Prioritizing Testing Instances to Enhance the Robustness of Object Detection Systems

Shihao Weng
shweng@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yang Feng*
fengyang@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Yining Yin
ynyin@smail.nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

Jia Liu
jialiu@nju.edu.cn
State Key Laboratory for Novel Software Technology
Nanjing University
Nanjing 210023, China

ABSTRACT

Object detection models have been widely deployed in military and life-related intelligent software systems. However, along with the outstanding success of object detection, it may exhibit abnormal behavior and lead to severe accidents and losses. During the development and evaluation process, training and evaluating an object detection model are computationally intensive, while preparing annotated tests requires extremely heavy manual labor. Therefore, reducing the annotation budget of test data collection becomes a challenging and necessary task. Although many test prioritization approaches for DNN-based systems have been proposed, the large differences between classification and object detection make them difficult to apply to testing object detection models.

In this paper, we propose `DeepView`, a novel instance-level test prioritization tool for object detection models to reduce data annotation costs. `DeepView` first splits the object detection results into instances, and then computes the localization and classification capabilities of the instances, respectively. Next, we design a test prioritization tool that enables testers to improve model performance by focusing on instances that may cause model errors from a large unlabeled dataset. To evaluate `DeepView`, we conduct extensive experiments on two kinds of object detection model architectures and two commonly used datasets. The experimental results show that `DeepView` outperforms existing test prioritization approaches regarding effectiveness and diversity. Also, we observe that using `DeepView` can effectively improve the accuracy and robustness of object detection models.

*Yang Feng is the corresponding author.

Internetware 2023, August 04–06, 2023, Hangzhou, China

ACM ISBN 979-8-4007-0894-7/23/08…$15.00
https://doi.org/10.1145/3609437.3609446

CCS CONCEPTS

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

KEYWORDS

Object Detection, Test Prioritization, Deep Learning Testing.

ACM Reference Format:
Shihao Weng, Yang Feng, Yining Yin, and Jia Liu. 2023. Prioritizing Testing Instances to Enhance the Robustness of Object Detection Systems. In *14th Asia-Pacific Symposium on Internetware (Internetware 2023), August 04–06, 2023, Hangzhou, China.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3609437.3609446

1 INTRODUCTION

Deep Neural Network (DNN) based object detection systems have been used in many fields to solve various tasks, such as Autonomous Driving [41], Face Recognition [43], and Robotics, etc. These systems are designed to identify and locate objects of interest in images or videos, which has enabled a wide range of applications. However, the vulnerability of DNN brings heavy quality and reliability issues to these systems. For example, Tesla's self-driving car [16] collided with a truck directly in 2019. The accident was caused by a failure in the object detection system, which mistakenly identified a white truck as the bright sky. Furthermore, in 2018, a woman was fatally struck by an Uber self-driving car which failed to detect her [27]. These accidents may threaten consumers' safety and cause significant economic damage. Therefore, testing and optimizing object detection systems has become an urgent and challenging task.

DNN-based systems are built based on a data-driven programming paradigm [22] that requires large amounts of data with ground truth for model training and evaluation. However, data annotation is particularly expensive for object detection tasks. Object detection models differ from traditional classification models in that they require manual annotators to specify bounding boxes for annotating object locations. Existing studies [29, 36] show that annotating a single object with tight bounding boxes can be ten times more expensive than answering a multiple-choice question. Moreover, the cost increases significantly when there are multiple objects present in a single image. Using all the data collected directly for annotating and retraining/fine-tuning to improve the model's performance and

robustness is a possible approach. Furthermore, such an approach is also very labor-intensive and budget-wasting. Because when having a well-trained model, most of the data collected will not improve the performance. Only a small amount of the data that is incorrectly detected by the model is new to the object model and can be used to improve robustness. Therefore, test prioritization methods are proposed to identify data worthy of the model's attention from a large, unlabeled dataset. These selected data points are then provided to testers for annotation, resulting in cost savings while simultaneously enhancing performance and robustness.

Up to now, researchers have proposed many novel prioritization techniques, such as *DeepGini* [11], *DeepState* [26] and *PRIMA* [38]. However, due to the substantial differences in evaluating and testing an object detection model compared to a classification model, most existing approaches for DNN-based systems are difficult to be migrated to object detection tasks. For the classification model, one input image only corresponds to a specific class in the candidate categories. Thus the model error exists in the image-level, which means the minimum unit to describe a model error is a single image. In object detection, the model's task is to locate and classify all objects present in a single input image. Therefore, the minimum unit for evaluating the object detection model should be a single object, which we refer to academically as *instances*.

Building on this concept, we introduce a novel test prioritization tool specifically designed for object detection systems, named `DeepView`. Since the errors in object detection models occur at the instance-level rather than the image-level, we adopt an approach that treats each detected object as a distinct instance. By doing so, we shift the focus of evaluation from the image-level to the instance-level. Both the classification and localization capabilities are combined to assess the model's instance detection performance. Building on this approach, we develop a test prioritization tool that aids testers in saving annotation time. To the best of our knowledge, we are the first to propose the object detection test prioritization tool and the first to test such a model from instance-level.

We implement `DeepView` and evaluate it under 12 experimental configurations. As pioneers in this task without any existing references, we compare `DeepView` with some existing object detection active learning technologies and classification model test prioritization tool that can be migrated to the object detection model. The experimental results demonstrate that `DeepView` outperforms the baselines in terms of effectiveness. Furthermore, we assess the diversity of `DeepView`'s results, and the experiments reveal that `DeepView` can prioritize a more diverse set of test cases compared to the baselines. Finally, we evaluate `DeepView`'s ability to improve the quality and robustness of the object detection model by selecting data at the top of the ranking for retraining. The experimental results demonstrate that `DeepView` can bring more remarkable accuracy improvement to the object detection model than baselines.

In summary, the main contributions of this paper are as follows.

Approach. We are the first to propose a prioritization technique for object detection models from instance-level. It integrally evaluates the classification capability and localization capability of a model for an instance.

Tool. Based on the instance-level perspective, we design and implement a tool called `DeepView`. Specifically, we split the model output into instances, sort them using our approach, map the Top-k instances back to the original image, and highlight them. The tester only needs to focus on the highlighted areas rather than the whole image. Therefore, `DeepView` enables more efficient use of the test budget. Our experimental codes are open source[1].

Study. We evaluated `DeepView` under 12 experimental configurations. The experimental results show that `DeepView` is good at prioritizing incorrectly detected instances. It could also be used to enhance the accuracy and robustness of object detection models through retraining.

2 BACKGROUND

This section introduces the basics of object detection models and the test prioritization method for DNN-based systems.

2.1 DNN-based Object Detection

Recently, with the advancement of Deep Learning (DL), Intelligent Software has flourished, especially in the areas of object localization and recognition. Object detection models have become the core technical dependency for many applications that require intelligent software with object detection and recognition capabilities [44].

The object detection task uses detectors to automatically locate particular objects in a scene and give them specific labels to assist humans in finding and identifying specific objects in complex scenes. For an input image or video, the object detection model annotates the object it finds with a rectangular box, labels it with the appropriate category, and then outputs the confidence. The metric used to evaluate the object detection performance is *mAP* (mean Average Precision) [44]. A high *mAP* usually means the detector has high accuracy.

Unlike most tasks, object detection models have many different architectures, each with its particularities. The most popular and widely used object detection models are usually divided into two-stage and one-stage architectures. The two-stage model first performs region generation, and the generated regions are called proposals, then use Convolution Neural Network (CNN) for region adjustment and object classification. Representative models within this category include SPP-Net [17], Faster R-CNN [31], and R-FCN [10]. One-stage models do not require the involvement of a proposal generator but extract features directly in the network and use the local features as proposals to perform region adjustment and object classification. Some of the representative models in this category are OverFeat[34], YOLO[30], SSD[25], and RetinaNet[24].

2.2 Test Prioritization for DNN-Based Systems

In traditional software testing, Rothermel et al. [32] first defined the concept of test prioritization. As Deep Learning techniques have been gradually applied in intelligent software in recent years, many researchers have designed test prioritization approaches for DNN-based systems, such as *DeepGini* [11], a generalized statistical method for DNN output confidence that outperforms many traditional techniques. In addition, there are also many test prioritization approaches for specific model architectures, such as *DeepState* [26] for Recurrent Neural Network (RNN).

With the development of Intelligent Software, many object detection models are used in areas such as intelligent assisted driving,

[1] https://github.com/wengshihao/DeepView

face payment, and area intrusion detection. Therefore, the security of such a system and the labor-intensive nature of test annotation is a matter of concern. However, to the best of our knowledge, researchers have not yet designed test prioritization techniques specifically for object detection models. As described in Section 2.1, object detection models are multi-tasking, with many widely different model architectures. During our experiments, we found that some of the currently available test prioritization tools for DNN-based systems perform poorly on the object detection model.

3 APPROACH

We design and implement a tool called DeepView to help developers test object detection systems and to annotate data that is more deserving of model attention with a lower annotation-budget. DeepView can also select instances from many unannotated data for object detection models to improve their performance. As shown in Figure 2, DeepView first analyzes the testing of object detection from the instance-level, which is introduced in Section 3.1. DeepView then computes a metric on instance-level, which is used to evaluate both the model's classification capability and localization capability. Next, prioritize the instances based on this metric, which is introduced in Section 3.2. Finally, Section 3.3 & 3.4 discusses how the test prioritization tool-DeepView works and improves the performance of the object detection model.

3.1 Instance-Level Analysis of Object Detection

Existing evaluation approaches available for object detection models, such as test prioritization approaches [11, 26, 38] for DNN-based systems, typically directly evaluate the image as a whole. Then, they obtain a score to measure whether an image is more valuable for testing or more informative for the model.

As described in Section 1 for the special case of the object detection system, if the whole image is the base unit for prioritization, it would take much time for testers to annotate all instances of the whole image one by one, most of which are very confident instances of the model and only a few of which are incorrectly detected. Obviously, such an approach is wasteful of the annotation-budget. For example, Google's Data Labeling Service [9] defines that the minimum unit for object detection annotation is the instance ($63 for 1000 instances) rather than the whole image, and an image with more object instances requires a larger payment in total, which is more expensive than classification dataset ($35 for 1000 images).

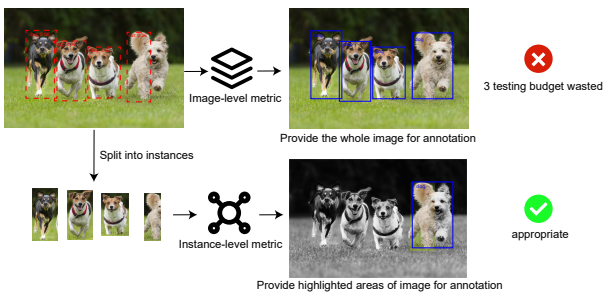


Figure 1: An example of the test prioritization in image-level and instance-level.

In this case, for test scenario, a natural idea to save the annotation-budget is to spend as much of the budget as possible on the more important instances rather than those in which the model is already confident. Therefore, unlike the above existing approaches that test at **image-level**, we propose a novel test prioritization approach for object detection models runs at **instance-level**, as shown in Figure 2. In the first stage, our framework first allows an object detection model with specified detection thresholds to perform inference on the input test cases (images), and then the results are split into instances. In the second stage, our framework analyzes the instances set to get the instances that are more worthy of manual annotation. Specifically, the detection capability of the model for each instance is evaluated from classification capability and localization capability. Then we sort the instances in descending order according to the value we get. The higher the ranking of the instance means that the model is less able to detect it, i.e., the instance is more worthy of manual annotation. Next, DeepView maps the Top-k (k is annotation-budget) instances back to their original image and highlights them, with the rest of the image grayed out. The tester only needs to annotate these highlighted instances to correct the model output. And for the gray part, the model's detection results are considered good enough that they do not require additional manual review. Figure 1 shows a comparison of a specific instance, where the model's output is shown on the left with red dashed boxes, and the two ways of testing it are shown on the right with solid blue boxes. Clearly, with the same annotation budget (the exact value of k), our approach is more budget-friendly.

3.2 Instance Prioritization Metric

In contrast to the traditional image-level evaluation, based on the above analysis of instance-level, we introduce a novel metric M for assessing the model's classification and localization capabilities at the instance-level. This new metric comprises two components: *Cls-capability* and *Loc-capability*. The calculation of the two properties and the principle of the composed M will be presented below, respectively.

3.2.1 ***Formal Representation.*** For a given test set $S = \{X_i\}_{i\in[n]}$, $[n] = \{1, \cdots, n\}$, n is the number of images in the test set, the detection of the object detector D with threshold T is:

$$D(X_i, T) = \{(s_{i,j}, box^p_{i,j}) | j \in [m_i]\}, \forall s_{i,j} \geq T \tag{1}$$

where $s_{i,j}$ and $box^p_{i,j}$ represent the confidence and location of the j-th instance of the detector output in i-th image, respectively. $[m_i]$ represents a set of natural numbers, i.e., $[m_i] = 1, ..., m_i$, and m_i is the number of instances detected under the model threshold $s \geq T$ in i-th image. The threshold T in the object detection system is used to control the number of output instances, which prevents the model from outputting redundant instances. In scenarios requiring high accuracy, such as autonomous driving, the system usually sets a high T value to ensure the detection of absolutely correct instances. Conversely, in other systems, such as Pedestrian Statistics System, a low threshold is set to ensure enough detected instances. In order to adapt DeepView to such different scenarios of the system, we will discuss the performance of DeepView under different T in the subsequent experimental section.

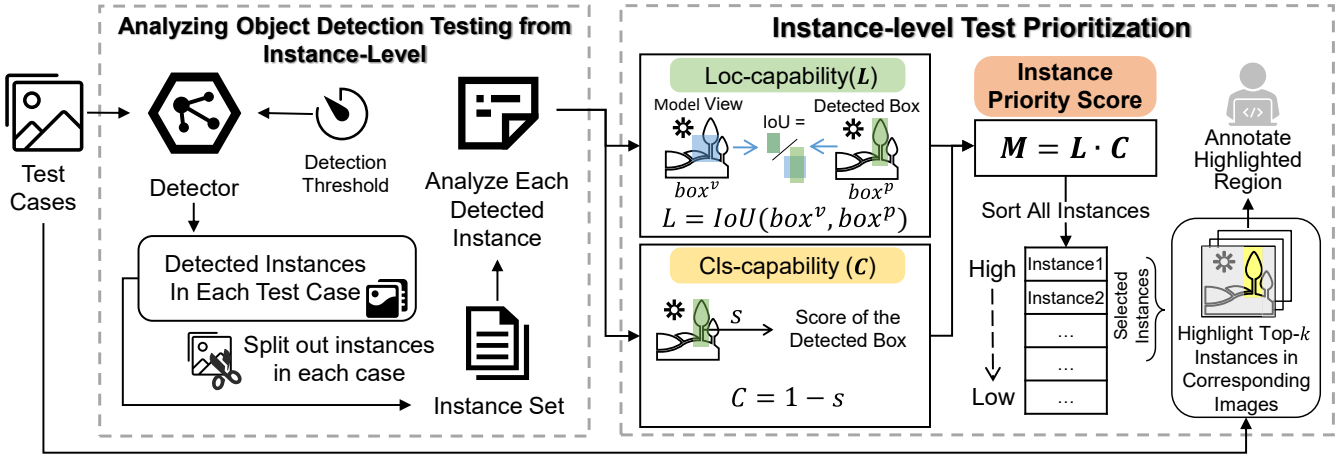


Figure 2: Overview of DeepView, which splits the inference results into instances and analyzes to get the instances that are more worthy of manual annotation from two detection capability aspects, then maps them back to the original image for annotating.

IoU (Intersection over Union) is a commonly used metric in object detection systems. Let $area$ and $area'$ represent two different areas, and the function $\mathbb{S}(\cdot)$ denotes the calculation of the size of a certain area, *IoU* is defined as:

$$IoU(area, area') = \frac{\mathbb{S}(area \cap area')}{\mathbb{S}(area \cup area')} \tag{2}$$

That is, *IoU* is the ratio of the size of the intersecting part of the two areas to the size of the merging part.

Let I denote the instance set detected by detector D for S, i.e:

$$(s_{i,j}, box^p_{i,j}) \in I, \forall (i,j) \in \{[n], [m_i]\} \tag{3}$$

We aim to design the prioritization metric M to obtain a ranked set $R = M(I)$, in which the higher the ranked test cases are, the more likely they cause errors in the object detection system. Thus, it is possible for testers to improve the model's performance by finding most of the errors by simply annotating and sequentially executing some of the test cases in R.

3.2.2 ***Cls-capability.*** For simplicity, we then treat each instance in I separately. From Equation 1, the model outputs a score s corresponding to an instance, which can be considered as the model's confidence in assigning the objects in the box to a specific class. So the measure of the model's capability to classify this instance is:

$$C = 1 - s \tag{4}$$

A larger C can be interpreted as a model with poorer classification capability for a specific instance, and vice versa. Similar concepts have also been employed in previous studies [13]. However, object detection is multitasking, and C only measures the capability of the model to classify the objects in the box it locates into a specific class, not the localization capability, so we further design a novel metric to measure the model's localization capability.

3.2.3 ***Loc-capability.*** Based on existing studies of object detection models [25, 31], it has been observed that a well-trained model is more proficient at precisely locating an object based on coarser proposal fields. Conversely, a poorly trained model's output tends to be closer to the original proposals. Inspired by this concept, our approach is designed. In this paper, we define the exact position of an object as *instance-location*, represented by its bounding box box^p, and the proposal's field as *model-view*, represented by its bounding box box^v. To better comprehend the concept of *model-view*, we introduce the detection process of a model for a single instance. Two-stage object detection models, such as Faster R-CNN [31] and SPP-Net [17], have a proposal generator, which generates a proposal on the original image, represents the area seen at first glance by the model. This first-glance-area is then fed to the detector, which adjusts its location according to the feature map within the proposal to get the final detection box. This step represents the model adjusting its previous rough first-glance-area to focus on locating the border of the object. For one-stage object detection models, such as SSD [25] and YOLO [30], these models can combine these two steps into one continuous operation. In summary, this first-glance area is the *model-view*, representing the region where the model initially receives.

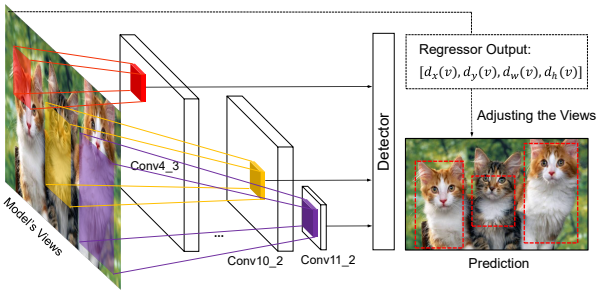


Figure 3: Relationship between the Model's View and the final Detected Box.

For example, Figure 3 shows a simplified SSD [25] model on a cat's image. On the left, we draw the *model-views* of each object

(denoted as Red, Yellow, Purple). On the right, we draw the *instance-location*. According to the above principle, the model's capability for locating these three instances should be ranked as Red, Purple, and Yellow. Because for Red, the model is able to locate the entire cat based on a very small view, while for Yellow, the model is only able to output its *model-view*.

To measure the concept above, we design a novel metric to quantify it. Which is calculated as:

$$L = IoU(box^v, box^p) \tag{5}$$

We use IoU to quantify the above concepts. A larger L means that the model is less capable of locating an instance.

However, obtaining the *model-view* of an instance becomes a problem again. Saving the *model-view* at all times during the inference process is a waste of computational resources in the application. To improve efficiency, we briefly consider the final proposals generated as their *model-view* so that the *model-view* can be inferred back from the instances and the corresponding regression information. Specifically, for a single instance, let $box^p = [p_x, p_y, p_w, p_h]$ be the output instance, where p_x, p_y, p_w, and p_h represent the coordinates of the center point and the width and height of the detected box, respectively. As shown in Figure 3, the model produces $[d_x(v), d_y(v), d_w(v), d_h(v)]$ in the hidden layer, which is computed based on the feature information from the *model-view*. Given $d_x(v), d_y(v), d_w(v), d_h(v)$, the computation formula of how model adjusting $box_v = [v_x, v_y, v_w, v_h]$ to $box^p = [p_x, p_y, p_w, p_h]$ is as follows:

$$box^p = \begin{cases} p_x = v_w d_x(v) + v_x \\ p_y = v_h d_y(v) + v_y \\ p_w = v_w \cdot \exp\left(d_w(v)\right) \\ p_h = v_h \cdot \exp\left(d_h(v)\right) \end{cases} \tag{6}$$

According to Equation 6 it can be deduced that box^v is:

$$box^v = \begin{cases} v_x = p_x - \frac{p_w}{\exp(d_w(v))} \cdot d_x(v) \\ v_y = p_y - \frac{p_h}{\exp(d_h(v))} \cdot d_y(v) \\ v_w = \frac{p_w}{\exp(d_w(v))} \\ v_h = \frac{p_h}{\exp(d_h(v))} \end{cases} \tag{7}$$

With Equation 7, we can obtain the *model-view* to which the instance corresponds by simply computing the output results. Thus, calculating L helps us measure the localization capability.

3.2.4 ***Metric.*** Now, we have a metric C that measures the model's classification capability and a metric L that can measure the model's localization capability for a particular output instance. We combine them to form our metrics as follows:

$$M = L \cdot C \tag{8}$$

M can be considered as a measure of the model's capability to detect an instance in terms of classification and localization, and a larger value of M means that the model is less able to detect an instance. Therefore, we can use M to sort the set of instances I in descending order. The higher-ranked instances are considered more important to test. Specifically, the insight of M is a value that measures the overall performance of an object detection model; if the model performs well in both classification and localization with respect to an instance, then the value of M will be low. Conversely, if the performance in either classification or localization is poor, the value of M will increase. This demonstrates the importance of classification and localization capabilities for the overall performance of an object detection model.

3.3 Test Prioritization Tool: `DeepView`

`DeepView` is a tool designed to prioritize test instances and save annotation-budget. As shown in Figure 2, for the collected dataset, `DeepView` first performs inference and then splits the results into multiple instances. Next, `DeepView` scores each instance using the metrics M in Section 3.2, then maps the top-k instances back to the original image and highlights them. The testers only need to focus on the highlighted areas and annotate them accordingly.

Algorithm 1: Process of `DeepView`

Data: $S = [X_1, X_2, \cdots, X_n]$: The original test set.
Input: D: The tested Object Detection Model; T: Model Threshold; k: annotation-budget.
Output: $TestData$: Data to be tested

1 R = []; `// Test prioritization set of instances`
2 **while** $i < n$ **do**
3 Ins = split_into_instances($D(X_i, T)$);
4 **while** $j < len(Ins)$ **do**
5 $Ins_j.img = X_i$;
6 $Ins_j.C$ = 1-$Ins_j.s$;`// Eq.4`
7 $Ins_j.box^v$ = getBoxV($Ins_j.box^p$, $Ins_j.d$);`// Eq.7`
8 $Ins_j.L$ = getIoU($Ins_j.box^v$, $Ins_j.box^p$);`// Eq.5`
9 $Ins_j.M = Ins_j.L \cdot Ins_j.C$;`// Eq.8`
10 R.add(Ins_j)
11 **end**
12 **end**
13 R = reverse_sortBy($R.M$);
`/* Highlight Top-`k` Instances in Corresponding Images */`
14 **while** $i < k$ **do**
15 image = Grayed_out($R_i.img$);
16 **if** $image \notin TestData$ **then**
17 $TestData$.add(image);
18 **end**
19 $TestData$.image = highlight($TestData$.image, R_i);
20 **end**
21 **return** $TestData$

Algorithm 1 outlines the process of `DeepView`. Given a specific object detection model D to be tested and the original test image set S, `DeepView` first inference each image $X \in S$ with the model D. And then, the outputs are split into individual instances and saved as the set Ins (Line 2-3). Secondly, for each instance in the instance set Ins, `DeepView` calculates its instance-level *Cls-capability*, *model-view*, and *Loc-capability* successively. So, the metric M for each instance can be computed (Line 4-11). Subsequently, all instances in Ins are sorted based on the metric M (Line 13). Finally, we use k to represent the annotation budget, `DeepView` selects the top-k instances. The corresponding images are then grayed out, mapped

back to the original images, and highlighted. These images are provided to the tester for annotation (Line 14-21).

3.4 Enhancing Object Detection with DeepView

In order to improve the performance of intelligent software systems with object detection models, a common approach is to collect data in usage scenarios, such as autonomous driving and face detection, and then annotate them by a large number of crowd-sourced workers. However, such an approach is obviously a costly and time-consuming task compared to simple classification tasks. DeepView provides a solution to this challenge, which is designed to prioritize an instance sub-set from a large dataset. Instances that appear earlier in the sorted list are more likely to reveal errors in the object detection model. Also, compared to the traditional image-level prioritization approach, DeepView ensures that the annotators do not waste budgets to annotate instances that are already well-trained. In practice, we can directly incorporate the model's detection results (pseudo-labels) as a part of the ground truth and combine them with the annotated portions from the testers. This combined dataset can then be used to retrain the model for accuracy improvement. In this context, the model can encounter more instances that it predicted incorrectly before. And the detection model can be refined with sufficient challenging data samples.

4 EXPERIMENT DESIGN

We conduct extensive experiments to evaluate the performance of DeepView. This section introduces the experiment settings. To conduct the experiments, we implement DeepView on Python 3.9.15 [2] with Pytorch 1.12.0 [3]. All experiments are performed on a Ubuntu 18.04.3 LTS server with Tesla V100-SXM2, one 10-core processor with 2.10GHz, and 64GB physical memory.

4.1 Studied Datasets and Models

As shown in Table 1, we conduct experiments on DeepView using two widely used object detection datasets. We apply each dataset to one-stage and two-stage object detection models to ensure the generality of the experimental results. Three thresholds T for each model to discuss the performance of DeepView in the different scenarios mentioned in Section 3.2.

MS COCO [28] is a widely used object detection dataset built by Microsoft. We use the standard COCO 2017 version, which has 80 object categories, 118,287 images for training, 5,000 images for validation, and about 940,000 objects annotated. The Pascal VOC [1] dataset is derived from the famous Pascal VOC Challenge, and we use the standard VOC 2012 version, which has 20 object classes, 5,717 images for training, 5,823 images used for validation, and about 27,000 objects are annotated. AI community generally adopts *AP*@[0.5:0.95] and *AP*@0.5 metrics to evaluate the accuracy of detection models for COCO and VOC, respectively. And in this paper, we use the mean average precision across the above evaluation metrics (*mAP*) as a proxy metric for unified discussion.

Faster R-CNN [31] is a representative two-stage model, which is the first to implement end-to-end training. The model is divided into three parts, backbone network to extract features, RPN to generate suggestion frames, and RCNN for classification and regression. SSD [25] is a representative one-stage model that performs box generation, classification, and regression on multiple feature maps of different depths and uses them to obtain the detection results. In this paper, we use the weights on the COCO officially provided by Pytorch. As officially reported, Faster R-CNN with backbone ResNet50 [18] has a *AP*@[0.5:0.95] of 46.7 [4] and SSD with backbone VGG16 [35] has a *AP*@[0.5:0.95] of 25.1 [5]. As shown in Table 1, we assume that this Faster R-CNN model works in high accuracy requiring scenarios (because it has a good *mAP* itself), so experiments are performed at three thresholds of 0.7, 0.8, and 0.9. SSD, on the contrary, we experiment it at three thresholds of 0.5, 0.6, and 0.7.

Table 1: The detailed information of experiment datasets and Object Detection Models.

Dataset	Description	Models	Detection Threshold
MS COCO	118K/5K images for train/val with over 900K objects	SSD	(0.5,0.6,0.7)
		Faster R-CNN	(0.7,0.8,0.9)
Pascal VOC	5.7K/5.8K images for train/val with over 27K objects	SSD	(0.5,0.6,0.7)
		Faster R-CNN	(0.7,0.8,0.9)

4.2 Baselines

To the best of our knowledge, researchers have not yet designed test prioritization techniques specifically for object detection models, and existing prioritization techniques are difficult to apply to object detection systems. We choose a barely usable test prioritization technique and two Active Learning (AL) techniques for object detection models as baselines to measure the performance of DeepView. These technologies are discussed from the image-level. In addition, we compare the effect of using only *Cls-capability*, thus demonstrating that *Loc-capability* is essential.

Specifically, we compare DeepView with *DeepGini* [11], a generic test prioritization approach for classification systems. Considering that its core idea is to calculate the *Gini* coefficient for the predicted results of the picture. In order to migrate this technique to the object detection model as much as possible, we calculate the *Gini* coefficient for multiple scores output by the model for one image. In addition, we select the two state-of-art active learning approaches for object detection models, and we use the ranking algorithms from these approaches to compare with DeepView. These two approaches are *Margin (1vs2-Sum)* [6] and *Entropy* [33], where the authors of *Margin* propose different computations, namely *1vs2-Max*, *1vs2-Avg*, *1vs2-Sum*, and we use the best algorithm reported in the paper. Furthermore, we use two random sampling methods as part of the baselines, and they are classified as image-level random and instance-level random. For simplification, we mark them as *Ran-Imag* and *Ran-Inst*, respectively.

5 RESULT ANALYSIS

DeepView is designed to help testers of object detection systems quickly identify tests that can trigger potentially erroneous behavior and annotate them to retrain/fine-tune the model to improve

model accuracy and robustness. To this target, we empirically evaluate its performance with three research questions (RQs) and answer these three RQs based on the experimental results, respectively.

5.1 Effectiveness

RQ1: Is DeepView more effective than the baseline in prioritizing those instances detected incorrectly?

The existing study [25] defines TP (True Positive)-instance as the instance that the model detects accurately, i.e., having *IoU* between the box of detected instance and the box of ground truth greater than 0.5 and predicted category is the same as the ground truth category. Other instances of the model output are considered to be incorrectly detected, which we need to prioritize and label to improve the model's accuracy and robustness.

Following the conventional test prioritization researches [11, 13], we use the metric **APFD** [37] (Average Percentage Fault Detected) to measure the overall performance of prioritization methods. We also use **RAUC-500** [38](Ratio of Area Under the Curve) to measure the effect of the first 500 instances of the result of the priority sorting method, because the sequential labeling of the sorted set makes the top instances more important for evaluation. Moreover, we conduct additional experiments to demonstrate the necessity of using *Loc-capability*.

Table 2: APFD values and variance for DeepView and baseline for different configurations.

APFD		Threshold	Ran-Inst	Ran-Imag	1vs2 Sum	Entropy	DeepGini	DeepView
MS COCO	SSD	*0.5*	0.498	0.502	0.604	0.584	0.586	**0.743**
		0.6	0.491	0.505	0.599	0.580	0.582	**0.746**
		0.7	0.482	0.505	0.593	0.575	0.577	**0.741**
	FRCNN	*0.7*	0.494	0.500	0.577	0.578	0.578	**0.724**
		0.8	0.502	0.499	0.578	0.579	0.579	**0.739**
		0.9	0.505	0.501	0.579	0.581	0.580	**0.747**
Pascal VOC	SSD	*0.5*	0.509	0.490	0.614	0.533	0.538	**0.747**
		0.6	0.485	0.491	0.588	0.521	0.525	**0.746**
		0.7	0.491	0.478	0.557	0.504	0.506	**0.746**
	FRCNN	*0.7*	0.495	0.499	0.542	0.548	0.548	**0.745**
		0.8	0.496	0.503	0.545	0.552	0.552	**0.751**
		0.9	0.502	0.496	0.547	0.551	0.551	**0.746**
σ (10^{-5})			6.65	6.13	59.02	71.04	66.68	4.73

As shown in Table 2, DeepView consistently outperforms all five baselines in terms of APFD values across all 12 experimental configurations. In particular, the performance of the two random-baselines is similar. Among the three state-of-the-art baselines, the 1vs2-Sum outperforms others, which is in line with related studies [6]. However, DeepView outperforms these methods by a wide margin, which demonstrates that DeepView can prioritize instances that are detected incorrectly by the model more effectively than baselines.

In addition, we calculate the variance (denoted σ) of each method under 12 configurations. DeepView also outperforms baselines. In particular, we use the two random-baselines as an upper bound on the measure of stability, since random should behave the same way under different configurations. We find that the variance of the three state-of-the-art baselines is much higher than the random baseline, which implies that they have poorer stability in different scenarios. However, the variance of DeepView is lower than the random-baselines. This means that DeepView is more stable for different application scenarios mentioned in Section 3.2.

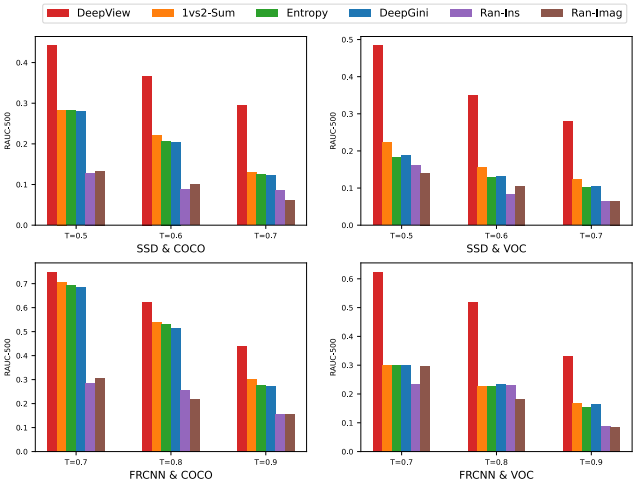


Figure 4: The RAUC-500 values for DeepView and baselines in different configurations.

Figure 4 illustrates the comparison of RAUC-500 values. Across all 12 configurations, DeepView consistently outperforms the five baselines. It exhibits a remarkable performance, particularly on the VOC dataset, where DeepView achieves a RAUC-500 value twice as high as that of the baselines. This further confirms that for the top 500 significant instances, the performance of DeepView is better than the baseline. Specifically, we find that for the same dataset, the higher the threshold is set for the detector, the progressively less effective approaches are. We conjecture that this is because when the requirements on the model are increased, i.e., a high threshold is set, the number of instances detected incorrectly decreases accordingly. Such a situation makes all approaches have more difficulty finding incorrect instances.

Table 3: APFD and RAUC-500 values for using *Cls-capability* only and complete DeepView.

Approach	Threshold	APFD COCO	APFD VOC	RAUC-500 COCO	RAUC-500 VOC
DeepView-*Cls-capability*	*0.5*	0.731	0.736	0.337	0.377
	0.6	0.736	0.735	0.278	0.300
	0.7	0.728	0.734	0.199	0.233
DeepView	*0.5*	**0.743**	**0.747**	**0.442**	**0.484**
	0.6	**0.746**	**0.746**	**0.367**	**0.351**
	0.7	**0.741**	**0.746**	**0.294**	**0.279**

To demonstrate the necessity of using *Loc-capability* for DeepView, we conduct a further experiment to analyze the effectiveness of only using *Cls-capability* in test prioritization. As shown in Table 3, the APFD value of the complete DeepView surpasses that of the *Cls-capability*-only version, providing strong evidence that incorporating *Loc-capability* leads to an enhancement in the overall performance of DeepView. Similarly, the higher RAUC-500 value of the complete DeepView in comparison to the *Cls-capability*-only version reinforces the conclusion that the inclusion of *Loc-capability* significantly improves the effectiveness in prioritizing instances that are incorrectly detected.

Answer to RQ1: DeepView can prioritize incorrectly detected instances more effective than baselines.

5.2 Diversity

RQ2: Can DeepView prioritize more types of errors than baselines?

Diversity is a critical metric in test prioritization [8]. When retraining data lacks sufficient diversity, it can lead to biases in the training results, ultimately diminishing the model's accuracy. Existing research [13] defines the concept of error types. We extend it to adjust object detection systems as follows:

$$Error\_Type(x) = (Annotation(x) \rightarrow Detection(x)) \quad (9)$$

Specifically, we divide the errors into two main types, namely classification errors and localization errors. For classification errors, $Annotation(x)$ denotes the ground truth class for x, and $Detection(x)$ denotes the detected class by the model for x. e.g., $Error\_Type(x) = (cat \rightarrow dog)$ indicates that a cat instance is detected as a dog, which is regarded as a classification error, marked as $cat \rightarrow dog$. For localization errors, according to object detection research [31], the instance having $IoU < 0.5$ with its ground truth box is defined as a location error. We use the category LE to denote these incorrectly located instances, e.g., $Error\_Type(x) = (cat \rightarrow LE)$ indicates that a cat instance is mislocated by the model, which is regarded as localization error, marked as $cat \rightarrow LE$.

We use a cumulative sum curve of fault types [13] to quantify the diversity of prioritization methods. The x-axis is the order of execution of the prioritized list, and the y-axis represents the number of different error types found after the execution of the top x instances. Therefore, the higher the curve is, the more diverse the prioritization method is. The results are shown in Figure 5, DeepView can reveal more types of incorrectly detected instances with fewer test cases than baselines, indicating that DeepView does not bias against some specific types of errors. On the contrary, we find that the curves of some baselines are even lower than the random-method. We guess that these methods have a bias for certain error types. Although they have some effect, this is of no value for practical applications if only a few types of incorrectly detected instances can be found. While with the assistance of DeepView, the developer only needs to execute a small number of instances in front of the sorted list to get most of the error types.

Answer to RQ2: DeepView can prioritize more types of errors instances than baselines and are not biased to specific types.

5.3 Guidance

RQ3: Can DeepView guide the retraining of an object detection model to improve its accuracy?

To answer RQ3, we divide the validation set of the VOC into two equal parts and double the data for each part using data augmentation methods[2]. We use one for running DeepView and retraining and the other for evaluating the performance of retraining. To reflect the improvement of DeepView's accuracy for object detection models under different labeling budgets, we set the values of annotation budget k to be 2.5%, 5%, 7.5%, and 10% of the total number of detected instances, respectively. And we use the pseudo-labels method. For example, 5% means that the top 5% instances prioritized by DeepView are taken for labeling and retraining. But we can't take these instances directly for training. We need to map them to the images they belong to. Since there are usually multiple instances in one image, for those instances that are not part of the 5%, we directly use the model's detection results as their labels. These labels are named pseudo-labels.

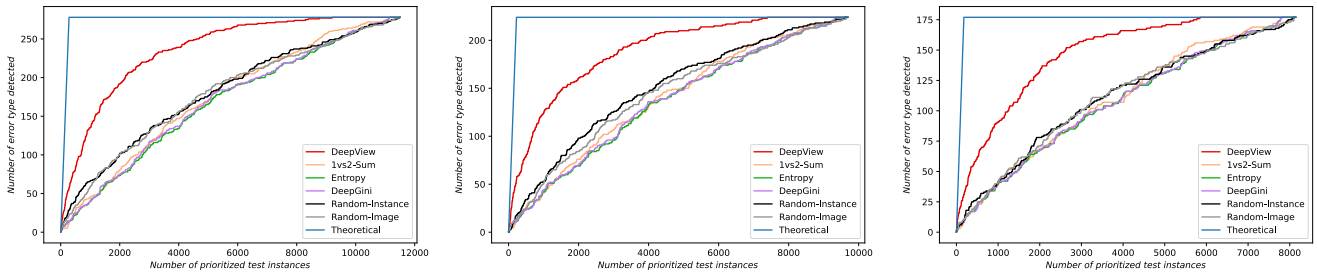
Table 4: *mAP* and Δ*mAP* values (%) of the model after retraining using DeepView and baselines.

Method	Annotation-Budget	FRCNN		SSD	
		mAP	Δ*mAP*	*mAP*	Δ*mAP*
Original	/	80		61.9	
Ran-Inst	10%	80	+0	62.6	+0.7
Ran-Imag		80	+0	62.4	+0.5
1vs2-Sum		81.1	+1.1	63.6	+1.7
Entropy		81	+1	63.4	+1.5
DeepGini		80.6	+0.6	63.1	+1.2
DeepView	2.5%	81.2	+1.2	65.7	+3.8
	10%	81.7	+1.7	66.2	+4.3

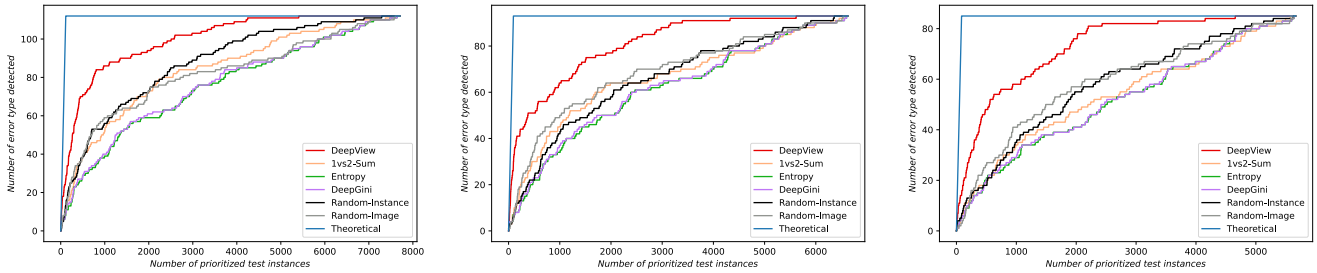
As described above, to measure the improvement in model performance using DeepView, we use DeepView and baselines to retrain the model separately. Figure 6 shows the performance improvement (denoted Δ*mAP*) of them relative to the original model. DeepView outperforms baselines in terms of improving model accuracy. In addition, we notice that the *DeepGini* and *Entropy* have negative Δ*mAP* in some cases, i.e., these techniques even lead to a decrease in model precision. We conjecture that the observed phenomenon is a result of a bias towards specific types. On the contrary, DeepView achieves a significant performance boost by striking a balance between effectiveness and diversity simultaneously.

Table 4 shows some of the specific values of the different methods for model performance improvement. We find that DeepView can outperform baselines with a 10% labeling budget using 2.5% of the annotation budget, which proves that DeepView can save more time and resources on manual annotation than baselines. In practice, we only need to provide testers with a small number of instances in front of DeepView's ranked list to improve the performance and robustness of the model.

[2]https://pytorch.org/vision/stable/transforms.html?highlight=augment



(a) MS COCO with model thresholds of 0.5, 0.6, and 0.7



(b) Pascal VOC with model thresholds of 0.5, 0.6, and 0.7

Figure 5: Cumulative sum curve of fault types for DeepView and baselines on SSD with two different dataset.

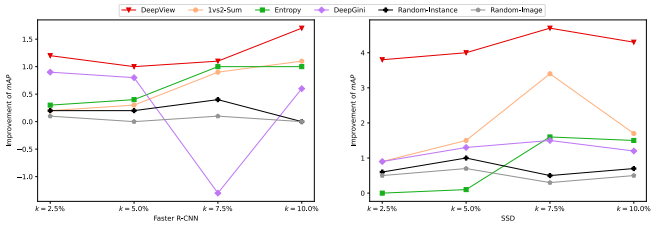


Figure 6: ΔmAP values for DeepView and baselines after retraining the model with different labeling budgets.

Answer to RQ3: Using DeepView can bring more performance and robustness improvements to the model than baselines and save manual labeling time.

6 DISCUSSIONS

This section discusses the comparison between DeepView and active learning methods, the potential of DeepView to save manual annotation time and the application scenarios of DeepView, and outlines the future work and potential threats.

6.1 Comparison with Active Learning Methods

Our experiments indicate that active learning methods are less effective than DeepView in test prioritization. This is primarily because active learning methods [33] are designed for model training rather than testing. Consequently, active learning techniques are usually focused on identifying data that is more informative for the model at the image-level. However, being more informative does not necessarily imply that the model is more likely to detect instances incorrectly. In application scenarios, active learning is primarily concerned with improving the model's performance during training, whereas DeepView is specifically designed for testing and evaluating object detection models that have been trained within a system. Additionally, active learning generally adopts an iterative sampling strategy, and only a small batch of extra data is selected for each training epoch. While DeepView is designed to prioritize and reveal enough model errors with only one-time model inference.

6.2 Application Scenarios and Future Work

DeepView is able to prioritize the instances that are detected incorrectly. These instances represent the incomplete and inaccurate parts of the model. Finding these instances by random-methods is time-consuming. DeepView provides a solution for this, thus reducing manual annotation time. Experimental results show that DeepView can achieve improvement in model performance and robustness using a small annotation budget. From this perspective, DeepView saves a lot of manual annotation time.

As introduced in Algorithm 1, DeepView prioritizes test cases from instance-level and highlights the ones most likely to be detected incorrectly by the model, making them available to testers for annotating. The traditional image-level approach is to provide the tester with the entire image, which obviously wastes the annotation-budget, since only a small number of instances in the entire image are worthy of the tester's attention. Therefore, DeepView saves the annotation-budget, i.e., the manual annotation time.

The application scenarios of DeepView mainly lie in two aspects. From the perspective of model optimization, due to problems such as distribution drift [23] and insufficient training samples [14], the object detection system still needs to be continuously optimized after training and deployment, and DeepView can timely discover instances of poor model performance, so as to achieve the maximum performance improvement at the lowest possible annotation

cost; from the perspective of model evaluation, in security-critical scenarios, users are usually more concerned about whether the object detection system exhibits serious performance degradation for a specific type of error, and by prioritizing the exposure of model error instances to the users through DeepView, the system's security issues can be revealed and modified faster.

In future work, we will explore the role of DeepView as an aid in large and complex systems and further optimize our approach. For example, investigating the use of DeepView to evaluate the prediction results of models on instances in application scenarios in a complex autonomous driving system to provide suggestions to the decision-making system.

6.3 Threats to Validity

The main threat to our findings lies in the choice of dataset and object detection model. Different architectures of object detection models produce different results, which introduces potential bias. To address this issue, we consider different model architectures when designing DeepView so that it can adapt to different object detection models. Furthermore, object detection datasets are of various distributions and tasks, and different datasets may lead to different results. To address this threat, we include two widely used datasets in our evaluation, employing representative one-stage and two-stage models for each of the employed datasets, thus mitigating this threat while also ensuring a comprehensive evaluation of DeepView's performance. The setting of our experiments also helps ensure the robustness and generalizability of our findings.

7 RELATED WORKS

In this section, we introduce the related work from two different aspects. First, we introduce the existing technologies for deep learning testing. After that, we provide a brief introduction to active learning methods for object detection systems.

7.1 Deep Learning Testing

Deep learning (DL) models and systems are widely used in various fields. To ensure their security and reliability, software engineering researchers have proposed techniques for DL testing. Due to the large input space of DL models, verifying their reliability and identifying defects often requires a significant number of manually labeled test cases [42]. Researchers have proposed various techniques for DL testing, including metamorphic testing [7] and reducing annotation costs. For example, Kim et al. [20] proposed the surprise adequacy metric, Feng et al. [11] proposed DeepGini based on output Gini impurity, and Wang et al. [38] proposed PRIMA based on mutation testing. However, most proposed deep learning test selection techniques can only be applied to limited deep learning tasks. Moreover, for more complex deep learning tasks and systems, existing selection methods cannot be simply migrated and applied. Kim et al. [21] emphasized and tried to solve this problem by testing an industrial semantic segmentation model in an autonomous driving system.

For object detection systems, the output of the model is inconsistent, which means there may be multiple objects in the input, or there may not be any objects. This characteristic inspired us to design the DeepView better to improve the test efficiency of the object detection system. Compared to existing techniques such as *DeepGini* that prioritize each input image [11], DeepView designs a finer granularity priority score metric specific for object detection systems. DeepView can highlight a small region of some test inputs for the testers to annotate and inspect prioritizing, thus significantly enhancing the efficiency of object detector testing and retraining.

7.2 Active Learning for Object Detection

Active learning is a popular technique in deep learning that aims to reduce the cost of manual labeling and select the most informative samples for annotation. Object detection is a complex and time-consuming task in computer vision, which requires large and diverse datasets for model training. Researchers in the computer vision community have combined active learning with object detection to improve the efficiency of data annotation. Several studies have proposed different methods for image selection in object detection tasks, such as query by committee [33], uncertainty-based active learning [6], and classification and localization results [19]. Recent studies have also introduced new active learning methods, such as entropy-based selection [39] and consistency-based weight [40]. Researchers have also studied the scalability of active learning in practical applications, such as autonomous driving [15]. Furthermore, a benchmark framework, *ALBench* [12], has been proposed to evaluate active learning in object detection.

Although both test prioritization method DeepView and active learning aim to reduce annotation efforts in object detection, they differ in their focus and application. Active learning is mainly used for building *training* data, while DeepView is used in the *testing* scenario to identify and evaluate model errors efficiently. Most of the active learning techniques are proposed at the image-level because most instances cannot be detected before the model is trained well. While in the testing scenario, the detectors to be tested have reached a relatively good performance for most of the detected instances, thus an instance-level view of DeepView is more reasonable for the test prioritization subject.

8 CONCLUSION

In this paper, we introduce DeepView, a test prioritization tool for object detection systems. We prioritize test instances from an instance-level perspective, using a metric to evaluate the model's capability to classify and localize each particular instance. Our experimental results demonstrate its effectiveness and diversity in detecting model errors, as well as guidance for enhancing model robustness. This work contributes to improving the quality and safety of object detection systems and fosters the development of object detection test methods. DeepView also provides practitioners with a tool to help them improve the reliability of their models. It enables them to save time and resources in the annotation process.

ACKNOWLEDGMENTS

We would like to thank anonymous reviewers for their insightful and constructive comments. This research was partially funded by the National Natural Science Foundation of China under Grant Nos. 62002158, 61832009, and 61932012, and the Science, Technology and Innovation Commission of Shenzhen Municipality (No. CJGJZD20200617103001003).

REFERENCES

[1] 2012. Pascal VOC. http://host.robots.ox.ac.uk/pascal/VOC/.

[2] 2022. Python 3.9.15. https://www.python.org/downloads/release/python-3915/.

[3] 2022. Pytorch 1.12.0. https://pytorch.org/get-started/previous-versions/.

[4] 2022. Pytorch Faster-RCNN on COCO. https://pytorch.org/vision/stable/models/generated/torchvision.models.detection.fasterrcnn_resnet50_fpn_v2.html.

[5] 2022. Pytorch SSD on COCO. https://pytorch.org/vision/stable/models/generated/torchvision.models.detection.ssd300_vgg16.html.

[6] Clemens-Alexander Brust, Christoph Käding, and Joachim Denzler. 2018. Active learning for deep object detection. *arXiv preprint arXiv:1809.09875* (2018).

[7] Tsong Y Chen, Shing C Cheung, and Shiu Ming Yiu. 2020. Metamorphic testing: a new approach for generating next test cases. *arXiv preprint arXiv:2002.12543* (2020).

[8] Tsong Yueh Chen, Hing Leung, and Ieng Kei Mak. 2005. Adaptive random testing. In *Advances in Computer Science-ASIAN 2004. Higher-Level Decision Making: 9th Asian Computing Science Conference. Dedicated to Jean-Louis Lassez on the Occasion of His 5th Birthday. Chiang Mai, Thailand, December 8-10, 2004. Proceedings 9*. Springer, 320–329.

[9] Google Cloud. 2022. AI Platform Data Labeling Service pricing. https://cloud.google.com/ai-platform/data-labeling/pricing.

[10] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. 2016. R-fcn: Object detection via region-based fully convolutional networks. *Advances in neural information processing systems* 29 (2016).

[11] Yang Feng, Qingkai Shi, Xinyu Gao, Jun Wan, Chunrong Fang, and Zhenyu Chen. 2020. DeepGini: prioritizing massive tests to enhance the robustness of deep neural networks. In *ISSTA '20: 29th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, USA, July 18-22, 2020*, Sarfraz Khurshid and Corina S. Pasareanu (Eds.). ACM, 177–188. https://doi.org/10.1145/3395363.3397357

[12] Zhanpeng Feng, Shiliang Zhang, Rinyoichi Takezoe, Wenze Hu, Manmohan Chandraker, Li-Jia Li, Vijay K Narayanan, and Xiaoyu Wang. 2022. ALBench: A Framework for Evaluating Active Learning in Object Detection. *arXiv preprint arXiv:2207.13339* (2022).

[13] Xinyu Gao, Yang Feng, Yining Yin, Zixi Liu, Zhenyu Chen, and Baowen Xu. 2022. Adaptive test selection for deep neural networks. In *Proceedings of the 44th International Conference on Software Engineering*. 73–85.

[14] Yuan Gao, Jiayi Ma, and Alan L Yuille. 2017. Semi-supervised sparse representation based classification for face recognition with insufficient labeled samples. *IEEE Transactions on Image Processing* 26, 5 (2017), 2545–2560.

[15] Elmar Haussmann, Michele Fenzi, Kashyap Chitta, Jan Ivanecky, Hanson Xu, Donna Roy, Akshita Mittel, Nicolas Koumchatzky, Clement Farabet, and Jose M Alvarez. 2020. Scalable active learning for object detection. In *2020 IEEE intelligent vehicles symposium (iv)*. IEEE, 1430–1435.

[16] Andrew J. Hawkins. 2019. Tesla's Autopilot was engaged when Model 3 crashed into truck, report states. https://www.theverge.com/2019/5/16/18627766/tesla-autopilot-fatal-crash-delray-florida-ntsb-model-3.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Spatial pyramid pooling in deep convolutional networks for visual recognition. *IEEE transactions on pattern analysis and machine intelligence* 37, 9 (2015), 1904–1916.

[18] 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.

[19] Chieh-Chi Kao, Teng-Yok Lee, Pradeep Sen, and Ming-Yu Liu. 2018. Localization-aware active learning for object detection. In *Asian Conference on Computer Vision*. Springer, 506–522.

[20] 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 2019, Montreal, QC, Canada, May 25-31, 2019*, Joanne M. Atlee, Tevfik Bultan, and Jon Whittle (Eds.). IEEE / ACM, 1039–1049. https://doi.org/10.1109/ICSE.2019.00108

[21] Jinhan Kim, Jeongil Ju, Robert Feldt, and Shin Yoo. 2020. Reducing DNN labelling cost using surprise adequacy: an industrial case study for autonomous driving. In *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, Prem Devanbu, Myra B. Cohen, and Thomas Zimmermann (Eds.). ACM, 1466–1476. https://doi.org/10.1145/3368089.3417065

[22] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. 2015. Deep learning. *Nature* (2015).

[23] Timothée Lesort, Massimo Caccia, and Irina Rish. 2021. Understanding continual learning settings with data distribution drift analysis. *arXiv preprint arXiv:2104.01678* (2021).

[24] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*. 2980–2988.

[25] 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.

[26] Zixi Liu, Yang Feng, Yining Yin, and Zhenyu Chen. 2022. DeepState: Selecting Test Suites to Enhance the Robustness of Recurrent Neural Networks. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 598–609. https://doi.org/10.1145/3510003.3510231

[27] Matt McFarland. 2020. Uber self-driving car operator charged in pedestrian death. https://www.cnn.com/2020/09/18/cars/uber-vasquez-charged/index.html.

[28] Microsoft. 2017. MS COCO. https://cocodataset.org/.

[29] Dim P. Papadopoulos, Jasper Uijlings, Frank Keller, and Vittorio Ferrari. 2016. We don't need no bounding-boxes: Training object class detectors using only human verification. *computer vision and pattern recognition* (2016).

[30] 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.

[31] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28 (2015).

[32] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[33] Soumya Roy, Asim Unmesh, and Vinay P Namboodiri. 2018. Deep active learning for object detection.. In *BMVC*. 91.

[34] Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. 2013. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229* (2013).

[35] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).

[36] Hao Su, Jia Deng, and Li Fei-Fei. 2012. Crowdsourcing annotations for visual object detection. In *Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[37] Manika Tyagi and Sona Malhotra. 2014. Test case prioritization using multi objective particle swarm optimizer. In *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT 2014)*. IEEE, 390–395.

[38] Zan Wang, Hanmo You, Junjie Chen, Yingyi Zhang, Xuyuan Dong, and Wenbin Zhang. 2021. Prioritizing Test Inputs for Deep Neural Networks via Mutation Analysis. In *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 397–409. https://doi.org/10.1109/ICSE43902.2021.00046

[39] Jiaxi Wu, Jiaxin Chen, and Di Huang. 2022. Entropy-based Active Learning for Object Detection with Progressive Diversity Constraint. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 9397–9406.

[40] Weiping Yu, Sijie Zhu, Taojiannan Yang, and Chen Chen. 2022. Consistency-based active learning for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 3951–3960.

[41] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. 2020. A Survey of Autonomous Driving: Common Practices and Emerging Technologies. *IEEE Access* 8 (2020), 58443–58469. https://doi.org/10.1109/ACCESS.2020.2983149

[42] Jie M. Zhang, Mark Harman, Lei Ma, and Yang Liu. 2022. Machine Learning Testing: Survey, Landscapes and Horizons. *IEEE Trans. Software Eng.* 48, 2 (2022), 1–36. https://doi.org/10.1109/TSE.2019.2962027

[43] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld. 2003. Face Recognition: A Literature Survey. *ACM Comput. Surv.* 35, 4 (dec 2003), 399–458. https://doi.org/10.1145/954339.954342

[44] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. 2019. Object Detection in 20 Years: A Survey. *arXiv: Computer Vision and Pattern Recognition* (2019).