insertion location found by the guided-insertion strategy that can trigger OD failures, we relocate the inserted "bird" to the centroid of existing objects on the background. If no error can be provoked regarding the newly synthetic image, we jump back to the middle and recheck the OD. In case this time prediction errors do occur, we search forward until the "bird" becomes too close to (1) the centroid, (2) the previous successful insertion (i.e., triggering prediction errors) with the longest distance from the starting point, or (3) the starting point itself. Again, for this step, we disallow any overlap between the inserted object and existing objects on the background: whenever overlapping occurs, we jump back as well.

It is worth mentioning that while the prototype implementation of MetaOD is equipped to use "centroid" as the exploration destination, any locations could be configured at this step to synthesize diverse and realistic images with respect to user requirements.

5 Implementation

MetaOD is implemented in Python in approximately 3,600 lines of code (see our released codebase at [9]). The object extraction module of MetaOD is implemented by extending a popular instance segmentation module, YOLACT [7, 14]. We extended YOLACT by using the instance mask to crop the input image and extract object instance images. YOLACT is built with Pytorch (ver. 1.0.1), and contains a model pretrained with a de facto object detection dataset, COCO [45]. This dataset contains objects with approximately 90 labels, and we use the pretrained model to perform instance segmentation. As aforementioned, one desirable feature of YOLACT is that it performs instance segmentation rapidly — indeed, all the instance segmentation tasks are launched on one Nvidia GeForce GTX 1070 GPU with promising processing time (see Sec. 6).

[4]The implementation details of the first four commercial ODs are not disclosed. See further discussions in Sec. 6.

6 Evaluation

Table 2 lists the ODs used in the evaluation (the "Speed" and "COCO mAP" of four Tensorflow pre-trained models are disclosed by Google [3], and "Speed" of four commercial APIs are estimated by us). We use four commercial OD APIs provided by industry giants for the evaluation [1, 2, 4, 6]. Python scripts are written to interact with these remote services and retrieve the prediction results (in JSON format). To the best of our knowledge, the OD models used within these commercial services are not disclosed; single-stage models are presumably employed given their rapid prediction speed (Sec. 2).

Google also supports directly deploying its TensorFlow OD APIs on Google Cloud [3] and provides the flexibility to choose different models pretrained on the COCO dataset [45]. We follow the official tutorial to setup TensorFlow OD on Google Cloud [3], and from a total of five pretrained models suggested in the tutorial, we choose four models (i.e., "Tensorflow" models in Table 2) and exclude another pre-trained model, RFCN Resnet. RCNN Inception ResNet model [34] yields the best accuracy (mAP 37) but has the slowest speed. We also chose another RCNN model [80] and two SSD models [37, 47] that exhibit medium prediction speed and good accuracy. The tutorial reports that RFCN Resnet has a very similar accuracy with faster RCNN Resnet, and therefore, we skip to evaluate RFCN Resnet. As mentioned in Sec. 2, the two RCNN-based models have two-stage region-based architectures, while the SSD models have single-stage architectures that are much faster.

6.1 Evaluation Overview

Table 1 summarizes the evaluation results. To acquire these data, we extracted object instances from 1,000 randomly selected images from the COCO 2017 image set [45]. We then randomly selected 500 images from the same dataset as background images. From the complete set of 1,000 images, MetaOD extracted a total of 5,843 object instances clustered with respect to 79 different categories (person, dog, etc.). As previously mentioned (Sec. 4.2), the object refinement module of MetaOD sorts object images with respect to their size and eliminates 90% of the small object images; the remaining 10% of the object images are kept as insertion candidates.

Given an image i, ODs can find different numbers of objects from i (the third column of Table 1 reports the total number of objects found by an OD). As discussed in Sec. 4.3, suppose that an OD finds M objects in i, then, MetaOD generates $10 \times M$ synthetic images following the "guided-insertion" strategy to test the OD. When a test image i' triggers prediction errors, that image is used to generate additional test inputs following the "delta-debugging"-style procedure (Sec. 4.3). The total number of images synthesized for each OD is reported in the second column of Table 1.

Table 1: Result overview. Due to the limited space, the detector names are simplified in this table and also in the rest of the paper. Note that "Processing Time" includes the prediction time of ODs.

OD	#Synthetic Images	#Detected Objects	#Images Causing Detection Failures	Processing Time (Hours)	Total Cost (USD)
Amazon	38,939	3,750	6,060 (15.6%)	11.9	$21.5
Google	18,655	1,801	2,738 (14.7%)	13.0	$18.8
Microsoft	20,453	1,985	2,494 (12.2%)	3.2	free
IBM	13,280	1,290	1,515 (11.4%)	2.2	free
SSD Mobilenet	24,796	2,387	3,460 (14.0%)	62.5	$53.4
SSD Inception	29,072	2,806	3,988 (13.7%)	64.3	$54.9
RCNN Resnet	70,754	6,914	7,442 (10.5%)	164.8	$140.7
RCNN Inception	76,257	7,349	10,648 (14.0%)	290.8	$248.3
Total	292,206	28,282	38,345 (13.1%)	612.7	$537.6

Table 2: ODs evaluated in this research.

OD Name	Speed	COCO mAP
Amazon Rekognintion API [1]	fast	N/A
Google AutoML Vision API [2]	fast	N/A
Microsoft Azure Vision API [4]	fast	N/A
IBM Vision API [6]	fast	N/A
TensorFlow SSD Mobilenet [37]	fast	21
TensorFlow SSD Inception [47]	fast	24
TensorFlow faster RCNN Resnet [29]	medium	32
TensorFlow faster RCNN Inception Resnet [72]	slow	37

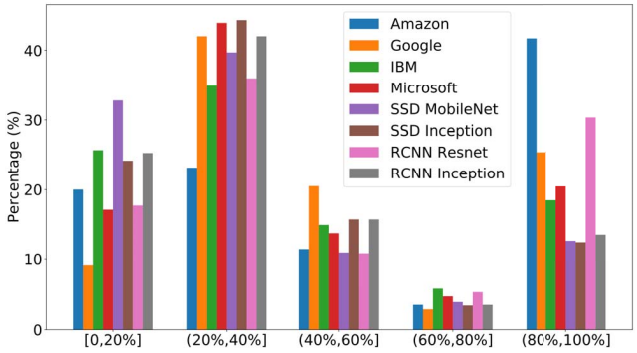

Figure 8: Efficiency of object relocation. Recall we leverage a "delta-debugging"-style method to relocate the inserted objects toward the centroid (Fig. 7). X-axis reports that how far the inserted object can proceed toward the centroid: 100% indicates that the object is placed at the centroid.

The number of images triggering prediction errors (not "number of bugs in the models") is reported in the fourth column of Table 1. At least 10% of the synthetic images triggered erroneous predictions of the evaluated ODs. Since we randomly decide locations to insert objects into background images (see Sec. 4.3), error-triggering images are very unlikely identical. Like typical software testing scenarios, not every test input can trigger a bug. Having 10-16% bug-triggering tests consistently across different models is very promising and shows that MetaOD provides a general, practical, and consistent approach to finding OD errors. Overall, Table 1 shows that OD failures are a general concern, regardless of the underlying model. Moreover, when a model detects more objects in images, the number of images that can trigger failures also increases. This is expected, as recall that, for an image of M objects, our "guided-insertion" strategy generates $10 \times M$ synthetic images (Sec. 4.3). Also, while the implementation of commercial APIs are unknown and are generally deemed as "fast" (see Table 2), we suspect that they are not using similar models, since their prediction capabilities are very different (cf. the third column in Table 1).

Processing Time. This evaluation was conducted on a machine equipped with an Intel i7-8700 CPU and 16GB RAM. The instance segmentation module runs on a single Nvidia GeForce GTX 1070 GPU with CUDA 9.0. Table 1 reports the processing time. Commercial APIs, particularly the Google and Microsoft services, take less time for prediction than the TensorFlow pretrained models. Although the implementation details of these remote services are not disclosed, from the results, we can assume that the commercial remote services presumably leverage highly optimized single-stage OD models that are faster but usually find fewer objects in images.

Financial Cost. Enabled by modern cloud computing infrastructures, all of these ODs are designed as "pay-as-you-go" models: users are charged based on how many queries they send to the services (for the first four services) or how many computing resources they use (for the TensorFlow services). We report the amount in terms of USD that we are charged by these services in Table 1. Due to erroneous behavior, some queries were indeed wasted. More importantly, given that commercial services have been adopted in supporting critical computer vision applications (e.g., surveillance cameras), we envision real-world scenarios where the prediction errors can cause financial loss or fatal errors.

6.2 Efficiency of Object Relocation

As discussed in Sec. 4.3, inspired by delta debugging, we propose techniques to mutate synthetic images by progressively moving an inserted object that triggers erroneous predictions toward the centroid of objects in the background image. We preserve the realism of synthetic images at our best effort by placing the inserted object into a realistic position, while finding more insertion locations and augmenting the visual diversity of the synthetic images.

We report the breakdown of synthetic images causing OD errors in Table 3. The second column reports the number of images triggering prediction errors that are synthesized by inserting objects into the background image, while the third column reports the number of images triggering prediction errors and are synthesized by relocating inserted objects toward the centroid. Since the same object could be inserted at different positions on a background image, and then reaching to the *same centroid*, we also measure the unique number of synthetic images at this step (the last column). As shown in Table 3, the object relocation step successfully finds a

Table 3: Breakdown of images causing OD failures. By adding the second and the last columns, we get the "#Images Causing Detection Failure" column in Table 1.

OD	#Synthetic Images With Inserted Obj.	#Synthetic Images With Relocated Obj.	#Unique Synthetic Images With Relocated Obj.
Amazon	4,621	2,501	1,439
Google	2,093	853	645
Microsoft	1,891	742	603
IBM	1,135	481	380
SSD Mobilenet	2,534	1,089	926
SSD Inception	2,976	1,192	1,012
RCNN Resnet	5,828	2,544	1,614
RCNN Inception	7,881	3,422	2,767

Table 4: Naturalness evaluation w.r.t. average HOG intersection (higher value is better). We report evaluation results of both synthetic images with inserted objects (second column) and relocated objects (third column).

OD	Average HOG Intersection Rate (%) for Inserted Obj.	Average HOG Intersection Rate (%) for Relocated Obj.
Amazon	98.9	98.8
Google	98.1	98.0
Microsoft	98.8	98.7
IBM	98.5	98.5
SSD Mobilenet	98.7	98.7
SSD Inception	98.6	98.6
RCNN Resnet	98.9	98.9
RCNN Inception	98.9	98.9

large number of unique images retaining prediction errors. Among a total of 28,959 synthetic images causing prediction errors, 9,386 (32.4%) unique images are created via object relocation. Moreover, we report the average distance (in terms of percentage) by which the inserted object can be relocated in Fig. 8; we consider arriving at the centroid as 100% and staying at the starting position as 0%. Note that Fig. 8 has excluded all the "0%" cases, where objects stay at the starting positions. As shown in Fig. 8, 21.9% of objects on average can be placed at the centroid while retaining errors, and 40.2% of objects are relocated by at least 40% of the distances. The overall results as promising, illustrating that a considerable number of synthetic images could be generated that retain prediction failures, and also make the images visually more diverse.

6.3 Naturalness of Synthetic Images

While the "naturalness" of a synthetic image may be subjective, as noted by existing research, natural images are deemed to have certain statistical regularities [38, 51, 52]. Therefore, following the commonly adopted convention in the computer vision literature [52], the "naturalness" of synthetic images is measured by first computing a histogram of oriented gradients (HOG [19]) of both synthetic images and their corresponding background images, and then computing the intersection of these two HOGs. HOG is a popular metric extracting distribution (histograms) of directions of gradients as "features" of an image. By summarizing the magnitude of gradients, this metric captures abrupt intensity changes in the image (object edges, object corners, etc.), and therefore is usually effective for comprehending high-level representations of images with multiple objects. In contrast, pixel-level similarity metrics (Sec. 4.2) leveraged in MetaOD focus on single object instance comparison, and are not applicable in this evaluation. Consistent with previous research [52], the comparison output (i.e., HOG intersection), a value between 0 and 1, illustrates the naturalness of synthetic images. We provide discussions regarding image naturalness in Sec. 7.

Table 4 reports HOG intersection rates (second column) by comparing synthetic images with inserted objects and their corresponding background images, and HOG intersection rates (third column) by comparing synthetic images with relocated objects and their corresponding backgrounds. Consistent with our intuition, all synthetic images have highly similar HOG regularities with their corresponding backgrounds, and are deemed "natural" (in contrast, we report that the HOG intersection rate of two randomly-selected images from the COCO dataset is below 50.0%). Also, while most synthetic images with relocated objects have HOG intersections identical to those of the synthetic images with inserted objects, there are three cases for which the synthetic images with relocated objects exhibit a slightly lower rate. Intuitively, relocation generates visually more diverse images and can potentially lead to lower "similarity" comparing to their corresponding background images.

6.4 Retraining with Error-Triggering Images

To capitalize on the synthetic images that triggered prediction failures, we show that such synthetic images can be used to retrain models and substantially improve their performances. We use a popular autonomous-driving dataset, Berkeley DeepDrive [5, 82], for this evaluation. This dataset contains images that depict real-time driving experiences under different weather conditions and at various times. Experiments in this section (i.e., model retraining) are conducted on a server equipped with an Intel Xeon CPU E5-2680 with 256 GB of RAM and eight Nvidia GeForce RTX 2080 GPUs.

We downloaded the SSD MobileNet object detection model pretrained by TensorFlow and retrained the model (with Tensorflow ver. 1.14.0) by using 900 images annotated with 10 common categories for traffic scenes from the DeepDrive training set. We actually imitated how OD models are customized and used in practice; based on transfer learning [54], pretrained models are adapted to similar tasks by fine-tuning the model parameters on a new dataset. At this step, we reuse the *default configuration* shipped with the MobileNet pretrained model; the batch size is 48 which means the whole training set will be processed once within 19 steps. We set up three retraining strategies ($Config_{1-3}$) as follows:

- We start by retraining the MobileNet model with 900 images for 200K steps (200K is the default setting in the model's configuration) and exporting the retrained model m_0. We also form an evaluation set by randomly selecting 100 images from the DeepDrive evaluation set.
- We then use MetaOD to generate new synthetic images from the 900 images and collect synthetic images that cause prediction failures of m_0. This step generates 18,707 images (denoted as $\mathcal{I}^*$) triggering prediction errors.
- $Config_1$: starting from m_0, we resume retraining with the 900 images for another 10K steps.
- $Config_2$: starting from m_0, we extend the original training set of 900 images with 900 images randomly selected from $\mathcal{I}^*$, and resume the model retraining with these 1,800 images for

another 10K steps. To label each synthetic image, we reuse the label of its reference input image (see Sec. 3.1).

- We also use MetaOD to generate another set of 900 images (denoted as $\mathcal{I}$). We do *not* check whether these images can trigger prediction failures or not.
- $Config_3$: starting from m_0, we extend the training set of 900 images with 900 images in $\mathcal{I}$, and resume the model retraining for another 10K steps. Again, to label each synthetic image, the label of its reference input is reused.

During the retraining of in total 210K steps, we measure the total loss and mAP score regarding the evaluation set of 100 images.

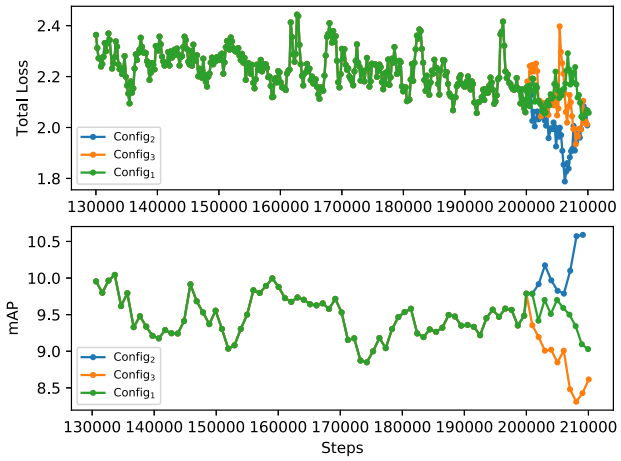

Figure 9: Smoothed total loss (first diagram) and mAP scores (second diagram) during model retraining. Due to limited space, we show performance starting from 130K steps.

As reported in Fig. 9, for the last 10K steps, $Config_1$ shows consistent trending compared to the first 200K steps. $Config_2$ outperforms the other two by having lower total loss. Moreover, the mAP score of $Config_2$ clearly outperforms those of the other configurations. $Config_3$ exhibits a slightly better total loss decrease than that of $Config_1$, but yields an even lower mAP score (which may be due to overfitting). We report that the average mAP scores of the three configurations within the last 10K steps are as follows:

	$Config_1$	$Config_2$	$Config_3$
mAP	9.3	10.5	8.6

As the table shows, model performance is increased by retraining with the synthetic images of the prediction errors. Note that according to OD surveys (e.g., Table Two in [86] and Table Seven in [46]), one point mAP score increase is significant. Overall, we interpret the evaluation result as promising: the failure-aware retraining demonstrated in this section sheds light on practical usages of the model prediction errors found by MetaOD and provides promising directions to improve model accuracy.

We acknowledge that the evaluation, while being *fair*, may not illustrate the *best practice* to promote model performance; Fig. 9 indicates that the "sweet spot" might be around 8K steps of retraining (where the mAP score is approximately 10.7). Overall, we consider that providing guidelines of best practice is beyond the scope of this research, but the reported results have illustrated the potential. Additionally, images are synthesized from the existing training set; in other words, we do *not* need new real images. Overall, "failure-aware" retraining is *orthogonal* to standard model retraining techniques and can potentially be orchestrated together.

7 Discussion and Future Work

In this paper, we presented the design, implementation and evaluation of MetaOD, a systematic workflow for automatically testing the erroneous behaviors of object detection systems. The proposed techniques can be adopted to promote object detector training and to motivate this emerging line of research. In this section, we present a discussion and several potentially promising directions for future research.

Comparison with Work in the CV Community. Parallel to SE community's efforts on testing deep learning systems, the CV community generates synthetic inputs by mutating real images to train DNNs. We compare and illustrate the novelty of MetaOD with related CV research along several aspects:

Blackbox vs. Whitebox. Most existing CV research considers a "white-box" setting (e.g., [59, 68]). Such efforts either require an in-depth understanding of the model structure to adaptively synthesize inputs [68], or use the hidden layers of DNNs and computed gradients to guide input synthesis [59]. As aforementioned, we consider blackbox setting for software testing and introduces MetaOD to effectively test commercial remote OD models.

Training with Synthetic Inputs vs. Re-training with Bug-triggering Synthetic Inputs. To our knowledge, related CV research directly uses synthetic inputs for model training. Contrarily, MetaOD suggests a novel failure-aware model retraining scheme (Sec. 6.4) to effectively improve model accuracy, which sheds light on future work to continue testing the "re-trained" model, i.e., the whole process would loop to iteratively re-train the model. We expect the model accuracy to further improve until reaching saturation. As evaluated in Fig. 9, model re-training with *arbitrarily generated* synthetic images may lead to decreased model accuracy, while re-training with *bug-triggering* synthetic inputs leads to significantly improved accuracy. This interesting observation suggests further research opportunities for both the CV and SE communities.

Fine-grained Modeling/Tuning on Synthetic Image for Training vs. Generic Framework to Synthesize Images for Testing. As Sec. 8 will review, existing CV research mostly performs heavyweight, fine-grained modeling to generate synthetic images, e.g., specifically modeling motion trajectory to synthetic images and train surveillance cameras [13]. These works usually focus on *specific application domains*, and leverage *domain knowledge* to fine-tune and optimize the synthetic images with statistical tools. We have a different design goal of proposing a general framework to efficiently produce a large amount of quality inputs for testing. MetaOD is, in general, agnostic of image "semantics" (except labels of existing objects in the image) and therefore can be more efficient and robust.

Novelty w.r.t. DNN Testing Work in the SE Community. As discussed in Sec. 8, most existing testing work in the SE community

focuses on image classification models (e.g., [58, 73]), or the underlying infrastructures of TensorFlow/PyTorch (e.g., [21, 39, 61, 85]). To our knowledge, no prior work focuses on designing a general, effective pipeline to test OD models, another class of fundamental models used in many real-world critical applications. MetaOD mutates and observes the detection of individual objects in an image, while existing research on testing image classification performs mostly whole image-wise mutations (e.g., adding foggy and rainy conditions). Some of these transformations are not applicable to our scenario (discussed in Sec. 3.3), and our approach is generally *orthogonal* to these whole image-wise mutations.

We also demonstrate the feasibility of "failure-driven retraining" (Sec. 6.4) with notably improved model accuracy. This evaluation initializes a promising step toward addressing a typical concern in the SE community on "how to use findings of DNN testing", which has not been systematically explored by previous works.

Measuring "Naturalness" of Synthetic Images. As noted in Sec. 6.3, we follow conventions in the CV community [52] to compare synthetic images with natural images w.r.t. HOG as a way of measuring the "naturalness". Nevertheless, as discussed in Sec. 4.3, it is challenging, if not impossible, to understand the "semantics" of an object and make it fully consistent with arbitrary backgrounds (i.e., more "natural"). For instance, the traffic image in Fig. 3 can be more natural, if the perspective of the inserted car slightly turns left rather than entering the sidewalk. To our knowledge, CV community is exploring statistical tools to indepthly comprehend only a few specific scenes and objects (e.g., mutating human gestures within indoor scenes [63]). In contrast, this work introduces a general, efficient pipeline for arbitrary images.

Reason to Preserve "Naturalness" of Synthetic Images. We note that "realism/naturalness" of synthetic images does *not* have direct influence on testing. Randomly mutating pixels to generate "fuzzy" and "unreal" images as the test inputs, which are challenging for human eyes to recognize objects, could also be used for stress-testing ODs.

However, we argue that preserving the "realism" of synthetic images at our best effort when designing the workflow of MetaOD is indeed *beneficial*. First, as noted in Sec. 3.3, we are not testing extreme cases to stress ODs (not like a typical fuzz testing setting). Stress testing of DNNs would be different. Second, synthesizing more "realistic" images facilitate the practical usage of MetaOD. As mentioned in Sec. 3, we aim to augment the realism to a certain extent, such that synthetic images triggering erroneous predictions could imply real-life failures of OD systems, leading to confusions during the daily usage. Also, we argue that using realistic and error-triggering images to *retrain* OD models (Sec. 6.4) deems a more reasonable approach than using arbitrarily mutated error-triggering inputs.

8 Related Work

Testing of Deep Learning Systems. Testing techniques for conventional software have been recently applied for deep learning systems, including fuzz testing [53, 81], mutation testing [49, 78], metamorphic testing [22, 73, 84], and also symbolic execution [31, 70, 71]. The majority of existing work focuses on image classification and its adoption on autonomous driving systems [58, 73, 84, 88]. NLP models, as well as typical downstream applications (e.g., machine translation), have also been tested using various testing schemes [27, 35, 50, 75]. We note that previous work on testing deep learning systems often adopts a "differential testing" scheme [58, 73]. However, OD models can usually recognize different number of objects from an image (due to the model capability), leading to the general challenge for cross comparison. For instance, when analyzing the "beach" sample image (https://git.io/fjFgI) provided by TensorFlow developers, Amazon Rekognition API [1] locates 13 objects while Microsoft Azure API [4] locates only four. In contrast, our work adopts metamorphic testing as an effective and adaptive testing strategy to reveal defects in these commercial ODs. Regarding testing oracle selection, neuron coverage [58, 73] and other finer-grained coverage metrics have been proposed [39, 48]. Also, in addition to the deep learning models, the underlying infrastructures (e.g., TensorFlow [10] and PyTorch [55]) have also been tested to find implementation bugs [21, 61, 85].

Data Augmentation for OD Model Training. In parallel to the SE community's progress in testing DNNs, data augmentation, which generates synthetic inputs by mutating real images, is an important technique to train DNNs. To train deep learning models (e.g., for object detection), augmentation methods vary from geometrical transformations such as horizontal flipping to color perturbations to adding noise to an image (e.g., mimic severe weather conditions) [23, 25, 26, 28, 36, 62, 74, 87]. Model accuracy can usually be improved by including such synthetic images into training data [28].

Most existing work prioritizes *local* rather than global consistency when augmenting images. For instance, when inserting new objects into training images, they focus more on the realism of inserted objects than on the context surrounding. Many synthetic images are unrealistic from a global point of view, such as putting a car on the table [74]. A few studies leverage heavyweight statistics methods to infer a "realistic" location for object insertion; they assumes a "white-box" setting and can handle only a few domain-specific scenes [63]. Our work proposes a lightweight and systematic new focus to promote synthetic images by considering both local and global realism. We take a "black-box" setting that facilitates the testing of commercial remote ODs. In addition, our testing focus enables "failure-aware" model retraining (Sec. 6.4), which effectively improves the model accuracy.

9 Conclusion

OD systems have been commonly used in real-world scenes such as building automated driving cars. This paper has introduced a novel metamorphic testing approach toward reliable ODs. Evaluation results are promising — MetaOD can find thousands of prediction errors from commercial APIs and OD models. We also show that the generated images can be used for retraining and substantially improving OD model accuracy.

Acknowledgments

We thank the anonymous ASE reviewers for their valuable feedback. We thank Yuyan Bao, Wei Luo, and Jiang Zhang for helping and commenting on this project.

References

[1] 2018. Amazon Rekognition. https://aws.amazon.com/rekognition/.

[2] 2018. Google Cloud Detecting Objects. https://cloud.google.com/vision/docs/detecting-objects.

[3] 2018. Tensorflow Object Detection API. https://cloud.google.com/solutions/creating-object-detection-application-tensorflow.

[4] 2019. Azure Computer Vision API. https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/.

[5] 2019. berkeley DeepDrive. https://bdd-data.berkeley.edu/.

[6] 2019. IBM Vision API. https://developer.ibm.com/exchanges/models/all/max-object-detector/.

[7] 2019. YOLACT. https://github.com/dbolya/yolact.

[8] 2020. Dropbox folder of all erroneous detection results found by MetaOD. https://www.dropbox.com/sh/ivpprt96nycokd9/AAAZ_3D8TVAF25L90Z9AqpRya?dl=0.

[9] 2020. MetaOD Codebase. https://github.com/MetaOD/MetaOD.

[10] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*. USENIX Association, Berkeley, CA, USA, 265–283. http://dl.acm.org/citation.cfm?id=3026877.3026899

[11] Raja Ben Abdessalem, Annibale Panichella, Shiva Nejati, Lionel C. Briand, and Thomas Stifter. 2018. Testing Autonomous Cars for Feature Interaction Failures Using Many-objective Search. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. 143–154.

[12] Michael A. Alcorn, Qi Li, Zhitao Gong, Chengfei Wang, Long Mai, Wei-Shinn Ku, and Anh Nguyen. 2019. Strike (with) a Pose: Neural Networks Are Easily Fooled by Strange Poses of Familiar Objects *(CVPR 2019)*.

[13] James Black, Tim Ellis, and Paul Rosin. 2003. A Novel Method for Video Tracking Performance Evaluation. In *In Joint IEEE Int. Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance (VS-PETS*. 125–132.

[14] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. 2019. YOLACT: Real-time Instance Segmentation. *arXiv preprint arXiv:1904.02689* (2019).

[15] Neal E. Boudette. 2017. Tesla's Self-Driving System Cleared in Deadly Crash. https://www.nytimes.com/2017/01/19/business/tesla-model-s-autopilot-fatal-crash.html.

[16] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases*. Technical Report. Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong ….

[17] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Comput. Surv.* 51, 1, Article 4 (Jan. 2018), 27 pages.

[18] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*. 379–387.

[19] Navneet Dalal and Bill Triggs. 2005. Histograms of oriented gradients for human detection.

[20] Xiaoning Du, Xiaofei Xie, Yi Li, Lei Ma, Yang Liu, and Jianjun Zhao. 2019. DeepStellar: Model-based Quantitative Analysis of Stateful Deep Learning Systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2019)*. 477–487.

[21] Saikat Dutta, Owolabi Legunsen, Zixin Huang, and Sasa Misailovic. 2018. Testing Probabilistic Programming Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*.

[22] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. 2018. Identifying Implementation Bugs in Machine Learning Based Image Classifiers Using Metamorphic Testing. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*. 118–128.

[23] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. 2017. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *Proceedings of the IEEE International Conference on Computer Vision*. 1301–1310.

[24] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88, 2 (2010), 303–338.

[25] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. 2018. Synthetic data augmentation using GAN for improved liver lesion classification. In *2018 IEEE 15th International Symposium on Biomedical Imaging (ISBI 2018)*. IEEE, 289–293.

[26] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. 2016. Virtual worlds as proxy for multi-object tracking analysis. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 4340–4349.

[27] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *ACM ESEC/FSE*. ACM, 498–510.

[28] Georgios Georgakis, Arsalan Mousavian, Alexander C Berg, and Jana Kosecka. 2017. Synthesizing training data for object detection in indoor scenes. *arXiv preprint arXiv:1702.07836* (2017).

[29] Ross B. Girshick. 2015. Fast R-CNN. *ICCV*.

[30] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR* (2014).

[31] Divya Gopinath, Kaiyuan Wang, Mengshi Zhang, Corina S Pasareanu, and Sarfraz Khurshid. 2018. Symbolic execution for deep neural networks. *arXiv preprint arXiv:1807.10439* (2018).

[32] David Grossman. 2018. Uber Self-Driving Car Kills Pedestrian in Arizona. https://www.popularmechanics.com/technology/infrastructure/a19482100/uber-self-driving-car-kills-pedestrian-in-arizona/.

[33] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. 2017. Mask R-CNN. In *Proceedings of the IEEE international conference on computer vision*. 2961–2969.

[34] 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*. 770–778.

[35] Pinjia He, Clara Meister, and Zhendong Su. 2020. Structure-invariant testing for machine translation. (2020).

[36] Stefan Hinterstoisser, Vincent Lepetit, Paul Wohlhart, and Kurt Konolige. 2018. On pre-trained image features and synthetic images for deep learning. In *Proceedings of the European Conference on Computer Vision (ECCV)*. 0–0.

[37] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861* (2017).

[38] Jinggang Huang and David Mumford. 1999. Statistics of natural images and models. In *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, Vol. 1. IEEE, 541–547.

[39] Jinhan Kim, Robert Feldt, and Shin Yoo. 2019. Guiding Deep Learning System Testing Using Surprise Adequacy. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. 1039–1049.

[40] Neal Krawetz. 2011. Average Hash. http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 60, 6 (May 2017), 84–90.

[42] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, et al. 2018. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982* (2018).

[43] Damon Lavrinc. 2018. This Is How Bad Self-Driving Cars Suck In The Rain. https://jalopnik.com/this-is-how-bad-self-driving-cars-suck-in-the-rain-1666268433.

[44] Yi Li, Haozhi Qi, Jifeng Dai, Xiangyang Ji, and Yichen Wei. 2017. Fully convolutional instance-aware semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2359–2367.

[45] 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.

[46] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2018. Deep Learning for Generic Object Detection: A Survey. *CoRR* abs/1809.02165 (2018).

[47] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. 2016. SSD: Single shot multibox detector. In *European conference on computer vision*. Springer, 21–37.

[48] Lei Ma, Felix Juefei-Xu, Jiyuan Sun, Chunyang Chen, Ting Su, Fuyuan Zhang, Minhui Xue, Bo Li, Li Li, Yang Liu, et al. [n. d.]. DeepGauge: Comprehensive and multi-granularity testing criteria for gauging the robustness of deep learning systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*.

[49] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, Felix Juefei-Xu, Chao Xie, Li Li, Yang Liu, Jianjun Zhao, et al. 2018. Deepmutation: Mutation testing of deep learning systems. In *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.

[50] Pingchuan Ma, Shuai Wang, and Jin Liu. 2020. Metamorphic Testing and Certified Mitigation of Fairness Violations in NLP Models. In *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*. 458–465.

[51] Aravindh Mahendran and Andrea Vedaldi. 2015. Understanding deep image representations by inverting them. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 5188–5196.

[52] Aravindh Mahendran and Andrea Vedaldi. 2016. Visualizing deep convolutional neural networks using natural pre-images. *International Journal of Computer Vision* 120, 3 (2016), 233–255.

[53] Augustus Odena and Ian Goodfellow. 2018. Tensorfuzz: Debugging neural networks with coverage-guided fuzzing. *arXiv preprint arXiv:1807.10875* (2018).

[54] Sinno Jialin Pan and Qiang Yang. 2009. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2009), 1345–1359.

[55] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in PyTorch. In *NIPS-W*.

[56] Priyadarshini Patil, Prashant Narayankar, DG Narayan, and S Md Meena. 2016. A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science* 78 (2016), 617–624.

[57] 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 (SOSP '17)*. ACM, 1–18.

[58] 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 (SOSP '17)*. ACM, New York, NY, USA, 1–18. https://doi.org/10.1145/3132747.3132785

[59] Xingchao Peng, Baochen Sun, Karim Ali, and Kate Saenko. 2014. Exploring Invariances in Deep Convolutional Neural Networks Using Synthetic Images. *CoRR* abs/1412.7122 (2014). arXiv:1412.7122 http://arxiv.org/abs/1412.7122

[60] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. 2010. Improving the Fisher Kernel for Large-scale Image Classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV (ECCV'10)*. 143–156.

[61] Hung Viet Pham, Thibaud Lutellier, Weizhen Qi, and Lin Tan. 2019. CRADLE: Cross-Backend Validation to Detect and Localize Bugs in Deep Learning Libraries. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*.

[62] Aayush Prakash, Shaad Boochoon, Mark Brophy, David Acuna, Eric Cameracci, Gavriel State, Omer Shapira, and Stan Birchfield. 2018. Structured Domain Randomization: Bridging the Reality Gap by Context-Aware Synthetic Data. *arXiv preprint arXiv:1810.10093* (2018).

[63] Siyuan Qi, Yixin Zhu, Siyuan Huang, Chenfanfu Jiang, and Song-Chun Zhu. 2018. Human-centric indoor scene synthesis using stochastic grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5899–5908.

[64] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 779–788.

[65] Joseph Redmon and Ali Farhadi. 2017. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 7263–7271.

[66] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018).

[67] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*. 91–99.

[68] Artem Rozantsev, Vincent Lepetit, and Pascal Fua. 2014. On Rendering Synthetic Images for Training an Object Detector. *CoRR* abs/1411.7911 (2014). arXiv:1411.7911 http://arxiv.org/abs/1411.7911

[69] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115, 3 (2015), 211–252.

[70] Youcheng Sun, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill, and Rob Ashmore. 2019. DeepConcolic: Testing and Debugging Deep Neural Networks. In *Proceedings of the 41st International Conference on Software Engineering: Companion Proceedings (ICSE '19)*. 111–114.

[71] Youcheng Sun, Min Wu, Wenjie Ruan, Xiaowei Huang, Marta Kwiatkowska, and Daniel Kroening. 2018. Concolic Testing for Deep Neural Networks. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. 109–119.

[72] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2818–2826.

[73] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. 303–314.

[74] Jonathan Tremblay, Aayush Prakash, David Acuna, Mark Brophy, Varun Jampani, Cem Anil, Thang To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. 2018. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*. 969–977.

[75] 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)*. 98–108.

[76] Jonas Uhrig, Eike Rehder, Björn Fröhlich, Uwe Franke, and Thomas Brox. 2018. Box2pix: Single-shot instance segmentation by assigning pixels to object boxes. In *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 292–299.

[77] J. R. Uijlings, K. E. Sande, T. Gevers, and A. W. Smeulders. 2013. Selective Search for Object Recognition. *Int. J. Comput. Vision* 104, 2 (Sept. 2013), 154–171.

[78] Jingyi Wang, Guoliang Dong, Jun Sun, Xinyu Wang, and Peixin Zhang. 2019. Adversarial Sample Detection for Deep Neural Network Through Model Mutation Testing. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. 1245–1256.

[79] Xiaoyu Wang, Tony X Han, and Shuicheng Yan. 2009. An HOG-LBP human detector with partial occlusion handling. In *2009 IEEE 12th international conference on computer vision*. IEEE, 32–39.

[80] Xiaolong Wang, Abhinav Shrivastava, and Abhinav Gupta. 2017. A-fast-rcnn: Hard positive generation via adversary for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2606–2615.

[81] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Hongxu Chen, Minhui Xue, Bo Li, Yang Liu, Jianjun Zhao, Jianxiong Yin, and Simon See. 2018. Coverage-guided fuzzing for deep neural networks. *arXiv preprint arXiv:1809.01266* (2018).

[82] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. 2018. BDD100K: A Diverse Driving Video Database with Scalable Annotation Tooling. *CoRR* (2018).

[83] Andreas Zeller. 1999. Yesterday, My Program Worked. Today, It Does Not. Why?. In *Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-7)*. 253–267.

[84] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. 2018. DeepRoad: GAN-based Metamorphic Testing and Input Validation Framework for Autonomous Driving Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*.

[85] Yuhao Zhang, Yifan Chen, Shing-Chi Cheung, Yingfei Xiong, and Lu Zhang. 2018. An Empirical Study on TensorFlow Program Bugs. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*. 129–140.

[86] Zhong-Qiu Zhao, Peng Zheng, Shou-tao Xu, and Xindong Wu. 2018. Object Detection with Deep Learning: A Review. *CoRR* (2018).

[87] Zhedong Zheng, Liang Zheng, and Yi Yang. 2017. Unlabeled samples generated by gan improve the person re-identification baseline in vitro. In *Proceedings of the IEEE International Conference on Computer Vision*. 3754–3762.

[88] Husheng Zhou, Wei Li, Yuankun Zhu, Yuqun Zhang, Bei Yu, Lingming Zhang, and Cong Liu. 2020. Deepbillboard: Systematic physical-world testing of autonomous driving systems. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE)*.