An Empirical Study to Repair Deep Object Detectors

by

Yuxin Yue

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

in

Software Engineering and Intelligent System

Department of Electrical and Computer Engineering
University of Alberta

Abstract

Visual object detection predicts the categories of objects in an image and estimates bounding boxes that can wrap those objects accurately, playing a crucial role in many vision-based AI systems like autonomous cars, robotics, and smart monitoring. Although achieving significant progress, even the state-of-the-art (SOTA) detectors could inevitably raise errors when we deploy them in the real-world application scenario due to the potential distribution shift between the training dataset and testing data. For example, the detectors trained on a clean dataset usually yield a significant precision reduction under the corrupted images (*e.g.*, noisy or rainy images). Such an error type is often inevitable in the operational environment where the system is deployed, and cannot be completely solved by modifying the network architecture or optimization algorithm either since the deployed detectors could unavoidably encounter unseen corruptions.

A promising and practical solution is to use the examples captured in the operational environment to guide the updating of the object detector and avoid similar errors. We, in this paper, denote such a task as ***detector repairing*** (DetRepair). So far, there are few works studying object detection from the repairing perspective, which hinders the deployment of SOTA object detectors and a better understanding of their limitations. To bridge this gap, we conduct an in-depth and extensive empirical study to benchmark the object detection repairing techniques for DetRepair tasks. Our goal is to understand the current status, limitations, and challenges of DetRepair, and to identify future opportunities for the research community.

Specifically, we build a detector repairing benchmark by considering different failure

patterns of objects and different error types and by collecting state-of-the-art detectors and various corruptions. The analysis enables us to gain a deep understanding and insight into the failure patterns and the underlying potential causes. Then based on clues from the analysis, we systematically and comprehensively study a series of repairing schemes and conduct extensive experiments on different schemes based on the constructed benchmark. We notice many inspirational facts on repairing schemes for object detectors. With all the above efforts and the evaluation results, we can understand the significance and challenges of this task and the strengths and weaknesses of different solutions. Furthermore, we propose several potential future directions based on our benchmark, opening new doors for developing robust object detectors.

Preface

This thesis is conducted under the supervision of Prof. Lei Ma. It's based on my work named as *How to Repair Object Detectors? A Benchmark, an Empirical Study, and a Way Forward*, which has been submitted to the *Software Engineering In Practice* track in *International Conference on Software Engineering(ICSE) 2024* under review. The article was completed in collaboration with Zhijie Wang, who helped with writing, Prof. Qing Guo, who contributed a lot to the structure and the manuscript, and Prof. Felix Juefei Xu, who offered valuable advice and assistance.

Acknowledgements

I would like to first thank my supervisor Prof. Lei Ma for his great kindness and support. I would also like to offer my sincere gratitude to my collaborators, Dr. Qing Guo and Dr. Felix Juefei Xu. I benefit so much from their enthusiasm, professionalism, and research insights. In addition, I'm grateful to my friends, Shuangzhi for the inspiration of denoising, Yuan and Xuan for mathematical tools and details, Zhijie for discussion and writing, Jiayang for figures and visualization, Yuheng for discussion, and all other members of the Momentum Lab for their help. Besides, I'm grateful to all the professors who gave very awesome lectures where I obtained knowledge and inspiration.

Contents

List of Tables

List of Figures

List of Symbols

Latin

A Architecture of a model.

$\mathbf{b}_i = (x_i, y_i, h_i, w_i)$ A bounding box annotated with its center point, height, and width.

D A detector(detection model).

I The input image.

$\mathbf{o}_i = (\mathbf{b}_i, c_i)$ An object denoted with its bounding box and category.

W Weights of a model.

$\mathscr{D}$ A dataset.

$\mathscr{D}^{\text{fail}}$ The failure dataset.

$\mathscr{D}_{\text{test}}$ The clean test dataset.

$\mathscr{D}_{\text{test}}^{\text{fail}}$ The failure test dataset.

$\mathscr{D}_{\text{train}}$ The original training dataset.

$\mathscr{D}_{\text{train}}^{\text{fail}}$ The failure training dataset.

$\mathscr{O} = \{\mathbf{o}_i\}_{i=1}^{N}$ The set of all objects inside one image.

$\mathscr{P} = \{\mathbf{b}'_j\}_{j=1}^{M}$ The proposals.

H Height of the input image.

W Width of the input image.

Greek

$\Phi(\cdot)$ The region of interest (RoI) feature extraction module

$\psi_{\text{cls}}(\cdot)$ The region classification module for the RoIs

$\psi_{\text{reg}}(\cdot)$ The bounding box regression module for the RoIs

$\varphi(\cdot)$ The deep image representation module

Abbreviations

AP Average Precision, evaluation metrics for object detection.

BN Batch Normalization, a type of operator in neural networks.

FPN Feature Pyramid Network.

NMS Non-Maximum Suppression. A technique to remove duplicated bounding box predictions.

R-CNN Region-based Convolutional Neural Networks.

RoI Regions of interest.

Glossary

Bounding Boxes. A bounding box is an axis-aligned tight box of an object in the image. A bounding box in an image is a 4-d vector, usually represented by its center point coordinates, height, and width.

Classification. The classification branch in the object detector serves to predict the object category of the RoI.

Corruption. Noise or distortion in the input images, e.g., Gaussian Noise, glass blur, fog.

Proposals (RoIs) A proposal in an image is a pre-defined region that is likely to contain one or no object of interest.

Regression. Refining predefined to obtain the bounding box of a RoI by predicting the relative offsets.

The RoI feature extraction module. The region of interest (RoI) feature extraction module generates features for each RoI.

The deep image representation module. The deep image representation module aims to extract features from the input image.

The input image The input image is usually a 3-d vector from an RGB digital image.

Chapter 1

Introduction

1.1 Motivation

Object detection [1] is a critical perception task in deep learning and general AI systems. It serves as the upstream task for many downstream tasks, such as object tracking [2], instance segmentation [3], pose estimation [4], *etc.* Moreover, it plays a significant role in real-world applied deep learning systems, such as autonomous driving, robotics, the smart city, video surveillance, and healthcare. Compared to object recognition, object detection is inherently more complex and challenging since it involves both classification and localization. However, similar to other machine learning models, deep object detectors are vulnerable to distortions, weather conditions, and adversarial perturbations that can frequently occur in the operational environment. Previous research [5, 6] have shown that models can experience performance degradation with corrupted input images for image classification and object detection. Furthermore, despite many detectors achieving high accuracy and performance, they still face challenges when deployed in real-world environments.

In the realm of traditional software development, program repair is a process that is typically conducted either manually or automatically subsequent to the detection of bugs during testing or deployment. However, in the context of the data-driven iterative model development process that underlies the AI industry, bug repair is a critical concern that frequently fails to receive adequate attention. For instance, consider a well-trained object detector that functions flawlessly within an application but abruptly fails to operate as

intended under unexpected conditions. In such instances, machine learning researchers and practitioners are obliged to undertake detector repair in order to forestall any further loss that may occur in the operational environment. Hence, it is paramount to prioritize the issue of bug repair and to develop effective approaches to resolving such challenges within the context of AI systems.

Different from traditional robustness enhancement, we focus on improving object detectors given images under a certain unknown common corruption instead of for general unknown cases. Real-world applications present numerous challenges that can cause object detectors to fail. For instance, if objects in the deployed environment differ significantly in scale from those in the training dataset, the detector may not be able to detect extremely large or small objects. Unseen background objects can also cause detection failures, such as dogs being detected as pedestrians by surveillance detectors. This is because the training set only includes categories of interest, such as pedestrians and vehicles, but fails to capture challenging background samples. Furthermore, illumination variation can pose a critical challenge for object detectors. High-quality images and videos captured at night are more difficult to obtain than those captured during the day, resulting in scarce datasets. Additionally, objects in night scenarios are often blurry and less salient, and digital captures in low-light environments tend to be noisier due to the need for higher ISO levels. Hence, multifarious factors, such as changes in illumination and weather patterns, present significant challenges to the development of robust object detection models.

Among these factors, the issue of image-wise corruptions has received significant attention in the realm of robust computer vision. In this thesis, we concentrate on repairing object detectors that fail due to such corruptions, which are prevalent in the operational environment. However, the number of failure samples is often small, as collecting such samples is a challenging task. Additionally, we make an assumption that there exist unknown deterministic connections between the failure data and the original data, similar to a previous work [7], based on industrial demands. In contrast to recent related works that focus on

unsupervised or weakly supervised problems [8, 9], we aim to systematically investigate and address corner cases to ensure the robustness and safety of object detection systems. While annotations for supervised training are often readily available, our focus is not on such cases but rather on repairing detectors in challenging real-world scenarios.

1.2 Thesis Objectives

In this thesis, our goal is to obtain a comprehensive understanding of state-of-the-art (SOTA) techniques for object detection repair. To accomplish this, we establish a benchmark of repairing techniques for SOTA object detectors, which we evaluate under various common corruptions, followed by a detailed analysis of the challenges and opportunities in this field. Our objective is to move beyond an empirical study and achieve a comprehensive understanding of repair in object detection. Through a large-scale evaluation and comparative study, we investigate the potential failures of these object detectors from various angles, such as corruptions, models, failure datasets, and error types. This analysis allows us to gain a deep understanding of the failure patterns and underlying potential causes. Based on this understanding, we propose potential approaches for repairing object detectors and expand the benchmark with our techniques for more comprehensive and comparative evaluation. Through a large-scale comparative evaluation, we identify key clues and problems that will be important for future work in this field.

Overall, this work aims to study object detection repair comprehensively and to highlight the challenges and opportunities in this area. We also provide a benchmark for repairing object detectors, which will enable further research and study along this line.

In summary, this thesis makes the following contributions:

- **Benchmark**. We establish the first benchmark for repairing deep object detectors against common image corruptions through a systematic and comprehensive evaluation of SOTA techniques, providing a common ground for the study and analysis in this direction.

- **Empirical Study.** On the basis of our benchmark, we perform an empirical study to better understand the SOTA repairing approaches' efficacy and efficiency.
- **Discussion.** Following the findings in the empirical study, we pose discussions and pinpoint challenges and opportunities for further research on detector repairing.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 introduces background information. We first introduce the object detection framework and milestone detection models. Then, we provide the formal definition of the failures in object detection. Chapter 3 then details the research topic based on my submitted work *How to Repair Object Detectors? A Benchmark, an Empirical Study, and a Way Forward*. We discuss potential techniques for repairing object detectors, after which the remaining sections are about evaluation results and our in-depth analysis. Later we provide the results of the large-scale comparative evaluation and results on diverse object detection repairing techniques. Based on this, we further discuss the challenges and future research opportunities of repair in object detection, followed by the conclusion of this work in Chapter 4.

Chapter 2

Background

2.1 General Information

2.1.1 Object Detection and Common Metrics for Evaluation

Object detection plays a key role in many vision-intelligent tasks, *e.g.*, it is the upstream task of object tracking [2], instance segmentation [3], pose estimation [4], *etc.* Given an input image denoted as $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$, a detector aims to localize the objects within the image and predict their categories. We denote the ith object as $\mathbf{o}_i = (\mathbf{b}_i, c_i)$, where the $\mathbf{b}_i = (x_i, y_i, h_i, w_i)$ represents a bounding box tightly wrapping the object and c_i is the object's category. Moreover, we use $\mathscr{O} = \{\mathbf{o}_i\}_{i=1}^{N}$ to denote the set of all objects in the image. To obtain a better understanding of the task of repairing object detectors, it is important to first introduce the existing general detection framework in Sec. 2.1.1. Based on this, we further discuss some milestone detection techniques in Sec. 2.1.1. We include the common and widely used evaluation metrics in Sec. 2.1.1.

Detection Framework

The main challenges of accurate object detection stem from four aspects:

(1) The large number of tentative object categories. The detector needs to have discrimination capability, especially under many open background objects.

(2) The intra-category variations caused by environmental conditions and intrinsic factors,

e.g., shape variations handling require the detector to be robust enough.

(3) Difficulty in achieving high-quality tight bounding boxes, where accurate box regression is demanded.

(4) The detection techniques often need to be efficient at real-time speed so that to be adopted in real-world efficiency-sensitive applications, *e.g.*, autonomous driving, and robotics.

Object scale variance is a critical issue for object detection, closely related to the aforementioned challenges. It encompasses two aspects: scale variance within a single image and scale variance across different images. Within a single image, objects of various scales can coexist, ranging from small to large. Detecting objects accurately across this scale range poses a challenge as the detector needs to effectively handle objects of different sizes and maintain consistent performance. Across different images, the same object may appear at different scales due to variations in camera angle, camera distance, or other factors. This variability in object scale further complicates the detection task as the detector needs to be robust enough to handle such scale variations and consistently identify the object regardless of its size. Object scale variance is not only a challenge but also an important consideration for lightweight detectors. In resource-constrained scenarios, such as embedded systems or edge devices, the detector's efficiency and computational complexity become crucial. Balancing the detection accuracy and the computational requirements becomes critical when addressing the issue of object scale variance in lightweight detectors.

A naive object detection framework is a natural extension of the image classification method, which employs the classifier to predict the categories of sub-regions sampled from the input image. For example, traditional object detection conducts the classification on sub-regions sampled by the sliding-window strategy [10–12]. However, such a framework still falls far from the aforementioned four requirements. Over the past few years, with the fast progress of cutting-edge deep learning techniques, researchers have performed

extensive research to design and enhance detectors in different directions, *e.g.*, enhancing detectors' discrimination powers, addressing the diverse object appearances, and achieving real-time running speeds. These modern object detectors are mostly based on the regions of interest (RoI). Given an image $\mathbf{I}$, a detector can sample regions of interest (*i.e.*, object proposals) that are usually rectangle samples in $\mathbf{I}$ and can be represented via the bounding boxes as $\mathscr{P} = \{\mathbf{b}'_j\}_{j=1}^{M}$, and then predict their categories. More specifically, we summarize existing detection methods comprised of three key components, including the deep image representation denoted as $\varphi(\cdot)$, region of interest (RoI) feature extraction denoted as $\Phi(\cdot)$, and region classification and bounding box regression, which are named as $\psi_{\text{cls}}(\cdot)$ and $\psi_{\text{reg}}(\cdot)$, respectively.

In the object detection process, a detector performs several steps to identify and localize objects in an input image. First, the detector samples a set of Regions of Interest (RoIs) denoted as $\mathscr{P}$ from the input image. These RoIs represent potential object locations in the image. Next, the detector extracts the features of these RoIs using a function $\Phi(\cdot)$ that takes as inputs the deep representation network $\varphi(\cdot)$, the input image $\mathbf{I}$, and the RoIs $\mathscr{P}$. The deep representation network, also known as the backbone network, is typically derived from a pre-trained classification network and serves as a feature extractor from the images. Finally, the output features of each RoI are further fed to the box prediction module $\psi_{\text{reg}}(\cdot)$ and classification prediction module $\psi_{\text{cls}}(\cdot)$, respectively.

To be specific, we start with the well-known region-based convolution neural network (R-CNN) [13, 14] as an example for discussion. Initially, the algorithm generates a set of M RoIs $\mathscr{P} = \{\mathbf{b}'_i\}_{i=1}^{M}$ using selective search [15], which identifies potential object locations in the image. These RoIs are then individually processed by a deep convolutional neural network (CNN) $\varphi(\cdot)$ to extract features specific to each region. The RoI feature extraction function $\Phi(\cdot)$ in the R-CNN algorithm involves two main steps. (1) It crops the image to obtain sub-regions and resizes the sub-regions to fit the input size of $\varphi(\cdot)$. (2) R-CNN uses the $\varphi(\cdot)$ to extract the features of each sub-region and get the M RoI features denoted as

$\mathscr{F} = \{\mathbf{F}_i\}_{i=1}^{M}$. We then summarize the whole detection process as

$$\begin{aligned} \mathscr{F} &= \Phi(\mathbf{I}, \mathscr{P}, \varphi) = \{\mathbf{F}_i = \varphi(\mathbf{X}_i)\}_{i=1}^{M}, \\ \text{s.t. } \mathbf{X}_i &= \text{CropResize}(\mathbf{I}, \mathbf{b}_i'), \forall \mathbf{b}_i' \in \mathscr{P}. \end{aligned} \tag{2.1}$$

To further enhance the performance of the object detection models, a deep representation network is first typically pre-trained on large-scale datasets such as ImageNet [16], which contains a vast collection of images spanning various object categories. The purpose of pre-training on ImageNet is to allow the network to learn general features that are relevant to a wide range of computer vision tasks. Once the deep representation network is pre-trained, it can be fine-tuned on specific object detection datasets, where the network learns to refine its features and further enhance the ability to detect and classify objects accurately.

Meanwhile, R-CNN trains a support vector machine (SVM) and a regressor as the $\psi_{\text{cls}}(\cdot)$ and $\psi_{\text{reg}}(\cdot)$, respectively. With the two functions, R-CNN feeds the extracted features of each RoI to the two functions, respectively, and outputs the refined bounding box (*i.e.*, $\mathbf{b}_i$) and category (*i.e.*, c_i) of that RoI (*i.e.*, $\mathbf{o}_i = (\mathbf{b}_i, c_i)$). If the region does not contain any object, c_i will be the background and the predicted bounding box will be discarded. The module can be represented as

$$c_i, s_i = \psi_{\text{cls}}(\mathbf{F}_i), \mathbf{b}_i = \psi_{\text{reg}}(\mathbf{F}_i, \mathbf{b}_i'), \forall \mathbf{b}_i' \in \mathscr{P}, \tag{2.2}$$

where s_i is the confidence score of the category prediction.

As the early work of using deep neural networks for object detection, R-CNN provides an intuitive idea but with obvious limitations: (1) The training method is a multi-stage strategy, which is slow and hard to optimize. (2) It's not efficient to crop and forward patches due to the large number of overlapping RoIs. Inspired by R-CNN, researchers developed a series of novel techniques to continuously address the efficiency issue and improve accuracy. Based on the modern object detection framework, a number of detection techniques are proposed to improve object detection from multiple aspects, *e.g.*two-stage (and multi-stage) detectors [17–21] *v.s.* single-stage detectors [22–28], anchor-based detectors *v.s.* anchor-free

detectors [29–32], CNN-based detectors *v.s.* multi-head-attention-based and transformer-based detectors [33–36], *etc.* In the following, we mainly discuss four recently developed representative detection methods, whose architectures are shown in Fig. 2.1.

Representative Detection Methods

Faster R-CNN. To address the key limitations of R-CNN, Girshick [17] develops Fast R-CNN that improves R-CNN's efficiency and accuracy by constructing an end-to-end network that includes a two-branch architecture for class prediction and bounding box regression, respectively. Meanwhile, it employs RoI pooling to enable extracted RoIs to share the deep feature calculation process and significantly reduce the computation. Later, [18] further pushes the efficiency of Fast R-CNN forward by proposing to remove the explicit RoI extraction process and replace it with a CNN, *i.e.*, region proposal network (RPN). Specifically, beyond extracting the features of RoIs independently like Eq. (2.1), Faster R-CNN designs a new RoI feature extraction process

$$\begin{aligned} &\mathscr{F} = \Phi(\mathbf{I}, \varphi) = \{\mathbf{F}_i = \text{RoIPool}(\varphi(\mathbf{I}), \mathbf{b}'_i)\}_{i=1}^{M}, \\ &\text{s.t. } \{\mathbf{b}'_i\}_{i=1}^{M} = \text{RPN}(\varphi(\mathbf{I})), \end{aligned} \tag{2.3}$$

where $\text{RPN}(\cdot)$ denotes the region proposal network and takes the feature of the whole input image (*i.e.*, $\varphi(\mathbf{I})$) as input, which addresses the feature via a convolution layer followed by two 1×1 convolutions for objectness classification head and regression head, respectively. Note that, the outputs of the two heads are defined based on a set of reference boxes called anchors. The anchors are pre-defined bounding boxes with different scales and aspect ratios and can cover objects with different shapes. Finally, the RPN can output feature-level RoIs (*i.e.*, $\mathbf{b}'_i$) that are filtered by non-maximum suppression (NMS) based on the predicted objectness scores. These RoIs are used to extract the RoI features via the RoIPooling layer.

Compared with the RoI feature extraction of R-CNN in Eq. (2.1), Faster R-CNN adopts the RPN to replace the search selective method and generate RoIs based on the feature representation of the whole image. Meanwhile, the extracted RoI features are fed to a

sequence of fully convolutional layers like the Fast RCNN, followed by a two-branch architecture: a softmax classifier for object category prediction, and class-specific bounding box regression offsets for proposal refinement. Such a design enables Faster R-CNN for end-to-end training, which is much faster than the training process of traditional R-CNN.

Feature Pyramid Network (FPN). [37] proposes the feature pyramid network (FPN) for a more powerful deep representation that employs the inherent multi-scale and pyramidal hierarchy structures of the deep convolution network to extract pyramidal features of input images. Specifically, based on a deep convolution neural network (CNN) like ResNet [38], FPN consists of a bottom-up pathway, a top-down pathway, and lateral connections. The bottom-up pathway outputs multi-scale features from different layers of the CNN while the top-down pathway upsamples the smallest feature to a higher resolution feature that is combined with the bottom-up's features with the same resolution via the element-wise addition. This process is recurrently conducted until the finest resolution is achieved. [37] equips the deep representation (*i.e.*, $\varphi(\cdot)$) of Faster R-CNN with the FPN architecture, achieving much higher detection accuracy, that is, we have

$$\begin{aligned} &\mathscr{F} = \Phi(\mathbf{I}, \varphi) = \{\mathbf{F}_i = \text{RoIPool}(\text{FPN}(\varphi(\mathbf{I})), \mathbf{b}'_i)\}_{i=1}^{M}, \\ &\text{s.t. } \{\mathbf{b}'_i\}_{i=1}^{M} = \text{RPN}(\text{FPN}(\varphi(\mathbf{I}))). \end{aligned} \tag{2.4}$$

The extracted features are further fed to the two-branch architecture to predict the category and regress the bounding box for each RoI as done in Faster R-CNN.

RetinaNet. Although Fast R-CNN, Faster R-CNN, and FPN have enhanced the detection accuracy as well as the running speed of the raw R-CNN significantly, it is still a two-stage detection framework that can be less efficient and may neglect some potential spatial locations in the extracted features. Therefore, [22] proposes a one-stage detector, *i.e.*, RetinaNet, based on the deep representations extracted from the FPN, which removes the RoIPooling layer and regresses the features to a category tensor and a bounding box tensor

straightforwardly. It is worth noting that RetinaNet conducts a dense sampling process where each location of the feature corresponds to A RoIs that all center at that location with different aspect ratios and scales, also known as A anchors. Specifically, we can reformulate the RoI feature extraction process as

$$\begin{aligned}
&\mathscr{F} = \Phi(\mathbf{I}, \varphi) = \{\mathscr{F}_s\}_{s=1}^{S}, \\
&\mathscr{F}_s = \{\mathbf{F}_{s,i} = \tilde{\mathbf{F}}_s[x_i', y_i'] | \mathbf{b}_i' = (\frac{H}{H_s}x_i', \frac{W}{W_s}y_i', \frac{H}{H_s}h_i', \frac{W}{W_s}w_i'), \\
&\forall (x_i', y_i') \in [(1,1), (H_s, W_s)], \forall (h_i', w_i') \in \mathscr{A}\}, \\
&\text{s.t. } \{\tilde{\mathbf{F}}_s\}_{s=1}^{S} = \text{FPN}(\varphi(\mathbf{I})),
\end{aligned} \tag{2.5}$$

where $\text{FPN}(\cdot)$ extracts features of the input image with S different scales (*i.e.*, $\{\tilde{\mathbf{F}}_s\}_{s=1}^{S}$). H_s and W_s are the height and width of the feature $\tilde{\mathbf{F}}_s$. For each location of $\tilde{\mathbf{F}}_s$ (*i.e.*, (x_i', y_i'), a $\frac{H}{H_s} \times \frac{W}{W_s}$ region of the input image), we have A anchors denoted as the set $\mathscr{A}$. Then, the Eq. (2.5) actually extracts the features of $H_s \cdot W_s \cdot A$ RoIs for each scale, and the anchors at each location share the same feature (*i.e.*, $\mathbf{F}_{s,i}$). The final set $\mathscr{F}_s$ for the s scale can form a tensor that is fed to two sub-networks for classification and bounding box regression, respectively. Compared with the Faster R-CNN or FPN, RetinaNet does not include the region proposal network and predicts the categories and tight bounding boxes for each feature location directly. Specifically, for the sth scale feature, there are two branches for classification and bounding box regression, respectively, *i.e.*,

$$\mathbf{C}_s = \psi_{\text{cls}}^s(\mathscr{F}_s), \mathbf{B}_s = \psi_{\text{reg}}^s(\mathscr{F}_s, \{\mathbf{b}_i'\}), \tag{2.6}$$

where $\mathbf{C}_s \in \mathbb{R}^{H_s \times W_s \times A \times C}$ and $\mathbf{B}_s \in \mathbb{R}^{H_s \times W_s \times A \times C \times 4}$, that is, $\mathbf{C}_s$ contains C category probabilities of all spatial locations of the sth scale feature and A anchors. $\mathbf{B}_s$ contains all regressed bounding boxes centered at all locations with different anchors.

Fully convolutional one-stage object detection (FCOS). [29] further introduces the semantic segmentation structure, *i.e.*, fully-convolutional network [39], to the object detection framework, which is anchor-free, reduces the number of anchor-related parameters,

and avoids the anchor-related complicated computation. In particular, FCOS follows a similar RoI feature extraction process with RetinaNet but adopts different classification and bounding box regression branches. Specifically, with S features $\mathscr{F}_s$ at different scales, FCOS sets three branches for each scale, that is, ψ^s_{cls}, ψ^s_{reg}, and ψ^s_{ctr}. The three branches output three tensors having the same resolutions as the input feature, that is,

$$\mathbf{C}_s = \psi^s_{\text{cls}}(\mathscr{F}_s), \mathbf{B}_s = \psi^s_{\text{reg}}(\mathscr{F}_s, \{\mathbf{b}'_i\}), \mathbf{T}_s = \psi^s_{\text{ctr}}(\mathscr{F}_s), \tag{2.7}$$

where $\mathbf{C}_s \in \mathbb{R}^{H_s \times W_s \times C}$, $\mathbf{B}_s \in \mathbb{R}^{H_s \times W_s \times 4}$, and $\mathbf{T}_s \in \mathbb{R}^{H_s \times W_s \times 1}$. $\mathbf{C}_s$ indicates the category probability of each location in the feature, which is similar to semantic segmentation. $\mathbf{B}_s$ means the absolute offsets from each location to the corresponding object center. $\mathbf{T}_s$ denotes the normalized distance from each location in the feature to the center of an object. Compared Eq. (2.7) with Eq. (2.6), we can see FCOS is not based on the anchor but treats it more like a semantic segmentation problem, which further benefits the accuracy.

Detection Transformer (DETR). [33] employs the transformer to detect objects due to the high capability of the transformer to perceive global information. Specifically, DETR uses a CNN as the deep representation $\varphi(\cdot)$ to extract the deep feature of the input image, which is fed to a transformer. The transformer could be regarded as the RoI feature extraction module (*i.e.*, the function $\Phi(\cdot)$) and outputs a fixed number of features for different RoIs. After that, each feature is further passed to a shared feed-forward network (FFN) that estimates the normalized center coordinates, height, and width of the bounding box *w.r.t.* the input, and also predicts the class label using a softmax function. DETR is also anchor-free but needs to pre-define a number to detect in an image.

Evaluation Metrics

Given a testing dataset and a detector, we employ the precision, recall, and average precision (AP) for accuracy evaluation. To make it more clear, we start with the definition of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) samples. Given

a detector and a testing image, we have the predicted objects denoted as the set $\mathscr{O} = \{\mathbf{o}_i = (\mathbf{b}_i, c_i)\}_{i=1}^{N}$ and we also have the ground truth object set denoted as $\mathscr{O}^* = \{\mathbf{o}_j^* = (\mathbf{b}_j^*, c_j^*)\}_{j=1}^{N^*}$, where the N^* is the real object number that may be different from the predicted number N. The predicted object (*i.e.*, $\mathbf{o}_i = (\mathbf{b}_i, c_i)$) is positive if and only if the following conditions are satisfied:

(1) There is a ground truth object having high bounding-box overlap with the predicted object (*i.e.*, $\mathbf{o}_j^* = (\mathbf{b}_j^*, c_j^*)$), that is, $\text{IoU}(\mathbf{b}_i, \mathbf{b}_j^*) > \delta$, where δ is a pre-defined threshold and usually set as 0.5.

(2) The category of the predicted object is the same as the one of the ground truth object, that is, $c_i = c_j^*$.

(3) A ground truth object can only match the predicted object with the highest score.

(4) If there exist multiple ground truth objects satisfying (1)-(3), the predicted object will only match the one with the highest IoU.

With the above definition of a positive predicted object, we can count the number of true positive (TP), false positive (FP), and false negative (FN) samples, and further define the precision, recall, and average precision (AP) as follows,

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{2.8}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{2.9}$$

$$\text{AP} = \frac{1}{N^*} \sum_{i=1}^{N} \frac{\text{TP}_i}{i}, \tag{2.10}$$

where TP_i is the number of true positive samples in the first i predicted objects. Note that, with different thresholds (*i.e.*, δ) for IoU evaluation, we get different AP scores and rename AP as AP@δ. For example, we usually use AP@0.5 and AP@0.75 for the AP scores under $\delta = 0.5$ and $\delta = 0.75$, respectively. A larger threshold can lead to a more strict overlap calculation and a lower AP score. In the COCO dataset [40], we usually compare the

averaging AP scores from $\delta = 0.5$ to $\delta = 0.95$. Besides, to conduct more comprehensive evaluations, we also define the AP scores for small, medium, and large objects, which are denoted as AP@S, AP@M, and AP@L, respectively, if we only consider the objects within a certain area range.

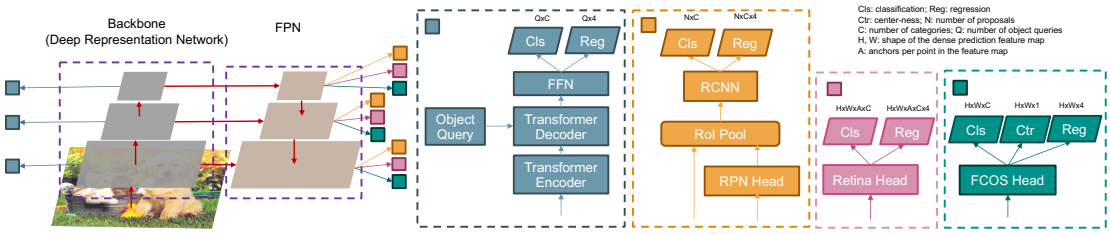


Figure 2.1: Framework of detectors. DETR, Faster R-CNN, RetinaNet, and FCOS are illustrated here.

2.2 Specific Information

2.2.1 Failure in Object Detection

Deploying a well-trained object detector in real-world scenarios can be challenging due to various degradation factors and common corruptions present in the operational environment, *e.g.*, noise, blur, rain, *etc.* These factors can significantly impact the detector's performance and lead to errors. To this end, we first introduce some common degradation types in the Sec. 2.2.1 that usually occur in the real world and will be used in subsequent studies. Meanwhile, we provide the failure definition of object detection in Sec. **??**.

Common Degradation

Previous works [5, 6] test deep neural networks on corrupted datasets and claim performance degradation and limited robustness. We would like first to confirm the existence of performance degradation, which is a critical pre-condition for repair. We perform evaluations of all the 48 collected and prepared models of RetinaNet, Faster R-CNN, FCOS, and DETR in MMDetection [41] model zoo across 15 corruption types including brightness, contrast, defocus blur, elastic transform, fog, frost, gaussian noise, impulse noise, snow, pixelate, jpeg

compression, motion blur, shot noise, and zoom blur. These corruptions are natural factors and usually appear in daily life. Hence, it is critical to study the accuracy of detectors under these corruptions. Intuitively, we follow the setups of [6] and add each corruption to the raw testing dataset of COCO. As a result, we have 15 corrupted testing datasets. We obtain a box plot with AP scores of all 48 detection models of interest over corruptions as shown in Fig. 2.2. The last column shows the performance of models on the clean test set. A higher AP score means better performance. Based on the figure, we can easily conclude that the detectors' performance drops when the input images are corrupted.

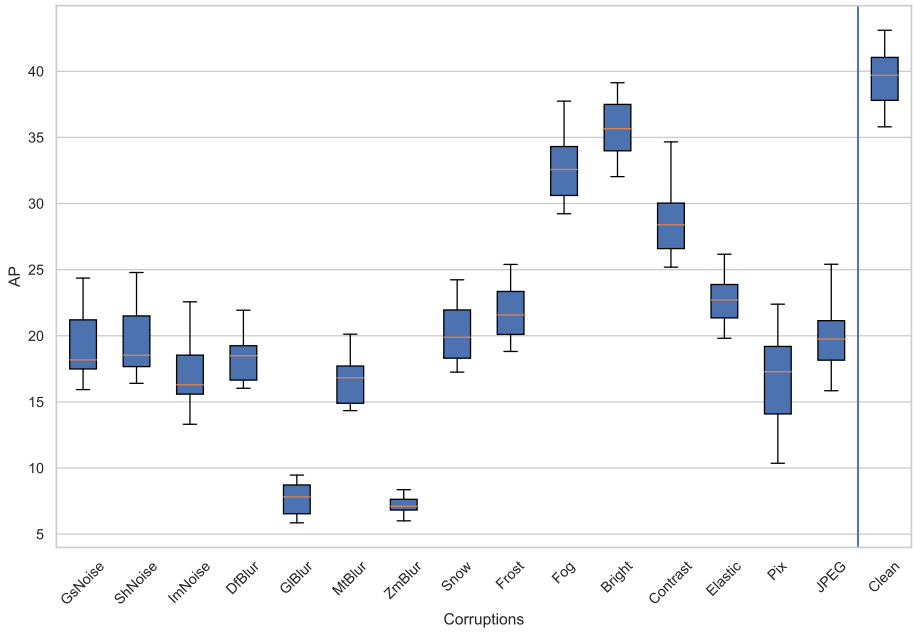


Figure 2.2: AP scores of detectors after corruptions. The last box shows those without any corruption. Performance degradation can be observed across corruptions.

2.2.2 Problem Formulation

We represent a deep object detector as $\mathbf{D} = \{\mathbf{W}, \mathbf{A}\}$ notated by its weights $\mathbf{W}$ and architecture-related parameters $\mathbf{A}$. The detector is trained with clean training dataset $\mathscr{D}_{\text{train}}$ by optimizer J and is tested with on clean testing dataset denoted as $\mathscr{D}_{\text{test}}$.

$$\mathbf{D} = \text{argmin}_{\mathbf{D}^*} J(\mathscr{D}_{\text{train}}, \mathbf{D}^*). \tag{2.11}$$

The trained deep detector obtains high detection accuracy on the clean test dataset $\mathscr{D}_{\text{test}}$, whose accuracy decreases when a type of natural corruption appears and is added to the clean testing dataset $\mathscr{D}_{\text{test}}$, leading to a corrupted testing dataset $\mathscr{D}^{\text{c}}_{\text{test}}$. Note that, we consider the situation where corruption patterns of all examples in the corrupted dataset belong to the same type. After evaluating the detector on $\mathscr{D}^{\text{c}}_{\text{test}}$, we can collect failure examples where the detector fails (See Sec. 2.2.1) and construct a dataset $\mathscr{D}^{\text{fail}}$. Then, we further split the failure dataset $\mathscr{D}^{\text{fail}}$ into failure training subset $\mathscr{D}^{\text{fail}}_{\text{train}}$ and failure testing subset $\mathscr{D}^{\text{fail}}_{\text{test}}$. We use the function $\text{AP}(\mathscr{D}, \mathbf{D})$ to calculate the performance of the detector $\mathbf{D}$ on the dataset $\mathscr{D}$. Then, we can evaluate the detector on the clean testing dataset $\mathscr{D}_{\text{test}}$, corrupted testing dataset $\mathscr{D}^{\text{c}}_{\text{test}}$, and the failure testing dataset $\mathscr{D}^{\text{fail}}_{\text{train}}$, and obtain $\text{AP}(\mathscr{D}_{\text{test}}, \mathbf{D})$, $\text{AP}(\mathscr{D}^{\text{c}}_{\text{test}}, \mathbf{D})$, and $\text{AP}(\mathscr{D}^{\text{fail}}_{\text{train}}, \mathbf{D})$, respectively. In general, we have the following results: ❶ $\text{AP}(\mathscr{D}_{\text{test}}, \mathbf{D}) \gg \text{AP}(\mathscr{D}^{\text{c}}_{\text{test}}, \mathbf{D})$, indicating that the corruptions can lead to significant accuracy degradation; ❷ We have $\text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D}) < \text{AP}(\mathscr{D}^{\text{c}}_{\text{test}}, \mathbf{D})$ since the $\mathscr{D}^{\text{fail}}$ is a part of $\mathscr{D}^{\text{c}}_{\text{test}}$ and fails the detector. ❸ We also have $\text{AP}(\mathscr{D}^{\text{fail}}_{\text{train}}, \mathbf{D}) \approx \text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D})$ since they are sampled from the $\mathscr{D}^{\text{fail}}$ in an independent way.

Given a pre-trained detector $\mathbf{D}$, the detection repair aims to update the detector's weights $\mathbf{W}$ or architecture $\mathbf{A}$ to get much higher accuracy on the failure test dataset $\mathscr{D}^{\text{fail}}_{\text{test}}$ while maintaining the accuracy on the original clean dataset $\mathscr{D}_{\text{test}}$. Such a repair problem commonly occurs in practice, and it is often important and necessary to repair a detector to avoid making similar errors when a type of corruption appears. In particular, the problem can be formulated as

$$\begin{aligned} \mathbf{D}' = &\text{argmin}_{\mathbf{D}^*} \Upsilon(\mathscr{T}(\mathscr{D}^{\text{fail}}_{\text{train}} \cup \mathscr{D}_{\text{train}}), \mathbf{D}^*), \\ &\text{s. t. } \text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D}') > \text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D}), \\ &\quad \text{AP}(\mathscr{D}_{\text{test}}, \mathbf{D}) - \text{AP}(\mathscr{D}_{\text{test}}, \mathbf{D}') < \varepsilon, \end{aligned} \tag{2.12}$$

where the function $\Upsilon(\cdot)$ represents a repair scheme, and ε is a small positive value. The function $\Upsilon(\cdot)$ denotes the repair optimization, the abstraction of model-oriented repair in

Sec. 3.3. $\mathscr{T}(\cdot)$ indicates how to combine the data, which leads to data-oriented repair in Sec. 3.3.

2.3 Discussion and Conclusion

The field of object detection has indeed made significant strides in academic and industrial applications. Previous attempts have developed the detection framework and various detector styles, contributing to the success of object detection systems. However, object detectors are vulnerable and can exhibit incorrect behavior when being deployed into operational environments. To fix the deficiency, one promising approach in both academic and industrial areas is DNN (deep neural networks) repair. Despite the substantial efforts dedicated to improving detection accuracy, the exploration of methods for repairing detectors has been relatively limited, with only a few research works delving into this crucial aspect. Therefore, it's important to take the first step of benchmarking DNN repair methods for object detectors.

Chapter 3

How to Repair Object Detectors? A Benchmark, an Empirical Study, and a Way Forward

3.1 Introduction

Object detection is a fundamental task in computer vision with widespread applications in many domains, including autonomous driving, surveillance, and robotics. Despite the success of deep learning-based object detectors in recent years, they are still prone to errors and inaccuracies due to common corruptions, *e.g.*, the inherent complexity and variability of object appearances, backgrounds, and lighting conditions. With the growing attention to robust and reliable object detection, it has become increasingly important to understand how to repair deep object detectors when they fail or underperform in challenging scenarios. Despite this need, there is currently no comprehensive and in-depth analysis of how to repair deep object detectors.

Common corruptions and distortions have been important issues in robust computer vision. Hence in this paper, we work on a specific but important scenario, where failure cases are caused by an **unknown** image-wise corruption (*e.g.*, ImageNet-C) [5] that received much attention recently. The aforementioned corruptions are widely used in various robustness-related efforts for perception. We attempt to repair the detectors under these common corruptions that could often occur in the operational environment. Nevertheless, the failure

dataset can still be small since failure samples are difficult to collect. We share similar assumptions of a previous work [7] that are based on industrial demands, *i.e.*, we have few supervised failure cases collected after the model is deployed in the operational environment, with unknown connections with the original data. In general, annotations for supervised training are often not difficult to obtain. Instead of focusing on unsupervised or weakly supervised problems like recent related works [8, 9], we aim to systematically investigate and deal with the corner cases to ensure the robustness and safety of the object detection systems.

While some previous works are related to DNN for image classifiers, it is still not clear how to repair the object detectors in a systematic way, *e.g.*, what the current status of existing techniques for repair is and what the gap is there, although there are some relevant work and direction that could be helpful and relevant to object detector repair.

Domain adaptation [8, 9] aims to improve the performance of a DNN in the target domain. It is usually assumed that sufficient unlabeled images are available in the target domain, while the scenarios we study only have very few labeled failure samples. As a result, domain adaptation methods may fall short when directly tasked to solve the repair problems. *Continual learning* methods [42] attempt to expand models to more tasks and they typically work on a task-based sequential learning setup. In our context, we focus on a single task with data from different domains. In other words, if we treat the task in different domains as different sub-tasks, the sub-tasks are not sequential but can be seen as connected or joint. Some works [43] dive deep into limited corruptions and have made some attempts to remedy the models with corrupted input images. They have only studied certain corruptions with much *a priori* knowledge, and thus the methods could not be directly generalized to other corruptions or unknown corruptions.

Different from image recognition, multiple instances with different scales and categories can appear in a single image. Besides, the extreme imbalance [22] between foreground and background boxes can be one of the key challenging properties of deep object detectors.

Together with other challenges, it requires more effort to repair object detectors. Although some previous attempts have claimed success in repairing neural networks [7, 44], they only consider image classification tasks and can not be directly applied to object detection tasks that often require more complex neural network architecture design. For example, DeepRepair [7] proposes style-transfer guided augmentation to repair neural networks by retraining on the whole training set. However, it can be much more expensive to train a detector than a classifier, making DeepRepair not efficient enough in large-scale object detector repair benchmarks.

To sum up, there is still a long way toward effective and efficient repair schemes for object detectors.

In this work, our goal is to benchmark the SOTA techniques for object detection repairing, based on which we further performed a comprehensive study to better understand the challenges and pinpoint the directions for further research in this direction. To achieve this, we establish and create a benchmark of repairing techniques for SOTA object detectors under various common corrections. Then, we performed a large-scale evaluation and comparative study to investigate the potential failures of these object detectors from various angles, including corruptions, models, failure datasets, and error types. The analysis enables us to gain a deep understanding and insight into the failure patterns and the underlying potential causes. Based on this, we further propose potential approaches for repairing object detectors and expand the benchmark with our techniques for more comprehensive and comparative evaluation. Through such a large-scale comparative evaluation, we identify a few key clues and problems that will be important for future work in this field. Overall, this paper aims to provide a deeper understanding of repair and to highlight the challenges and opportunities in this area. We also provide a benchmark for repairing object detectors, to enable further research and study along this line. The contents of this paper are organized toward addressing the three research questions as follows.

RQ1: Are there any failure patterns for robustness and repair of object detectors? To begin with, we confirm the performance degradation of object detectors when the input images are corrupted across diverse categories and families of object detectors and types of corruptions as shown in Fig. 3.4. For common corruptions, we notice that shot_noise and gaussian_noise are highly correlated in terms of their effect on performance decrease. For detectors, DETR [33] shows different erroneous patterns from others in object scales and box qualities. Additionally, we further study the distributions of object scales on the failure datasets and obtain clues for repair. Besides, we compute the major error type for the detectors after corruptions and find that the missing detection error increases the most in general. These observations confirm the importance and necessity of developing novel repair schemes for object detection.

RQ2: Are current SOTA techniques able to repair the object detectors effectively and efficiently? To obtain more comprehensive insights and understanding, we carefully design our study, establish an object detection repairing benchmark, and conduct a systematic and comprehensive evaluation, the workflow of which is summarized in Fig. 3.1. In particular, we perform a comparative analysis of all the potential repairing techniques (highlighted in orange color). Although we tried our best to include as many techniques as possible, we found that some were not feasible in our study. Eventually, We include as many as seven different techniques in our benchmark for comparative evaluation in terms of their effectiveness, efficiency, and inference impact. At a high level, we have the following observations.

- For fine-tuning-based methods, it's better to fine-tune on combined datasets than failure datasets. We observe catastrophic forgetting in fine-tuning-based methods.
- For augmentation-based methods, we work on MixUp [45] and Mosaic [26] to boost information fusion between the original training data and failure data. However, both of them fail to achieve outstanding performance in repair due to their impacts on the

object scales and instance densities.

- For denoisers-based techniques, we find the self-supervised method, Neighbor2Neighbor [46], works on limited cases.
- BN (Batch Normalization) calibration techniques cannot be directly applied to our experiments, and we tried our best to modify it with a practical version. Its performance varies across models and corruptions.
- For DeepRepair [7], we study a naive version and find its enormous potential relying on the style transfer method.

Among all the repair techniques, the performance of fine-tuning with combined data stands out. BN calibration and denoising can benefit multiple detectors with one repair.

RQ3: What are the challenges and opportunities of future work for repairing object detectors? Based on results in RQ1 and RQ2, we highlight some potential directions and opportunities for future work, *e.g.*,

- For fine-tuning-based and augmentation-based repairing techniques, catastrophic forgetting can be a bottleneck, and it would be important and quite beneficial to design novel techniques to address such problems.
- Robust denoising techniques and style transfer-based techniques can be helpful for improving the repairing effectiveness.
- In addition, more advanced better image fusion schemes, *e.g.*, better augmentations can also be beneficial for object detection repairing.
- To accelerate the deployment of object detectors with possible version updates and evolution, it is highly desirable the repairing techniques could be efficient, to be able to quickly adapt to potential changes in both system-level and operational environments.

Overall, the contributions of this paper are summarized as follows.

- We systematically investigate the failure patterns of object detection techniques based on diverse corruptions, detectors, object distributions on the failure datasets, and the error types.
- We construct an object detection repairing benchmark and performed a comprehensive evaluation and comparative studies on the SOTA repair schemes.
- We identify current potential challenges and pinpoint future and opportunities directions for repairing object detectors.

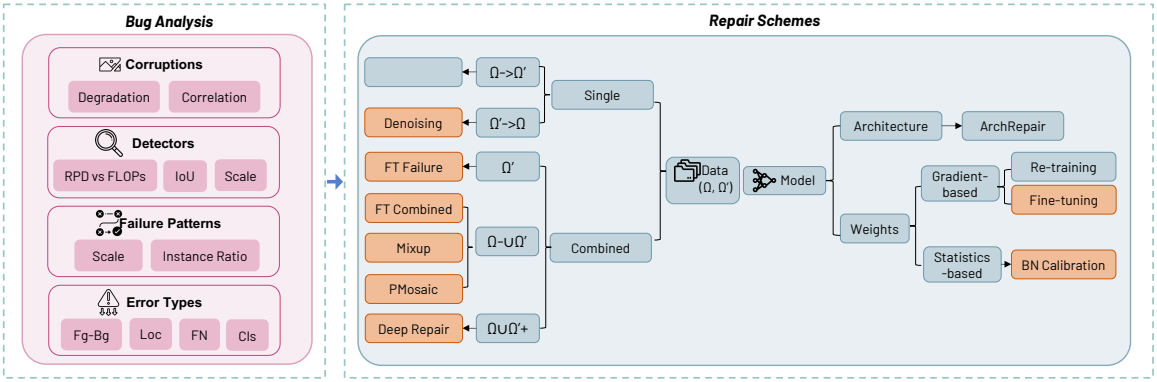


Figure 3.1: Overview of our workflow. We initiate the analysis of failures in object detection from multiple aspects. Next, we conduct an extensive study of potential repair methods for addressing these failures. Methods are located in leaf nodes and the orange ones are included in our benchmark.

3.2 Related Work

Similar to traditional software repairing, the purpose of machine learning (ML) model/system (including deep learning) repairing is to fix the identified erroneous data, so as to ensure the ML models can be continuously deployed and function with high performance. Up to the present, there have been some early and recent efforts to repair image classifiers. However, the repairing of object detectors can be much more challenging, and has not been comprehensively investigated so far. Furthermore, there could be quite a few research

Table 3.1: Methods in the benchmark and their counterparts.

Method	Better Version	Upper-bound/Baseline
Fine-tuning on failure data		
Fine-tuning on combined data		
BN calibration	Calibration on classification	Calibration on ImageNet-C [5]
Neighbor2Neighbor [46]		Supervised denoising
Mosaic [26]	PMosaic	
MixUp [47]		
DeepRepair [7]		

approaches that could be relevant and helpful for the repairing purpose. In this section, we discuss the most relevant work of repairing object detectors by categories.

Deep Neural Network Repair. Apricot [48] is an early work on repairing deep neural networks (DNN), which presents a weight-adaptation approach to merge several reduced models trained on the subset of all training data. MODE [49] performs fault localization by model state differential analysis and repair by training input selection. Both Apricot and MODE aim to improve accuracy instead of handling system errors caused by corruption. NNRepair [50] proposes a constraint-based approach to repairing classifiers. Before solving the constraints, it first conducts fault localization to mark neurons for repair. It repairs only a single layer of the whole network, while actual faults might be located in multiple layers. [51] follows a similar workflow, which applies causality-based fault localization and PSO optimization. Archane [52] conducts bidirectional fault localization and repair with a population-based heuristic optimization algorithm. However, heuristic methods are often quite challenging for deep object detectors. DeepRepair [7] is an augmentation-based repair method to conduct data-driven repair for DNNs. It employs style transfer to augment the datasets so that it solves the problem of insufficient data. ArchRepair [44] repairs the classifiers by searching for better architectures. In addition, some other works [53, 54]

attempt to repair more neural networks than feed-forward networks.

Robustness and Augmentation. Besides the aftermentioned work that directly repairs DNNs, other techniques can also be helpful to achieve the repairing effects under different contexts. For example, some recent works [5, 6] on robustness study the overall accuracy performance of detectors under various corruptions. Augmix [55] and AutoAug [56] propose automatic augmentations for image classification to help improve accuracy performance and robustness. [57] and [58] make attempts to boost deep object detectors with augmentation. Safety, where repair works, revolves around a restricted specification. On the contrary, robustness revolves around everything not specified. Robustness usually concerns reducing failure when the input is corrupted or fake. However, repair aims to fix bugs when failure cases occur.

Domain Adaptation. Domain adaptation transfers a model from the source domain to the target domain. Batch normalization(BN) [59] is a critical component in the neural network architecture. AdaBN [60] assumes that models from different domains share similar weights except for BN and statistics in BN capture information in diverse domains. Some related papers [61, 62] also show great success in domain adaptation with BN calibration. Recent works on domain adaptation for object detection [8, 9, 63] employ schemes in detection frameworks to improve the detectors' performance in the target domain. There can be several differences between repair and domain adaptation. Domain adaptation mostly focuses on performance in the target domain, while repair has to take performance on both the failure set and clean set into consideration. Domain adaptation is usually formulated as an unsupervised or semi-supervised problem, and repair is treated more as supervised few-shot learning. Besides, it would be less preferred to conduct repair by large architecture modification as is usually used in domain adaptation.

Continual Learning. Continual learning [42, 64] works on a sequence of well-defined tasks. [42] clarifies three scenarios for continual learning, *i.e.*, task-incremental learning, domain-incremental learning, and class-incremental learning. In continual learning, tasks are sequential and the models can only access data from the current task. However, in repair scenarios, both clean training data and corrupted data are available. As a result of task sequences, the key point of continual learning is the problem of catastrophic forgetting. They utilize regularization-based methods [65, 66], data modification [67, 68], and an episodic memory [69, 70]. Although we observe catastrophic forgetting in some repair methods, it's beyond our discussion to solve catastrophic forgetting.

Out-of-distribution in Object Detection. Out-of-distribution problems are critical in data-centric machine learning. [71–73] work on open-set object detection. Open-set problems occur when unseen categories exist after deployment. Most related works [74] aim to detect out-of-distribution samples but fail to work on how to solve the failure prediction, which is the main goal of the repair.

Image Denoising. Image denoising has been a popular research topic, and it has been revived recently by deep learning techniques. As noted in [75], some deep denoisers are recently proposed for additive white noisy images [76–79], real noisy images [80–82], blind denoising [83–85], and hybrid-noisy images (*i.e.*, the combination of noisy, blurred, and low-resolution images) [86, 87]. Image denoising models could serve to solve some of the corruptions in our study. Due to limited available computational resources at our hands, we eventually include Neighbor2Neighbor [46] in our benchmark. It is a self-supervised model, achieving state-of-the-art results on image denoising. Intuitively, better denoisers would be beneficial for repairing detectors under diverse common corruptions.

3.3 Object Detector Repairing Schemes

As shown in Fig. 3.1, we propose to investigate two directions to solve the above repair problem based on the detector formulation $\mathbf{D} = \{\mathbf{W}, \mathbf{A}\}$, that is, data-oriented repair and model-oriented repair. Intuitively, data-oriented repair focuses on modifying or enhancing the training dataset, such as through data augmentation techniques. This approach aims to update the weights of the detector or preprocess the input data during the testing process to improve its performance against corruption examples. On the other hand, model-oriented repair aims to update the weights or architectures within the detector itself to enhance its robustness against corruption examples. This approach involves making modifications directly to the model's parameters or structure to improve its ability to handle and mitigate the impact of corruption.

Model-oriented Detector Repairing

In general, a model of a detector is comprised of the architecture (*i.e.*, $\mathbf{A}$) and its weights (*i.e.*, $\mathbf{W}$). So based on the model view, we identify four potential solutions based on the characteristic training of object detectors, *i.e.*, *fine-tuning-based weight repairing*, *retraining-based weight repairing*, *statistic-based weight repairing*, and *neural architecture search-based architecture repairing*.

Retraining-based weight repairing. A straightforward idea for improving robustness against a specific corruption is to incorporate the collected failure training dataset $\mathscr{D}_{\text{train}}^{\text{fail}}$ to the clean training dataset $\mathscr{D}_{\text{train}}$ and then to retrain the model from scratch. This method is denoted as retraining-based weight repairing. In our benchmark, we have excluded retraining as a repairing technique due to its high time and computation cost.

Fine-tuning-based weight repairing. One potential solution for repairing object detectors is to perform fine-tuning on the pre-trained detector $\mathbf{D}$ with the failure training dataset $\mathscr{D}_{\text{train}}^{\text{fail}}$. This process allows the detector to adapt and learn from the failure cases, effectively

repairing and enhancing its capabilities. Fine-tuning provides a practical and efficient solution for repairing object detectors without the need for extensive retraining, as it focuses on updating the pre-existing model using the collected failure data. While fine-tuning the detector with failure data can help improve its performance and reduce the occurrence of similar errors, there is a potential risk of introducing errors in the original testing dataset. This is because the fine-tuning process may lead to overfitting on the failure data, causing the detector to become less accurate or reliable when applied to the original dataset. We will detail the training configurations for fine-tuning in Sec. 3.3.

Neural architecture search-based repairing. On the other side, we can also achieve the goal of repair by modifying the architectures **A** of the detectors. For example, ArchRepair [44] proposes to localize the vulnerable block of a network and use the neural architecture search (NAS) to repair the localized block with the guidance of the collected failure training dataset. However, it is not easy to repair the architecture of a detector since the modifications on operators and modules might lead to a negative impact on inference. For example, ArchRepair searches the architectures without any consideration for computation. But the repaired network directly adds extra convolutions to the failed model. It increases the network's computation and inference time, and it is an unfair comparison between models before and after repair. It could result in more severe problems in deployment after repairing the detectors, especially when the deployed hardware device has a gap with the ones that are used for training and development. For instance, group convolution, a widely-used operator in NAS search spaces, is not optimized across all devices and backends. Considering that it can be risky to repair by modifying the architectures in many cases, especially those without specifying the hardware configurations of the deployed environment. Therefore, in this paper, we don't include neural architecture search-based repairing methods.

Statistics-based repairing. Like many state-of-the-art neural networks for image classification, object detectors also often employ statistics-based operators, *e.g.*, batch nor-

malization (BN). Recent works [60, 61] hypothesize that running statistics in BN layers store the domain-specific knowledge while other weights of networks capture semantic pattern information across domains. They highlight the effectiveness of Batch normalization (BN) calibration in handling distribution shift. BN calibration updates a network's running statistics while keeping other weights unchanged. Acknowledging the significance of statistics-based operators in object detectors and their ability to capture domain-specific knowledge, it is indeed worth exploring the potential of BN calibration as a candidate for repairing object detection models.

However, in our pilot experiment, we found that BN calibration failed to repair the detector and resulted in a significant drop in performance on both the clean and failure test sets. This failure may be due to inconsistency between the training and repairing BN status. If the detector is trained with frozen BN statistics, meaning that the statistics are directly copied from its classification pre-trained model instead of calculated on the detection dataset, we cannot calibrate the BN statistics. Unfortunately, most models in the experimental model zoo are trained with frozen BN statistics, which could explain the failure of BN calibration for object detection repairing.

To further study the reason why BN calibration fails and explore its potential, we conduct an ablation experiment in Table 3.2 based on BN status (frozen or updated) during training and repairing. Our conjecture is that BN calibration fails in repairing due to inconsistency between training and repairing BN status. To begin with, we confirm that different BN statuses during training result in similar performance. They can achieve comparable performance after being repaired with frozen BN statistics, too. However, the detector trained with updated BN statistics succeeds in achieving improvement after being repaired with updated BN statistics, while the other one trained with frozen BN statistics fails. It confirms our conjecture, and we conclude that BN calibration can only be applied to the detectors trained with updated BN statistics.

Given our experimental models are trained with frozen BNs, direct BN calibration is not

practical in our benchmark. Therefore, we propose an improved version of BN calibration, *i.e.*, BN calibration on classification. We follow three steps to detector repairing via BN calibration on classification. First, generate images as the input of the classification model. A simple example is to transfer images from ImageNet using the failure images as the style. Second, calibrate batch normalization layers of the pre-trained classification model. Finally, copy the running BN statistics to the detector. As shown in Table 3.2, it improves the detector successfully. Eventually, we adopt this approach in our benchmark.

Only once repair would benefit all of the object detectors sharing the same pre-trained model. On the other side, BN calibration on classification also has explicit limitations. To repair the detector, the pre-trained classification model is supposed to be accessible due to privacy or other reasons. For example, our benchmark fails to include experiments of BN calibration on classification on FCOS [29] because they utilize Caffe versions of ResNets as pre-trained models.

Table 3.2: A pilot study for consistency between BN updating schemes during training and repair. BN@Train is the status of batch normalization layers when training the detectors to be repaired. And BN@Repair is the BN status when repairing detectors. AP@Cl, AP@Fl, and AP@Avg mean the mean average precision of the object detectors on the clean test set, the failure test set, and their average values. BN calibration on detection can be only applied to models trained with updated BN. BN calibration on classification improves average performance on failure and clean test datasets.

BN @ Train	BN @ Repair	AP@Cl	AP@Fl	AP@Avg
Frozen	-	36.5	26.0	31.25
Updated	-	36.1	26.3	31.2
Frozen	Frozen	35.2	31.0	33.1
Frozen	Updated	30.0	26.2	28.1
Updated	Updated	35.2	30.8	33.0
Frozen	Updated on Classification	35.6	27.8	31.7

Data-oriented Detector Repairing

As noted before, the original training dataset $\mathscr{D}_{\text{train}}$ and failure dataset $\mathscr{D}_{fail}$ are available in repairing tasks. For clarity, we assume they belong to different domains, *i.e.*, Ω and Ω' respectively. To resolve problems across domains, we can transfer the images into a single domain or combine the images to enable the models to capture knowledge in both domains. For repairing proposals in a single domain, we can repair the detectors by transferring the images to Ω' or Ω, discussed in Sec. 3.3. For domain combination, we first propose two approaches based on naive fine-tuning. Then we step further to boost domain fusion with advanced techniques like data augmentations. Among all the transforms and augmentations in MMDetection [41], we study those that can help the detectors fuse images from different domains, *i.e.*, MixUp [45] and Mosaic [26]. Finally, we study the repairing method on classification, DeepRepair [7].

Single domain: transferring images to Ω'. In order to transfer the images into a single domain, we can merge the images into Ω or Ω'. As failure samples are not frequently observed in the application environment, it's not reasonable to transfer all images into Ω'.

Single domain: denoising. We include a denoising method to transfer the images Ω' to Ω. In detail, we study one of the best self-supervised methods, Neighbor2Neighbor [46]. It removes noises based on the locality and smoothness of clean images. To further estimate the technical opportunities of denoising methods, we study a supervised version of Neighbor2Neighbor as an appropriate upper bound.

We first conducted a pilot experiment to study its effectiveness on denoising. Here we employ the widely-used PSNR(Peak Signal-to-Noise Ratio) to evaluate the output image quality and report AP scores as the final evaluation. For gaussian_noise, it achieves a very high PSNR score and succeeds in improving detection performance. However, it leads to a much lower PSNR score on defocus_blur and fails to repair the detector. Therefore, to

further investigate the opportunities in denoising for repair, we experiment on a supervised version as an estimated upper bound. In the pilot study, we find that Neighbor2Neighbor performs close to its supervised counterpart on gaussian_noise but far less than the upper bound on defocus_blur.

Table 3.3: On gaussian_noise, both the self-supervised and supervised denoisers work and achieve comparable performance. But the self-supervised one fails on defocus_blur and there is a large gap between the denoisers.

Corruption	Repair	PSNR	AP@Fl
GsNoise	-	-	16.3
	Self-supervised	26.74	26.3
	Supervised	26.74	26.8
DfBlur	-	-	15.0
	Self-supervised	22.20	15.5
	Supervised	30.55	31.9

Multiple domains: fine-tuning with failure data. Multiple schemes could be applied to combine images from two domains (see Fig. 3.1). A naive solution is to fine-tune the model on the failure dataset directly. The model to be repaired is well-trained and captures sufficient information in Ω. Fine-tuning will only result in limited changes from the original optima. Then, the model owns knowledge across both domains. We include this method in the benchmark as a simple baseline.

Multiple domains: fine-tuning with combined data. In our pilot experiments, we find that for fine-tuning with only failure data, the performance improves in Ω' but drops in Ω severely. Therefore, we propose to fine-tune the detectors with both failure data and sampled clean data. It explicitly updates the models with data from both domains. It is a simple but effective repairing method.However, as Fig. 3.12 shows, we notice catastrophic forgetting in these two fine-tuning-based methods, which we will discuss later in Sec. 3.4.

Multiple domains: MixUp. MixUp [45] is a data augmentation to boost training by mixing two images with random weights. In object detection, MixUp merges two images by weighted sum and aggregates all ground truth bounding boxes. [47] is the first to explore MixUp in object detection. YOLOv4 [26] and YOLOX [27] utilize MixUp as one of the transform pipelines to boost training. However, as our later experiments show, it would result in some problems in repair.

Multiple domains: Mosaic. Mosaic [26] mixes four images into a collage one. Sticher [57] is a similar augmentation based on the observation of object scale statistics. Compared to MixUp, it enriches the context without fusing semantic information of objects. As noted in Stitcher [57], it would also benefit the training of small objects, one of the critical problems in object detection.

We conduct a pilot study on Mosaic as shown in Table 3.4 as well, where the detector after being repaired with Mosaic shows some advantages over that before repair. However, it performs much worse than the fine-tuning one on the failure test set. In Stitcher [57], we find similar experimental results. They notice all collaged inputs (similar to Mosaic) would lead to severe performance degradation compared to all regular inputs. They propose random sampling between collaged and regular input images and find it better than the all-regular baseline. In our work, we implement a random-sampling-based version of Mosaic, noted as PMosaic. To be more specific,

$$\text{PMosaic}(\mathbf{I}) = \begin{cases} \text{Mosaic}(\mathbf{I}), & \text{w.p. } 0.5, \\ \mathbf{I}, & \text{w.p. } 0.5. \end{cases} \tag{3.1}$$

In the pilot experiment in Table 3.4, PMosaic exceeds Mosaic greatly on the failure test set, but fails to achieve better performance than fine-tuning. However, we notice that Mosaic and Stitcher converge slower in training an object detector. It motivates us to observe its behavior on a longer schedule. In Table 3.4, PMosaic at epoch 49, notated as PMosaic$^+$, shows comparable performance with fine-tuning on the failure test set. But the model suffers from catastrophic forgetting, *i.e.*, the performance on the original clean test set drops with

the fine-tuning process. Fig. 3.12 shows the forgetting curves. To conclude, PMosaic can be promising for detector repair if either a slow convergence rate or catastrophic forgetting is solved or relieved.

They [57] argue that all collaged inputs would harm the learning of large objects. In our experiments, we validate it by observing a severe performance drop of large objects on both clean and failure test sets. For instance, the detectors repaired by naive fine-tuning and Mosaic share comparable performance in overall mean AP. However, there is a remarkable gap between their performance in AP@L. It motivates them to develop the random sampling method. In Table 3.4, we find that it relieves the problem in Mosaic(AP@L increases from 26.7 to 31.4). But there remains much scope for improvement.

Table 3.4: Pilot experiments on Mosaic. Mosaic fails to exceed fine-tuning due to its degradation in the learning of large objects. We utilize an improved version, PMosaic, and achieve much better AP scores than Mosaic. It suffers from slow convergence and catastrophic forgetting, too.

Repair	AP@Cl	AP@Fl	AP@Fl@SML			AP@Cl@SML		
-	36.6	16.3	6.9	18.5	24.8	21.6	41.0	49.8
Fine-tune	35.8	21.4	9.1	23.8	32.7	21.4	40.4	48.8
Mosaic	35.6	18.4	7.9	21.1	26.7	21.4	39.9	47.0
PMosaic	35.4	20.8	9.3	23.1	31.4	20.8	39.7	47.1
PMosaic^{+}	34.4	21.3	9.4	23.8	31.6	20.0	38.4	46.5

Multiple domains: DeepRepair. When we attempt to combine data in Ω and Ω', we have to handle the lack of failure data. Moreover, there exists an imbalance between sufficient clean data and insufficient failure data. DeepRepair [7] generates data in Ω' with style transfer and fuse images with AugMix [55]. Since DeepRepair is implemented on classification, here we apply a naive version for detection. We directly fine-tune the detector with both clean data and generated failure data. Based on our pilot experiments in Table 3.5, we notice that the performance increases as the size of the training set increases for both

naive DeepRepair and its upper bound. The training dataset of COCO contains 118,287 images. To achieve the upper bound, we generated the training images with corruptions instead of style transfer. When compared with the upper-bound, naive DeepRepair shows comparable performance on clean data. On the other hand, it performs much worse on the failure test set. Here we utilize the same style transfer model as DeepRepair and we suggest that a better style transfer method would help even more. However, it's far beyond our discussion in this paper to improve style transfer. But we still encourage further attempts at detection repair to follow and improve it.

Table 3.5: Pilot experiments for the naive version of DeepRepair. 1000*8 means 1000 images trained for 8 epochs. And all the others are trained for 1 epoch. 118287 is the size of the COCO training dataset. For the naive version DeepRepair, the performance increases as the number of training images increases. For the upper bound, training images are generated with corruptions instead of style transfer. When compared with the upper-bound, naive DeepRepair shows comparable performance on clean data, but it has a large performance gap on the failure test set. Besides, it fails to outperform fine-tuning despite it costs about 30 times more GPU hours.

Method	Images	AP@Avg	AP@Fl	AP@Cl
Fine-tuning	1000*8	28.6	21.4	35.8
Naive DeepRepair	8000	26.3	16.4	36.2
	50000	26.45	16.7	36.2
	118287	26.6	16.9	36.3
Upper bound	8000	28.8	21.6	36.0
	50000	29.95	23.7	36.2
	118287	30.45	24.6	36.3

Summary

To sum up, we systematically analyze and discuss potential methods to repair object detectors as shown in Fig. 3.1. We attempt nine different candidate methods in this section. Two of them are eventually not included in the benchmark due to huge computation costs and one is

not included as a result of the unclear and common mismatching of training and deployment hardware configurations. In our pilot experiments, we find that:

- Mosaic harms the performance of large objects and PMosaic suffers from slow convergence.
- BN calibration can only be directly applied to models trained with updated BN.
- The self-supervised denoiser works on some corruptions but fails on others.
- The naive DeepRepair can retain the performance on the clean test set and there is a performance gap between this approach and its upper bound.

Finally, we will include the seven repairing methods in our benchmark, *i.e.*, fine-tuning on the failure dataset, fine-tuning on the combined dataset, MixUp, PMosaic, BN calibration on classification, denoising, and the naive DeepRepair. More information is available in Table 3.1.

3.4 Detector Repairing Benchmark Construction

We've discussed the problem formulation and our repairing proposals. Based on these proposals, we'll continue to build our benchmark. In this section, we'll show how we construct our repairing benchmark. In Sec. 3.4.1, we first set up experimental configurations, including the dataset, corruptions, detectors, and training schedule. More implementation details about the repairing methods are in Sec. 3.6. We introduce the evaluation metrics for repair in object detection in Sec. 3.4.2.

In the following sections, we'll discuss our analysis and comparative experimental results on repairing object detectors. In Sec. 3.5, we aim to confirm and synthesize failure patterns to benefit repair further. In Sec. 3.6, we empirically study the performance of repairing methods and accumulate deep insights into these methods and the task.

3.4.1 Experimental Setup

We will present our experimental settings in this section. We involve most details in the experimental environment and more information for the repairing methods can be found in the implementation details of the methods in Sec. 3.6.

Dataset. We perform all of our experiments on COCO [40], one of the most important and widely-used dataset in object detection. COCO includes common objects and annotations across 80 categories. It captures objects with a wide range of scales, from very small objects to very large ones. It's challenging and has a great influence on the object detection research community.

Failure Definition. In terms of the image classification task, we can easily define a failure sample of a classifier by checking whether the predicted category is the same as the ground truth. However, it is more complicated for object detection. The widely-used metric like average precision (AP) is calculated based on the ranking of bounding boxes across images, and cannot be used to determine the failure sample directly. Then, we need a new way to define a failure sample for a detector under corruption. To address this issue, we first define the image-wise AP of a detector $\mathbf{D}$ on an input image $\mathbf{I}$ as $\mathrm{AP}_{\mathrm{img}}(\mathbf{I},\mathbf{D})$ with Eq. (2.8) by evaluating AP of the objects within an image instead of the whole dataset, *i.e.*, $\mathrm{AP}_{\mathrm{img}}(\mathbf{I},\mathbf{D}) = \mathrm{AP}(\mathscr{D}_{\mathbf{I}},\mathbf{D})$ where $\mathscr{D}_{\mathbf{I}} = \{\mathbf{I}\}$. Besides, we define the failure samples in COCO-C [6], which is constructed by adding corruption to clean images in COCO [40]. Given a detector $\mathbf{D}$ and an image $\mathbf{I}_c$ in COCO-C corrupted by the corruption c, we name this image as a failure sample of the detector under the corruption if and only if its image-wise AP drops when compared to its clean counterpart $\mathbf{I}_o$ in COCO. To validate the rationality of the definition of failure sample under a kind of corruption, we check whether the image-wise AP variation is consistent with the dataset-level AP variation. Specifically, given a corrupted testing dataset, we evaluate a detector on that dataset and collect the failure samples based on

the failure sample definition. Then, we construct a failure dataset and can calculate the size of the dataset (*i.e.*, the number of samples within the failure dataset). Then, we compute the correlation between the size of the failure dataset and overall performance (*i.e.*, the AP on the whole corrupted testing dataset). The size of the failure dataset is significantly negatively correlated (Kendall's Tau=-0.896) to the AP scores on the whole corrupted testing dataset. Intuitively, when a detector shows lower performance on the whole testing dataset, it's more likely to collect more failure samples. When a detector fails at all the images, its AP score falls to 0 and the size of the failure dataset reaches the maximum. To sum up, our definition of failure is consistent with performance degradation on the whole corrupted test dataset.

Failure Collection. As stated before, we assume that failure is caused by a single image-wise corruption. Following COCO-C [6], we experiment on all 15 corruption types. The corruptions include brightness, contrast, defocus_blur, elastic_transform, fog, frost, gaussian_noise, impulse_noise, snow, pixelate, jpeg_compression, motion_blur, shot_noise, and zoom_blur. The abbreviations of corruptions are summarized in Table A.1. For simplicity, we only study severity at level three, the middle one across all five levels. All of our experiments and analysis methods can be easily extended to other levels without any extra workload except GPU hours. We collect all failure samples based on the definition in Sec. **??**. Following DeepRepair [7], we randomly sample 1000 failure images as the failure training set and leave the others as the failure test set. We repeat each repair experiment five times. Empirically, we find the randomness comes from the split of training and test sets while the results of repeated single experiments are stable.

Detectors. We conduct our experiments on MMDetection [41], one of the most popular open-source detection repositories. We directly utilize detection models from MMDetection model zoo, where almost all state-of-the-art detectors are available. Due to the limitation of computation resources and time, we only focus on RetinaNet, Faster R-CNN, FCOS, and DETR as the representatives of single-stage detectors, two-stage detectors, anchor-free

detectors, and transformer-based detectors. We use their abbreviations as noted in Table A.2 in figures. In the study of repairing methods, we experiment on RetinaNet, Faster R-CNN, FCOS, and DETR with ResNet50 and ResNet101 backbone. DETR-ResNet101 is not included because of its absence in the MMDetection model zoo. In detail, we experiment to evaluate repairing methods on retinanet_r50_fpn_1x (RetinaNet-ResNet50), retinanet_-r101_fpn_1x (RetinaNet-ResNet101), fcos_r50_caffe_fpn_gn-head_1x (FCOS-ResNet50), fcos_r101_caffe_fpn_gn-head_1x (FCOS-ResNet101), and detr_r50_8x2_150e (DETR-ResNet50). To sum up, we have 105 experimental configurations with 5 repetitions, 525 experiments in total, to evaluate each repairing technique.

Training schedule. For fine-tuning-based and augmentation-based methods, we report the ones with the best average AP. For denoising, we utilize the denoiser with the best PSNR. For BN calibration, we apply the classifier with the best average accuracy. We finally report the AP scores for denoising and BN calibration. When fine-tuning the detectors, we inherit the training settings and load the pre-trained models provided by the model zoo. The batch size during fine-tuning is set as 8 on a single NVIDIA A6000 GPU. In most experiments, we fine-tune the detectors for 8 epochs. The learning rates are set the same as the ones at the end of training and we do not apply decay to learning rates.

3.4.2 Evaluation Metrics for Repairing

We employ relative performance decrease to conduct our quantitative analysis for performance degradation and relative performance improvement to evaluate repairing methods.

Relative Performance Decrease. To investigate performance degradation in a quantitative way, the performance decrease is formally defined as Eq. (3.2), where **D** represents an object detector, $\mathscr{D}_{\text{test}}$ is the clean test dataset, and $\mathscr{D}^{\text{c}}_{\text{test}}$ is the corrupted dataset. The notations are the same as those in Sec. 2.2.2. It's obvious the smaller RPD is, the better the model performs.

$$\text{RPD} = 1 - \frac{\text{AP}(\mathscr{D}^{\text{c}}_{\text{test}}, \mathbf{D})}{\text{AP}(\mathscr{D}_{test}, \mathbf{D})} \tag{3.2}$$

MRPD of one model refers to average RPD values over the corruptions. We utilize RPD to analyze degradation for a single model with one corruption and MRPD for the overall impact of the corruptions on one model. In Fig. 3.4, we present MRPD values for 48 models under 15 corruptions.

Relative Performance Improvement. Based on definition in Eq. (2.12), we evaluate the repairing methods with Eq. (3.3).

$$\frac{\lambda \text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D}') + (1-\lambda)\text{AP}(\mathscr{D}_{test}, \mathbf{D}')}{\lambda \text{AP}(\mathscr{D}^{\text{fail}}_{\text{test}}, \mathbf{D}) + (1-\lambda)\text{AP}(\mathscr{D}_{test}, \mathbf{D}))} - 1 \tag{3.3}$$

, where we use $\lambda = 0.5$ in this paper for equal importance.

Different from RPD, a higher value in RPI means a higher performance gain. Similarly, a higher overall value indicates a better repair method.

3.5 Analysis on Failure

To further discover knowledge in object detectors with corrupted input images, we propose four sub-research questions from multiple aspects. Different from existing papers [6] work on differences between images before and after corruption, we pay attention to failure patterns closely related to object detection.

RQ1.1. How do common corruptions impact deep object detectors?

RQ1.2. How do detectors perform under common corruptions?

RQ1.3. Are there any patterns in the failure datasets?

RQ1.4. What are the major error types of detectors under common corruptions?

3.5.1 RQ1.1. How do common corruptions impact deep object detectors?

In RQ1.1, we aim to study the corruptions based on RPD values and statistics. In Fig. 2.2, the corrupted detectors have lower AP scores than clean ones. In Fig. 3.4, no negative RPD value is observed. Therefore, all corruptions lead to performance degradation, although to a different extent. From the colors of the heatmap in Fig. 3.4, we find that different types of corruption can lead to different levels of degradation in performance. In general, zoom blur leads to the most degradation in performance across models, while brightness results in the slightest performance drop.

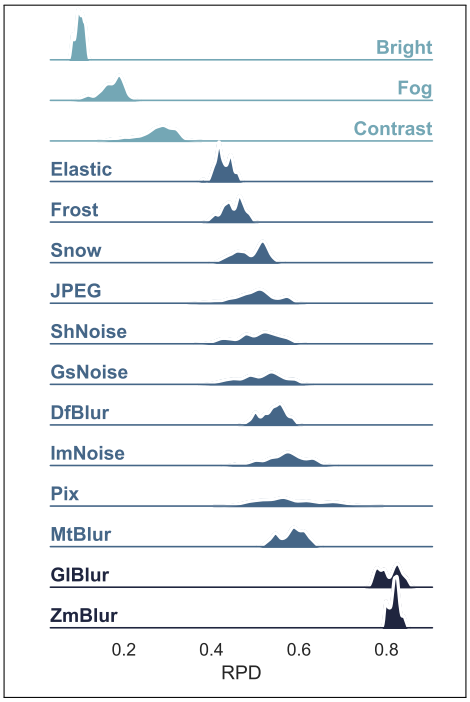


Figure 3.2: Correlation between corruptions based on relative performance decrease values.

In Fig. 3.2, we observe three clusters in corruptions based on performance degradation. Brightness, fog, and contrast form the first cluster, where all models have much smaller relative performance decrease values. Glass blur and zoom blur locate in the third cluster and they both lead to large RPD scores. The others with overlapped RPD values in distribution belong to the second cluster, which locates in the middle of the figure. The clusters imply

possible connections among the corruptions.

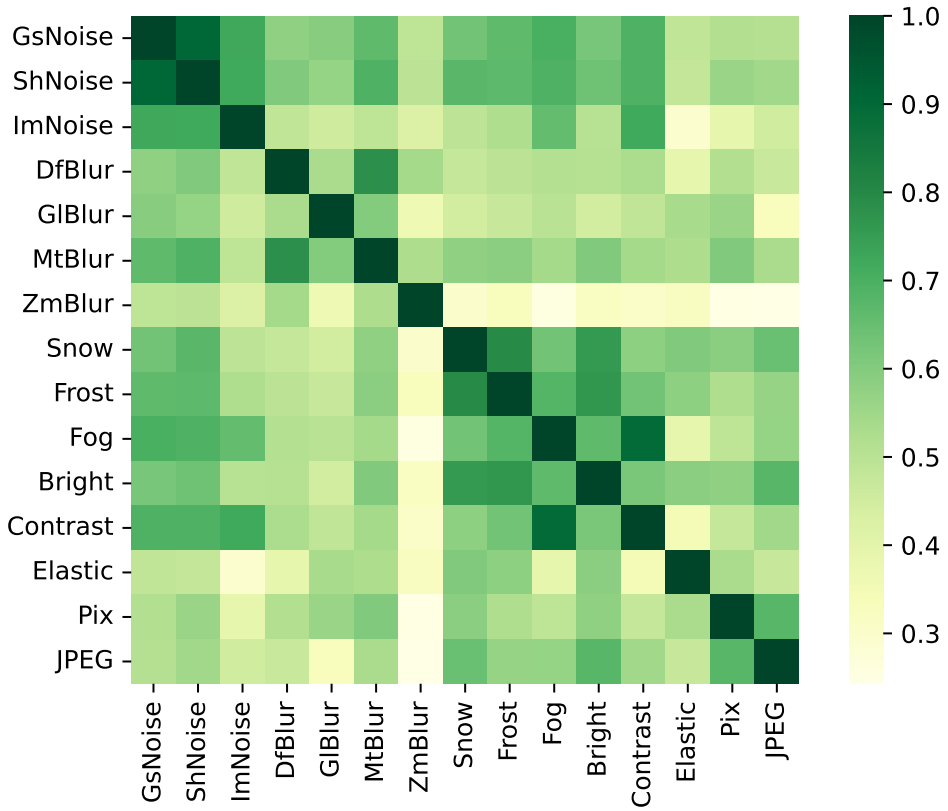


Figure 3.3: Distributions of RPD values for corruptions. They form three clusters as shown in different colors.

Correlation Analysis between Corruptions We further study the relationships between corruptions by analyzing the correlations between their relative performance decreases. The heatmap is located in Fig. 3.3. The minimal correlation appears between zoom_blur and jpeg_compression(0.24), and the maximal one is between shot_noise and gaussian_-noise(0.91). In the later study of repairing methods, we notice that the detectors achieve comparable performance gain after being repaired under shot_noise and gaussian_noise. Fog and contrast are another pair of corruptions with a high correlation in the relative performance decrease. The co-repair of different corruptions could be conducted based on correlations between corruption types in further works.

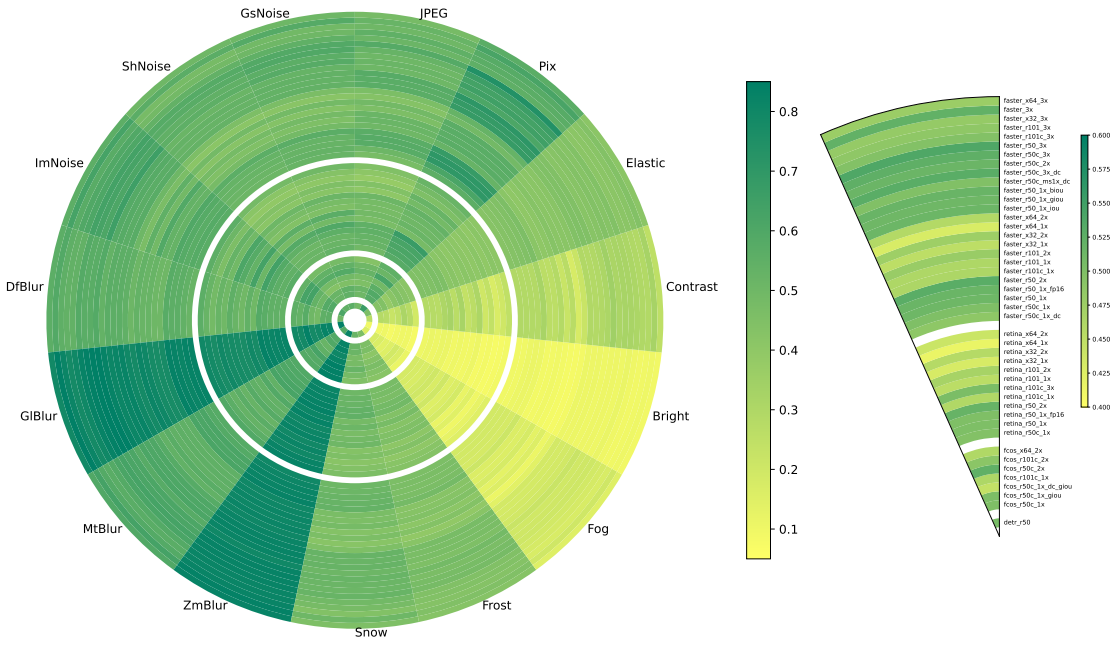


Figure 3.4: Left: RPD(relative performance decrease) values for corruptions and detectors. Right: MRPD of detectors.

Answer to RQ1.1: All 15 common corruptions we study result in performance degradation. In general, zoom blur leads to the most degradation in performance across models while brightness results in the slightest performance drop. Based on relative performance decrease, shot_noise and gaussian_noise are highly correlated.

3.5.2 RQ1.2. How do detectors perform under common corruptions?

In this section, we focus on models to understand performance degradation. Instead of working on only the overall AP decrease, we dive deeper to study the details of detectors under common corruptions.

From Fig. 3.4, we find that retinanet_x101_64x4d_fpn_1x has the least performance decrease among the models and faster_rcnn_r50 _fpn_mstrain_3x yields the most degradation. In Fig. 3.5, we plot the models as scatters based on MRPD and FLOPs. The gray line is the linear regression results based on MRPD and FLOPs and it shows a negative correlation between performance drops and model computation. In general, larger detectors are more likely to survive degradation under corruption. We group the models based on their detection heads and training schedules. We notice that RetinaNets seem to be marginally less

vulnerable than Faster R-CNNs. What's more, a longer training schedule hurts robustness.

Object Scales. As one of the most important features that distinguish object detection from image classification, objects with different scales locate inside a single image. Thus, it has long been a critical methodology for object detection to synthesize performance across different scales. Our experimented dataset, COCO [40], defines the object scale based on the area of bounding boxes, *i.e.*, small, medium, and large objects. In the left plot in Fig. 3.6, we analyze the performance degradation for different scales, *i.e.*, AP@S, AP@M, and AP@L. It can be easily observed that smaller objects suffer from more degradation while larger ones are more robust. The relative performance decreases of medium objects agree with that of overall mAP in general. There are significantly different patterns between the impact of common corruptions across scales and overall mAP. The curves share similar trends but differ in detail. We rank the models along the x-axis based on the performance decrease of overall mAP, and thus the curve of MRPD@AP is increasing. Therefore, the bumpy curves show differences in local rankings between decreases in small and large objects and overall performance. We can also find many interesting points in the figure. For example, retinanet_r101_fpn_1x and retinanet_r101_caffe_fpn_1x, with the same backbone architecture but different pre-trained models, have comparable MRPD values, but the latter one decreases much more in large objects and vice versa. Moreover, compared with other models, detr_r50_8x2_150e_coco suffers more performance decreases in large objects, as we highlight in the figure.

IoU Threshold. As we noted in Sec. 2.1.1, IoU threshold, δ in Sec. 2.1.1, is used to match predicted bounding boxes and ground truths. The IoU threshold implies the box quality. Unlike previous works that only focus on the overall performance, we obtain relative performance decrease values based on different values of δ to study the impact of corruptions for bounding boxes on different levels of quality. The right plot in Fig. 3.6 visualizes the RPD values at different δ. It shows that detectors with more strict box-matching strategies

drop more in performance. It might indicate why Michael *et al.* [6] find Cascade R-CNN [21], which heavily relies on boxes with high quality, suffers from severe performance degradation. Besides, different from what we observe about scales, MRPD values own similar curves across IoU thresholds. The only point here is a significantly lower MRPD at IoU $\delta = 0.75$ for detr_r50_8x2_150e_coco. DETR behaves differently from others in detailed analysis. We suggest there would be more interesting facts and observations in further analysis between CNN-based and transformer-based detectors and we leave it for further work.

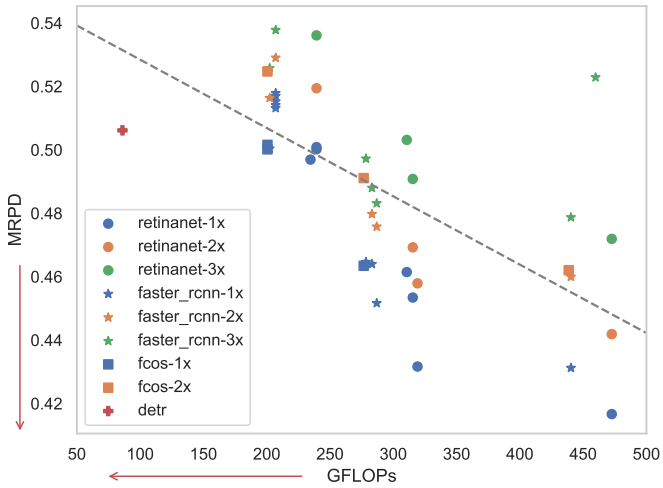


Figure 3.5: Mean relative performance decrease values of detectors. The gray dotted line is estimated by linear regression. In general, larger detectors are more likely to survive degradation under common corruptions. RetinaNets seem to be marginally less vulnerable than Faster R-CNNs. A longer training schedule hurts robustness.

Answer to RQ1.2: Common corruptions cause performance degradation across models. In general, larger detectors are more likely to survive degradation under common corruptions. A longer training schedule might hurt robustness. Smaller objects and more accurate boxes are more sensitive to corruption. DETR-ResNet50 shows different degradation patterns from others in object scales and IoU thresholds.

3.5.3 RQ1.3. Are there any patterns in the failure datasets?

This research question is to analyze the errors in failure samples. In this section, we only study the seven models of interest in our study for repair, as noted in Sec. 3.4.1. Following

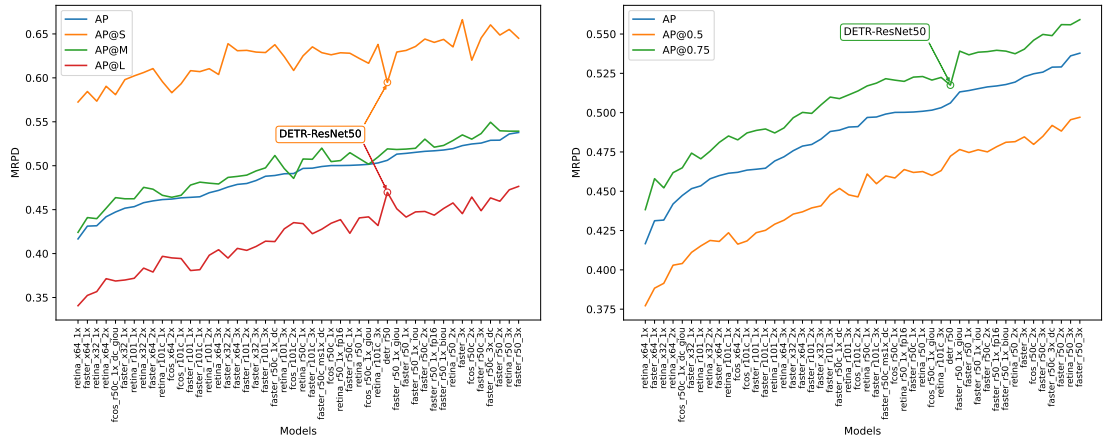


Figure 3.6: The values of RPD *w.r.t.* different object scales and IoU thresholds. Left: the smaller objects are more sensitive to corruption while the larger ones are more robust. Small and large objects show different degradation patterns from overall mAP. DETR-ResNet50 shows different patterns from others in performance on object scales. Right: detectors with more strict box matching strategies drop more in performance.

the failure definition in Sec. 2.2.2, we analyze the failure patterns based on the statistics of objects on the failure datasets.

We attempt to discover knowledge towards a better understanding of model failure by synthesizing and observing the relationships between performance decreases and potential statistical reasons or factors on the failure dataset. Based on experience with object detection, we study three factors, *i.e.*, the average number of instances per image(instance ratio), the ratio of instances marked as crowded over all instances(crowd ratio), and the ratio of small, medium, and large objects over all instances(small ratio, medium ratio, and large ratio). To obtain observations of their relationships with the performance decrease, we plot the scatters in Fig. 3.7. The blue points are the results of our experiments and the red 'x' markers are the extreme cases when the detector never achieves any correct prediction. It is often that the failure dataset of a detector is a proper subset of the test set. However, if a detector fails at every image on the test set, its failure dataset equals the test set. Therefore, we compute the statistics on the test set as a reference for later analysis.

Scales. Plots in the first row in Fig. 3.7 show results of object ratio with different scales. We can easily observe that the ratios of small objects on the failure test sets are higher than that on the whole test set. And as the RPD value increases, the ratio of small objects decreases. For large objects, we can obtain similar observations but in the reverse direction. We infer that small objects are difficult for all the models across corruptions. On the contrary, large objects might be the key point for performance drop gaps among models and corruptions. In terms of medium objects, several cases show lower ratios than the extreme case. Five of all seven cases are Faster R-CNN with ResNet50 backbone, showing great advantages on medium objects.

Instance Ratio and Crowd Ratio. The bottom left plot shows a negative correlation between RPD and the instance ratio. We further measure their correlation by Kendall's tau with a coefficient of -0.715 and p-value of $3.10e-27$, which confirms our observation of a strong negative correlation. It implies images with more instances are vulnerable across corruptions and models, while those with fewer instances contribute to differences in RPD between models and corruptions. In the bottom right figure in Fig. 3.7, most of the failure datasets are more crowded than the whole test set. We notice that only three points fall below the dashed line. All three cases occur on DETR-ResNet50 under weather corruptions (snow, fog, and frost).

Core Failure Set. To validate the intuition, we obtain a core failure set for all 7 models and 15 corruptions. The core failure set is the intersection of all failure datasets, as defined in Eq. (3.4)

$$\mathscr{D}^* = \bigcap_{e \in \mathscr{E}} \mathscr{D}_e^{\text{fail}} \tag{3.4}$$

, where $\mathscr{E}$ is the set of all the experiments(105 in total here) and $\mathscr{D}_{\text{fail}}^e$ is the failure set of experiment e. The core failure set contains vulnerable samples for all models and corruptions. There are 114 images and 1,824 instances in $\mathscr{D}^*$, and thus the instance ratio is 16.0. The instance ratio is quite large and even much larger than the maximum on the experimental

failure datasets. It's consistent with our conjectures about degradation and instance ratio. The crowd ratio of $\mathscr{D}^*$ is 0.01261, close to those suffering from severe degradation. For object scales, the small, medium, and large ratios are respectively 0.471, 0.356, and 0.160. Compared with results in the first row of Fig. 3.7, $\mathscr{D}^*$ has fewer large objects and more small and medium objects. It agrees with our inference on object scales.

What's more, we find that $\mathscr{D}^*$ only includes 69 of all 80 categories. Categories not involved in $\mathscr{D}^*$ include airplane, stop sign, parking meter, bear, zebra, giraffe, frisbee, skis, snowboard, surfboard, and hair drier. In Fig. 3.8, we observe differences in category distributions between the core failure dataset and the whole test set. The percentage of dominant person instances decreases from 30% to 25%. In the meantime, we notice increased percentages of cups, bottles, dining tables, books, and bananas. As instances of these categories frequently appear in occluded scenarios, we argue the category distribution shift agrees with the increase in instance ratio.

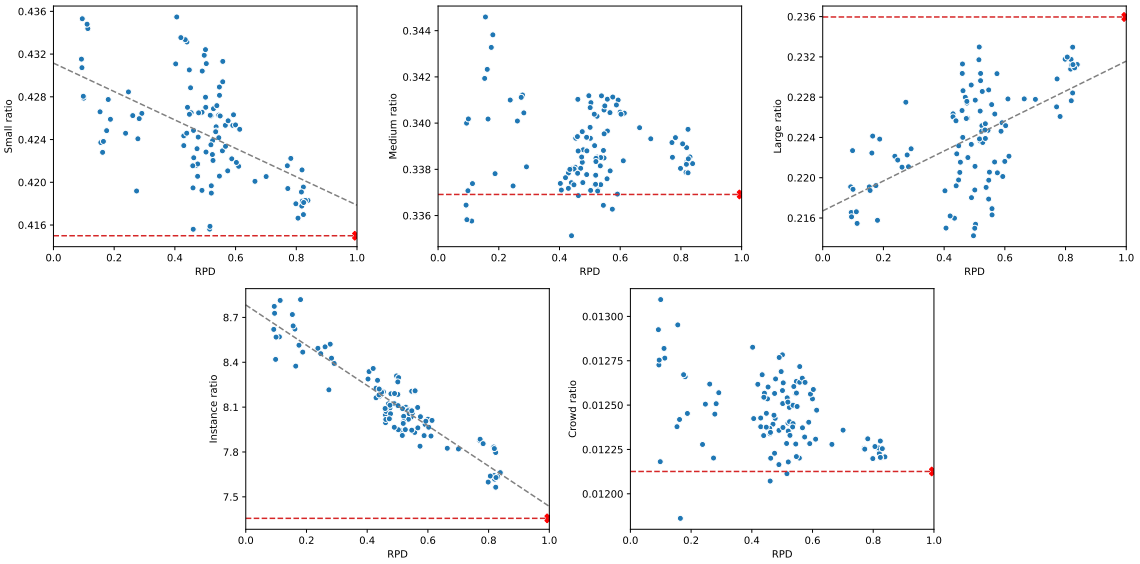


Figure 3.7: Failure patterns for object scales, instance ratio, and crowd ratio.

Clues for Repair. Based on our conjectures from the analysis of the failure datasets, we summarize some clues for repairing detectors. Methods that help detection with dense instances could benefit all models under all corruptions, while those that can improve

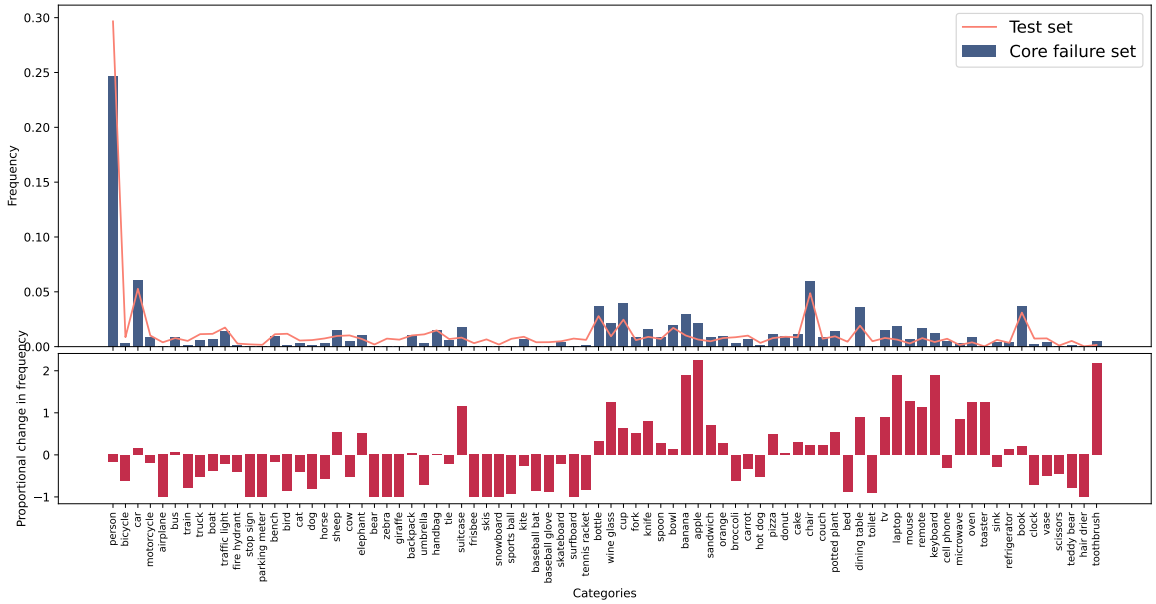


Figure 3.8: Object categories on the core failure dataset have a different distribution with the whole test set.

detection with sparse instances could benefit some of the models. Methods with better small object detection could help to repair object detectors, and those with a preference for detecting large objects could be beneficial for some models.

Answer to RQ1.3: Yes. We observe a negative correlation between performance degradation and average instances per image of the failure dataset. In addition, when compared with the whole test dataset, there are more small objects and fewer large objects in the failure datasets. Based on the observations of correlations between performance degradation and object scales, we infer that small objects are vulnerable across models and corruptions while large ones contribute to the degradation gaps among experiments. We find some clues for better detector repairing as well.

3.5.4 RQ1.4: What are the major error types of detectors under common corruptions?

We try to synthesize the error types based on official COCO toolkits [40] in this section. In object detection, performance and error analysis are roughly defined by several error types based on AP on the test set. Localization error means inaccurate predicted bounding boxes overlapping with the ground truth at an IoU value lower than the threshold. Classification error is caused by predicting the objects into incorrect categories. More specifically, the

official COCO Toolkit [40] divides the classification error into errors in or out of a super category as shown in Fig. 3.9. False positive samples are boxes that cannot match any ground truth. False positive error is a result of misclassification between foreground and background objects. Although it seems like another type of classification error, background objects are not a specific category but include all categories beyond interest. False negative error is a result of missing detection. The quantitative error is estimated by the gap between its potential upper bound. We obtain the estimated error values for each type across models corrupted by various distortions. Then we compute the error that increases the most after corruption for each experiment and treat it as the major error type. We find that the false negative error (missing detection) increases the most in general. It confirms the importance of developing repair schemes for object detection.

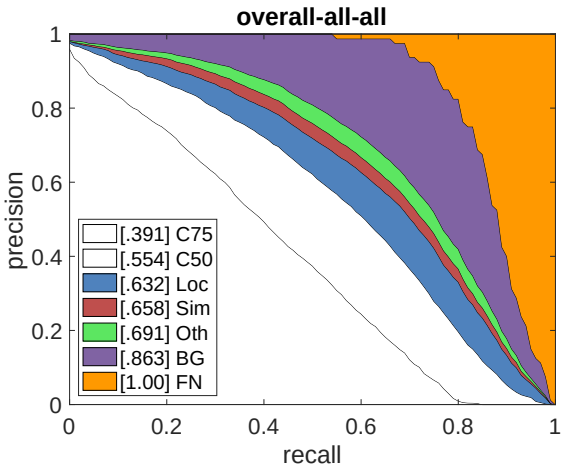


Figure 3.9: An example of error analysis results provided by COCO toolkit. Errors are shown in colored regions.

Answer to RQ1.4: The missing detection error increases the most in general, and Faster R-CNNs show different error patterns from the others.

3.6 Repair Experimental Results

Armed with analysis in the last section, we aim to conduct a comprehensive study and come up with inspirational results by analyzing repairing methods in object detection.

In this section, we will study each repairing method in detail. For each method, we study its performance on repairing and step further to obtain its characteristics and clues for improvement. In addition, we conduct a comparative analysis for a comprehensive understanding of repair. Finally, we highlight that we find limitations and possible solutions for repairing methods.

3.6.1 Fine-tuning

Fine-tuning is a naive and efficient scheme for repair. As we discussed in Sec. 3.3, the key point here is the construction of the training datasets. In this section, we focus on repairing detectors by fine-tuning the models on the failure datasets and combined datasets. The first one refers to directly fine-tuning the models on the failure training datasets. The training datasets of the latter method are composed of the failure training sets and the corresponding clean samples.

Fine-tuning on combined data shows its effectiveness on all models and corruptions. However, fine-tuning on failure data works on lots of configurations but fails to repair some models.

Detectors. In Fig. 3.10, both of the two fine-tuning methods work for repairing detectors and result in about 10% improvement for all the detectors. Faster R-CNN with ResNet50 and FPN gains the most improvement in general. Fine-tuning on combined datasets shows advantages over the other one across models. Specifically, fine-tuning only with failure data on DETR-ResNet50 would result in some lower performance after repair. In addition, we notice DETR-ResNet50 suffers from a 41% performance drop on the clean test set after fine-tuning only on the failure dataset of zoom_blur.

Corruptions. We perform analysis of the two methods in the radar plots based on corruption types in Fig. 3.11. We notice fine-tuning on combined datasets improves models under all corruptions. However, fine-tuning only with the failure data fails on zoom_blur. They

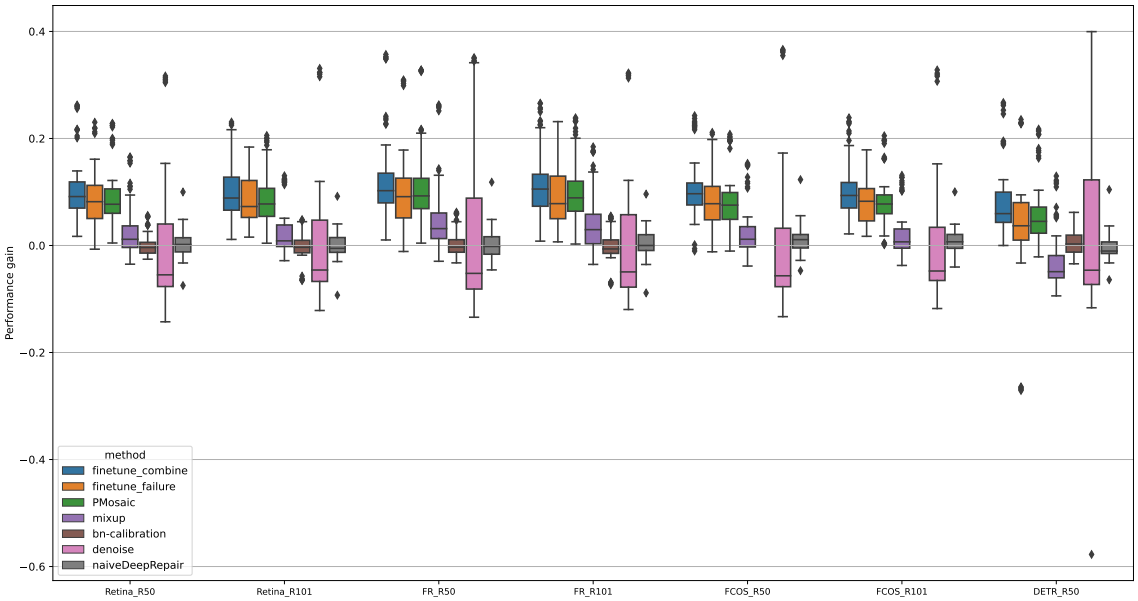


Figure 3.10: Performance improvement for detection models after being repaired by methods.

show comparable performance gains for brightness. For other corruptions, fine-tuning on combined datasets performs better. For both methods, pixelate obtains the most performance gains. Besides, among the three most degraded corruptions(glass_blur, zoom_blur, and pixelate), only zoom_blur shows improvement of less than 10% after fine-tuning. Glass_blur and zoom_blur suffer much more degradation than others on the clean test set after fine-tuning.

Statistical Tests. To validate the improvement, we conduct t-tests for each configuration with repeated experiments. We assume repeated experiments are normally distributed. For fine-tuning on combined datasets, FCOS-ResNet50 and DETR-ResNet50 corrupted by brightness cannot reject the hypothesis of no improvement. For fine-tuning with the failure training sets, there are seven cases of failing to reject the null hypothesis. Three of them were on zoom_blur and four of them occur on DETR-ResNet50.

Analysis on Forgetting. To better understand fine-tuning-based methods, we conduct a case study on RetinaNet-ResNet50 corrupted by gaussian_noise. We plot the learning curves of AP scores on the clean, corrupted test sets, and their mean performance, as shown in

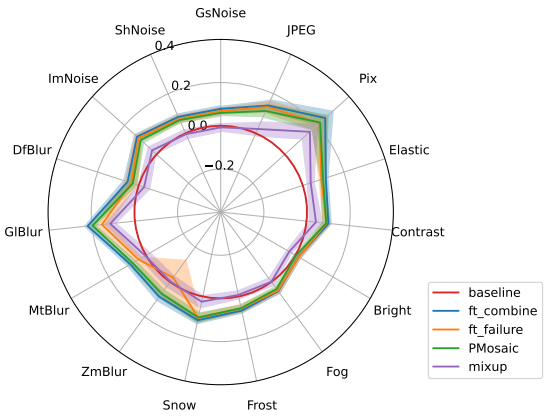


Figure 3.11: Performance gain on corruptions for fine-tuning-based and augmentation-based methods.

Fig. 3.12. We notice that both fine-tuning-based methods suffer from catastrophic forgetting. The performance keeps decreasing on the clean test set as fine-tuning progress goes on. The model converges on the corrupted datasets after training for some epochs. Combined datasets can be helpful for slowing down catastrophic forgetting while it suffers from severe forgetting to fine-tune with only failure samples. That might be the reason that fine-tuning with combined datasets achieves better performance gains. Fine-tuning with only the failure datasets shows much more improvement on the failure test set. However, we notice it drops too much on the clean test set. Common catastrophic forgetting occurs across models and corruptions when fine-tuning the detectors with only failure data.

Challenges and Opportunities. We observe catastrophic forgetting in fine-tuning-based repairing methods. Repairing can benefit from works to solve catastrophic forgetting, *e.g.*, continual learning [42].

3.6.2 Fusing Images with Augmentations

In this part, we study methods based on augmentations to fuse images from different domains. As noted before, we focus on PMosaic and MixUp in this paper and conduct analysis based on pilot experiments. We discuss the problem of Mosaic in Sec. 3.3 and propose PMosaic

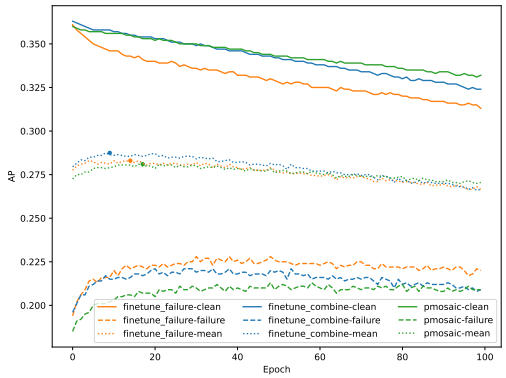


Figure 3.12: AP scores on clean, failure datasets and their average values across training epochs. The fine-tuning-based models suffer from catastrophic forgetting. It's better to fine-tune with combined data than on only failure datasets. PMosaic converges slower than the other two methods.

as an improved version. Following two research questions, the pilot experiments show that MixUp is not an effective approach to repairing detectors. In this section, we'll experiment with more configurations to expand our provisional conclusions.

Implementation Details. Our experimental settings are similar to those in fine-tuning with combined data. As the two augmentations require square images, we set the input image scales to 1024x1024, very similar to 800x1333 in naive fine-tuning. The padding values for both augmentations are 0. The ratio range of MixUp is 0.8 to 1.6, the same as that in YOLOX. In PMosaic, inputs are sampled uniformly from the raw images and the mixed ones. For PMosaic, we train the detectors for 24 epochs due to slow convergence.

PMosaic

In general, PMosaic shows its effectiveness to repair detectors. The maximal performance gain, 32.7%, appears at Faster R-CNN with ResNet50 corrupted by pixelate. However, based on Table A.5, PMosaic fails to repair the DETR-ResNet50 corrupted by defocus_blur and brightness and FCOS-ResNet50 corrupted by brightness. For the last one, its improvement on the failure test set and decrease on the clean test set are very close, and thus it results in a marginal performance drop. In Sec. 3.6.1, fine-tuning on combined datasets fails to

obtain remarkable performance gain on brightness. Therefore, it's not surprising to find PMosaic with no improvement in brightness since we study PMosaic based on fine-tuning with combined data.

In Table 3.6, PMosaic does not lead to much more performance drops of DETR-ResNet50 on brightness. And we notice that it obtains very a marginal performance gain after PMosaic. Therefore, we observe AP scores of different object scales as shown in Table 3.6. Compared with that before repair, PMosaic improves the detector on small and medium objects but shows no improvement on large objects. PMosaic performs better than fine-tuning with combined datasets on small objects. However, it has much lower performance on medium and large objects. As discussed in Sec. 3.3, PMosaic prefers small objects while degrading large objects. This indicates that it explains the failure of PMosaic on DETR-ResNet50 under brightness.

Different from what happens on brightness, DETR-ResNet50 after PMosaic on defocus_-blur shows comparable improvement with fine-tuning on combined datasets. However, it suffers from much more performance drop on the clean test set. We can observe these from Table 3.7. In fact, the results are reported at epoch 10 and epoch 7 for PMosaic and fine-tuning respectively. It seems not a result of catastrophic forgetting. Furthermore, we notice PMosaic has a much lower AP score on the clean test set even at the first epoch. We do not find clear reasons for its severe degradation and we leave it for further study.

Table 3.6: Results of DETR-ResNet50 corrupted by brightness. PMosaic prefers small objects but degrades medium and large ones.

Repair	AP@Cl	AP@Fl	AP@Avg	AP@Fl@SML		
-	40.0	33.0	36.5	17.1	37.5	50.1
PMosaic	38.5	33.1	35.8	17.6	37.4	50.8
Fine-tune	39.0	34.0	36.5	17.3	38.8	51.6

Table 3.7: Results of DETR-ResNet50 corrupted by defocus_blur. PMosaic improves the detector on failure data. But the performance on the clean test set drops too much when compared with fine-tuning on the combined datasets. It might be a result of slow convergence and catastrophic forgetting.

Repair	AP@Cl	AP@Fl	AP@Avg
-	40.0	18.0	29.0
PMosaic	37.5	19.6	28.55
Fine-tune	38.8	20.0	29.4

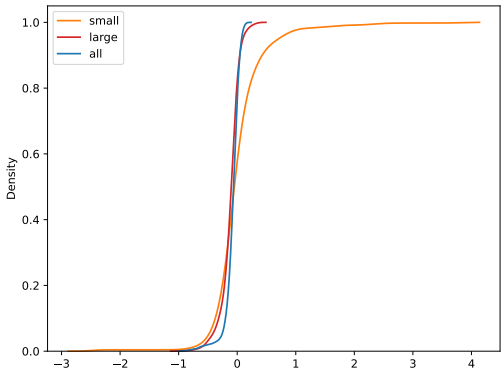
Detectors. In Fig. 3.10, Faster R-CNN with ResNet50 as the backbone benefits the most from PMosaic. Among the seven models, CNN-based detectors, *i.e.*, all detectors except DETR-ResNet50 achieve similar improvement, around 8.6%. DETR-ResNet50 has slightly lower performance, as for brightness and defocus_blur, it shows no improvement after being repaired by PMosaic. In addition, we compare the results of PMosaic with fine-tuning on the combined datasets. In Fig. 3.10, we can see that most models have fewer performance gains with PMosaic. For RetinaNets and Faster R-CNNs, PMosaic performs closer to naive fine-tuning than others.

Corruptions. With the help of PMosaic, we can repair the detectors and improve their AP scores on both the clean test sets and the failure datasets for most corruptions. Models corrupted by pixelate achieve the most performance gains, while models with brightness show the least improvement. Two of the three cases, when PMosaic fails to conduct repairing, occur in brightness. Based on observations in Sec. 3.5.3, most of the failure datasets generated by brightness own much fewer medium objects.

Statistical Tests. Similar to what we do in the last section, we attempt to validate the improvement of PMosaic. We find that all models under brightness fail to pass t-tests. It is consistent with the observations when we conduct corruption-based analysis. Besides, DETR-ResNet50 corrupted by defocus_blur cannot reject the null hypothesis either.

Analysis on Scales. As noted before, PMosaic generates more small objects while it suppresses large objects. In Sec. 3.5.3, we infer that methods of improving small object detection would benefit most detectors. The general improvement with PMosaic agrees with our conjecture. And about 50% of the configurations obtain improvement in AP scores of small objects. For large objects, we suggest the performance changes vary for different configurations. In Fig. 3.13, we plot the kernel-estimated cumulative distribution function of relative improvement for PMosaic *w.r.t.* fine-tuning based on AP scores on different object scales. First of all, the curve of small objects locates on the right of that of large ones, indicating that small objects benefit more than large objects. In some cases, large objects show to have improvement in performance. However, in most configurations, AP scores on large objects drop when compared to naive fine-tuning. In addition, we notice that the curve of the overall AP score is close to that of large objects. It indicates that large objects might be to blame for overall performance degradation when compared PMosaic with naive fine-tuning.

Figure 3.13: The kernel-estimated cumulative distribution function of relative improvement for PMosaic *w.r.t.* fine-tuning based on AP scores on different object scales. It indicates that PMosaic improves small objects in general but degrades large objects. The overall performance of PMosaic is correlated to its performance drop on large objects.



MixUp

Unlike previous methods, MixUp fails to repair lots of models. The relative performance gain varies from -9.4% to 26.3%. Around 60% of the configurations suffer from performance

drop after being repaired by MixUp. To investigate the reasons for its failure, we study a case in detail before presenting results on more models and corruptions.

Case Analysis. In particular, to further investigate the reasons, we propose two research questions and conduct an empirical analysis on a pilot experiment.

- Can MixUp boost detection performance during repair?
- Can MixUp help information fusion across domains?

The first question aims to investigate whether MixUp can reduce the detectors' learning error if applied to repair. To answer the first question, we fine-tune the model with MixUp on only the clean dataset and the results are available in the fourth line of the Table 3.8. It would result in a severe performance drop on both the clean test set and the corrupted test set even when compared to the original detector before repair. It implies that the augmentation itself would harm the detection performance when applied in repair. Zhang *et al.* [47] find the correlation between the performance gains of MixUp on the detectors and the utilization of MixUp in their ImageNet pre-trained models. Based on the previous results, we infer that it might be the inconsistency of the augmentation utilization between the detector's training and repair that is to blame for the degradation. As a result, we find that MixUp cannot boost detection during the repair.

As for the second question, we first fuse images both with or without corruption and forbid fusion between images across domains. By comparing results in the third and fifth line of Table 3.8, we notice that MixUp doesn't contribute to information fusion across domains as expected. Moreover, we add another pilot experiment, noted as the Corrupted-oriented MixUp in Table 3.8, to roughly estimate the upper bound of knowledge fusion between domains for MixUp in repairing object detectors. In this experiment, one clean image and its corrupted counterpart are fused by a random weight without extra resizing and cropping. There is no inter-image fusion as the input of the detector. Hence, the ground

truth bounding boxes are not duplicated. It achieves better AP scores on both test datasets than naive MixUp but fails to exceed the fine-tuning baseline. Therefore, we confirm that MixUp cannot effectively help improve information fusion across domains.

Table 3.8: Pilot experiments on MixUp. MixUp degrades the performance compared with naive fine-tuning. Even worse, we notice that the augmentation itself would harm the detection performance when applied in repair, as is shown in the fourth line of the table. From the results in the last two lines, we find MixUp cannot effectively help improve information fusion across domains.

Repair	AP@Cl	AP@Fl	AP@Avg
-	36.6	16.3	26.45
Fine-tune	35.8	21.4	28.6
MixUp	34.3	18.6	26.45
MixUp w. clean data	35.2	14.8	25.0
MixUp in individual domains	34.3	18.6	26.45
Corruption-oriented MixUp	35.5	21.3	28.4
BGMixUp	34.8	19.2	27.0
MixUp w. small weights	34.9	15.0	24.95

Detectors. In Fig. 3.10, we find that DETR with ResNet50 fails to obtain improvement with MixUp and suffers from about 5% performance drop. Faster R-CNN with ResNet50 benefits the most from MixUp. All other five models have marginal improvement after being repaired by MixUp. Similar to the analysis for PMosaic, we also compare MixUp's results with naive fine-tuning. Six of all seven detectors show about 100% less performance gain than fine-tuning. However, DETR-ResNet50 has about 300% less improvement compared to fine-tuning.

Corruptions. Models corrupted by pixelate benefit the most from MixUp. MixUp fails to repair the detectors under brightness and defocus_blur. The shape in its radar figure

on different corruptions is quite similar to those of the previous three methods. Besides, we notice that MixUp shows less improvement than fine-tuning on failure datasets on 14 corruptions except zoom_blur.

Statistical Tests. MixUp only passes the t-tests for DETR-ResNet50 under pixelate and elastic_transform. It fails to pass the tests for Faster R-CNN with ResNet50 corrupted by defocus_blur and brightness. For RetinaNet-ResNet50, RetinaNet-ResNet101, and Faster R-CNN ResNet101, it cannot reject the null hypothesis under 6, 8, and 6 corruptions. In addition, all models fail to achieve statistically remarkable improvement under brightness. For elastic_transform and pixelate, MixUp succeeds in repairing all the detectors. For impulse_noise, contrast, jpeg_compression, and snow, all detectors except DETR-ResNet50 pass the tests.

Analysis on Instance Ratio. In the case analysis, we empirically study the functionality of MixUp on one pilot configuration with two research questions. Here we aim to turn to the instance ratio based on analysis in Sec. 3.5.3. MixUp includes all the boxes of merged images and thus it drastically increases the instance ratio. In Sec. 3.5.3, we find that the failure dataset has a much larger instance ratio. Then MixUp aggravates it and makes the detectors more difficult to repair since modern deep object detectors perform poorly on crowded images. In Table 3.8, corruption-oriented MixUp, which mixes the images without increasing the instance ratio, achieves good performance. To validate it, we further conduct another experiment, notated as BGMixUp in Table 3.8. We add one image $\mathbf{I}_a$ with a very small weight(0.01) to the original image $\mathbf{I}_o$ and drop all the boxes in $\mathbf{I}_a$. When compared with MixUp, BGMixUp improves AP scores on both the clean and failure test sets. In addition, we conduct another experiment, where we directly apply the small weight without dropping the boxes, and it shows terrible performance. The three experiments together indicate the impact on instance ratio is a potential reason why MixUp fails.

Summary

In this section, we study two potential augmentations for repair. PMosaic is closely related to object scales, one of the most important and challenging problems in object detection. MixUp impacts on instance ratio, and thus is correlated with the occlusion problem. According to clues as noted in Sec. 3.5.3, we suggest the necessity of augmentations designed for repair.

3.6.3 BN Calibration

In this part, we explore the possibility of repairing the detectors with BN calibration. In Sec. 3.3, we discuss why we calibrate batch normalization layers on classification. In this work, we utilize the style transfer model to generate the images for classification and later works can take more effective and efficient methods for image generation. We also conduct BN calibration on clean and corrupted images from ImageNet [16] to estimate the upper bound of BN calibration in repairing object detectors.

Implementation Details. For BN calibration on classification, we calibrate the classification models with batch-size 32 for 2,000 iterations on the dataset combined with clean and corrupted samples from ImageNet. In style-transfer-based BN calibration, the model for style transfer is the same as Deep Repair [7]. We randomly sample 32k images from ImageNet as the content and 10 corrupted images in the core failure dataset as the style to generate 32k corrupted images for the classification model. We conduct the experiments on the checkpoints from the official torch hub, the same ones as the pre-trained models in the training process. FCOS-ResNet50 and FCOS-ResNet101 are not included in BN calibration because they utilize the Caffe versions of pre-trained classification models.

Detectors. In general, BN calibration improves some of the failure configurations. All six CNN-based models show similar performance gain and Faster R-CNN with ResNet101 as backbone has slightly lower improvement. Notably, DETR-ResNet50 benefits much more

than other detectors from BN calibration across corruptions. In the meantime, its performance shows a larger variance than others. Transformers usually prefer layer normalization to batch normalization. It inspires us to take robustness into consideration when choosing normalization operations in neural networks.

Corruptions. BN calibration improves six of all fifteen corruptions. Contrast benefits the most after being repaired by BN calibration. Moreover, it achieves obvious performance gain on image distortions, *e.g.*jpeg_compression, pixelate, and contrast. And it shows only marginal improvement in the weather, like snow, frost, and fog. However, it fails on usual noises and blurs except impulse_noise.

Statistical Tests. In all 105 failure configurations, only 22 of them pass t-tests of significant improvement. All the models corrupted by jpeg_compression, pixelate, and contrast succeed in the tests. All the detectors pass the t-tests for four to five corruptions. Note that DETR-ResNet50 doesn't show significant advantages over other models in terms of corruption types. Together with our observations in the last paragraph, DETR-ResNet50 benefits more from BN calibration than others under some corruption types. In Table A.5, we notice that it gains more improvement for pixelate and jpeg_compression after repair.

Compared with Estimated Upper Bound. We estimate the upper bound by calibrating the classifier with ImageNet-C [5]. In Fig. 3.14 and Fig. 3.15, we notice that it achieves comparable or better performance when compared with the estimated upper bound. In shot_noise, impulse_noise, and jpeg_compression, our approach significantly outperforms the estimated upper bound. For the classifier's accuracy after calibration, the estimated upper bound is much better than our approach. Therefore, the outperformance should be a result of mismatching between tasks. As noted in Sec. 3.3, it would be better to calibrate the detectors with detection data if the detectors are trained with updated BN layers.

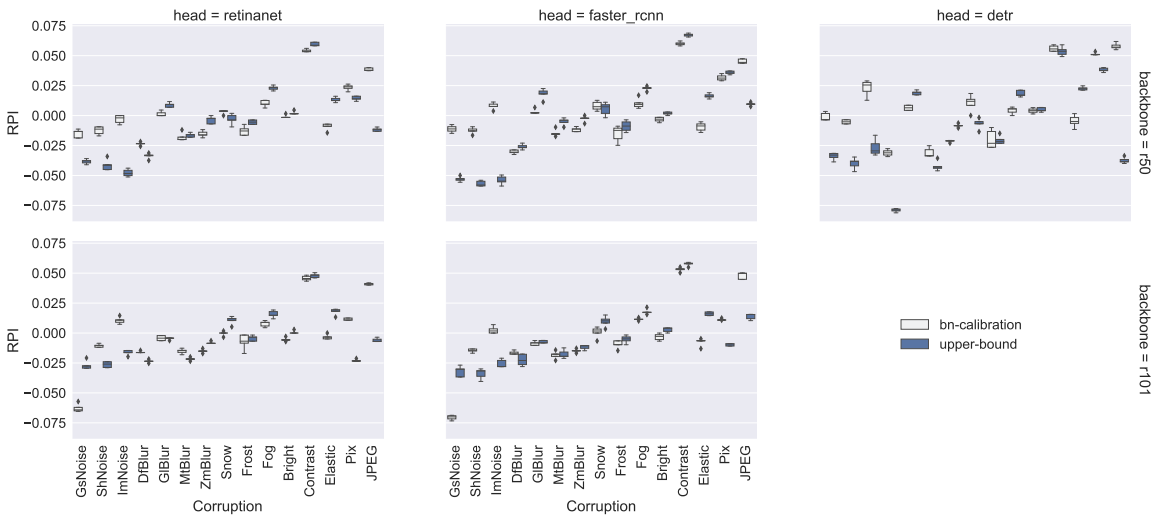


Figure 3.14: Comparison between BN calibration and its estimated upper bound.

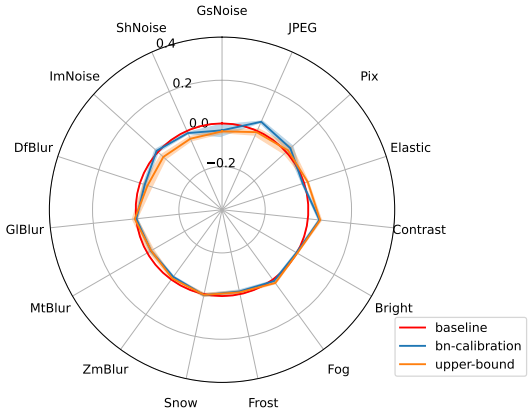


Figure 3.15: Comparison between BN calibration and its estimated upper bound in terms of corruptions. It sometimes outperforms its upper bound as a result of task transfer.

Challenges and Opportunities. Although BN calibration fails to show great improvement for repair, it's still attractive to develop similar simple and efficient methods for repair. What's more, its diverse performance on different corruptions motivates us to study the semantic meaning of statistical parameters in neural networks. It might further benefit interpretable neural networks, especially when very few works aim to interpret object detectors.

3.6.4 Denoising

Denoising aims to obtain clean images from corrupted ones. Lots of related works can act as a denoiser and we select a self-supervised method, Neighbor2Neighbor [46]. During training, it optimizes the differences between two sub-image samples to smooth the images. We provide a supervised version as a possible upper bound to inspire further repairing work based on denoising.

Implementation Details. We train an independent denoiser in each experiment for 1,000 epochs. The supervised versions share the same architectures as the self-supervised ones. We notice that the failure training set with 1,000 random samples is sufficient to train a well-performed denoising model. So, there is no concern about the size of the datasets. In addition, we empirically find that the supervised denoiser trained with only corrupted images would fail to conduct identity mapping for clean images. Therefore, we train the supervised denoiser with images equally sampled from corrupted and clean data.

Detectors. All the detectors show similar performance gains with denoising. We observe Faster R-CNN-ResNet50 and DETR-ResNet50 have larger variance in improvement. Note that denoising has much larger values than other methods in variance in Fig. 3.10. When we later turn to analyze denoising based on the corruptions, we can know the reason for the variance.

Corruptions. The performance of the self-supervised denoiser is closely related to the corruption types. The denoiser improves remarkably on gaussian_noise, shot_noise, impulse_noise, pixelate, and jpeg_compression. However, it fails to work well on other corruptions. The correlation between performance and corruption types is related to the optimization target for smoothness. For those corrupted in the frequency domain or other dimensions, they require different strategies.

Statistical Tests. We also conduct t-tests for denoising. 70 of all 105 configurations fail to pass the t-tests. Gaussian_noise, shot_noise, impulse_noise, pixelate, and jpeg_compression succeed in the tests for all the detectors. The other ten corruptions fail the tests for all the detectors. The results agree with our observations in the last paragraph.

Compared with the Supervised Upper Bound. Although the self-supervised denoiser works on some corruptions, its supervised counterpart achieves great performance for most corruptions. However, brightness fails to benefit from the supervised denoiser. On gaussian_-noise, shot_noise, impulse_noise, and jpeg_compression, the self-supervised denoiser has comparable performance with the supervised one. And the results are available in Fig. 3.16 and Fig. 3.17.

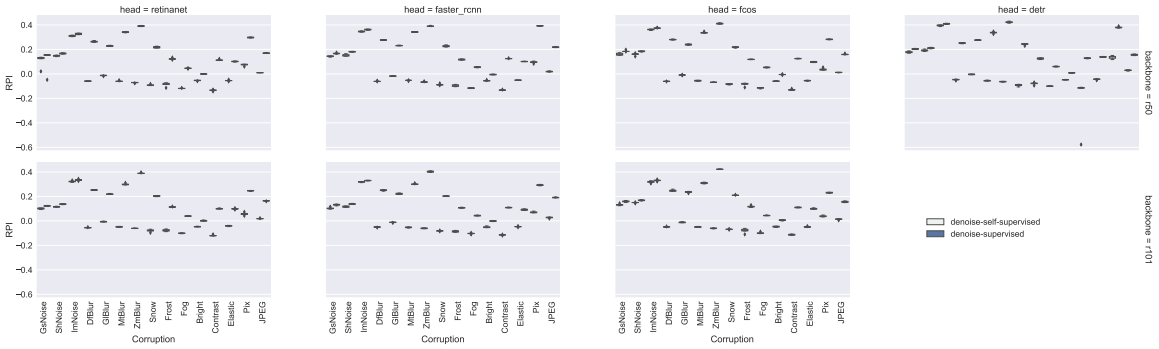


Figure 3.16: Comparison between self-supervised denoising and its supervised counterpart.

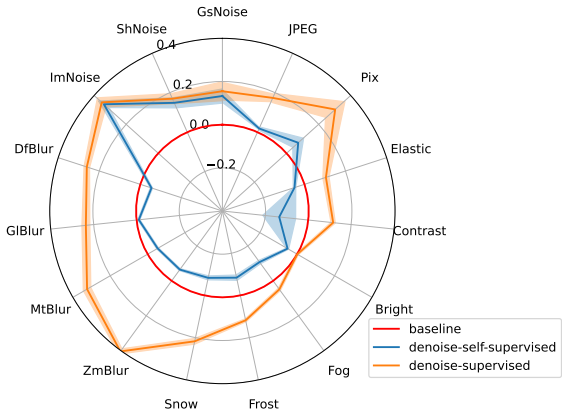


Figure 3.17: Comparison between self-supervised denoising and its supervised counterpart in terms of corruption. There are large performance gaps in terms of blurs and weather.

Challenges and Opportunities. Denoisers with better performance will potentially boost more on the repair. Here we study one self-supervised denoising method. It fixes failures in some corruptions but fails in other cases. In the meanwhile, its supervised counterpart shows great performance in all the corruptions. Therefore, we expect further work on denoising to bridge the gaps and enhance repair.

3.6.5 Naive DeepRepair

DeepRepair [7] is an approach to repairing classifiers and it cannot be directly applied to detection. We study a naive version of DeepRepair, which costs much more time than other methods. In general, it does not achieve excellent performance gains, which might be a result of performance problems in style transfer. However, it still stands out in some cases, *e.g.*, contrast. Notably, it does not cause performance drops on the clean test set and it is not impacted by catastrophic forgetting.

Implementation Details. We adopt a very simple baseline to study the feasibility of augmenting the training dataset $\mathscr{D}_{\text{train}}$. DeepRepair [7] utilizes style transfer on the training dataset and Augmix [55] to further fuse the images. As AugMix is seldom observed in detection models and not available in MMDetection, we directly fine-tune the model with the clean and transferred data without augmentations. Similar to DeepRepair, we employ the WCT2 [88] to transfer the COCO training set, *i.e.*, $\mathscr{D}_{\text{train}}$, with the failure samples as the style to extend the failure dataset to the augmented training set $\mathscr{D}^{c}_{\text{train}}$. We then fine-tune the detectors with both $\mathscr{D}_{\text{train}}$ and $\mathscr{D}^{c}_{\text{train}}$ for one epoch. Due to limited time and computation resources, we does not repeat the experiments five times as others.

Detectors. Following a similar pipeline as previous methods, we first analyze the repair results based on detection models. It shows limited and comparable performance gains for the seven detectors. The performance improvement for FCOS-ResNet50 is marginally higher than others and that for DETR-ResNet50 is slightly lower than others. However, we

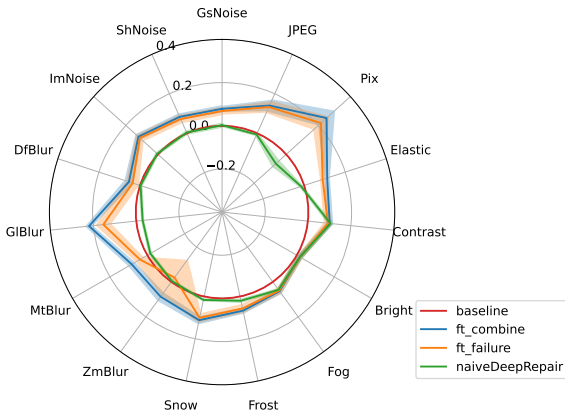


Figure 3.18: Corruption-oriented analysis for the naive DeepRepair. It's outstanding in contrast.

notice that it's the best method in DETR-ResNet50 corrupted by fog and contrast.

Corruptions. Then, we step to conduct a corruption-oriented analysis as illustrated in Fig. 3.18. For fog, snow, brightness, and contrast, it improves performance remarkably. However, on glass_blur and pixelate, the performance drops after applying the naive DeepRepair. In most cases, the naive DeepRepair does not have a significant negative impact on the performance of the clean data. When observing its results on the failure test set, it achieves significant performance gains on contrast, frost, and fog. And the performance of the detectors drops on defocus_blur, glass_blur, motion_blur, pixelate, jpeg_compression, and elastic_transform, especially on glass_blur and pixelate. When compared to the other two fine-tuning-based methods, it has better performance on contrast. As we discussed in Sec. 3.3, we find its upper bound is rather attractive, which might be caused by style transfer methods.

Challenges and Opportunities. Note that the naive DeepRepair successfully retains the performance on the clean test sets for all the detectors. The performance drops on the clean data are less than 2.5% and are within 1% in most cases, which are much smaller than the others. It is not limited by catastrophic forgetting and leaves us with incremental updating

without concerns. We also notice that it results in the most improvement on the failure test set and the least performance drop on the clean test set for contrast. It indicates the method has enormous potential and is rather promising for repair on object detection. In the meantime, it achieves less improvement in other corruptions than fine-tuning-based methods. We are convinced a better style transfer approach would result in significant improvement for repair. What's more, as style transfer methods [89] usually do not focus on transferring image corruptions, other approaches, *e.g.*, an inverted denoiser, could be applied to improve DeepRepair.

3.6.6 Comparative Analysis

So far, we've studied the proposed repairing approaches in detail based on their performance from the viewpoint of corruption, detectors, and statistical test results. We've discussed the strengths and weaknesses, and challenges and opportunities for each method. To summarize our experimental results above, we conduct a comparative analysis of the methods based on effectiveness, efficiency, and their impact on inference.

Corruption	Retina_R50	Retina_R101	Faster_R50	Faster_R101	FCOS_R50	FCOS_R101	DETR_R50	Acc	Avg RPI
GsNoise	❶❷❸④⑤⑥❼ (0.109)	❶❷❸④⑤⑥⑦ (0.101)	❶❷❸❹⑤⑥❼ (0.144)	❶❷❸④⑤⑥⑦ (0.104)	❶❷❸④ - ⑥❼ (0.161)	❶❷❸④ - ⑥❼ (0.133)	❶❷❸④⑤⑥⑦ (0.178)	33/49	0.133
ShNoise	❶❷❸④⑤⑥❼ (0.148)	❶❷❸④⑤⑥❼ (0.116)	❶❷❸❹⑤⑥⑦ (0.153)	❶❷❸④⑤⑥⑦ (0.116)	❶❷❸④ - ⑥❼ (0.159)	❶❷❸④ - ⑥❼ (0.149)	❶❷❸④⑤⑥⑦ (0.192)	33/49	0.148
ImNoise	❶❷❸❹⑤⑥❼ (0.310)	❶❷❸❹❺⑥⑦ (0.323)	❶❷❸❹❺⑥❼ (0.346)	❶❷❸❹⑤⑥⑦ (0.318)	❶❷❸❹ - ⑥❼ (0.362)	❶❷❸❹ - ⑥❼ (0.318)	❶❷❸④❺⑥❼ (0.395)	42/49	0.339
DfBlur	❶❷❸④⑤⑥⑦ (0.052)	❶❷❸④⑤⑥⑦ (0.057)	❶❷❸④⑤⑥⑦ (0.065)	❶❷❸④⑤⑥⑦ (0.063)	❶❷❸④ - ⑥❼ (0.071)	❶❷❸④ - ⑥❼ (0.054)	❶②③④⑤⑥⑦ (0.014)	21/49	0.054
GlBlur	❶❷❸❹⑤⑥⑦ (0.211)	❶❷❸❹⑤⑥⑦ (0.223)	❶❷❸❹⑤⑥⑦ (0.234)	❶❷❸❹⑤⑥⑦ (0.227)	❶❷❸❹ - ⑥⑦ (0.223)	❶❷❸❹ - ⑥⑦ (0.229)	❶❷❸❹⑤⑥⑦ (0.192)	28/49	0.22
MtBlur	❶❷❸❹⑤⑥⑦ (0.090)	❶❷❸④⑤⑥⑦ (0.082)	❶❷❸❹⑤⑥⑦ (0.103)	❶❷❸❹⑤⑥⑦ (0.088)	❶❷❸④ - ⑥⑦ (0.086)	❶❷❸❹ - ⑥⑦ (0.083)	❶②❸④⑤⑥⑦ (0.052)	24/49	0.083
ZmBlur	❶②❸④⑤⑥⑦ (0.074)	❶❷❸④⑤⑥⑦ (0.089)	❶②❸❹⑤⑥⑦ (0.094)	❶❷❸❹⑤⑥⑦ (0.111)	❶❷❸④ - ⑥⑦ (0.085)	❶❷❸④ - ⑥❼ (0.095)	❶②❸④⑤⑥⑦ (0.042)	21/49	0.084
Snow	❶❷❸❹⑤⑥❼ (0.119)	❶❷❸❹⑤⑥❼ (0.114)	❶❷❸❹⑤⑥❼ (0.131)	❶❷❸❹⑤⑥❼ (0.121)	❶❷❸❹ - ⑥❼ (0.111)	❶❷❸❹ - ⑥❼ (0.108)	❶❷❸④❺⑥❼ (0.079)	35/49	0.112
Frost	❶❷❸④⑤⑥❼ (0.069)	❶❷❸④⑤⑥❼ (0.071)	❶❷❸❹⑤⑥❼ (0.073)	❶❷❸④⑤⑥❼ (0.072)	❶❷❸④ - ⑥❼ (0.063)	❶❷❸④ - ⑥❼ (0.074)	❶❷❸④⑤⑥❼ (0.046)	29/49	0.067
Fog	❶❷❸❹❺⑥❼ (0.057)	❶❷❸④❺⑥❼ (0.052)	❶❷❸❹⑤⑥❼ (0.061)	❶❷❸④❺⑥❼ (0.051)	❶❷❸❹ - ⑥❼ (0.070)	❶❷❸④ - ⑥❼ (0.047)	❶❷❸④⑤⑥❼ (0.037)	34/49	0.053
Bright	❶❷③④⑤⑥❼ (0.021)	❶❷③④⑤⑥❼ (0.021)	❶❷③④⑤⑥❼ (0.015)	❶❷③④⑤⑥❼ (0.019)	①②③④ - ⑥❼ (0.026)	❶❷③④ - ⑥❼ (0.027)	①②③④⑤⑥⑦ (0.007)	16/49	0.019
Contrast	❶❷❸❹❺⑥❼ (0.110)	❶❷❸❹❺⑥❼ (0.096)	❶❷❸❹❺⑥❼ (0.123)	❶❷❸❹❺⑥❼ (0.109)	❶❷❸❹ - ⑥❼ (0.123)	❶❷❸❹ - ⑥❼ (0.100)	❶❷❸④❺⑥❼ (0.104)	39/49	0.109
Elastic	❶❷❸❹⑤⑥⑦ (0.115)	❶❷❸❹⑤⑥⑦ (0.106)	❶❷❸❹⑤⑥⑦ (0.124)	❶❷❸❹⑤⑥⑦ (0.109)	❶❷❸❹ - ⑥⑦ (0.098)	❶❷❸❹ - ⑥⑦ (0.114)	❶❷❸❹⑤⑥⑦ (0.116)	28/49	0.112
Pix	❶❷❸❹❺❻⑦ (0.260)	❶❷❸❹❺❻⑦ (0.206)	❶❷❸❹❺❻⑦ (0.352)	❶❷❸❹❺❻⑦ (0.259)	❶❷❸❹ - ❻⑦ (0.232)	❶❷❸❹ - ❻⑦ (0.192)	❶❷❸❹❺❻⑦ (0.258)	40/49	0.251
JPEG	❶❷❸❹❺❻⑦ (0.137)	❶❷❸❹❺❻⑦ (0.134)	❶❷❸❹❺❻⑦ (0.185)	❶❷❸❹❺❻⑦ (0.170)	❶❷❸❹ - ❻❼ (0.138)	❶❷❸❹ - ❻⑦ (0.128)	❶❷❸④❺❻⑦ (0.091)	40/49	0.14
Acc	69/105	67/105	72/105	67/105	65/105	67/105	56/105	463/735	
Avg RPI	0.125	0.119	0.147	0.129	0.134	0.123	0.12		0.128

Table 3.9: The marks 1-7 are fine-tuning with combined data, fine-tuning with failure data, PMosaic, MixUp, BN calibration, denoising, and the naive DeepRepair, respectively. The darker mark means the approach successfully repairs the detector under one corruption, while the lighter one indicates repair failure. "-" means no results. The best repair approaches are in red. The relative performance improvement values of the best methods are noted within parentheses.

Effectiveness

In Table 3.9, we present the results of repair success using statistical t-tests, with the null hypothesis assuming that a repair method does not statistically improve the detector's performance. In this study, we did not apply BN calibration to FCOS-ResNet50 and FCOS-ResNet101, as their backbone classification models were unavailable for calibration. As a result, their respective repair results are denoted with "-" in the table. According to the data presented in Table 3.9, the detectors exhibit performance gains ranging from 0.7% to 39.5%, with an average improvement of 12.8% achieved through the best repair methods.

In general, fine-tuning with combined data improves the most for repair while self-supervised denoising shows the least improvement after repair. Most repairing methods can be applied to all detectors and corruptions in our experiments. In this work, we do not apply BN calibration to FCOS-ResNet50 and FCOS-ResNet101 since their backbone classification models are not available for calibration. We compute the overall average RPI, as noted in Eq. (3.3), to evaluate each repairing method and the results are shown in Table 3.10.

For fine-tuning and augmentation-based methods, pixelate benefits the most, and brightness shows the least improvement. And Faster R-CNN with ResNet101 backbone always shows the most improvement. Among the four approaches, fine-tuning with combined data performs the best and MixUp performs the worst. These methods could roughly work across corruptions and models.

BN calibration and the self-supervised denoiser are closely related to the corruption types. BN calibration achieves remarkable performance gains on image distortions, *e.g.*jpeg_-compression, pixelate, and contrast. But it shows only marginal or no improvement on other corruptions except impulse_noise. On the contrary, the denoiser improves a lot on noises, *e.g.*, gaussian_noise, shot_noise, and impulse_noise, while it fails on other corruptions. The naive DeepRepair, which also relies on style transfer, is correlated with the corruption type. It achieves outstanding performance on the contrast. Notably, it doesn't result in a performance drop on the clean test set as other fine-tuning-based methods.

Table 3.10: Overall relative performance improvement for the repairing methods. The winning configurations mean the method achieves the best in our benchmark under the configuration.

Method	Overall RPI	Winning ConFigs
Fine-tuning with combined data	0.105	72
PMosaic	0.087	0
Fine-tuning with failure data	0.081	5
MixUp	0.023	0
Naive DeepRepair	0.005	7
BN calibration	0.002	0
Denoising	0.001	21

Impact on Inference

Training-time strategies are free lunch for inference and we don't have any concerns after deployment. Six of the repairing methods in this paper are training-time strategies. However, denoising would put some extra workload on inference speed. The denoiser works at around 4 fps on a single NVIDIA A6000 GPU.

Efficiency

In Table 3.11, we report roughly estimated costs for the approaches on a single NVIDIA A6000 GPU. For fine-tuning-based and augmentation-based methods, the repair cost depends on the detector. For denoising and BN calibration, repairing once could benefit many related models. A denoiser could be applied to multiple detectors corrupted by the same corruption. BN calibration on classification can repair all the detectors with the same backbone. Therefore, in the table, we include the average repair costs per configuration in our work in parentheses. In conclusion, fine-tuning with failure data is the most efficient among all the proposed methods.

Table 3.11: Repair costs of the repairing methods. Values in parentheses are average costs shared across detectors.

Method	Cost
Fine-tuning w. combined data	0.33 hour
PMosaic	1.4 hours
Fine-tuning w. failure data	0.2 hour
MixUp	0.33 hour
Naive DeepRepair	9.6hours
BN calibration	10hours (0.67 hour)
Denoising	23hours(0.66 hour)

3.7 Discussion

3.7.1 Future directions

We've discussed the repairing approaches in detail and compared them in the benchmark. What's more, we offer some clues to further improve detection repair. In this section, we highlight some problems and topics to inspire better repair methods for object detection.

- In our benchmark, five of all the involved methods work with fine-tuning. But most of them suffer from catastrophic forgetting, as illustrated in Fig. 3.12. There is a compelling need to handle catastrophic forgetting on object detection.

- Building upon our analysis, we anticipate the potential effectiveness of novel augmentation techniques or alternative information fusion schemes for repair, particularly in cases where the distributions of objects in terms of scales, density, and categories undergo significant changes.

- In addition to denoising and style transfer techniques tailored to patterns of common corruption, there is a pressing need to devise transformation models capable of addressing unknown types of corruption. Developing unsupervised or self-supervised robust transfer

models presents an even greater challenge, but it is crucial for handling unforeseen corruption scenarios effectively.

- It is essential to explore efficient repair schemes, as they represent a significant and promising direction for research.

3.8 Conclusion

In this paper, our focus has been on the critical task of repairing object detection models when they encounter failures caused by common corruptions. Our work has provided a comprehensive benchmark and empirical study of object detection repair approaches, shedding light on their effectiveness and highlighting areas for improvement. We have identified strengths and limitations in fine-tuning, augmentation, denoising, and calibration-based methods, offering insights into the complex landscape of repair techniques. Moving forward, addressing challenges such as catastrophic forgetting and innovating augmentation strategies, will be essential for advancing the field. As the demands for resilient object detection systems continue to grow, these future directions will play a pivotal role in enhancing the reliability and effectiveness of detection repair mechanisms.

Chapter 4

Conclusions & Future Work

4.1 Conclusion

The goal of this paper is to address the issue of object detection models failing under common corruptions and to propose methods for repairing these failures. To begin, we formally define failure and repair in the context of object detection. Our objective is to improve the performance of the model on the failure dataset while minimizing degradation on the clean set.

We first systematically analyze failure patterns in object detection and identify clues for repairing detectors. We suggest that better localization and improvement in detecting small and occluded objects would be beneficial for repairing object detectors. We then investigate potential methods for repair and construct a comprehensive benchmark for evaluating their performance.

Our experiments reveal that fine-tuning with combined datasets performs the best, while other approaches also show promise after we attempt to optimize their upper bounds. Furthermore, we highlight key points for repairing object detectors and summarize the challenges and future directions for further research.

We acknowledge that all the approaches in our benchmark have room for improvement, and we hope that our benchmark, comparative results, and identified challenges and opportunities can inspire future research in this area. Our ultimate goal is to develop high-quality object detectors that can function continuously in operational environments.

4.2 Future Directions

In summary, we have discussed various repairing approaches for object detectors and compared them in our benchmark. Based on our observations, we can provide some clues for repairing object detectors. However, there are still several key challenges and potential directions that need to be addressed in future research. Here, we outline some of these challenges and topics to inspire better repairing methods for object detection.

4.2.1 Catastrophic Forgetting

In the last section, five of all the involved methods work with fine-tuning, but most of them suffer from catastrophic forgetting. Many approaches [42, 67, 70] could be applied in repairing to handle catastrophic forgetting.

It's usually expensive to train the detection models; thus it's more important to deal with catastrophic forgetting in object detection. Repairing would be much more efficient if the detectors could only focus on the failure data without concerns about forgetting. It's a critical problem for object detection, even more than repair. However, it's more challenging to handle catastrophic forgetting on object detection than classification. For example, there exists an extreme imbalance between categories, and regularization working on the logits might fail. If we turn to feature-level regularization, the detection framework is more complex and we have to take both the image-wise features and RoI-wise features into account. It's quite interesting to solve the problem with the properties of object detection.

4.2.2 Better Augmentations

In this paper, we attempt two widely-used augmentations, MixUp, and Mosaic. However, both of them failed to achieve outstanding performance in our repairing benchmark. Augmentations designed for detection usually focus on mixing between objects and background scenes, and they don't match our problem formulation. Following the clues, we expect a novel augmentation, or other information fusion schemes, to work for repair, when the distri-

butions of objects in scales, density, and category change significantly. Their improvement would benefit generic object detection and many other detection problems.

4.2.3 Denoisers and Style Transfer for Common Corruptions

In Sec. 3.6.4, we find the self-supervised denoiser [46] in our benchmark works pretty well on some corruptions but fails on others. Similarly, the style transfer method [88] involved in Sec. 3.6.3 and Sec. 3.6.5 performs varying from the corruptions, too. These methods are designed with corruption properties and thus may not work among the common corruptions. Therefore, it is challenging and critical to propose methods for transfer between corrupted and clean images. In addition to denoising and style transfer based on patterns of common corruptions, it's important to design transformation models for unknown corruption types. It is even more difficult to develop unsupervised or self-supervised robust transferring models.

4.2.4 Efficient Repair

Repairing happens when the deployed model encounters challenges in the operational environment. Sometimes it can be urgent to repair the model. Therefore, efficiency is a crucial point in repair. In traditional detection tasks, we usually only care about the inference speed but ignore the training time. However, repairing takes both the inference and training time into account. Hence efficient repair schemes can be an important and promising direction to explore. For example, as we noted in Sec. 3.3, it is an efficient proposal to calibrate BN layers directly on detectors trained without frozen BNs, which takes about 1/8 time of fine-tuning-based methods. Moreover, it's a critical but neglected topic to achieve efficient training in all the deep learning tasks beyond repair.

Bibliography

[1] L. Jiao *et al.*, "A survey of deep learning-based object detection," *IEEE access*, vol. 7, pp. 128 837–128 868, 2019.

[2] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 8971–8980.

[3] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: State of the art," *International journal of multimedia information retrieval*, vol. 9, no. 3, pp. 171–189, 2020.

[4] Q. Dang, J. Yin, B. Wang, and W. Zheng, "Deep learning based 2d human pose estimation: A survey," *Tsinghua Science and Technology*, vol. 24, no. 6, pp. 663–676, 2019.

[5] D. Hendrycks and T. Dietterich, "Benchmarking neural network robustness to common corruptions and perturbations," *arXiv preprint arXiv:1903.12261*, 2019.

[6] C. Michaelis *et al.*, "Benchmarking robustness in object detection: Autonomous driving when winter is coming," *arXiv preprint arXiv:1907.07484*, 2019.

[7] B. Yu *et al.*, "Deeprepair: Style-guided repairing for deep neural networks in the real-world operational environment," *IEEE Transactions on Reliability*, 2021.

[8] R. Ramamonjison, A. Banitalebi-Dehkordi, X. Kang, X. Bai, and Y. Zhang, "Simrod: A simple adaptation method for robust object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 3570–3579.

[9] P. Oza, V. A. Sindagi, V. VS, and V. M. Patel, "Unsupervised domain adaptation of object detectors: A survey," *arXiv preprint arXiv:2105.13502*, 2021.

[10] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, Ieee, vol. 1, 2005, pp. 886–893.

[11] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*, Ieee, 2008, pp. 1–8.

[12] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[14] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 1, pp. 142–158, 2015.

[15] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[17] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[18] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[19] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," *Advances in neural information processing systems*, vol. 29, 2016.

[20] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[21] Z. Cai and N. Vasconcelos, "Cascade r-cnn: Delving into high quality object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6154–6162.

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[24] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European conference on computer vision*, Springer, 2016, pp. 21–37.

[25] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[26] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, 2020.

[27] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, "Yolox: Exceeding yolo series in 2021," *arXiv preprint arXiv:2107.08430*, 2021.

[28] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "Dssd: Deconvolutional single shot detector," *arXiv preprint arXiv:1701.06659*, 2017.

[29] Z. Tian, C. Shen, H. Chen, and T. He, "Fcos: Fully convolutional one-stage object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 9627–9636.

[30] T. Kong, F. Sun, H. Liu, Y. Jiang, L. Li, and J. Shi, "Foveabox: Beyound anchor-based object detection," *IEEE Transactions on Image Processing*, vol. 29, pp. 7389–7398, 2020.

[31] X. Zhou, D. Wang, and P. Krähenbühl, "Objects as points," *arXiv preprint arXiv:1904.07850*, 2019.

[32] K. Duan, S. Bai, L. Xie, H. Qi, Q. Huang, and Q. Tian, "Centernet: Keypoint triplets for object detection," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6569–6578.

[33] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," in *European conference on computer vision*, Springer, 2020, pp. 213–229.

[34] H. Hu, J. Gu, Z. Zhang, J. Dai, and Y. Wei, "Relation networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3588–3597.

[35] Z. Liu *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 012–10 022.

[36] Y. Li, H. Mao, R. Girshick, and K. He, "Exploring plain vision transformer backbones for object detection," *arXiv preprint arXiv:2203.16527*, 2022.

[37] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[39] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[40] T.-Y. Lin *et al.*, "Microsoft coco: Common objects in context," in *European conference on computer vision*, Springer, 2014, pp. 740–755.

[41] K. Chen *et al.*, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[42] G. M. Van de Ven and A. S. Tolias, "Three scenarios for continual learning," *arXiv preprint arXiv:1904.07734*, 2019.

[43] M. Sayed and G. Brostow, "Improved handling of motion blur in online object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 1706–1716.

[44] H. Qi *et al.*, "Archrepair: Block-level architecture-oriented repairing for deep neural networks," *arXiv preprint arXiv:2111.13330*, 2021.

[45] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[46] T. Huang, S. Li, X. Jia, H. Lu, and J. Liu, "Neighbor2neighbor: Self-supervised denoising from single noisy images," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 14 781–14 790.

[47] Z. Zhang, T. He, H. Zhang, Z. Zhang, J. Xie, and M. Li, "Bag of freebies for training object detection neural networks," *arXiv preprint arXiv:1902.04103*, 2019.

[48] H. Zhang and W. Chan, "Apricot: A weight-adaptation approach to fixing deep learning models," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2019, pp. 376–387.

[49] S. Ma, Y. Liu, W.-C. Lee, X. Zhang, and A. Grama, "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*, 2018, pp. 175–186.

[50] M. Usman, D. Gopinath, Y. Sun, Y. Noller, and C. S. Păsăreanu, "Nn repair: Constraint-based repair of neural network classifiers," in *International Conference on Computer Aided Verification*, Springer, 2021, pp. 3–25.

[51] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov, "Neural network repair with reachability analysis," *arXiv preprint arXiv:2108.04214*, 2021.

[52] J. Sohn, S. Kang, and S. Yoo, "Arachne: Search based repair of deep neural networks," *ACM Transactions on Software Engineering and Methodology*, 2022.

[53] X. Xie *et al.*, "Rnnrepair: Automatic rnn repair via model-based analysis," in *Proceedings of the 38th International Conference on Machine Learning*, M. Meila and T. Zhang, Eds., ser. Proceedings of Machine Learning Research, vol. 139, PMLR, 2021, pp. 11 383–11 392. [Online]. Available: https://proceedings.mlr.press/v139/xie21b.html.

[54] B Taylor, "Neural network repair with reachability analysis," in *Formal Modeling and Analysis of Timed Systems: 20th International Conference, FORMATS 2022, Warsaw, Poland, September 13-15, 2022, Proceedings*, Springer Nature, vol. 13465, 2022, p. 221.

[55] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan, "Augmix: A simple data processing method to improve robustness and uncertainty," *arXiv preprint arXiv:1912.02781*, 2019.

[56] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 113–123.

[57] Y. Chen *et al.*, "Dynamic scale training for object detection," *arXiv preprint arXiv:2004.12432*, 2020.

[58] B. Zoph, E. D. Cubuk, G. Ghiasi, T.-Y. Lin, J. Shlens, and Q. V. Le, "Learning data augmentation strategies for object detection," in *European conference on computer vision*, Springer, 2020, pp. 566–583.

[59] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[60] Y. Li, N. Wang, J. Shi, J. Liu, and X. Hou, "Revisiting batch normalization for practical domain adaptation," *arXiv preprint arXiv:1603.04779*, 2016.

[61] D. Wang, E. Shelhamer, S. Liu, B. Olshausen, and T. Darrell, "Tent: Fully test-time adaptation by entropy minimization," *arXiv preprint arXiv:2006.10726*, 2020.

[62] S. Schneider, E. Rusak, L. Eck, O. Bringmann, W. Brendel, and M. Bethge, "Improving robustness against common corruptions by covariate shift adaptation," *Advances in Neural Information Processing Systems*, vol. 33, pp. 11 539–11 551, 2020.

[63] K. Saito, Y. Ushiku, T. Harada, and K. Saenko, "Strong-weak distribution alignment for adaptive object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 6956–6965.

[64] Y.-C. Hsu, Y.-C. Liu, A. Ramasamy, and Z. Kira, "Re-evaluating continual learning scenarios: A categorization and case for strong baselines," *arXiv preprint arXiv:1810.12488*, 2018.

[65] J. Kirkpatrick *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the national academy of sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[66] F. Zenke, B. Poole, and S. Ganguli, *Improved multitask learning through synaptic intelligence*, 2017.

[67] Z. Li and D. Hoiem, "Learning without forgetting," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 12, pp. 2935–2947, 2017.

[68] A. Rannen, R. Aljundi, M. B. Blaschko, and T. Tuytelaars, "Encoder based lifelong learning," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1320–1328.

[69] P. Sprechmann *et al.*, "Memory-based parameter adaptation," *arXiv preprint arXiv:1802.10542*, 2018.

[70] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "Icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2017, pp. 2001–2010.

[71] K. Joseph, S. Khan, F. S. Khan, and V. N. Balasubramanian, "Towards open world object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5830–5840.

[72] X. Du, G. Gozum, Y. Ming, and Y. Li, "Siren: Shaping representations for detecting out-of-distribution objects," in *Advances in Neural Information Processing Systems*, 2022.

[73] X. Du, Z. Wang, M. Cai, and Y. Li, "Vos: Learning what you don't know by virtual outlier synthesis," *arXiv preprint arXiv:2202.01197*, 2022.

[74] J. Yang, K. Zhou, Y. Li, and Z. Liu, "Generalized out-of-distribution detection: A survey," *arXiv preprint arXiv:2110.11334*, 2021.

[75] C. Tian, L. Fei, W. Zheng, Y. Xu, W. Zuo, and C.-W. Lin, "Deep learning on image denoising: An overview," *Neural Networks*, vol. 131, pp. 251–275, 2020.

[76] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[77] K. H. Jin, M. T. McCann, E. Froustey, and M. Unser, "Deep convolutional neural network for inverse problems in imaging," *IEEE Transactions on Image Processing*, vol. 26, no. 9, pp. 4509–4522, 2017.

[78] P. Liu, H. Zhang, K. Zhang, L. Lin, and W. Zuo, "Multi-level wavelet-cnn for image restoration," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2018, pp. 773–782.

[79] R. A. Yeh, T. Y. Lim, C. Chen, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, "Image restoration with deep generative models," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2018, pp. 6772–6776.

[80] J. Chen, J. Chen, H. Chao, and M. Yang, "Image blind denoising with generative adversarial network based noise modeling," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3155–3164.

[81] S. Guo, Z. Yan, K. Zhang, W. Zuo, and L. Zhang, "Toward convolutional blind denoising of real photographs," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 1712–1722.

[82] K. Zhang, W. Zuo, S. Gu, and L. Zhang, "Learning deep cnn denoiser prior for image restoration," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 3929–3938.

[83] K. Zhang, W. Zuo, and L. Zhang, "Ffdnet: Toward a fast and flexible solution for cnn-based image denoising," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4608–4622, 2018.

[84] S. Soltanayev and S. Y. Chun, "Training deep learning based denoisers without ground truth data," *Advances in neural information processing systems*, vol. 31, 2018.

[85] J. Yang, X. Liu, X. Song, and K. Li, "Estimation of signal-dependent noise level function using multi-column convolutional neural network," in *2017 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2017, pp. 2418–2422.

[86] X. Li, M. Liu, Y. Ye, W. Zuo, L. Lin, and R. Yang, "Learning warped guidance for blind face restoration," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 272–289.

[87] K. Zhang, W. Zuo, and L. Zhang, "Learning a single convolutional super-resolution network for multiple degradations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 3262–3271.

[88] J. Yoo, Y. Uh, S. Chun, B. Kang, and J.-W. Ha, "Photorealistic style transfer via wavelet transforms," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019, pp. 9036–9045.

[89] Y. Jing, Y. Yang, Z. Feng, J. Ye, Y. Yu, and M. Song, "Neural style transfer: A review," *IEEE transactions on visualization and computer graphics*, vol. 26, no. 11, pp. 3365–3385, 2019.

Appendix A: Appendix

A.1 Abbreviations

Abbreviations of models in our work are available in Table A.2. And abbreviations of corruptions in our work are available in Table A.1.

Table A.1: Abbreviations of corruptions.

gaussian_noise	GsNoise	shot_noise	ShNoise	impulse_noise	ImNoise
defocus_blur	DfBlur	glass_blur	GlBlur	motion_blur	MtBlur
zoom_blur	ZmBlur	snow	Snow	fog	Fog
frost	Frost	brightness	Bright	contrast	Contrast
elastic_transform	Elastic	pixelate	Pix	jpeg_compression	JPEG

Table A.2: Abbreviation of models.

model name in mmdetection	abbr				
retinanet_x101_64x4d_fpn_1x_coco	retina_x64_1x	faster_rcnn_x101_64x4d_fpn_1x_coco	faster_x64_1x	retinanet_x101_32x4d_fpn_1x_coco	retina_x32_1x
retinanet_x101_64x4d_fpn_2x_coco	retina_x64_2x	fcos_center-normbbox-centeronreg-giou_r50_caffe_fpn_gn-head_dcn_1x_coco	fcos_r50c_1x_dc_giou	faster_rcnn_x101_32x4d_fpn_1x_coco	faster_x32_1x
retinanet_r101_fpn_1x_coco	retina_r101_1x	retinanet_x101_32x4d_fpn_2x_coco	retina_x32_2x	faster_rcnn_x101_64x4d_fpn_2x_coco	faster_x64_2x
retinanet_r101_caffe_fpn_1x_coco	retina_r101c_1x	fcos_x101_64x4d_fpn_gn-head_mstrain_640-800_2x_coco	fcos_x64_2x	fcos_r101_caffe_fpn_gn-head_1x_coco	fcos_r101c_1x
faster_rcnn_r101_fpn_1x_coco	faster_r101_1x	faster_rcnn_r101_caffe_fpn_1x_coco	faster_r101c_1x	retinanet_r101_fpn_2x_coco	retina_r101_2x
retinanet_x101_64x4d_fpn_mstrain_3x_coco	retina_x64_3x	faster_rcnn_x101_32x4d_fpn_2x_coco	faster_x32_2x	faster_rcnn_x101_64x4d_fpn_mstrain_3x_coco	faster_x64_3x
faster_rcnn_r101_fpn_2x_coco	faster_r101_2x	faster_rcnn_x101_32x4d_fpn_mstrain_3x_coco	faster_x32_3x	faster_rcnn_r101_fpn_mstrain_3x_coco	faster_r101_3x
faster_rcnn_r50_caffe_dc5_1x_coco	faster_r50c_1x_dc	retinanet_r101_fpn_mstrain_3x_coco	retina_r101_3x	fcos_r101_caffe_fpn_gn-head_mstrain_640-800_2x_coco	fcos_r101c_2x
retinanet_r50_caffe_fpn_1x_coco	retina_r50c_1x	faster_rcnn_r101_caffe_fpn_mstrain_3x_coco	faster_r101c_3x	faster_rcnn_r50_caffe_dc5_mstrain_1x_coco	faster_r50c_ms1x_dc
fcos_r50_caffe_fpn_gn-head_1x_coco	fcos_r50c_1x	retinanet_r50_fpn_fp16_1x_coco	retina_r50_1x_fp16	faster_rcnn_r50_caffe_fpn_1x_coco	faster_r50c_1x
retinanet_r50_fpn_1x_coco	retina_r50_1x	fcos_center-normbbox-centeronreg-giou_r50_caffe_fpn_gn-head_1x_coco	fcos_r50c_1x_giou	retinanet_r101_caffe_fpn_mstrain_3x_coco	retina_r101c_3x
detr_r50_8x2_150e_coco	detr_r50	faster_rcnn_r50_fpn_giou_1x_coco	faster_r50_1x_giou	faster_rcnn_r50_fpn_1x_coco	faster_r50_1x
faster_rcnn_r50_fpn_iou_1x_coco	faster_r50_1x_iou	faster_rcnn_r50_caffe_fpn_mstrain_2x_coco	faster_r50c_2x	faster_rcnn_r50_fpn_fp16_1x_coco	faster_r50_1x_fp16
faster_rcnn_r50_fpn_bounded_iou_1x_coco	faster_r50_1x_biou	retinanet_r50_fpn_2x_coco	retina_r50_2x	faster_rcnn_x101_32x8d_fpn_mstrain_3x_coco	faster_3x
fcos_r50_caffe_fpn_gn-head_mstrain_640-800_2x_coco	fcos_r50c_2x	faster_rcnn_r50_caffe_fpn_mstrain_3x_coco	faster_r50c_3x	faster_rcnn_r50_caffe_dc5_mstrain_3x_coco	faster_r50c_3x_dc
faster_rcnn_r50_fpn_2x_coco	faster_r50_2x	retinanet_r50_fpn_mstrain_3x_coco	retina_r50_3x	faster_rcnn_r50_fpn_mstrain_3x_coco	faster_r50_3x

A.2 More Results

A.2.1 Results on the clean and failure datasets

Relative performance improvement values of detectors across corruptions on the clean and failure test sets are available on Table A.4 and Table A.3.

A.2.2 Results on corruptions for all methods

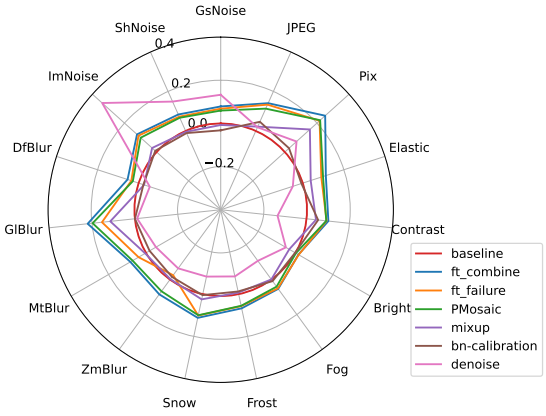


Plate A.1: Improvement on corruptions for all methods

Table A.3: All experimental results of repair methods for detectors under common corruptions. We present relative performance improvement for AP on the failure test sets.

model	method	GsNoise	ShNoise	ImNoise	DfBlur	GlBlur	MtBlur	ZmBlur	Snow	Frost	Fog	Bright	Contrast	Elastic	Pix	JPEG
retinanet_r50_fpn_1x_coco	finetune_failure	0.318	0.325	0.408	0.227	**1.801**	0.369	0.518	**0.470**	**0.283**	**0.170**	**0.064**	**0.333**	**0.449**	**1.056**	**0.526**
	bn-calibration	-0.012	-0.007	0.027	-0.057	0.052	-0.032	-0.061	0.045	0.006	0.048	0.032	0.175	-0.004	0.114	0.152
	finetune_combine	0.299	0.323	0.401	**0.235**	1.706	**0.426**	0.913	0.443	0.266	0.161	0.054	0.314	0.415	1.040	0.496
	mixup	0.108	0.130	0.193	0.020	1.200	0.229	0.463	0.264	0.115	0.114	-0.002	0.246	0.260	0.781	0.176
	PMosaic	0.285	0.309	0.366	0.190	1.701	0.426	**0.933**	0.456	0.280	0.157	0.041	0.312	0.385	0.916	0.433
	denoise	**0.584**	**0.617**	**1.179**	-0.057	0.177	-0.059	-0.183	-0.130	-0.112	-0.176	-0.025	-0.232	-0.044	0.418	0.163
	naiveDeepRepair	0.027	0.049	0.025	-0.027	-0.197	-0.074	-0.032	0.060	0.107	0.124	0.047	0.261	-0.031	-0.261	-0.036
retinanet_r101_fpn_1x_coco	finetune_failure	0.227	0.261	0.484	**0.223**	**1.442**	0.323	**0.905**	**0.411**	**0.253**	**0.142**	**0.055**	**0.275**	**0.364**	**0.745**	**0.454**
	bn-calibration	-0.062	-0.003	0.057	-0.033	0.017	-0.034	-0.050	0.039	0.012	0.045	0.017	0.151	0.007	0.056	0.145
	finetune_combine	0.219	0.249	0.469	0.222	1.421	**0.327**	0.809	0.393	0.244	0.139	0.044	0.262	0.355	0.712	0.448
	mixup	0.070	0.101	0.253	0.028	1.012	0.151	0.398	0.225	0.093	0.085	0.003	0.194	0.212	0.532	0.194
	PMosaic	0.204	0.237	0.436	0.183	1.360	0.314	0.828	0.399	0.247	0.131	0.037	0.264	0.329	0.623	0.383
	denoise	**0.420**	**0.453**	**1.163**	-0.063	0.162	-0.053	-0.159	-0.108	-0.114	-0.144	-0.027	-0.203	-0.027	0.279	0.160
	naiveDeepRepair	0.005	0.025	0.000	-0.012	-0.160	-0.041	-0.013	0.057	0.063	0.102	0.044	0.237	-0.028	-0.282	-0.032
faster_rcnn_r50_fpn_1x_coco	finetune_failure	0.347	0.393	0.543	0.293	1.832	0.359	0.826	0.514	**0.301**	**0.205**	**0.060**	**0.379**	**0.455**	**1.806**	**0.749**
	bn-calibration	-0.006	-0.010	0.060	-0.080	0.056	-0.030	-0.045	0.056	-0.002	0.049	0.030	0.197	-0.003	0.176	0.182
	finetune_combine	0.362	0.374	0.529	**0.301**	**1.838**	**0.494**	1.009	0.506	0.276	0.179	0.055	0.360	0.438	1.746	0.721
	mixup	0.173	0.180	0.299	0.137	1.379	0.306	0.682	0.336	0.152	0.146	0.013	0.332	0.328	1.423	0.341
	PMosaic	0.328	0.353	0.504	0.252	1.792	0.492	**1.048**	**0.517**	0.276	0.182	0.049	0.354	0.422	1.618	0.649
	denoise	**0.626**	**0.660**	**1.341**	-0.062	0.165	-0.044	-0.147	-0.127	-0.128	-0.169	-0.020	-0.234	-0.039	0.616	0.203
	naiveDeepRepair	0.042	0.018	0.013	-0.041	-0.154	-0.050	-0.030	0.069	0.062	0.131	0.052	0.315	-0.056	-0.198	-0.026
faster_rcnn_r101_fpn_1x_coco	finetune_failure	0.265	0.293	0.515	**0.258**	**1.435**	**0.377**	**1.090**	**0.439**	**0.277**	**0.160**	**0.060**	**0.327**	**0.397**	**1.004**	**0.618**
	bn-calibration	-0.064	-0.017	0.028	-0.031	-0.009	-0.044	-0.038	0.040	0.007	0.052	0.025	0.180	0.001	0.063	0.176
	finetune_combine	0.261	0.285	0.500	0.253	1.418	0.366	1.022	0.431	0.259	0.148	0.053	0.309	0.372	0.973	0.596
	mixup	0.114	0.132	0.313	0.129	1.093	0.227	0.639	0.291	0.145	0.105	0.002	0.271	0.263	0.781	0.328
	PMosaic	0.232	0.254	0.473	0.210	1.362	0.358	0.987	0.422	0.251	0.146	0.041	0.306	0.350	0.890	0.530
	denoise	**0.441**	**0.465**	**1.164**	-0.047	0.133	-0.063	-0.152	-0.115	-0.127	-0.143	-0.024	-0.194	-0.031	0.370	0.193
	naiveDeepRepair	-0.010	0.010	0.017	0.006	-0.202	-0.024	0.000	0.074	0.065	0.116	0.058	0.252	-0.044	-0.287	-0.011
fcos_r50_caffe_fpn_gn-head_1x_coco	finetune_failure	0.385	0.357	0.540	**0.319**	**1.866**	0.406	**0.947**	**0.441**	**0.247**	**0.188**	0.055	**0.319**	**0.391**	**0.948**	**0.487**
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	0.365	0.340	0.520	0.302	1.787	**0.408**	0.918	0.422	0.241	0.181	0.045	0.303	0.369	0.908	0.472
	mixup	0.182	0.159	0.287	0.108	1.256	0.214	0.422	0.246	0.117	0.120	-0.006	0.268	0.242	0.717	0.217
	PMosaic	0.328	0.314	0.480	0.241	1.715	0.404	0.874	0.425	0.225	0.148	0.021	0.302	0.349	0.816	0.394
	denoise	**0.712**	**0.668**	**1.439**	-0.060	0.230	-0.049	-0.179	-0.119	-0.116	-0.162	-0.033	-0.224	-0.047	0.281	0.159
	naiveDeepRepair	0.065	0.030	0.043	0.007	-0.213	-0.050	-0.034	0.059	0.070	0.141	**0.057**	0.318	-0.025	-0.174	0.023
fcos_r101_caffe_fpn_gn-head_1x_coco	finetune_failure	0.291	0.296	0.441	**0.221**	**1.562**	**0.336**	**1.031**	**0.393**	**0.272**	**0.139**	**0.072**	**0.287**	**0.386**	**0.663**	**0.444**
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	0.282	0.292	0.418	0.208	1.518	0.328	0.971	0.381	0.255	0.132	0.062	0.266	0.372	0.644	0.432
	mixup	0.106	0.126	0.217	0.046	1.130	0.182	0.493	0.223	0.139	0.083	0.001	0.217	0.242	0.489	0.198
	PMosaic	0.250	0.271	0.376	0.149	1.453	0.329	0.931	0.369	0.259	0.112	0.036	0.246	0.343	0.581	0.375
	denoise	**0.542**	**0.566**	**1.150**	-0.054	0.150	-0.056	-0.176	-0.099	-0.110	-0.143	-0.026	-0.195	-0.048	0.217	0.143
	naiveDeepRepair	0.027	0.047	0.017	0.006	-0.187	-0.036	0.015	0.051	0.078	0.094	0.057	0.245	-0.027	-0.132	-0.005
detr_r50_8x2_150e_coco	finetune_failure	0.253	0.261	0.448	0.097	1.866	0.280	0.592	0.314	0.180	**0.123**	0.037	0.258	**0.456**	**1.215**	0.325
	bn-calibration	0.025	0.010	0.122	-0.074	0.095	-0.076	-0.093	0.071	-0.002	0.043	**0.042**	0.202	0.007	0.237	0.197
	finetune_combine	0.271	0.284	0.462	**0.120**	**1.946**	0.282	**0.680**	**0.325**	**0.216**	0.123	0.033	0.263	0.446	1.199	**0.325**
	mixup	0.073	0.053	0.204	-0.099	1.275	0.049	0.151	0.112	0.027	0.016	-0.048	0.129	0.240	0.825	0.021
	PMosaic	0.254	0.296	0.429	0.099	1.926	**0.291**	0.652	0.318	0.209	0.107	0.008	0.262	0.431	1.070	0.265
	denoise	**0.760**	**0.776**	**1.610**	-0.045	0.244	-0.066	-0.180	-0.149	-0.120	-0.143	-0.022	-0.203	-0.026	0.702	0.196
	naiveDeepRepair	0.018	0.006	0.098	0.023	-0.203	-0.044	0.015	0.053	0.059	0.118	0.027	**0.302**	-0.020	-0.190	-0.021

Table A.4: All experimental results of repair methods for detectors under common corruptions. We present relative performance improvement for AP on the clean test sets.

model	method	GsNoise	ShNoise	ImNoise	DfBlur	GlBlur	MtBlur	ZmBlur	Snow	Frost	Fog	Bright	Contrast	Elastic	Pix	JPEG
retinanet_r50_fpn_1x_coco	finetune_failure	-0.036	-0.041	-0.036	-0.035	-0.125	-0.063	-0.086	-0.052	-0.048	-0.032	-0.015	-0.046	-0.091	-0.093	-0.054
	bn-calibration	-0.016	-0.015	-0.016	-0.010	-0.007	-0.013	-0.008	-0.016	-0.022	-0.019	-0.028	-0.026	-0.012	-0.010	-0.013
	finetune_combine	-0.020	-0.020	-0.019	-0.023	-0.040	-0.036	-0.067	-0.030	-0.028	-0.023	-0.008	-0.026	-0.043	-0.031	-0.025
	mixup	-0.054	-0.056	-0.060	-0.049	-0.077	-0.072	-0.087	-0.073	-0.060	-0.068	-0.043	-0.075	-0.074	-0.071	-0.055
	PMosaic	-0.036	-0.037	-0.040	-0.039	-0.059	-0.051	-0.081	-0.049	-0.046	-0.036	-0.018	-0.038	-0.047	-0.032	-0.035
	denoise	-0.104	-0.066	-0.060	-0.060	-0.046	-0.059	-0.053	-0.073	-0.075	-0.074	-0.078	-0.069	-0.060	-0.055	-0.059
	naiveDeepRepair	**-0.008**	**-0.005**	**-0.005**	**0.000**	**-0.005**	**-0.003**	**-0.005**	**-0.008**	**-0.008**	**-0.008**	**-0.006**	**-0.008**	**-0.005**	**-0.006**	**0.000**
retinanet_r101_fpn_1x_coco	finetune_failure	-0.023	-0.031	-0.030	-0.032	-0.101	-0.074	-0.146	-0.040	-0.031	-0.019	-0.008	-0.035	-0.066	-0.084	-0.033
	bn-calibration	-0.063	-0.015	-0.010	-0.008	-0.009	-0.007	-0.008	-0.020	-0.017	-0.022	-0.024	-0.028	-0.009	-0.009	-0.010
	finetune_combine	-0.016	-0.019	-0.016	-0.017	-0.033	-0.026	-0.050	-0.024	-0.022	-0.017	**-0.007**	-0.020	-0.035	-0.025	-0.020
	mixup	-0.047	-0.055	-0.047	-0.038	-0.069	-0.056	-0.071	-0.070	-0.050	-0.059	-0.045	-0.067	-0.071	-0.068	-0.052
	PMosaic	-0.030	-0.031	-0.031	-0.032	-0.050	-0.038	-0.067	-0.041	-0.039	-0.027	-0.016	-0.033	-0.044	-0.028	-0.029
	denoise	-0.064	-0.060	-0.044	-0.051	-0.042	-0.046	-0.042	-0.066	-0.057	-0.067	-0.063	-0.060	-0.048	-0.045	-0.050
	naiveDeepRepair	**-0.003**	**-0.008**	**-0.008**	**-0.005**	**-0.003**	**-0.005**	**-0.008**	**-0.003**	**-0.010**	**-0.008**	-0.010	**-0.008**	**-0.003**	**-0.008**	**-0.005**
faster_rcnn_r50_fpn_1x_coco	finetune_failure	-0.038	-0.047	-0.038	-0.050	-0.117	-0.081	-0.150	-0.061	-0.059	-0.049	-0.022	-0.057	-0.090	-0.112	-0.055
	bn-calibration	-0.013	-0.014	-0.013	-0.011	-0.006	-0.009	-0.006	-0.014	-0.022	-0.019	-0.031	-0.028	-0.013	-0.009	-0.010
	finetune_combine	-0.022	-0.022	-0.023	-0.028	-0.039	-0.042	-0.066	-0.040	-0.030	-0.027	-0.019	-0.029	-0.041	-0.036	-0.030
	mixup	-0.055	-0.060	-0.056	-0.066	-0.077	-0.070	-0.091	-0.082	-0.065	-0.079	-0.058	-0.081	-0.075	-0.066	-0.063
	PMosaic	-0.027	-0.035	-0.038	-0.038	-0.056	-0.058	-0.095	-0.052	-0.040	-0.031	-0.021	-0.034	-0.043	-0.033	-0.037
	denoise	-0.068	-0.073	-0.052	-0.059	-0.049	-0.056	-0.052	-0.069	-0.078	-0.077	-0.083	-0.066	-0.058	-0.050	-0.054
	naiveDeepRepair	**-0.008**	**-0.011**	**-0.003**	**-0.005**	**0.003**	**-0.005**	**-0.003**	**-0.005**	**-0.003**	**-0.013**	**-0.016**	**-0.008**	**-0.011**	**-0.003**	**0.000**
faster_rcnn_r101_fpn_1x_coco	finetune_failure	-0.031	-0.032	-0.030	-0.042	-0.107	-0.088	-0.168	-0.054	-0.043	-0.034	-0.021	-0.049	-0.076	-0.085	-0.042
	bn-calibration	-0.074	-0.014	**-0.009**	-0.010	-0.009	-0.008	-0.011	-0.019	-0.019	-0.020	-0.027	-0.032	-0.012	**-0.010**	-0.011
	finetune_combine	-0.018	-0.021	-0.020	-0.022	-0.034	-0.031	-0.057	-0.036	-0.030	-0.024	-0.014	-0.026	-0.039	-0.027	-0.024
	mixup	-0.056	-0.059	-0.057	-0.062	-0.069	-0.068	-0.080	-0.071	-0.075	-0.071	-0.062	-0.076	-0.068	-0.064	-0.063
	PMosaic	-0.023	-0.025	-0.030	-0.032	-0.044	-0.042	-0.075	-0.043	-0.037	-0.030	-0.018	-0.035	-0.037	-0.028	-0.024
	denoise	-0.069	-0.064	-0.049	-0.052	-0.044	-0.049	-0.044	-0.064	-0.063	-0.072	-0.072	-0.061	-0.055	-0.048	-0.049
	naiveDeepRepair	**-0.010**	**-0.005**	-0.010	**-0.003**	**0.000**	**-0.003**	**0.000**	**-0.005**	**-0.003**	**-0.008**	**-0.013**	**-0.010**	**-0.008**	-0.010	**-0.005**
fcos_r50_caffe_fpn_gn-head_1x_coco	finetune_failure	-0.040	-0.041	-0.048	-0.050	-0.118	-0.095	-0.138	-0.058	-0.052	-0.032	-0.022	-0.067	-0.096	-0.090	-0.033
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	-0.016	-0.017	-0.026	-0.019	-0.035	-0.035	-0.050	-0.033	-0.027	-0.014	-0.014	-0.041	-0.047	-0.030	-0.018
	mixup	-0.069	-0.067	-0.065	-0.058	-0.073	-0.069	-0.079	-0.071	-0.070	-0.076	-0.059	-0.089	-0.076	-0.068	-0.061
	PMosaic	-0.041	-0.046	-0.047	-0.039	-0.059	-0.044	-0.077	-0.055	-0.048	-0.043	-0.028	-0.049	-0.056	-0.037	-0.033
	denoise	-0.072	-0.069	-0.052	-0.061	-0.049	-0.058	-0.050	-0.067	-0.070	-0.079	-0.081	-0.068	-0.059	-0.056	-0.057
	naiveDeepRepair	**0.003**	**-0.003**	**0.000**	**0.008**	**0.003**	**0.000**	**0.003**	**0.000**	**-0.005**	**-0.008**	**0.000**	**-0.003**	**0.003**	**0.003**	**0.005**
fcos_r101_caffe_fpn_gn-head_1x_coco	finetune_failure	-0.032	-0.031	-0.042	-0.045	-0.102	-0.085	-0.141	-0.052	-0.044	-0.033	-0.009	-0.045	-0.072	-0.077	-0.036
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	-0.016	-0.014	-0.021	-0.015	-0.025	-0.021	-0.054	-0.027	-0.021	-0.020	-0.005	-0.021	-0.030	-0.020	-0.021
	mixup	-0.065	-0.064	-0.068	-0.062	-0.073	-0.067	-0.083	-0.075	-0.076	-0.075	-0.057	-0.080	-0.080	-0.074	-0.066
	PMosaic	-0.032	-0.031	-0.036	-0.036	-0.050	-0.040	-0.063	-0.044	-0.042	-0.038	-0.024	-0.040	-0.047	-0.033	-0.030
	denoise	-0.063	-0.055	-0.047	-0.046	-0.043	-0.047	-0.041	-0.053	-0.063	-0.063	-0.064	-0.057	-0.048	-0.045	-0.052
	naiveDeepRepair	**-0.003**	**-0.003**	**0.003**	**0.000**	**0.000**	**0.003**	**0.003**	**0.000**	**0.003**	**-0.003**	**-0.003**	**0.000**	**0.000**	**0.003**	**0.000**
detr_r50_8x2_150e_coco	finetune_failure	-0.057	-0.059	-0.050	-0.085	-0.186	-0.105	-0.412	-0.059	-0.060	-0.045	**-0.017**	-0.061	-0.100	-0.080	-0.035
	bn-calibration	**-0.012**	**-0.012**	**-0.013**	-0.012	**-0.006**	**-0.013**	**-0.009**	-0.017	-0.029	-0.025	-0.027	-0.039	**-0.010**	**-0.007**	**-0.009**
	finetune_combine	-0.030	-0.037	-0.026	-0.033	-0.064	-0.038	-0.066	-0.033	-0.040	-0.032	-0.018	-0.034	-0.050	-0.038	-0.022
	mixup	-0.104	-0.100	-0.099	-0.085	-0.119	-0.096	-0.103	-0.111	-0.091	-0.097	-0.074	-0.119	-0.105	-0.103	-0.095
	PMosaic	-0.061	-0.062	-0.057	-0.063	-0.083	-0.066	-0.084	-0.067	-0.067	-0.050	-0.039	-0.060	-0.072	-0.058	-0.044
	denoise	-0.063	-0.055	-0.045	-0.048	-0.040	-0.051	-0.044	-0.066	-0.059	-0.068	-0.069	-0.208	-0.053	-0.043	-0.052
	naiveDeepRepair	-0.022	-0.017	-0.018	**-0.010**	-0.007	-0.020	-0.015	**-0.015**	**-0.020**	**-0.025**	-0.025	**-0.022**	-0.012	-0.025	-0.012

Table A.5: All experimental results of repair methods for detectors under common corruptions. We present relative performance improvement for AP averaged on the clean and failure test sets.

model	method	GsNoise	ShNoise	ImNoise	DfBlur	GlBlur	MtBlur	ZmBlur	Snow	Frost	Fog	Bright	Contrast	Elastic	Pix	JPEG
retinanet_r50_fpn_1x_coco	finetune_failure	0.074	0.074	0.097	0.041	0.153	0.055	0.001	0.112	0.061	0.055	**0.021**	0.105	0.095	0.218	0.126
	bn-calibration	-0.015	-0.012	-0.003	-0.024	0.001	-0.018	-0.015	0.003	-0.013	0.010	-0.001	0.054	-0.009	0.024	0.039
	finetune_combine	0.079	0.087	0.107	**0.052**	**0.211**	**0.090**	**0.074**	**0.119**	**0.069**	**0.057**	0.020	**0.110**	**0.115**	**0.260**	**0.137**
	mixup	-0.003	0.002	0.016	-0.029	0.107	0.010	-0.007	0.033	-0.002	0.011	-0.024	0.053	0.041	0.160	0.017
	PMosaic	0.064	0.071	0.081	0.027	0.195	0.079	0.065	0.110	0.062	0.048	0.008	0.101	0.103	0.225	0.111
	denoise	**0.109**	**0.148**	**0.310**	-0.059	-0.013	-0.059	-0.071	-0.091	-0.087	-0.118	-0.054	-0.134	-0.054	0.074	0.010
	naiveDeepRepair	0.003	0.011	0.004	-0.008	-0.033	-0.022	-0.009	0.013	0.029	0.049	0.018	0.100	-0.014	-0.075	-0.011
retinanet_r101_fpn_1x_coco	finetune_failure	0.062	0.069	0.126	0.047	0.171	0.046	0.025	0.110	0.068	**0.052**	**0.021**	0.093	0.089	0.176	0.127
	bn-calibration	-0.063	-0.011	0.011	-0.016	-0.004	-0.015	-0.015	-0.000	-0.007	0.008	-0.005	0.046	-0.003	0.011	0.041
	finetune_combine	0.064	0.073	0.131	**0.057**	**0.223**	**0.082**	**0.089**	**0.114**	**0.071**	0.051	0.016	**0.096**	**0.106**	**0.206**	**0.134**
	mixup	-0.007	-0.002	0.044	-0.018	0.122	0.007	0.005	0.028	-0.000	0.004	-0.023	0.040	0.031	0.120	0.030
	PMosaic	0.050	0.061	0.111	0.034	0.199	0.069	0.078	0.106	0.061	0.043	0.008	0.089	0.091	0.177	0.107
	denoise	**0.101**	**0.116**	**0.323**	-0.055	-0.006	-0.048	-0.061	-0.080	-0.077	-0.101	-0.047	-0.119	-0.041	0.057	0.019
	naiveDeepRepair	0.000	0.003	-0.005	-0.007	-0.030	-0.016	-0.009	0.017	0.015	0.040	0.014	0.092	-0.012	-0.093	-0.014
faster_rcnn_r50_fpn_1x_coco	finetune_failure	0.080	0.089	0.129	0.047	0.167	0.038	-0.005	0.120	0.063	0.060	0.014	0.113	0.098	0.305	0.175
	bn-calibration	-0.011	-0.013	0.008	-0.030	0.003	-0.015	-0.012	0.008	-0.015	0.010	-0.003	0.060	-0.009	0.032	0.045
	finetune_combine	0.095	0.100	0.135	**0.065**	**0.234**	**0.103**	**0.094**	**0.131**	**0.073**	**0.061**	0.014	**0.123**	**0.124**	**0.352**	**0.185**
	mixup	0.015	0.014	0.045	-0.008	0.135	0.031	0.025	0.049	0.008	0.017	-0.026	0.081	0.064	0.258	0.053
	PMosaic	0.081	0.085	0.117	0.044	0.213	0.090	0.075	0.126	0.066	0.060	0.010	0.117	0.117	0.327	0.160
	denoise	**0.144**	**0.153**	**0.346**	-0.060	-0.018	-0.053	-0.066	-0.087	-0.095	-0.116	-0.055	-0.132	-0.051	0.095	0.019
	naiveDeepRepair	0.007	-0.002	0.002	-0.015	-0.021	-0.017	-0.007	0.018	0.019	0.049	**0.015**	0.118	-0.026	-0.046	-0.008
faster_rcnn_r101_fpn_1x_coco	finetune_failure	0.069	0.079	0.135	0.050	0.170	0.051	0.028	0.111	0.070	0.051	0.016	0.102	0.094	0.226	0.165
	bn-calibration	-0.071	-0.015	0.002	-0.017	-0.009	-0.018	-0.015	0.001	-0.009	0.011	-0.003	0.053	-0.007	0.011	0.048
	finetune_combine	0.077	0.083	0.138	**0.063**	**0.227**	**0.088**	**0.111**	**0.121**	**0.072**	**0.051**	0.016	**0.109**	**0.109**	**0.259**	**0.170**
	mixup	0.001	0.006	0.055	-0.003	0.139	0.021	0.032	0.050	0.003	0.006	-0.033	0.063	0.051	0.177	0.060
	PMosaic	0.064	0.070	0.122	0.043	0.208	0.078	0.090	0.113	0.065	0.047	0.009	0.102	0.102	0.235	0.149
	denoise	**0.104**	**0.116**	**0.318**	-0.051	-0.012	-0.053	-0.060	-0.081	-0.086	-0.103	-0.050	-0.115	-0.047	0.071	0.027
	naiveDeepRepair	-0.010	0.000	-0.002	0.000	-0.035	-0.009	0.000	0.022	0.021	0.046	**0.019**	0.096	-0.021	-0.089	-0.007
fcos_r50_caffe_fpn_gn-head_1x_coco	finetune_failure	0.086	0.082	0.115	0.053	0.164	0.043	0.013	0.100	0.049	0.062	0.013	0.084	0.075	0.200	0.132
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	0.097	0.093	0.126	**0.071**	**0.223**	**0.086**	**0.085**	**0.111**	**0.063**	**0.070**	0.013	0.094	**0.098**	**0.232**	**0.138**
	mixup	0.005	0.003	0.033	-0.012	0.116	0.009	-0.009	0.029	-0.007	0.008	-0.035	0.051	0.035	0.151	0.027
	PMosaic	0.069	0.066	0.100	0.040	0.193	0.079	0.056	0.097	0.045	0.039	-0.006	0.088	0.086	0.201	0.103
	denoise	**0.161**	**0.159**	**0.362**	-0.061	-0.009	-0.056	-0.068	-0.084	-0.086	-0.115	-0.059	-0.129	-0.055	0.038	0.012
	naiveDeepRepair	0.021	0.008	0.012	0.008	-0.028	-0.014	-0.002	0.019	0.020	0.056	**0.026**	**0.123**	-0.007	-0.047	0.011
fcos_r101_caffe_fpn_gn-head_1x_coco	finetune_failure	0.073	0.077	0.105	0.037	0.172	0.041	0.029	0.096	0.065	0.043	**0.027**	0.090	0.092	0.160	0.123
	bn-calibration	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	finetune_combine	0.080	0.087	0.113	**0.054**	**0.229**	**0.083**	**0.095**	**0.108**	**0.074**	**0.047**	0.026	0.096	**0.114**	**0.192**	**0.128**
	mixup	-0.010	-0.002	0.019	-0.028	0.125	0.008	0.000	0.024	-0.002	-0.005	-0.031	0.041	0.035	0.106	0.022
	PMosaic	0.059	0.068	0.090	0.021	0.197	0.070	0.082	0.093	0.061	0.028	0.003	0.076	0.093	0.163	0.104
	denoise	**0.133**	**0.149**	**0.318**	-0.048	-0.011	-0.050	-0.061	-0.069	-0.079	-0.098	-0.047	-0.113	-0.048	0.039	0.013
	naiveDeepRepair	0.007	0.014	0.007	0.002	-0.030	-0.009	0.004	0.017	0.028	0.040	0.024	**0.100**	-0.010	-0.040	-0.002
detr_r50_8x2_150e_coco	finetune_failure	0.034	0.036	0.082	-0.029	0.075	0.003	-0.268	0.058	0.020	0.027	**0.007**	0.064	0.087	0.230	0.082
	bn-calibration	-0.001	-0.005	0.023	-0.031	0.006	-0.031	-0.021	0.010	-0.020	0.004	0.004	0.056	-0.005	0.051	0.058
	finetune_combine	0.058	0.058	0.104	**0.014**	**0.192**	**0.052**	**0.042**	**0.079**	**0.046**	0.035	0.005	0.083	**0.116**	**0.258**	**0.091**
	mixup	-0.053	-0.054	-0.018	-0.089	0.058	-0.056	-0.066	-0.041	-0.052	-0.048	-0.062	-0.021	0.011	0.119	-0.057
	PMosaic	0.031	0.045	0.072	-0.013	0.173	0.034	0.022	0.053	0.026	0.018	-0.018	0.067	0.097	0.212	0.057
	denoise	**0.178**	**0.192**	**0.395**	-0.047	-0.004	-0.055	-0.063	-0.092	-0.079	-0.100	-0.047	-0.206	-0.044	0.135	0.029
	naiveDeepRepair	-0.011	-0.010	0.013	0.000	-0.033	-0.027	-0.011	0.007	0.007	**0.037**	-0.001	**0.104**	-0.015	-0.064	-0.015