Automated Testing of DNN Models for Unreliable Inferences

YONGQIANG TIAN, The Hong Kong University of Science and Technology, China
MEIZINIU LI, The Hong Kong University of Science and Technology, China , China
TSZ ON LI, The Hong Kong University of Science and Technology, China
SHING-CHI CHEUNG*, The Hong Kong University of Science and Technology, China

Deep Neural Network (DNN) models are widely used for object detection. While they offer high performance in terms of accuracy, researchers are often concerned about whether a DNN model inappropriately made an inference using features irrelevant to a target object on a given image. If so, such an inference is considered unreliable. To address this concern, we propose two metamorphic relations (MRs) to detect an unreliable inference. The first MR expects that after an image's features relevant to a target object are corrupted, a DNN model should no longer detect the target object from the image. The second MR expects that after an image's features irrelevant to a target object (e.g., background features of the image) are corrupted, a DNN model should still detect the target object from the image. An inference that violates any of these MRs is regarded as an unreliable inference.

We evaluated the MRs on YOLOv3, a widely-adopted object detection model. The datasets used in our evaluation consist of a hand detection dataset collected from an industrial company (named "the company' dataset") and a publicly-available hand detection dataset (named "EgoHands dataset"). Our evaluation shows that the MRs are effective in identifying unreliable inferences: the MRs together detected 1057 and 1436 unreliable inferences from YOLOv3's inferences on the company's dataset and EgoHands dataset, respectively. For the unreliable inferences detected by the MRs, we identify the images that led to these, and then use the identified images to retrain YOLOv3. After retraining, YOLOv3's inference accuracy on the company's dataset and EgoHands dataset increased by 6.44% and 3.55%, respectively.

1 INTRODUCTION

Deep Neural Network (DNN) models have been widely deployed for object detection tasks [14, 28]. These models outperform classic algorithms, such as SIFT+FV [33] and Sparse Coding [16], in terms of object detection accuracy (i.e., the proportion of target objects being successfully detected by a model from input images). Recent studies have raised concerns about other properties of such models, including reliability [22, 30, 35], fairness [1, 39, 45], robustness [4]. To help detect the inappropriate behaviors of DNN models, various testing techniques [8, 9, 18, 25, 38, 41, 44] have been proposed. For instance, Pei et al. [25] proposed an optimization strategy to generate adversarial test inputs for image classification. Dwarakanath et al. [9] leveraged metamorphic testing to detect bugs in model implementations.

These techniques, however, do not consider a key property when evaluating a DNN-based object detection, that is, whether the inferences made by the model are based on the features encoded from the target objects or the features encoded from these objects' background. We refer to the former features as *object-relevant features*, the latter as *object-irrelevant features*, and the property as *object-relevancy property*. Intuitively, a reliable inference made by a DNN model should be mostly based on object-relevant features instead of object-irrelevant features.

For instance, let us assume that the *mouse* shown on Figure 1a is the *target object*. The object-relevant features are encoded from the mouse, while the object-irrelevant features are encoded from the rest of this image. Let us further assume that a model detected a mouse on Figure 1a. This inference is reliable on the condition that it is made mostly based on the object-relevant features instead of the object-irrelevant features. If the inference is majorly based on the object-irrelevant features but not the object-relevant features, the same model is likely to detect a mouse on Figure 1b, because this image has the same object-irrelevant features as Figure 1a. However, in reality, "mouse" should not be

* Corresponding author.

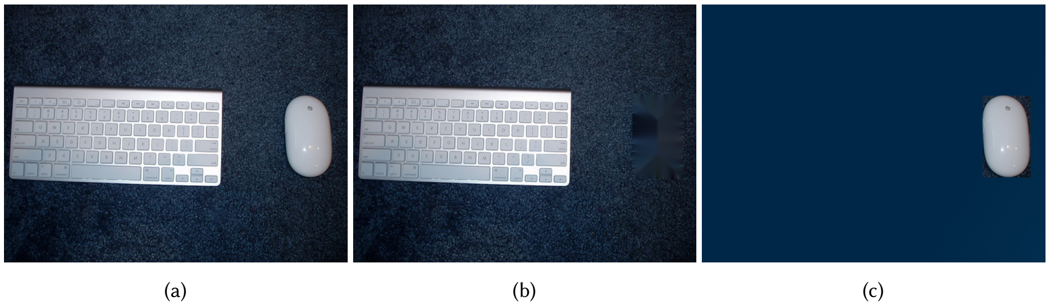


Fig. 1. (a): The Original Image. (b): Object (Mouse) Corrupting Mutation. (c): Object (Mouse) Preserving Mutation.

detected because Figure 1b does not have any "mouse" object. Further, the model is also likely to detect an object other than "mouse" on Figure 1c, since this image does not have the object-irrelevant features of Figure 1a. Such an inference result does not make sense because Figure 1c clearly has the target object *mouse*.

Due to their stochastic nature, many DNN models do not necessarily make inferences based on object-relevant features, which may lead to various problems. For instance, a recent study showed that an animal classification model would classify any image with bright backgrounds as "wolf", regardless of the objects in the image [30]. This raises the concern of reliability and overfitting for this model [19, 30]. Another work showed that attackers could inject a backdoor trigger, such as a yellow square in an image's background, to a deep neural network (DNN) model [11]. Such a backdoor exploits a DNN model which makes inferences based on object-irrelevant features (e.g., yellow square in the background), so that the DNN model can still detect an object from an image, even if the object does not exist on the image. Thus, the DNN model is not robust and can cause catastrophic consequences when being deployed in mission-critical applications. Based on the above analysis, we conjecture that the violation of the object-relevancy property might be the root cause of many issues in DNN models, including but not limited to the aforementioned ones. Therefore, it is important to develop effective techniques to assess DNN models' inference results from the perspective of object relevancy, so as to help improve the trustworthiness of the models.

Validating DNN models' inference results with respect to object relevancy is challenging. It is well-known that DNN models behave as black boxes [25, 30]. Their logic is learned from data and represented as model structures and weight values. It is non-trivial for human beings to examine the inference process of such models and check what kind of features determines the inference results. Some existing techniques [30, 34] try to explain the inferences for individual input. However, these techniques still require manual efforts to make the final assessment for each input due to the lack of test oracles. In contrast, in our work, we first try to generate both test inputs and test oracles for DNN models, and then leverage them to identify unreliable inferences that violate the object-relevancy property automatically. However, generating test oracles is a long-standing challenge for software testing [3], especially in the testing of the deep learning systems [23, 25, 26, 38], where the expected probability outputted from DNN models is unknown.

To tackle these challenges, we resort to metamorphic testing [5], which has been popularly leveraged to test DNN models [8, 9, 41, 44]. Specifically, we propose two metamorphic relations (MRs) to quantitatively assess a model's inferences from the perspective of object relevancy as follows:

- **MR-1** An image mutated by corrupting the features of target object(s) should lead to a different object detection result.

- **MR-2** An image mutated by not corrupting the features of target object(s) should lead to the same object detection result.

The two metamorphic relations will be formally defined in Section 3. For the purpose of metamorphic testing, we designed image mutation operations to generate test inputs with respect to the two relations. Applying these operations to a given image allows us to check if the pair of the original inference and the inference on a generated mutant satisfies the metamorphic relations. Violations of such relations will be deemed as the indication of unreliable inferences. We note that applying metamorphic testing to evaluate DNN-based image processing tasks is not new. However, existing work [38, 44] mutates the whole image (e.g., blurring or rotating) to test the model robustness. In comparison, our MRs focus on object-relevant/irrelevant features in one input image, and hence our image mutation is regional, semantic and more targeted. Besides, our goal is to assess whether an inference violates the object-relevancy property, which is a new property proposed by us.

Note that the metamorphic testing approach proposed in this technical report is a refinement of the metamorphic testing approach proposed in our EmSE'21 paper [37]. The approach in [37] is originally designed for image classification tasks, and we refine its mutation operations and the MRs to formulate a new metamorphic testing approach for object detection tasks (as described in this technical report).

To validate the effectiveness of our proposed approach, we applied it to YOLOv3 [29], a popular object detection DNN model, trained on a company's hand dataset and a publicly-available hand detection dataset (named EgoHands dataset [2]). The evaluation results show that for the YOLOv3 model, our approach detected 1057 and 1436 unreliable inferences from YOLOv3's inferences. We further repaired YOLOv3 with the image data, which caused the unreliable inferences identified by MR1 or MR2. After repairing, YOLOv3's inference accuracy increased by 6.44% and 3.55%, respectively.

2 PRELIMINARIES

2.1 Metamorphic Testing

Metamorphic testing [5, 6] was proposed to address the test oracle problem. It works in two steps. First, it constructs a new set of test inputs (called *follow-up inputs*) from a given set of test inputs (called *source inputs*) based on some properties that should be satisfied by the program under test. Second, it checks whether the program outputs based on the source inputs and the ones based on the follow-up inputs satisfy certain desirable properties, known as *metamorphic relations* (MRs).

For example, let us suppose p is a program implementing the sin () function. We know that the equation $\sin(\pi + x) = -\sin(x)$ holds for any numeric value x. Leveraging this knowledge, we can apply metamorphic testing to p as follows. Given a set of source inputs $I_s = \{i_1, i_2, \ldots, i_n\}$, we first construct a set of follow-up inputs $I_f = \left\{i'_1, i'_2, \ldots, i'_n\right\}$, where $i'_j = \pi + i_j$, $j \in [1, n]$. Then, we check whether the metamorphic relation $\forall j \in [1, n], p(i_j) = -p(i'_j)$ holds. A violation of it indicates the presence of faults in p.

2.2 DNN-based Object Detection

Object detection is a key application of Deep Learning. Its objective is to identify a region on an image which contains a target object. Specifically, an object detection model identifies the coordinates of a rectangle (a.k.a., bounding box) on an image which is completely overlapped with a target object. Figure 2 shows an example of a bounding box, which is the red dash-lined rectangle completely overlaps with the target object "rabbit". Figure 3 shows the format of a bounding box, which consists of four x-y coordinates representing a bounding box's four corners on an image.

Popular DNN models for object detection include YOLOv3 [29], Mask R-CNN [13] and so on. The performance of these models is mostly evaluated based on mean Average Precision (mAP): simply speaking, mAP is a metric which multiplies *precision* and *recall* of a model's inference results: *precision* essentially measures how many objects detected by a model are target objects (instead of irrelevant objects), and *recall* essentially measures how many target objects are successfully detected by a model (rather than being missed by the model). More details will be discussed in the following paragraphs. Hence, a higher mAP value implies a model obtains a higher *precision* and *recall*, and thus is more accurate.

Since the formal definition of mAP is complicated and abstract, we illustrate the idea of mAP by describing the workflow of how mAP is computed. Given a set of images (denoted by *I*) and an object detection model (denoted by $\mathcal{M}$), for each image $i \in I$, $\mathcal{M}(i)$ outputs the coordinates of a bounding box which can cover a targeted object on i (let $\{(x_1^i, y_1^i), ..., (x_4^i, y_4^i)\}$ denote the coordinates of the bounding box's four corners). Suppose for each $i \in I$, the ground-truth coordinates of a bounding box are $\{(x_1'^i, y_1'^i), ..., (x_4'^i, y_4'^i)\}$, to compute mAP, the first step is to compute *Intersection over Union (IoU)*, which is the proportion of area that $\{(x_1^i, y_1^i), ..., (x_4^i, y_4^i)\}$ and $\{(x_1'^i, y_1'^i), ..., (x_4'^i, y_4'^i)\}$ are overlapped. Specifically, let R denote the region on an image indicated by $\{(x_1^i, y_1^i), ..., (x_4^i, y_4^i)\}$ is A_i, and let R' denote the region on an image indicated by $\{(x_1'^i, y_1'^i), ..., (x_4'^i, y_4'^i)\}$, *IoU* is computed as the ratio whose numerator is the overlapped area of R_i and R_i', and the denominator is the total area (i.e., overlapped area plus non-overlapped area) of R_i and R_i':

$$IoU = \frac{Area(R_i \cap R_i')}{Area(R_i \cup R_i')}$$

Hence, IoU equals 1.0 means R_i and R_i' are completely overlapped, which implies $\mathcal{M}(i)$ is the completely accurate because $\mathcal{M}(i)$ equals the ground truth. On the other hand, IOU equals 0.0 means $\mathcal{M}(i)$ is the least accurate, because $\mathcal{M}(i)$ is totally different from the ground truth.

Since IoU rarely equals 1.0 in practice, a threshold value (e.g., 0.7) will be set manually in order to assess whether IoU is sufficiently large. For *IoU* larger than the threshold, $\mathcal{M}$ will be regarded as successful in accurately locating a target object on i (i.e., *True Positive, TP*). For *IoU* less than the threshold while a target object does not exist on an image, $\mathcal{M}$ will be regarded as failed to identify the absence of a target object on i (i.e., *False Positive, FP*). For *IoU* larger than the threshold while a target object does not exist on an image, $\mathcal{M}$ will be regarded as successful in identifying the absence of a target object on i (i.e., *True Negative, TN*). For *IoU* less than the threshold while a target object does exist on an image, $\mathcal{M}$ will be regarded as failed to identify the presence of a target object on i (i.e., *False Negative, FN*).

Hence, for $i \in I$, $\mathcal{M}(i)$ is either a TP, FP, TN or FN. After the model performs inferences for all $i \in I$, *precision* and *recall* can be computed. Following the conventional definitions, the metric *precision* is defined as

$$precision = \frac{\text{Total number of TP}}{\text{Total number of TP } + \text{ Total number of FP}}$$

the metric *recall* is defined as

$$recall = \frac{\text{Total number of TP}}{\text{Total number of TP } + \text{ Total number of FN}}$$

Roughly speaking, mAP is computed as

$$mAP = \frac{1}{|C|} \Sigma_{c \in C} \text{precision} * \text{recall}$$

where C is number of types of target object(s) (e.g., keyboard and mouse are two different types of object in Figure 1a). Hence, higher mAP implies a model makes more correct inference results.

2.3 Data Mutation

Data mutation is a popular technique to generate a new dataset by mutating an original dataset. It is widely-adopted in DL training in order to address an over-fitting problem of a DL model. There are many different ways to mutate image data. For instance, an image can be mutated by changing the image's pixel values, or rotating, shearing and resizing the image.

In our approach, we adopt two popular data augmentation techniques: *random erase* [46] and *Gaussian noise* [36]. **Random erase** essentially mutates an image by converting specific pixel values of the image into random pixel values. In other words, random erase corrupts an image's features (the features to be corrupted are usually specified manually in the format of a bounding box). Figure 4 shows examples of our mutation operators.

In our approach, we apply random erase to mutate an image in order to check whether a model's inference result on the mutated image would violate MR1 (which essentially assures that when object-relevant features are corrupted, a model should no longer detect a target object, §4.4.1). Specifically, we erase an image's features within a target object's bounding box (an annotation of a bounding box is usually available in a training dataset). Since a bounding box is usually a rectangle, a bounding box inevitably includes not only object-relevant features but also object-irrelevant features, especially when a target object is non-rectangular (e.g., hands).

To mitigate the repercussion of accidentally erasing object-irrelevant features, we specify the proportion of features being erased in a bounding box (we name such a proportion as "mutation ratio"). The range of mutation ratio's value is from 0.0 to 1.0. Mutation ratio equals 0.0 means that no features in a bounding box will be erased. Mutation ratio equals 0.5 means half of the features in the bounding box will be erased. Note that features chosen to be erased are those closest to the center of a bounding box, because those features are likely the object-relevant features. Mutation ratio equals 1.0 implies all features in a bounding box will be erased. In our evaluation, we empirically search for an optimal value of mutation ration (§4.5).

Gaussian noise adds random noise that follows Gaussian distribution to the pixel values of an image. The noise intensity is determined by the variance of Gaussian distribution (while the mean of Gaussian noise should always be zero). Greater variance value of a Gaussian distribution implies greater noise intensity. Note that Gaussian noise can be used to corrupt and preserve features of an image, depending on the noise intensity. Specifically, applying Gaussian noise of large variance value (i.e., high noise intensity) corrupts an image's features, because the pixel values which encode the features would be drastically randomized. On the other hand, applying Gaussian noise of small variance value (i.e., low noise intensity) preserves important features of an image, because subtle mutation on an image will not corrupt object-relevant features but object-irrelevant features (object-relevant features are usually more robust to noise than object-irrelevant features [20].

Therefore, in our approach, we apply Gaussian noise of low noise intensity to target object, and Gaussian noise of high intensity to non-target object. By doing so, we preserve the features of target object and corrupt the feature of non-target object, so that we can assess whether a model's inference result would violate MR2 (which essentially assures that when object-irrelevant features are corrupted, a model still detects a target object, see §4.4.2). In our evaluation, we empirically search for an optimal value of variance (§4.5). Besides a variance value, the Gaussian noise mutation operator also has the mutation ratio parameter, similar to random erase.

3 OBJECT-RELEVANT METAMORPHIC RELATIONS

With the aim to identify an unreliable inference made by a DNN model, we are motivated to propose two metamorphic relations (MRs) as mentioned in Section 1. This section presents the details of these MRs, starting with a motivating example. Specifically, we follow a common metamorphic testing framework to define the two metamorphic relations [5, 6]. Following the formulation in §2.2, let $\mathcal{M}(i)$ and $\mathcal{M}(i')$ respectively denote the inferences made by a DNN model $\mathcal{M}$ on an input image i and follow-up image i' which is generated by mutating i (§2.3). Let $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'))$ denote the distance between $\mathcal{M}(i)$ and $\mathcal{M}(i')$.

3.1 MR-1

Motivating Example-1 Given a source input as shown in Figure 1a, let us assume a model correctly locates the "mouse" object on the figure (i.e., the model outputs the coordinates of a bounding box close to the ground-truth bounding box, see §2.2). A follow-up input is constructed by corrupting the object *mouse*, as shown in Figure 1b. After feeding the follow-up input into the previous model, one of the following two cases could happen. First, it is possible that the model still correctly locates the mouse object on Figure 1b. Such a situation indicates that the inference on the source input is not based on the object(*mouse*)-relevant features. If it is based on the object(*mouse*)-relevant features, it does not make sense that the model can still locate the mouse object on the figure when there is no such an object(*mouse*). Second, it is possible that the model does not detect a mouse object at all on Figure 1b. It implies that the inference on the source input is based on the object(*mouse*)-relevant features. This situation is in line with human expectations. Since the object has been removed or corrupted, humans are likely to identify the absence of the object. Motivated by the above example, we proposed the following MR-1. In the first situation aforementioned, the MR-1 is violated while in the second situation, MR-1 is satisfied.

MR-1 An image mutated by corrupting the features of target object(s) should lead to a different object detection result.

Relation Formulation of MR-1 Let i'_c be a follow-up input constructed from a source input i for a model $\mathcal{M}$ by $\underline{c}$orrupting the target object but preserving its background. We consider such a mutation as *object-corrupting*. An example of object-corrupting mutation is shown in Figure 1a (source input) and Figure 1b (follow-up input). MR-1 mandates that $\mathcal{M}(i)$ and $\mathcal{M}(i'_c)$ should satisfy the relation: $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_c)) \geq \Delta_c$. Here D takes one factor of $\mathcal{M}(i)$ and $\mathcal{M}(i'_c)$ into consideration, i.e., the coordinates of a bounding box. The detailed definition of D for MR-1 is introduced in Section 4.4.1. Δ_c denotes a threshold for the distance between two inference results made by a model under metamorphic testing using object-corrupting mutations.

Explanation of MR-1 If an inference made by a specific model is based on object-relevant features, after object-corrupting mutations, the new inference results should be affected since those object-relevant features have been corrupted, and thus those features cannot be further utilized by the model anymore. Such effects could cause two consequences. First, the model can still make the same inference as the inference of the original input. Second, the model cannot make the same inference as the inference of the original input if the corruption is very severe. Consequently, the new inference result should be different from the original one.

3.2 MR-2

Motivating Example-2 Given a source input shown in Figure 1a, assume a model correctly locates "mouse" on the image. A follow-up input is constructed by preserving the object, as shown in Figure 1c. After feeding the follow-up input into the previous model, one of the following two cases could happen. First, the model cannot detect "mouse" anymore. It indicates that the inference on the source input is not based on the object(*mouse*)-relevant features. Since the object *mouse* is still in the input, if the inference on the source input is based on the object(*mouse*)-relevant features, the model should still detect and locate "mouse" correctly. Second, the inference on follow-up input remains the same as the source input. It implies that the inference on the source input is based on the object(*mouse*)-relevant features. When the object-relevant features are preserved, the model can leverage them to make the correct inference. Such a situation is in line with human expectations. Motivated by this example, we propose the following MR-2. In the above example, MR-2 is violated in the first situation and satisfied in the second situation.

MR-2 An image mutated by not corrupting the features of target object(s) should lead to the same object detection result.

Relation Formulation of MR-2 Let i'_p be a follow-up input constructed from a source input i for a model $\mathcal{M}$ by preserving the target object(s) but mutating the other parts. We consider such a mutation *object-preserving*. An example of object-preserving mutation is shown in Figure 1a (source input) and Figure 1c (follow-up input). MR-2 mandates that $\mathcal{M}(i)$ and $\mathcal{M}(i'_p)$ should satisfy the relation: $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_p)) \leq \Delta_p$. Here, Δ_p denotes a threshold for the distance between two inference results made by a model under metamorphic testing using object-preserving mutations. The detailed definition of D for MR-2 is introduced in Section 4.4.2.

Explanation of MR-2 If an inference made by a specific model is based on object-relevant features, after object-preserving mutations, the new inference result should remain the same, since the object-relevant features are preserved and a reliable model should be able to make the same prediction using these features.

4 APPROACH

We present our metamorphic testing approach in this section, starting with an overview, followed by an explanation of each stage.

4.1 Overview

Figure 2 shows an overview of our approach, including the following three stages:

① **Object-Relevant Feature Identification** Given an inference to be examined, we regard its input image as the source input. We semantically divide the input into two parts, a *target-object region* and a *background region*. The *target-object region* is where the target object(s) is located and where the object-relevant features are encoded. The *background region* is where the object-irrelevant features are encoded.

② **Follow-up Inputs Construction** Mutation functions are leveraged to generate follow-up inputs from the source inputs, based on the proposed metamorphic relations. Specifically, these mutation functions will corrupt or preserve the object-relevant features in the source input. The corresponding testing oracles will also be generated based on the metamorphic relations.

③ **Metamorphic Relation Validation** We validate if the distance between the inference result of a source input and the inferences of its follow-up inputs violates the test oracles. If so, the inference of the source input is flagged as an *unreliable inference*, which means this inference is made mainly based on object-irrelevant features.

④ **Model Repairing** After the *unreliable inferences* are detected, we further propose a model repairing strategy to enhance the target DNN model in learning object-relevant features. Specifically, we augment the original training data by adding these *unreliable inferences* with corresponding labels. The augmented set will be used to retrain the DNN model.

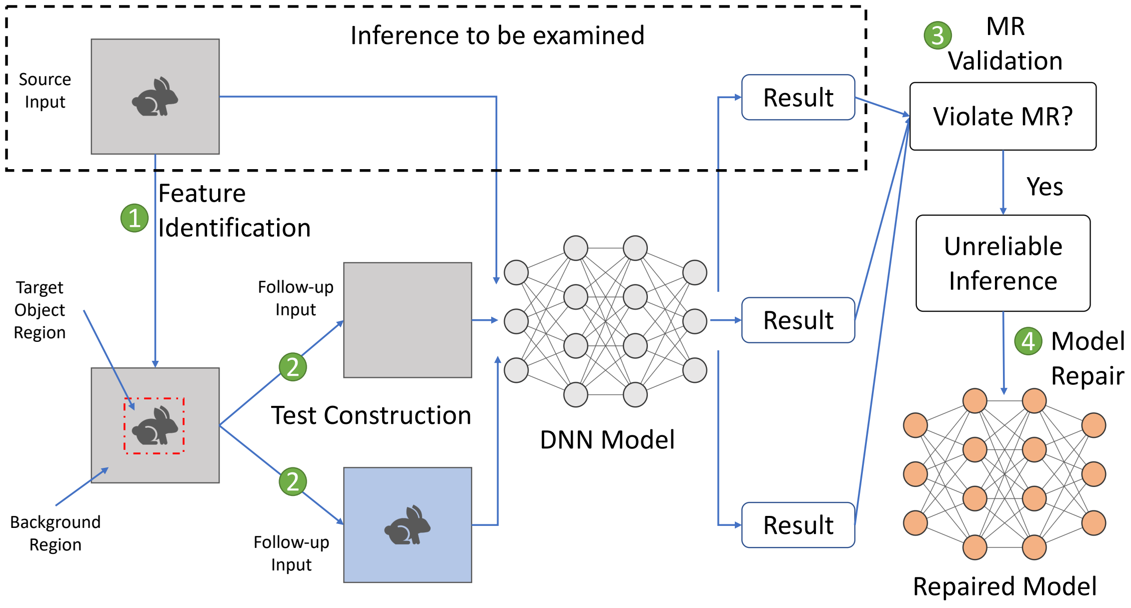
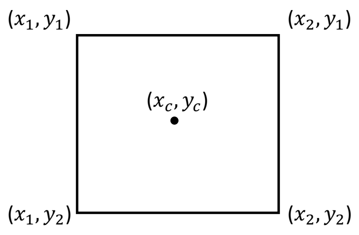


Fig. 2. The Overview of Our Metamorphic Testing Approach



Annotation Format:

[class_id x y width height]

$$x = \frac{x_c}{image\ width} \qquad width = \frac{box\ width}{image\ width}$$

$$y = \frac{y_c}{image\ height} \qquad height = \frac{box\ height}{image\ height}$$

Fig. 3. The Annotation In Bounding Box Format

4.2 Object-Relevant Feature Identification

Pixels where the target object(s) reside are treated as the *target-object region* and the others are regarded as the *background region.* The annotations of the target objects are usually available in the dataset of object detection tasks (e.g., EgoHands hand dataset [2]), and the annotations are usually in the format of a *bounding box* (§2.2). Some datasets, such as COCO, annotate the object using the object mask, which draws the boundary of each object with a finer granularity. These annotations provide the exact target-object region that does not contain any pixels belonging to the background region.

Both annotation formats can be used in our approach. If the annotations are provided as bounding boxes (as is shown in Figure 3), we regard the region of the bounding boxes as the target-object region. Although the target-object region could contain some pixels that do not belong to the target object(s), the majority of the region represents the target object. If the annotations are object masks, we regard the region covered by the object masks as the target-object region.

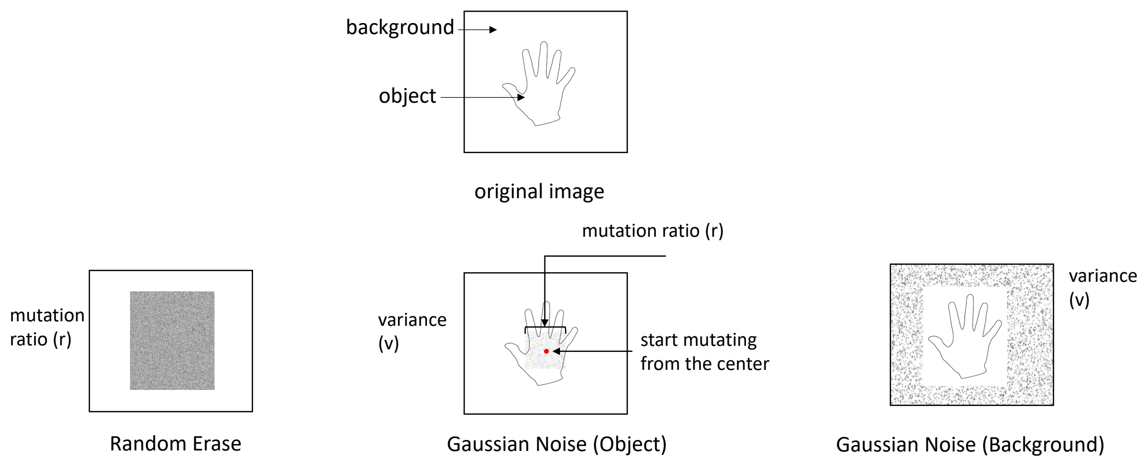


Fig. 4. Illustration of Input Construction Strategies

4.3 Follow-up Inputs Construction

We generate the follow-up inputs by semantically corrupting or preserving the object-relevant features of a source image using the two aforementioned image mutations: *object-corrupting* mutation and *object-preserving* mutation. As discussed in §2.3, our approach adopts two popular mutation operators: random erase and Gaussian noise. In our approach, random erase is an *object-corrupting* mutation by adding random noise to replace the image. In contrast, gaussian noise is an *object-preserving* mutation, and it can either add gaussian noise to the object or to the background without corrupting object-relevant features. Specifically, in our approach, random erase is designed for MR-1, gaussian noise (object), and gaussian noise (background) are designed for MR-2. Here we provide examples for each input construction strategy (as is shown in Figure 4:

Random Erase randomizes pixel values inside a bounding box (i.e., erasing object-relevant features). A parameter *mutation ratio* needs to be specified in order to control the proportion of features inside a bounding box to be erased.

Gaussian Noise (Object) applies small Gaussian noise to the object region. The amount of Gaussian noise added is determined by the variance (v); increasing v will result in more perturbations. The size of the affected region is determined by the mutation ratio (r), which ranges from 0.0 to 1.0. Increasing r will increase the size of the affected region; when setting r to be 1, gaussian noise will be added into the whole object region.

Gaussian Noise (Background) applies Gaussian noise to the background region. The strength of gaussian noise is controlled by the variance (v).

4.4 Metamorphic Relation Validation

In this subsection, we introduce the metamorphic relation validation process. Note that in our experiments, for each source input i, we generate follow-up inputs i' (s) using corresponding input construction strategies.

Note that for object detection model, the inference result is a bounding box containing the target object. We use a function T to denote whether the generated bounding box indeed contains the expected object:

$$\mathcal{T}(\mathcal{M}(i)) = \begin{cases} 1, & \text{if } IoU(\mathcal{M}(i), \text{bounding box}) \geq \Delta_c \\ 0, & \text{otherwise} \end{cases}$$

Specifically, for object inside the image i, we will compare the IoU between the bounding box generated by the DNN model and the ground truth bounding box manually annotated. IoU is a function which computes the overlapped area between $\mathcal{M}(i)$ and $\mathcal{M}(i'_c)$ (see §2.2). The default value of Δ_c is chosen as 0.3, because it is also the default IoU threshold value adopted by YOLOv3. For Δ_c larger than 0.3, the inference result of YOLOv3 is considered significantly different from the ground truth (i.e., a target object is not detected by YOLOv3).

Then we will validate the MRs, if any MR is violated, we consider an unreliable inference is detected. We will regard $\mathcal{M}(i)$ as an unreliable inference if and only if MR-1 is violated at least two out of three times. The same strategy is applied to MR-2.

4.4.1 Validation of MR-1. e.g., **MR-1** An image mutated by corrupting the features of target object(s) should lead to a different object detection result.

Here we use the same notation as Section 3. We define the distance function $\mathcal{D}$ as follows:

$$\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_c)) = \begin{cases} 1, & \text{if } \mathcal{T}(\mathcal{M}(i)) = \mathcal{T}(\mathcal{M}(i'_c)) \\ 0, & \text{otherwise} \end{cases}$$

As discussed in §2.3, i'_c is generated from i by applying random erase on i. The features on i being erased are the features within a bounding box of a target object (§4.2). if $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_c)) = 1$, i.e., the label of the inference on the source input $l_{\mathcal{M}(i)}$ is different from the one of the inference on the follow-up input $l_{\mathcal{M}(i'_c)}$, the MR-1 is satisfied. Otherwise, if $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_c)) = 0$, i.e., $l_{\mathcal{M}(i)}$ and $l_{\mathcal{M}(i'_c)}$ are almost the same, it implies that after corrupting the object-relevant features in the source input, the model can still detect the target object. In other words, the examined inference $\mathcal{M}(i)$ is made based on features irrelevant to the objects. This conclusion violates our MR-1, and thus $\mathcal{M}(i)$ is labeled as an unreliable inference.

4.4.2 Validation of MR-2. **MR-2** An image mutated by not corrupting the features of target object(s) should lead to the same object detection result.

For object detection, we define $\mathcal{D}$ as follows:

$$\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_p)) = \begin{cases} 1, & \text{otherwise} \\ 0, & \text{if } \mathcal{T}(\mathcal{M}(i)) = \mathcal{T}(\mathcal{M}(i'_p)) \end{cases}$$

As discussed in §2.3, i'_p can be generated from i by applying Gaussian noise of low noise intensity on object-relevant features of i, or applying Gaussian noise of high noise intensity on object-irrelevant features (e.g., background). Note that to apply Gaussian noise on i for preserving object-relevant features (instead of corrupting them), the intensity of the noise has to be low enough. On the other hand, to corrupt object-irrelevant features on i, the noise intensity has to be large enough. In our experiment, we empirically search for an optimal variance value: we sample a small set of image data, and experiment with different variance values which allow an object detection model to achieve optimal performance (e.g., allow a model to have the highest inference accuracy after retraining).

If $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_p)) = 0$, it means that the label of the inference on the source input $l_{\mathcal{M}(i)}$ is different from the one of the inference on the follow-up input $l_{\mathcal{M}(i'_p)}$. In other words, after preserving the features of the target object and corrupting the remaining features in the source input, the model detects a different object. This conclusion is opposite to our MR-2, and thus the examined inference $\mathcal{M}(i)$ is labeled as an unreliable inference. If $\mathcal{D}(\mathcal{M}(i),\ \mathcal{M}(i'_p)) = 1$, it implies that after preserving the features of the target object and corrupting the others, the model still classifies the input into the same label as the one of the source input. This result is in line with our MR-2, and thus the examined inference will not be labeled as an unreliable inference by us.

4.5 Model Repairing

The mutated images i'_c and i'_p which result in violations of MR1 or MR2 can be used to retrain (repair) a model, in order to reduce the model's unreliable inference (i.e., increase the model's inference accuracy). Specifically, retraining a model with those i'_c and i'_p allows the model to be more precise in distinguishing object-relevant features and object-irrelevant features. Figure 5 shows an end-to-end workflow of model repairing, which is divided into three phases: Model Training, MR Validation and Model Retraining. The first two phases (Model Training and MR Validation) are described in §4.2 and §4.4. Essentially, these two phases take a pre-trained object detection model and the model's training dataset as inputs, and output all the i'_c and i'_p which result in violations of MR1 or MR2.

In the Model Retraining phase, the first step is to form an augmented dataset (denoted by S), by combining the original training dataset and the i'_c and i'_p outputted by the first two phases. Typically, the proportion of i'_c and i'_p in S accounts for 1% to 27% , depending on the value of mutation ratio or variance (§2.3). In particular, the higher the value of mutation ratio or variance, more object-relevant features on i will be corrupted. Hence, the generated i'_c or i'_p is more likely to violate the MRs, and thus more i'_c or i'_p will be passed to the augmented dataset S. Table 3 and Table 4 shows the correlation between mutation ratio values and the number of i'_c violated the MRs.

Note that the proportion of i'_c and i'_p in S has to be precisely controlled: too many mutated images in S can cause a model to have lower inference accuracy after retraining, because too many mutated images could drastically change the features' distribution of the original training dataset. On the other hand, too few mutated images could result in negligible effectiveness of model retraining [7]. Hence, to avoid the proportion of i'_c and i'_p in S being too large or too small, we dynamically search for an optimal value of mutation ratio or variance. The details will be discussed in the following paragraphs.

The second step of Model Retraining is to train the pre-trained object detection model with S. In order to make a model to achieve its highest inference accuracy after retraining, the value of mutation ratio or variance has to be set carefully. However, choosing the right value for mutation ratio or variance is challenging, because the search space of the value is usually huge (e.g., the value of variance can be any positive real number), while a suboptimal value of mutation ratio or variance could decrease a model's inference accuracy after the model is retrained. For instance, setting a mutation value too large not only causes S to contain too many mutated data as discussed in the previous paragraphs, but also causes object-irrelevant features in i to be mistakenly erased (§2.3).

We address this challenge by sampling. Our rationale is that a small sample of training dataset already consists of most object-relevant or irrelevant features encoded in the training dataset (this phenomenon has been observed in many previous works [7, 15, 36]. Hence, to investigate whether a mutation approach can effectively preserve nearly-all possible object-relevant features or corrupt only object-irrelevant features encoded in a training dataset, we assess a mutation approach on a small sample of training dataset.

Specifically, we first sample a small set of images from the original training dataset. Then, we automatically test different values of mutation ratio and variance, to see which combination of mutation ratio value and variance value achieves the optimal performance (i.e., allow a model to achieve the highest inference accuracy after the model is

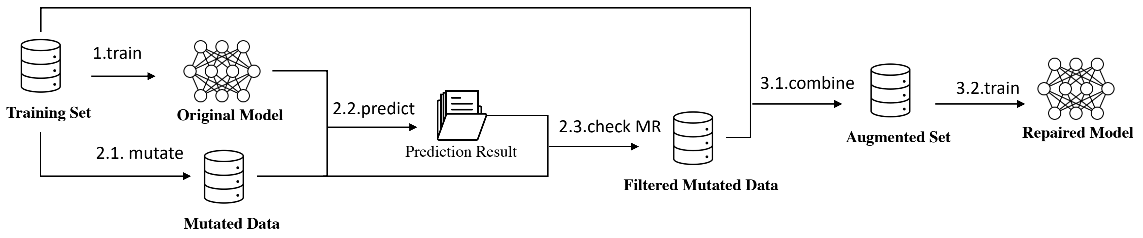


Fig. 5. The Overview of our Model Repairing Approach

Table 1. Introduction of Two Datasets

	Total Images	Training Images/Testing Images	Total Hands	Average Hands/Images	Images With No Hands	Average Hand Size
Company	10000	8000/2000	16994	1.6994	3140	TBA
EgoHands	4800	3840/480	10256	2.1367	94	TBA

retrained with the corresponding i'_c and i'_p). The values of mutation ratio and variance which achieve the optimal performance on the sample, will be adopted to mutate the entire original training dataset (i.e., forming S). After S is formed, the object detection model is trained with S, until the model's training accuracy is saturated (i.e., no longer increases).

5 EVALUATION

In this section, we evaluate our approach from the perspective of effectiveness. First, we investigate the effectiveness of our proposed approach to see whether it can detect unreliable inferences that are made based on object-irrelevant features. Specifically, we measure the total number of unreliable inferences detected by us on the famous object detection model: YOLOv3 and two hand datasets: the company's dataset and EgoHands dataset. We aim to answer the following question:

RQ1 How effectively can our approach detect unreliable inference for object detection model?

Further, we investigate how effective can these detected unreliable inferences help improve model's accuracy. Specifically, we applied our designed repair framework and detected unreliable inferences to retrain the object detection model: YOLOv3. Then, we compare the mAP achieved by original YOLOv3 model and the repaired version to measure the effectiveness of our approach in terms of repairing the DNN model. These results will help us to answer the following question:

RQ2 How effective is our repair methodology in terms of improving DNN model's accuracy?

Our experiments were conducted on two datasets, one dataset is collected from the industrial company (we refer to it as the company dataset.), and the other one is the EgoHands hand dataset [2]. Both datasets contain the hand label for the object detection task. The details of these two datasets are presented in Table 1. Specifically, since the company's dataset is captured using the control monitor, the hands are usually small while the backgrounds are complicated and various. The EgoHands dataset is collected by extracting frames from 48 google glass videos of complex, first-person interactions between two people. Compared with the company's dataset, the size of hands in EgoHands dataset is usually large, but some hands are not complete. We randomly split 8,000 out of 10,000 as the training set while the remaining 2,000 images as the testing set; Same as existing works [2] we take the training set with 3,840 for model training and 480 images for model testing. We choose YOLOv3 model as the target model to be evaluated and repaired.

Table 2. Performance of YoloV3 on Two Datasets

	mAP@0.5 (%)	Precision	Recall	Average IoU (%)
company	38.62	0.73	0.26	49.93
egohands	85.26	0.97	0.65	73.92

YOLOv3 is a wide-known deep learning model utilizing CNN to provide real-time object detection. In particular, we adapt the YOLOv3 model to hand detection task as our test subject. The reason to choose YOLOv3 is that it is a popularly used object detection model and some works have already adapted it for hand detection.

5.1 Effectiveness in Detecting Unreliable Inferences

In order to evaluate whether our metamorphic testing approach can effectively identify unreliable inferences, we train the YOLOv3 model on the training set of the two dataset, respectively. Then we apply our mutation operators to the training set and further use our MR to identify the unreliable inference. Given the pre-trained model, we apply our mutation operators to the training images and inference the mutated images with the pre-trained model. An unreliable inference is detected if the inference result on mutated images violates the corresponding MR.

5.1.1 Experiment Setup. **Model Training** For each dataset, we train the YOLOv3 model on its training set following the default hyper-parameters declared by its official website [1]. For both dataset, we set the training iteration to be 3000, where the performance of YOLOv3 stops increasing. Following the default training scheme, we save a checkpoint during every 1000 iterations and consider the best mAP on the test set as the final model. The performance of YOLOv3 on these two datasets is presented in Table 2.

Metamorphic Relation and Image Mutations We use MR2 as the criteria to evaluate the YOLOv3 model. Specifically, to use MR-2, we apply two types of mutation operators to mutate images while preserving object-relevant features: 1) applying gaussian noise on the background, and 2) applying gaussian noise on the object (see details in Section 4.3). For gaussian noise on object, we set the gaussian variance to be 160 while changing the mutation ratio from 0.1 to 0.9. For gaussian noise on background, we apply the gaussian noise to the whole background and change the mutation ratio from 2 to 160.

Unreliable Inference Detection For each training image, our tool will apply the mutation operator to generate mutated images. We further make inferences on the original images and mutated images using the trained YOLOv3 model, respectively. Since we are using MR-2 as the criteria for unreliable inference detection, we compare the inference result between the original image and mutated image following Section 4.4; if the detection result is inconsistent, an unreliable inference is detected. Specifically, two types of unreliable inference are detected: 1) Missing Detection: the model can successfully detect the hand on the original image while failing on the mutated image; 2) Incorrect Detection: the model cannot detect the hand on the original image while success on the mutated image.

5.1.2 Results and Discussions. **Company's Dataset** The evaluation result on the company's dataset is presented in Table 3. We observe a large number of unreliable inferences that can be detected by our tool when evaluating YOLOv3 model on the company's dataset. One possible reason is that images in the company's dataset have various scenarios such as different light, different backgrounds, and different hand colors. As a result, the current YOLOv3 model fails to achieve a satisfactory result in detecting these hands (as presented in Table 2, the YOLOv3 model can

[1]https://github.com/pjreddie/darknet

Table 3. Evaluation Result On Company's Dataset

Mutation Ratio	Inference Violating MR-2		Inference Not Violating MR-2
	Missing Detection	Incorrect Detection	
0.1	317	96	5046
0.2	521	88	4975
0.3	682	100	4899
0.4	803	90	4831
0.5	891	82	4793
0.6	952	85	4777
0.7	969	82	4780
0.8	972	85	4790
0.9	951	96	4805

Table 4. Evaluation Result On EgoHands' Dataset

Variance	Inference Violating MR-2		Inference Not Violating MR-2
	Missing Detection	Incorrect Detection	
0.5	24	17	5189
1.0	46	43	5163
2.0	52	57	5157
4.0	60	108	5125
8.0	89	203	5059
16.0	127	383	4927
32.0	246	617	4706
64.0	590	655	4523
128.0	831	605	4454

only achieve 38.62 mAP with only 0.26 recall.) Specifically, we evaluate our tool under different mutation ratios, and the experiment result shows that the higher the mutation ratio is, the more unreliable inferences are detected. In particular, at least 7.6% inferences made by the YOLOv3 are unreliable inferences (when setting the mutation ratio to be 0.1), and at most 17.9% are unreliable inferences (when setting the mutation ratio to be 0.9). One possible reason is that, the more perturbations are applied to the original model, the more likely that YOLOv3 model outputs different inference result. Moreover, the total number of *Missing Detection* violations are significantly larger than the total number of *Incorrect Detection* violations, which means that after mutation, the target YOLOv3 model fails to detect the hands even though it can successfully detect them on the original model. This finding shows that YOLOv3 model depends on some hand-irrelevant features when detecting hands. Therefore, after we apply our mutation operator to corrupt these hand-irrelevant features, the existing YOLOv3 model fails to detect the variance of these hands. The experiment results indicate the ineffectiveness of YOLOv3 model in learning the hand-relevant features. One possible reason to improve its accuracy is to generate a high-quality training dataset to help learn object-relevant features.

EgoHands Dataset To mutate EgoHands dataset, we apply the gaussian noise to the background. Same as our violation detection strategy for the company's dataset, if the inference result is inconsistent between the original image and the mutated image, the violation is detected. Specifically, we evaluate our tool under different gaussian variances; the higher the gaussian variance is, the more perturbations are applied to the background. The evaluation result on EgoHands dataset is presented in Table 4. We observe that when applying small perturbations (i.e., 0.5, 1.0, 2.0), the unreliable inferences only take up a small portion (i.e, from 0.78% to 2.1%). In contrast, when increasing the perturbations (i.e., gaussian noise larger than 4.0), more unreliable inferences are detected. The observation is similar to the YOLOv3's behavior on the company's dataset, the more perturbations are applied to the original model, the more likely that YOLOv3 model outputs different inference results.

Answer to RQ1 For hand detection task, our approach is effective in identifying unreliable inferences that violate MR-2. We observe that the target YOLOv3 models trained on the company's dataset and EgoHands dataset fail to learn the hand-relevant features when performing the hand detection, which may be the bottleneck for increasing its accuracy in hand detection. By evaluating our tools under different mutation ratios, we observe that the more perturbations are applied to the original model, the more likely that model outputs different inference results.

5.2 Effectiveness in Improving DNN Model's Accuracy

This subsection evaluates our tools' effectiveness in repairing the DNN model using detected unreliable inferences. Same as RQ1, we target the YOLOv3 model trained on the two datasets: the company's dataset and EgoHands dataset, respectively. Note that we repair the YOLOv3 model by deriving an augmented training set based on the original training set. Specifically, the enhanced training set is composed of two parts: 1) original training set; 2) mutated images that expose the unreliable inference of the original YOLOv3 model. However, it is not often the case that the more mutated images we add into the enhanced training set, the better performance the repaired DL model will have. The reason is that adding a large number of mutated images may influence the original training data distribution. Largely influencing the original distribution may force the YOLOv3 model to be unable to learn the target hand-relevant features and thereby decrease the accuracy of target model. Besides, the quality of mutated images will also influence the repairing result. Therefore, we apply hyperparameter tuning to select the most suitable mutation ratios for YOLOv3 models trained on each dataset.

5.2.1 Experiment Setup. Same as RQ1, for the company's dataset, we evaluate our tool under different mutation ratios (0.1, 0.2, ..., 0.9); for the EgoHands dataset, we evaluate our tool under different gaussian noise (0.5, 1.0, 2.0, ..., 128.0). After each repair, we evaluate the repaired DNN model on the test set, which has never been used during model repairing. Same as the default setting [2], we use the mAP at IoU=0.5 as the evaluation metric to evaluate the performance of the YOLOv3 model; the higher the mAP@0.5, the better performance the DNN model has. The parameter tuning process will iterate all possible hyper-parameters and select the best that can achieve the highest mAP@0.5.

5.2.2 Results and Discussion. **Company's Dataset** As is shown in Table 5, our repair tool can improve original model's performance under most mutation ratios. In particular, it can achieve at most 6.44% improvement (setting mutation ratio to be 0.3). We observe that when the mutation ratio increase, the repairing performance decreases. One possible reason is that increasing the mutation ratio will introduce more mutated images with large gaussian noise. Although adding mutated images with large noise into the training set may improve the DNN model's robustness, too many of these mutated images may result in a training data distribution shift, thus decreasing the repairing performance. Besides, images generated using high mutation ratios are likely to be unnatural.

EgoHands Dataset The repairing result for the EgoHands dataset is presented in Table 6. Overall, our tool can improve the original model's performance under all gaussian noises. In particular, the original YOLOv3 model can be improved by at most 3.55% mAP. We also observe that when the gaussian noise increases, the repairing performance

[2]https://github.com/pjreddie/darknet

Table 5. Performance on Repairing YoloV3 Model on Company's Dataset

Mutation Ratio	mAP@0.5 (%)	Precision	Recall	TP	FP	FN	Average IoU (%)
0.0 (original model)	38.95	0.71	0.28	948	381	2436	48.44
0.1	39.86	0.70	0.30	1002	434	2382	47.76
0.2	33.64	0.78	0.20	673	194	2711	53.76
0.3	**45.39**	0.77	0.32	1093	323	2291	52.75
0.4	39.84	0.80	0.19	651	162	2733	55.02
0.5	41.11	0.79	0.23	791	216	2593	54.14
0.6	38.84	0.74	0.26	874	309	2510	51.48
0.7	42.40	0.81	0.23	772	186	2612	55.93
0.8	37.98	0.76	0.21	711	221	2673	52.83
0.9	40.59	0.79	0.25	842	230	2542	54.62

Table 6. Performance on Repairing YoloV3 Model on EgoHands Dataset

Variance	mAP@0.5 (%)	Precision	Recall	TP	FP	FN	Average IoU (%)
0.0 (original model)	85.26	0.97	0.65	991	34	523	73.92
0.5	**88.81**	0.95	0.77	1170	67	344	72.49
1.0	87.10	0.97	0.65	987	33	527	74.32
2.0	88.79	0.97	0.74	1115	37	399	74.15
4.0	88.14	0.94	0.74	1124	71	390	70.72
8.0	86.92	0.95	0.69	1038	55	476	72.38
16.0	88.44	0.97	0.73	1111	37	403	74.52
32.0	87.16	0.89	0.79	1189	142	325	67.10
64.0	87.77	0.96	0.68	1032	38	482	74.36
128.0	87.44	0.91	0.77	1159	112	355	69.52

decreases. The possible reason is that when the gaussian noise increases, more unreliable inferences will be detected (see Table 4), adding a large number of mutated images into the training set may affect the training data distribution. For instance, when setting the gaussian noise to be 128.0, our tool can detect 1436 unreliable inferences so 1436 mutated images will be added to the enhanced training set, taking up 27.2% images of the enhanced training set. In contrast, when setting the gaussian noise to be 0.5, only 41 mutated images will be appended into the enhanced training set, taking up 1.1%.

Answer to RQ2 Our repair tool can successfully repair the YOLOv3 model trained on both the company's dataset and EgoHands dataset by achieving higher mAP at 0.5 IoU. Specifically, we can repair the YOLOv3 model's mAP at 0.5 IoU by at most 6.44% on the company's dataset and repair the YOLOv3 model's mAP by at most 3.55% on the EgoHands dataset.

6 RELATED WORK

6.1 Metamorphic Testing in DNN models

Several studies have applied metamorphic testing to validate DNN models [8, 9, 38, 41, 44]. Dwarakanath et al. [9] leveraged two sets of metamorphic relations to identify faults in machine learning implementations. For example, one metamorphic relation is that the "permutation of input channels (i.e. RGB channels) for the training and test data" would not affect inference results. To validate whether a specific implementation of DNN satisfies this relation, they re-ordered the RGB channel of images in both the training set and test set. They examine the impact on the accuracy or precision of the DNN model after it is trained using the permuted dataset. Their relations treat the pixels in an image as independent units and they do not consider objects and background in the image.

Xie et al. [41] performed metamorphic testing on two machine learning algorithms: k-Nearest Neighbors and Naïve Bayes Classifier. Their work targets testing attribute-based machine learning models instead of deep learning systems. Ding et al.[8] proposed metamorphic relations for DNN at three different validation levels: system level, data set level and data item level. For example, a metamorphic relation on system level asserts that DNNs should perform better than SVM classifiers for image classification. Their technique requires retraining the systems and is inapplicable to testing pre-trained models.

Other studies [38, 44, 48] leveraged metamorphic testing to validate autonomous driving systems. DeepTest [38] designed a systematic testing approach to detecting the inconsistent behaviors of autonomous driving systems using metamorphic relation. Their relations focus on general image transformation, including scale, shear, rotation and so on. Further, DeepRoad [44] leverages Generative Adversarial Networks to improve the quality of the transformed images. Given an autonomous driving system, DeepRoad mutates the original images to simulate weather conditions such as adding fog to an image. An inconsistency is identified if a DNN model and its mutant make an inconsistent decision on an image (e.g., the difference of the steering degrees exceeds a certain threshold). Tian et al. [37] took the first step to design metamorphic relations to assess whether an inference is based on object-relevant features for DNN-based image classification models. Our approach is a refinement of the approach presented in [37].

6.2 Testing Deep Learning Systems

Besides metamorphic testing, studies have also been made to adapt other classical testing techniques for DNN models. A recent survey [43] summarizes the latest work in this direction. DeepXplore [25] proposed neuron coverage to quantify the adequacy of a testing dataset. DeepGauge [17] proposed a collection of testing criteria. TensorFuzz [24], DLFuzz [12] and DeepHunter [42] leveraged fuzz testing to facilitate the debugging process in DNN. DeepMutation [18] applied mutation testing to measure the quality of test data in DNN. Our study falls into the research direction of testing DNN systems. One of our major contributions is that we test DNN models from a new perspective, i.e., the object relevancy of inferences.

6.3 Background Dependence of Computer Vision Systems

Some existing work studied the background dependence of computer vision systems, even before the DNN becomes popular [27, 31]. Qin et al. [27] found that removing the background in street scene images can improve the performance of object recognition systems. Rosenfeld et al. [32] demonstrated that after transplanting an object from the training set to the background of another image, the state-of-the-art object detectors could fail to identity the inserted object. Later, Wang and Su [40] proposed an automated approach to test the object detectors. Their approach generates test inputs by inserting objects to another image's background. Our study focuses on image classification applications, and we conduct a large-scale empirical study to understand the problem.

6.4 Heatmap-based Testing of DNN Models

Researchers have proposed ideas of generating *HeatMaps* for DNN testing and debugging [10, 19, 21, 30, 34, 47]. These HeatMaps essentially capture the *importance* of individual neurons [19] or layers [10, 21] in a given DNN model. Based on different definitions of *importance*, these methods generate different types of HeatMaps. Some of them directly use neuron activation values, gradient values etc. for HeatMap generation [34, 47]. Others perform some extra processing on such raw data, such as calculating the Jacobian matrix or using differential analysis to extract the differences between correctly classified and misclassified samples [19]. A common drawback of such methods is that there is no standard definition of neuron/layer importance and it is hard to evaluate whether the generated HeatMaps are correct. As a result, these HeatMaps may or may not accurately reflect neuron/layer importance. Compared to their work, the effectiveness of our approach is properly evaluated.

Moreover, some HeatMap generation techniques require the intermediate information from the models and can only be applied for some specific types of models. For example, CAM [47] and GradCAM [47] requires access to the pooling layer of neural networks, which may not always be available. Different from these methods, our method does not need extra intermediate results from models and thus can be applied to any DNN-based image classification models.

7 ACKNOWLEDGMENT

This work was supported by the Hong Kong ITF (Grant No: MHP/055/19). Any opinions, findings, and conclusions in this paper are those of the authors only and do not necessarily reflect the views of our sponsors.

8 CONCLUSION

In this work, we proposed to leverage metamorphic testing to identify unreliable object detection made by DNN models based on object-irrelevant features. We proposed two metamorphic relations, from the perspective of object relevancy. We evaluated the effectiveness of our approach and showed that it achieves high precision. We applied our approach to a public dataset and a proprietary dataset collected by a company in real-life scenes. Our experiments found that the phenomenon of unreliable inferences was pervasive. The pervasiveness caused significant bias in model evaluation. We demonstrated that the identified unreliable inferences made by a model can be useful to improve the performance of the model.

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 (ESEC/FSE 2019)*. Association for Computing Machinery, New York, NY, USA, 625–635. https://doi.org/10.1145/3338906.3338937

[2] Sven Bambach, Stefan Lee, David J. Crandall, and Chen Yu. 2015. Lending A Hand: Detecting Hands and Recognizing Activities in Complex Egocentric Interactions. In *The IEEE International Conference on Computer Vision (ICCV)*.

[3] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525.

[4] Nicholas Carlini and David A. Wagner. 2017. Towards Evaluating the Robustness of Neural Networks. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*. IEEE Computer Society, 39–57. https://doi.org/10.1109/SP.2017.49

[5] Tsong Yueh Chen, Shing-Chi Cheung, and Siu Ming Yiu. 1998. *Metamorphic testing: a new approach for generating next test cases.* Technical Report HKUST-CS98-01. Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong.

[6] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, et al. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *ACM Comput. Surv.* 51, 1, Article 4 (Jan. 2018), 27 pages. https://doi.org/10.1145/3143561

[7] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. 2019. Autoaugment: Learning augmentation strategies from data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 113–123.

[8] J. Ding, X. Kang, and X. Hu. 2017. Validating a Deep Learning Framework by Metamorphic Testing. In *2017 IEEE/ACM 2nd International Workshop on Metamorphic Testing (MET)*. 28–34. https://doi.org/10.1109/MET.2017.2

[9] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, et al. 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)*. ACM, New York, NY, USA, 118–128. https://doi.org/10.1145/3213846.3213858

[10] Hazem Fahmy, Fabrizio Pastore, Mojtaba Bagherzadeh, and Lionel Briand. 2020. Supporting DNN Safety Analysis and Retraining through Heatmap-based Unsupervised Learning. arXiv:cs.SE/2002.00863

[11] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* 7 (2019), 47230–47244. https://doi.org/10.1109/ACCESS.2019.2909068

[12] Jianmin Guo, Yu Jiang, Yue Zhao, Quan Chen, and Jiaguang Sun. 2018. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 739–743. https://doi.org/10.1145/3236024.3264835

[13] 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.

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (Eds.). Curran Associates, Inc., 1097–1105. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[15] Sungbin Lim, Ildoo Kim, Taesup Kim, Chiheon Kim, and Sungwoong Kim. 2019. Fast autoaugment. *Advances in Neural Information Processing Systems* 32 (2019).

[16] Y. Lin, F. Lv, S. Zhu, M. Yang, T. Cour, et al. 2011. Large-scale image classification: Fast feature extraction and SVM training. In *CVPR 2011*. 1689–1696.

[17] Lei Ma, Felix Juefei-Xu, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, et al. 2018. DeepGauge: Multi-granularity Testing Criteria for Deep Learning Systems. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering (ASE 2018)*. ACM, New York, NY, USA, 120–131. https://doi.org/10.1145/3238147.3238202

[18] Lei Ma, Fuyuan Zhang, Jiyuan Sun, Minhui Xue, Bo Li, et al. 2018. DeepMutation: Mutation Testing of Deep Learning Systems. In *29th IEEE International Symposium on Software Reliability Engineering, ISSRE 2018, Memphis, TN, USA, October 15-18, 2018*, Sudipto Ghosh, Roberto Natella, Bojan Cukic, Robin Poston, and Nuno Laranjeiro (Eds.). IEEE Computer Society, 100–111. https://doi.org/10.1109/ISSRE.2018.00021

[19] Shiqing Ma, Yingqi Liu, Wen-Chuan Lee, Xiangyu Zhang, and Ananth Grama. 2018. MODE: Automated Neural Network Model Debugging via State Differential Analysis and Input Selection. 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)*. Association for Computing Machinery, New York, NY, USA, 175–186. https://doi.org/10.1145/3236024.3236082

[20] Rohan Reddy Mekala, Gudjon Einar Magnusson, Adam Porter, Mikael Lindvall, and Madeline Diep. 2019. Metamorphic detection of adversarial examples in deep learning models with affine transformations. In *2019 IEEE/ACM 4th International Workshop on Metamorphic Testing (MET)*. IEEE, 55–62.

[21] Grégoire Montavon, Alexander Binder, Sebastian Lapuschkin, Wojciech Samek, and Klaus-Robert Müller. 2019. Layer-wise relevance propagation: an overview. In *Explainable AI: interpreting, explaining and visualizing deep learning*. Springer, 193–209.

[22] S. Moosavi-Dezfooli, A. Fawzi, and P. Frossard. 2016. DeepFool: A Simple and Accurate Method to Fool Deep Neural Networks. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2574–2582. https://doi.org/10.1109/CVPR.2016.282

[23] M. Nejadgholi and J. Yang. 2019. A Study of Oracle Approximations in Testing Deep Learning Libraries. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 785–796. https://doi.org/10.1109/ASE.2019.00078

[24] Augustus Odena, Catherine Olsson, David Andersen, and Ian J. Goodfellow. 2019. TensorFuzz: Debugging Neural Networks with Coverage-Guided Fuzzing. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, 4901–4911. http://proceedings.mlr.press/v97/odena19a.html

[25] 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

[26] 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)*. IEEE Press, 1027–1038. https://doi.org/10.1109/ICSE.2019.00107

[27] G. Qin, B. Vrusias, and L. Gillam. 2010. Background Filtering for Improving of Object Detection in Images. In *2010 20th International Conference on Pattern Recognition*. 922–925. https://doi.org/10.1109/ICPR.2010.231

[28] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 779–788. https://doi.org/10.1109/CVPR.2016.91

[29] Joseph Redmon and Ali Farhadi. 2018. YOLOv3: An Incremental Improvement. *CoRR* abs/1804.02767 (2018). arXiv:1804.02767 http://arxiv.org/abs/1804.02767

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*. 1135–1144.

[31] D. Roobaert, M. Zillich, and J. . Eklundh. 2001. A pure learning approach to background-invariant object recognition using pedagogical support vector learning. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, Vol. 2. II–II. https://doi.org/10.1109/CVPR.2001.990982

[32] Amir Rosenfeld, Richard S. Zemel, and John K. Tsotsos. 2018. The Elephant in the Room. *CoRR* abs/1808.03305 (2018). arXiv:1808.03305 http://arxiv.org/abs/1808.03305

[33] J. Sanchez and F. Perronnin. 2011. High-Dimensional Signature Compression for Large-Scale Image Classification. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '11)*. IEEE Computer Society, USA, 1665–1672. https://doi.org/10.1109/CVPR.2011.

5995504

[34] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, et al. 2017. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 618–626. https://doi.org/10.1109/ICCV.2017.74

[35] Pierre Stock and Moustapha Cissé. 2018. ConvNets and ImageNet Beyond Accuracy: Understanding Mistakes and Uncovering Biases. In *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part VI (Lecture Notes in Computer Science)*, Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss (Eds.), Vol. 11210. Springer, 504–519. https://doi.org/10.1007/978-3-030-01231-1_31

[36] Zhiqiang Tang, Yunhe Gao, Leonid Karlinsky, Prasanna Sattigeri, Rogerio Feris, et al. 2020. OnlineAugment: Online data augmentation with less domain knowledge. In *European Conference on Computer Vision*. Springer, 313–329.

[37] Yongqiang Tian, Shiqing Ma, Ming Wen, Yepang Liu, Shing-Chi Cheung, et al. 2021. To what extent do DNN-based image classification models make unreliable inferences? *Empirical Software Engineering* 26, 5 (2021), 1–40.

[38] Yuchi Tian, Kexin Pei, Suman Jana, and Baishakhi Ray. 2018. DeepTest: Automated Testing of Deep-neural-network-driven Autonomous Cars. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 303–314. https://doi.org/10.1145/3180155.3180220

[39] F. Tramèr, V. Atlidakis, R. Geambasu, D. Hsu, J. Hubaux, et al. 2017. FairTest: Discovering Unwarranted Associations in Data-Driven Applications. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*. 401–416. https://doi.org/10.1109/EuroSP.2017.29

[40] Shuai Wang and Zhendong Su. 2020. Metamorphic Object Insertion for Testing Object Detection Systems. In *Proceedings of the 35th ACM/IEEE International Conference on Automated Software Engineering (ASE 2020)*. ACM, New York, NY, USA, 1053–1065. https://doi.org/10.1145/3324884.3416584

[41] Xiaoyuan Xie, Joshua W.K. Ho, Christian Murphy, Gail Kaiser, Baowen Xu, et al. 2011. Testing and validating machine learning classifiers by metamorphic testing. *Journal of Systems and Software* 84, 4 (2011), 544 – 558. https://doi.org/10.1016/j.jss.2010.11.920 The Ninth International Conference on Quality Software.

[42] Xiaofei Xie, Lei Ma, Felix Juefei-Xu, Minhui Xue, Hongxu Chen, et al. 2019. DeepHunter: a coverage-guided fuzz testing framework for deep neural networks. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2019, Beijing, China, July 15-19, 2019*, Dongmei Zhang and Anders Møller (Eds.). ACM, 146–157. https://doi.org/10.1145/3293882.3330579

[43] J. M. Zhang, M. Harman, L. Ma, and Y. Liu. 2020. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Transactions on Software Engineering* (2020), 1–1. https://doi.org/10.1109/TSE.2019.2962027

[44] 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)*. ACM, New York, NY, USA, 132–142. https://doi.org/10.1145/3238147.3238187

[45] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, et al. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the 42nd International Conference on Software Engineering (ICSE '20)*. Association for Computing Machinery, New York, NY, USA.

[46] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. 2020. Random erasing data augmentation. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 13001–13008.

[47] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. 2016. Learning Deep Features for Discriminative Localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2921–2929. https://doi.org/10.1109/CVPR.2016.319

[48] Zhi Quan Zhou and Liqun Sun. 2019. Metamorphic Testing of Driverless Cars. *Commun. ACM* 62, 3 (Feb. 2019), 61–67. https://doi.org/10.1145/3241979