A High-Resolution Dataset for Instance Detection with Multi-View Instance Capture

Qianqian Shen[1] **Yuhan Zhao**[2] **Nahyun Kwon**[3] **Jeeeun Kim**[3] **Yanan Li**[1] **Shu Kong**[3]
[1]Zhejiang Lab [2]UC-Irvine [3]Texas A&M University

Dataset and open-source code in webpage

Abstract

Instance detection (InsDet) is a long-lasting problem in robotics and computer vision, aiming to detect object instances (predefined by some visual examples) in a cluttered scene. Despite its practical significance, its advancement is overshadowed by Object Detection, which aims to detect objects belonging to some predefined classes. One major reason is that current InsDet datasets are too small in scale by today's standards. For example, the popular InsDet dataset GMU (published in 2016) has only 23 instances, far less than COCO (80 classes), a well-known object detection dataset published in 2014. We are motivated to introduce a new InsDet dataset and protocol. First, we define a realistic setup for InsDet: training data consists of multi-view instance captures, along with diverse scene images allowing synthesizing training images by pasting instance images on them with free box annotations. Second, we release a real-world database, which contains multi-view capture of 100 object instances, and high-resolution (6k$\times$8k) testing images. Third, we extensively study baseline methods for InsDet on our dataset, analyze their performance and suggest future work. Somewhat surprisingly, using the off-the-shelf class-agnostic segmentation model (Segment Anything Model, SAM) and the self-supervised feature representation DINOv2 performs the best, achieving $>$10 AP better than end-to-end trained InsDet models that repurpose object detectors (e.g., FasterRCNN and RetinaNet).

1 Introduction

Instance detection (InsDet) requires detecting specific object instances (defined by some visual examples) from a scene image [12]. It is practically important in robotics, e.g., elderly-assistant robots need to fetch specific items (*my*-cup vs. *your*-cup) from a cluttered kitchen [41], micro-fulfillment robots for the retail need to pick items from mixed boxes or shelves [4].

Motivation. InsDet receives much less attention than the related problem of Object Detection (ObjDet), which aims to detect all objects belonging to some predefined classes [29, 38, 30, 49]. Fig. 1 compares the two problems. *One major reason is that there are not large-enough InsDet datasets by today's standards.* For example, the popular InsDet dataset GMU (published in 2016) [15] has only 23 object instances while the popular ObjDet dataset COCO has 80 object classes (published in 2014) [29]. Moreover, *there are no unified protocols in the literature of InsDet.* The current InsDet literature mixes multiple datasets to simulate training images and testing scenarios [12]. Note that the training protocol of InsDet does not follow that of ObjDet, which has training images annotated with

Submitted to the 37th Conference on Neural Information Processing Systems (NeurIPS 2023) Track on Datasets and Benchmarks. Do not distribute.

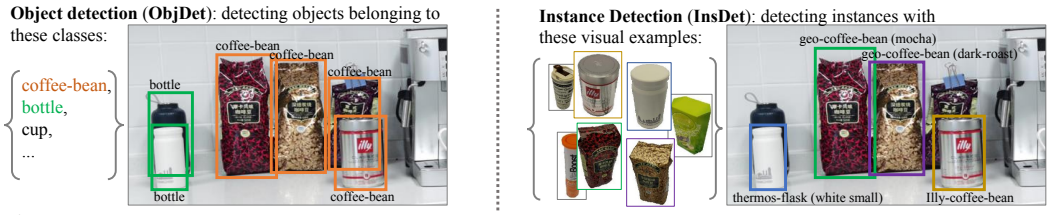

Figure 1: **Object detection (ObjDet) vs. instance detection (InsDet)**. ObjDet aims to detect all objects belonging to some predefined classes, whereas InsDet requires detecting specific object instances defined by some visual examples. Loosely speaking, InsDet treats a single object instance as a class compared to ObjDet. Please refer to Fig. 2-right for the challenge of InsDet, which is the focus of our work.

bounding boxes. Differently, for InsDet,[1] its setup should have profile images of instances (cf. right in Fig. 1) and optionally diverse background images not containing such instances [12]. We release a new dataset and present a unified protocol to foster the InsDet research.

Overview of our dataset is presented in Fig. 2. In our dataset, profile images (3072x3072) of object instances and testing images (6144x8192) are high-resolution captured by a Leica camera (commonly used in today's cellphones). This inexpensive camera is deployable in current or future robot devices. Hence, our dataset simulates real-world scenarios, e.g., robotic navigation in indoor scenes. Even with high-resolution images, objects in testing images appear small, taking only a tiny region in the high-res images. This demonstrates a clear challenge of InsDet in our dataset. Therefore, our dataset allows studying InsDet methods towards real-time operation on high-res (as future work).

Preview of technical insights. On our dataset, we revisit existing InsDet methods [27, 12, 17]. Perhaps the only InsDet framework is cut-paste-learn [12], which cuts instances from their profile images, pastes them on random background images (so being able to derive "free" bounding boxes annotations), and trains InsDet detectors on such data by following that of ObjDet (e.g., Faster-RCNN [38]). We study this framework, train different detectors, and confirm that the state-of-the-art transformer-based detector DINO [49] performs the best, achieving 27.99 AP, significantly better than CNN-based detector FasterRCNN (19.52 AP). Further, we present a non-learned method that runs off-the-shelf proposal detectors (SAM [24] in our work) to generate object proposals and use self-supervised learned features (DINO$_f$ [8][2] and DINOv2$_f$ [34]) to find matched proposals to instances' profile images. Perhaps surprisingly, this non-learned method resoundingly outperforms end-to-end learning methods, i.e., SAM+DINOv2$_f$ achieves 41.61 AP, much better than DINO (27.99 AP) [49].

Contributions. We make three major contributions.

1. We formulate the InsDet problem with a unified protocol and release a challenging dataset consisting of both high-resolution profile images and high-res testing images.
2. We conduct extensive experiments on our dataset and benchmark representative methods following the cut-paste-learn framework [12], showing that stronger detectors perform better.
3. We present a non-learned method that uses an off-the-shelf proposal detector (i.e., SAM [24]) to produce proposals, and self-supervised learned features (e.g., DINOv2$_f$ [34]) to find instances (which are well matched to their profile images). This simple method significantly outperforms the end-to-end InsDet models.

2 Related Work

Instance Detection (InsDet) is a long-lasting problem in computer vision and robotics [50, 12, 33, 3, 16, 22, 4], referring to detecting specific object instances in a scene image. Traditional InsDet methods use keypoint matching [35] or template matching [20]; more recent ones train deep neural networks

[1] In real-world applications (e.g., robot learning), it is infeasible to place objects in diverse scenes, take scene photos, then annotate instances using boxes towards training images (cf. training data in object detection).

[2] We add subscript $_f$ to indicate that DINO$_f$ [8] is the self-supervised learned feature extractor; distinguishing it from a well-known object detector DINO [49].

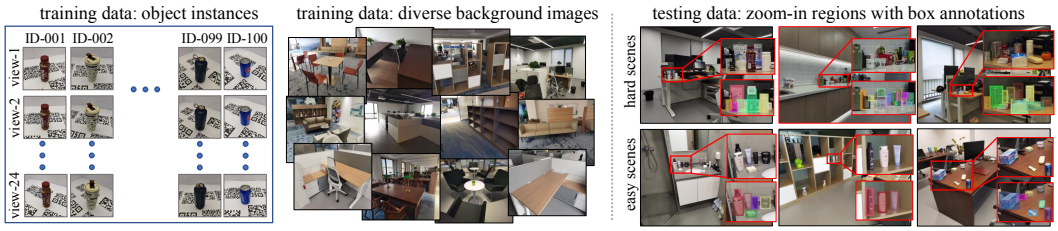

Figure 2: **Overview of our instance detection dataset**. **Left**: It contains 100 distinct object instances. For each of them, we capture 24 profile photos from multiple views. We paste QR code images beneath objects to allow relative camera estimation (e.g., by COLMAP [42]), just like other existing datasets [21, 5]. **Middle**: We take photos in random scenes (which do not contain any of the 100 instances) as background images. The background images can be optionally used to synthesize training data, e.g., pasting the foreground instances on them towards box-annotated training images [27, 12, 17] as used in the object detection literature [29]. **Right**: high-resolution (6k×8k) testing images of clutter scenes contain diverse instances, including some of the 100 predefined instances and other uninterested ones. The goal of InsDet is to detect the predefined instances in these testing images. From the zoom-in regions, we see the scene clutters make InsDet a rather challenging problem.

to approach InsDet [33]. Some others focus on obtaining more training samples by rendering realistic instance examples [23, 22], data augmentation [12], and synthesizing training images by cutting instances as foregrounds and pasting them to background images [27, 12, 17]. Speaking of InsDet datasets, [15] collects scene images from 9 kitchen scenes with RGB-D cameras and defines 23 instances of interest to annotate with 2D boxes on scene images; [22] creates 3D models of 29 instances from 6 indoor scenes, and uses them to synthesize training and testing data; [4] creates 3D mesh models of 100 grocery store objects, renders 80 views of images for each instance, and uses them to synthesize training data.

As for benchmarking protocol of InsDet, [12] synthesizes training data from BigBird [44] and UW Scenes [26] and tests on the GMU dataset [15]; [22] trains on their in-house data and test on LM-O [5] and Rutgers APC [39] datasets. Moreover, some works require hardware-demanding setups [4], some synthesize both training and testing data [22, 27], while others mix existing datasets for benchmarking [12]. Given that the modern literature on InsDet lacks a unified benchmarking protocol (till now!), we introduce a more realistic unified protocol along with our InsDet dataset, allowing fairly benchmarking methods and fostering research of InsDet.

Object Detection (ObjDet) is a fundamental computer vision problem [13, 29, 38], requiring detecting all objects belonging to some predefined categories. The prevalent ObjDet detectors adopt convolutional neural networks (CNNs) as a backbone and a detector-head for proposal detection and classification, typically using bounding box regression and a softmax-classifier. Approaches can be grouped into two categories: one-stage detectors [37, 31, 36, 47] and two-stage detectors [18, 6]. One-stage detectors predict candidate detection proposals using bounding boxes and labels at regular spatial positions over feature maps; two-stage detectors first produce detection proposals, then perform classification and bounding box regression for each proposal. Recently, the transformer-based detectors transcend CNN-based detectors [7, 52, 49], yielding much better performance on various ObjDet benchmarks. Different from ObjDet, InsDet requires distinguishing individual object instances within a class. Nevertheless, to approach InsDet, the common practice is to repurpose ObjDet detectors by treating unique instances as individual classes. We follow this practice and benchmark various ObjDet methods on our InsDet dataset.

Pretrained Models. Pretraining is an effective way to learn features from diverse data. For example, training on the large-scale ImageNet dataset for image classification [10], a neural network can serve as a powerful feature extractor for various vision tasks [11, 43]. Object detectors trained on the COCO dataset [29] can serve as a backbone allowing finetuning on a target domain to improve detection performance [28]. Such pretraining requires human annotations which can be costly. Therefore, self-supervised pretraining has attracted increasing attention and achieved remarkable progress [9, 19, 8, 34]. Moreover, the recent literature shows that pretraining on much larger-scale data can serve as a foundation model for being able to perform well across domains and tasks. For example, the Segment Anything Model (SAM) pretrains a class-agnostic proposal detector on

web-scale data and shows an impressive ability to detect and segment diverse objects in the wild [24]. In this work, with our high-res InsDet dataset, we explore a non-learned method by using publicly available pretrained models. We show that such a simple method significantly outperforms end-to-end learned InsDet detectors.

3 Instance Detection: Protocol and Dataset

In this section, we formulate a realistic unified InsDet protocol and introduce the new dataset. We release our dataset under the MIT License, hoping to contribute to the broader research community.

3.1 The Protocol

Our InsDet protocol is motivated by real-world indoor robotic applications. In particular, we consider the scenario that assistive robots must locate and recognize instances to fetch them in a cluttered indoor scene [41], where InsDet is a crucial component. Realistically, for a given object instance, the robots should see it only from a few views (*at the training stage*), and then accurately detect it *in a distance* in *any* scenes (*at the testing stage*). Therefore, we suggest the protocol specifying the training and testing setups below. We refer the readers to Fig. 2 for an illustration of this protocol.

- **Training**. There are profile images of each instance captured at different views and diverse background images. The background images can be used to synthesize training images with free 2D-box annotations, as done by the cut-paste-learn methods [27, 12, 17].
- **Testing**. InsDet algorithms are required to precisely detect all predefined instances from real-world images of cluttered scenes.

Evaluation metrics. The InsDet literature commonly uses average precision (AP) at IoU=0.5 [12, 2, 33]; others use different metrics, e.g., AP at IoU=0.75 [22], mean AP [3, 16], and F1 score [4]. As a single metric appears to be insufficient to benchmark methods, we follow the literature of ObjDet that uses multiple metrics altogether [29].

- **AP** averages the precision at IoU thresholds from 0.5 to 0.95 with the step size 0.05. It is the *primary metric* in the most well-known COCO Object Detection dataset [29].
- $\mathbf{AP}_{50}$ and $\mathbf{AP}_{75}$ are the precision averaged over all instances with IoU threshold as 0.5 and 0.75, respectively. In particular, $\mathbf{AP}_{50}$ is the widely used metric in the literature of InsDet.
- **AR** (average recall) averages the proposal recall at IoU threshold from 0.5 to 1.0 with the step size 0.05, regardless of the classification accuracy. AR measures the localization performance (excluding classification accuracy) of an InsDet model.

Moreover, we tag *hard* and *easy* scenes in the testing images based on the level of clutter and occlusion, as shown by the right panel of Fig. 2. Following the COCO dataset [29], we further tag testing object instances as *small*, *medium*, and *large* according to their bounding box area (cf. details in the supplement). These tags allow a breakdown analysis to better analyze methods.

3.2 The Dataset

We introduce a challenging real-world dataset of indoor scenes (motivated by indoor assistive robots), including high-resolution photos of 100 distinct object instances, and high-resolution testing images captured from 14 indoor scenes where there are such 100 instances defined for InsDet. Table 1 summarizes the statistics compared with existing datasets, showing that our dataset is larger in scale and more challenging than existing InsDet datasets. Importantly, object instances are located far from the camera in cluttered scenes; this is realistic because robots must detect

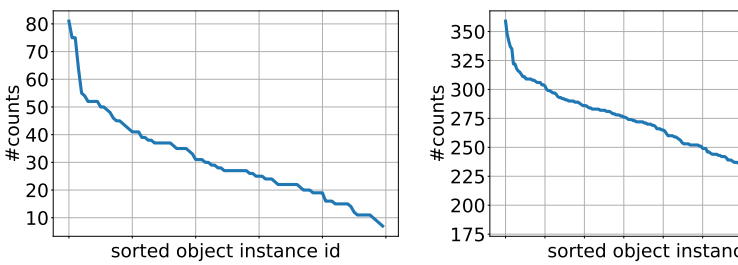

Figure 3: Imbalanced distribution of instances in test-set. Yet, instances have the same number of profile images in training and the metrics average over all instances. So, the evaluation is unbiased.

Table 1: **Comparison of our dataset to existing ones**. Several datasets are used in the InsDet literature although they are designed for different tasks. For example, BigBird and LM are designed to study algorithms of object recognition and object pose estimation, hence they contain instances that are close to the camera. Naively repurposing them for InsDet leads to saturated performance, impoverishing the exploration space of InsDet. Instead, ours is more challenging as instances are placed far from the camera, simulating realistic scenarios where robots must detect instances at a distance. Importantly, our dataset contains far more instances than other publicly available InsDet datasets.

	for what task	publicly available	#instances	#scenes	published year	resolution
BigBird [44]	recognition	✓	100	N/A	2014	1280x1024
RGBD [27]	scene label.	✓	300	14	2017	N/A
LM [21]	6D pose est.	✓	15	1	2012	480x640
LM-O [5]	6D pose est.	✓	20	1	2017	480x640
RU-APC [39]	3D pose est.	✓	14	1	2016	480x640
GMU [15]	InsDet	✓	23	9	2016	1080x1920
AVD [1]	InsDet	✓	33	9	2017	1080x1920
Grocery [4]	InsDet	✗	100	10	2021	unknown
Ours	InsDet	✓	100	14	2023	6144x8192

objects in a distance before approaching them [1]. Perhaps surprisingly, only a few InsDet datasets exist in the literature. Among them, Grocery [4], which is the latest and has the most instances like our dataset, is not publicly available.

Our InsDet dataset contains 100 object instances. When capturing photos for each instance, inspired by prior arts [44, 21, 5], we paste a QR code on the tabletop, which enables pose estimation, e.g., using COLMAP [42]. Yet, we note more realistic scenarios can be hand-holding instances for capturing [25], which we think of as future work. Each instance photo is of 3072×3072 pixel resolution. For each instance, we capture 24 photos from multiple views. The left panel of Fig. 2 shows some random photos for some instances. For the testing set, we capture high-resolution images (6144×8192) in cluttered scenes, where some instances are placed in reasonable locations, as shown in the right panel of Fig. 2. We tag these images as *easy* or *hard* based on scene clutter and object occlusion levels. When objects are placed sparsely, we tag the testing images as *easy*; otherwise, we tag them as *hard*. Our InsDet dataset also contains 200 high-res background images of indoor scenes (cf. Fig. 2-middle). These indoor scenes are not included in testing images. They allow using the cut-paste-learn framework to synthesize training images [27, 12, 17]. Following this framework, we segment foreground instances using GrabCut [40] to paste them on background images. It is worth noting that the recent vision foundation model SAM [24] makes interactive segmentation much more efficient. Yet, this work is made public after we collected our dataset. In Fig. 3, we plot the per-instance frequency in the testing set.

4 Methodology

4.1 The Strong Baseline: Cut-Paste-Learn

Cut-Paste-Learn serves as a strong baseline that synthesizes training images with 2D-box annotations [12]. This allows one to train InsDet detectors in the same way as training normal ObjDet detectors, by simply treating the K unique instances as K distinct classes. It cuts and pastes foreground instances at various aspect ratios and scales on diverse background images, yielding synthetic training images, as shown in Fig. 4. Cut-paste-learn is model-agnostic, allowing one to adopt any state-of-the-art detector architecture. In this work, we study five popular detectors, covering the two-stage detector FasterRCNN [38], and one-stage anchor-based detector RetinaNet [30], and one-stage anchor-free detectors CenterNet [50], and FCOS [46]; and the transformer-based detector DINO [49]. There are multiple factors in the cut-paste-learn framework, such as the number of inserted objects in each background image, their relative size, the number of generated training images and blending methods. We conduct comprehensive ablation studies and report results using the best-tuned choices. We refer interested readers to the supplement for the ablation studies.

(a) box (b) Gaussian blurring (c) Motion (d) naive pasting

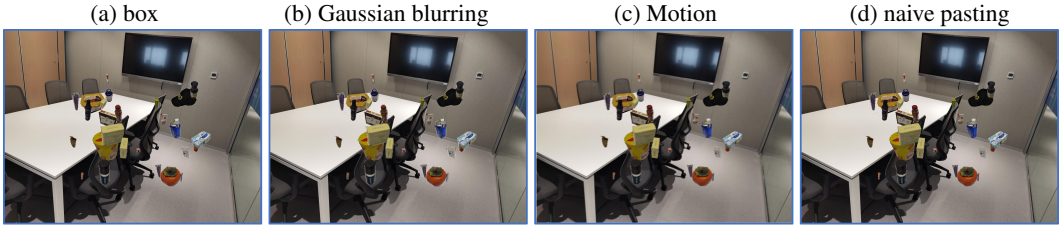

Figure 4: Synthetic training images for cut-paste-learn methods. We use different blending methods to paste object instances on the same background. We recommend that interested readers refer to the supplement for an ablation study using different blending methods.

4.2 The Simple, Non-Learned Method

We introduce a simple, non-learned InsDet method by exploiting publicly available pretrained models. This method consists of three main steps: (1) proposal generation on testing images, (2) matching proposals and profile images, (3) selecting the best-matched proposals as the detected instances.

Proposal generation. We use the recently released Segment Anything Model (SAM) [24] to generate proposals. For a proposal, we define a minimum bounding square box encapsulating the masked instance, and then crop the region from the high-resolution testing image. SAM not only achieves high recall (Table 3) on our InsDet dataset but detects objects not belonging to the instances of interest. So the next step is to find interested instances from the proposals.

Feature representation of proposals and profile images. Intuitively, among the pool of proposals, we are interested in those that are well-matched to any profile images of any instance. The well-matched ones are more likely to be predefined instances. To match proposals and profile images, we use off-the-shelf features to represent them. In this work, we study two self-supervised learned models as feature extractors, i.e. DINO$_f$ [8], and DINOv2$_f$ [34]. We feed a square crop (of a proposal) or a profile image to the feature extractor to obtain its feature representation. We use cosine similarity over the features as the similarity measure between a proposal and a profile image.

Proposal matching and selection. As each instance has multiple profile images, we need to design the similarity between a proposal and an instance. For a proposal, we compute the cosine similarities of its feature to all the profile images of an instance and use the maximum as its final similarity to this instance. We then filter out proposals and instances if they have similarities lower than a threshold, indicating that they are not matched to any instances or proposals. Finally, we obtain a similarity matrix between all remaining proposals and all remaining instances. Over this matrix, we study two matching algorithms to find the best match (hence the final InsDet results), i.e. Rank & Select, and Stable Matching [14, 32]. The former is a greedy algorithm that iteratively selects the best match (highest cosine similarity) between a proposal and an instance and removes the corresponding proposal until no proposal/instance is left. The latter produces an optimal list of matched proposals and instances, such that there exist no pair of instances and proposals which both prefer each other to their current correspondence under the matching.

5 Experiments

Synthesizing training images for cut-paste-learn baselines. Our baseline method trains state-of-the-art ObjDet detectors on data synthesized using the cut-paste-learn strategy [12]. For evaluating on our InsDet dataset, we generate 19k training examples and 6k validation examples. For each example, various numbers of foreground objects ranging from 25 to 35 are pasted to a randomly selected background image. The objects are randomly resized with a scale from 0.15 to 0.5. We use four blending options [12], including Gaussian blurring, motion blurring, box blurring, and naive pasting. Fig. 4 shows some random synthetic images. The above factors have a notable impact on the final performance of trained models, and we have conducted a comprehensive ablation study. We refer interested readers to the supplement for the study.

Table 2: **Benchmarking results on our dataset**. We summarize three salient conclusions. (1) End-to-end trained detectors perform better with stronger detector architectures, e.g., the transformer DINO (27.99 AP) outperforms FasterRCNN (19.54 AP). (2) Interestingly, the non-learned method SAM+DINOv2$_f$ performs the best (41.61 AP), significantly better than end-to-end learned detectors including DINO (27.99 AP). (3) All methods have much lower AP on `hard` testing images or `small` objects (e.g., SAM+DINOv2$_f$ yields 28.03 AP on `hard` vs. 47.57 AP on `easy`), showing that future work should focus on `hard` situations or `small` instances.

	AP						**AP**$_{50}$	**AP**$_{75}$
	`avg`	`hard`	`easy`	`small`	`medium`	`large`		
FasterRCNN [38]	19.54	10.26	23.75	5.03	22.20	37.97	29.21	23.26
RetinaNet [30]	22.22	14.92	26.49	5.48	25.80	42.71	31.19	24.98
CenterNet [50]	21.12	11.85	25.70	5.90	24.15	40.38	32.72	23.60
FCOS [46]	22.40	13.22	28.68	6.17	26.46	38.13	32.80	25.47
DINO [49]	27.99	17.89	32.65	11.51	31.60	48.35	39.62	32.19
SAM + DINO$_f$	36.97	22.38	43.88	11.93	40.85	62.67	44.13	40.42
SAM + DINOv2$_f$	**41.61**	**28.03**	**47.57**	**14.58**	**45.83**	**69.14**	**49.10**	**45.95**

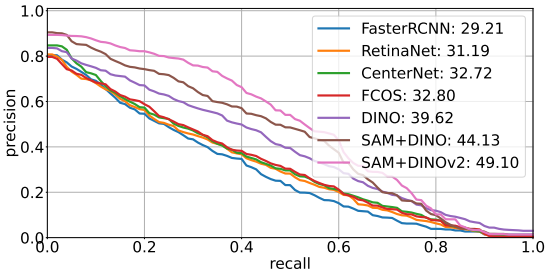

Figure 5: Precision-recall curves with IoU=0.5 (AP50 in the legend) on our InsDet dataset. Stronger detectors perform better, e.g., DINO, a transformer-based detector significantly outperforms FasterRCNN. Furthermore, even with a simple non-learned method, leveraging pretrained models, e.g., SAM+DINOv2$_f$, outperforms end-to-end learned methods.

Implementation details. We conduct all the experiments based on open-source implementations, such as Detectron2 [48] (for FasterRCNN and RetinaNet), CenterNet [51], FCOS [45] and DINO [49]. The CNN-based end-to-end detectors are initialized with pretrained weights on COCO [29]. We fine-tune CNN-based models using SGD and the transformer-based model using AdamW with a learning rate of 1e-3 and a batch size of 16. We fine-tune all the models for 5 epochs (which are enough for training to converge) and evaluate checkpoints after each epoch for model selection. The models are trained on a single Tesla V100 GPU with 32G memory.

If applied, we preprocess object instance profile images and proposals. Specifically, for a profile image, we remove the background pixels (e.g., pixels of QR code) using foreground segmentation (i.e., GrabCut). For each proposal, we crop its minimum bounding square box. We also study whether removing background pixels by using SAM's mask output performs better. We use DINO$_f$ and DINOv2$_f$ to compute feature representations.

5.1 Benchmarking Results

Quantitative results. To evaluate the proposed InsDet protocol and dataset, we first train detectors from a COCO-pretrained backbone following the cut-past-learn baseline. Table 2 lists detailed comparisons and Fig. 5 plots the precision-recall curves for the compared methods. We can see that detectors with stronger architectures perform better, e.g. DINO (27.99% AP) vs. FasterRCNN (19.54% AP). Second, non-learned methods outperform end-to-end trained models, e.g., SAM+DINOv2$_f$ (41.61% AP) vs. DINO (27.99% AP). Third, all the methods perform poorly on *hard* and *small* instances, suggesting future work focusing on such cases.

Table 3 compares methods w.r.t the average recall (AR) metric. "AR@max10" means AR within the top-10 ranked detections. In computing AR, we rank detections by using the detection confidence scores of the learning-based methods (e.g., FasterRCNN) or similarity scores in the non-learned methods (e.g., SAM+DINO$_f$). AR$_s$, AR$_m$, and AR$_l$ are breakdowns of AR for small, medium, and large testing object instances. Results show that (1) the non-learned methods that use SAM generally recall more instances than others, and (2) all methods suffer from small instances. In sum, results show that methods yielding higher recall achieve higher AP metrics (cf. Table 2).

Table 3: **Benchmarking results w.r.t average recall (AR)**. "AR@max10" means AR within the top-10 ranked detections. In computing AR, we rank detections by using the detection confidence scores of the learning-based methods (e.g., FasterRCNN) or similarity scores in the non-learned methods (e.g., SAM+DINO$_f$). AR$_s$, AR$_m$, and AR$_l$ are breakdowns of AR for small, medium and large testing object instances. Results show that (1) the non-learned methods that use SAM generally recall more instances than others, and (2) all methods suffer from small instances. In sum, results show that methods yielding higher recall achieve higher AP metrics (cf. Table 2).

	AR@max10	**AR@max100**	**AR$_s$@max100**	**AR$_m$@max100**	**AR$_l$@max100**
FasterRCNN [38]	26.24	39.24	14.83	44.87	60.05
RetinaNet [30]	26.33	49.38	22.04	56.76	69.69
CenterNet [50]	23.55	44.72	17.84	52.03	64.58
FCOS [46]	25.82	46.28	22.09	52.85	64.11
DINO [49]	29.84	54.22	**32.00**	59.43	72.92
SAM + DINO$_f$	31.25	63.05	31.65	70.01	**90.63**
SAM + DINOv2$_f$	**40.02**	**63.06**	31.11	**70.40**	90.36

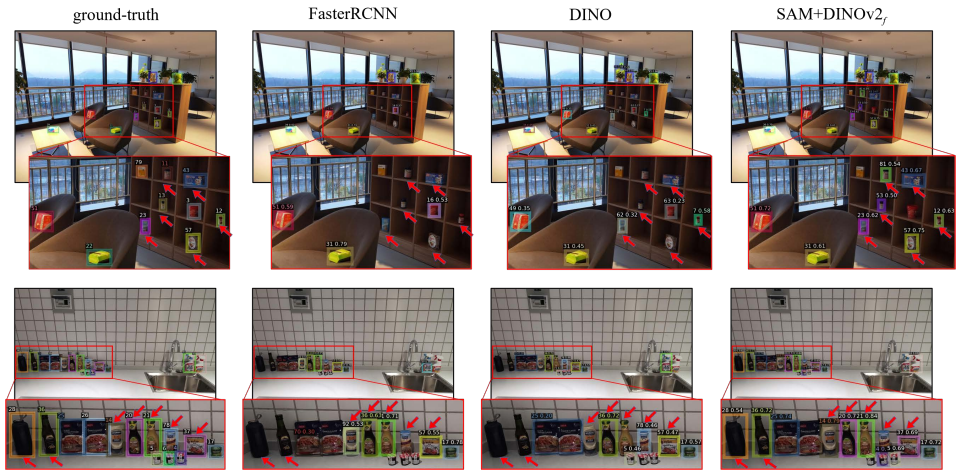

Figure 6: Visual results of FasterRCNN, DINO, and SAM+DINOv2$_f$ on our InsDet dataset. The top row illustrates the sparse placement of instances (i.e., `easy` scenario), while the bottom contains more cluttered instances (i.e., `hard` scenario). We drop predicted instance names for brevity. SAM helps localize instances with more precise bounding boxes, e.g., as arrows labeled in the upper row. DINOv2$_f$ provides more precise recognition of localized instances, e.g., five instances in the right of the bottom row. Compared with DINO, SAM+DINOv2$_f$ is better at locating occluded instances.

Qualitative results. Fig. 6 visualizes qualitative results on two testing examples from the InsDet dataset. Stronger detectors, e.g., the non-learned method SAM+DINOv2$_f$, produce fewer false negatives. Even so, all detectors still struggle to detect instances with presented barriers such as heavy occlusion, instance size being too small, etc. As shown in Fig. 5, the non-learned method SAM+DINOv2$_f$ outperforms end-to-end learned methods in a wide range of recall thresholds.

5.2 Ablation Study

Due to the space limit, we ablate the instance crop and stable matching in the main paper and put more (including ablation studies for the cut-paste-learn methods) in the supplement.

Proposal feature extraction in the non-learned method. Given a box crop (encapsulating the proposal) generated by SAM in the non-learned method, we study how to process the crop to improve InsDet performance. Here, we can either crop and feed its minimum bounding box to compute DINOv2$_f$ features, or we can use the mask to remove the background in the box. Table 4 shows the comparison. Clearly, the latter performs remarkably better in both "hard" and "easy" scenarios.

Proposal-instance match in the non-learned method. After generating proposals by SAM, we need to compare them with instance profile images to get the final detection results. We study the InsDet performance of the two matching algorithms. Rank & Select is a greedy algorithm that iteratively finds the best match between any proposals and instances until no instances/proposals

Table 4: **Ablation study: whether to remove background in crops for feature computation**. Based on a proposal given by SAM, we can crop and feed its minimum bounding square to compute DINOv2$_f$ feature, or we can use the mask to remove the background in the square before computing the feature. Clearly, the latter performs remarkably better.

strategy	AP			AP_{50}			AP_{75}		
	avg	hard	easy	avg	hard	easy	avg	hard	easy
w/o background removal	36.04	23.04	42.37	43.84	29.12	51.00	39.59	25.74	46.13
w/ background removal	39.12	24.00	47.17	46.72	30.81	54.66	42.86	26.40	51.58

Table 5: **Ablation study: whether to generate unique proposal-instance match**. In contrast to Rank&Select, Stable Matching produces a unique match to proposal/instance for each instance/proposal, yielding better performance than Rank&Select.

strategy	AP			AP_{50}			AP_{75}		
	avg	hard	easy	avg	hard	easy	avg	hard	easy
Rank & Select	38.62	23.95	46.31	46.04	30.77	53.64	42.37	26.39	50.61
Stable Matching	39.12	24.00	47.17	46.72	30.81	54.66	42.86	26.40	51.58

are left unmatched; stable matching produces an optimal list of matched proposals and instances such that there does not exist a pair in which both prefer other proposals/instances to their current correspondence under the matching. Table 5 compares these two methods, clearly showing that stable matching works better.

5.3 Discussions

Societal Impact. InsDet is a crucial component in various robotic applications such as elderly-assistive agents. Hence, releasing a unified benchmarking protocol contributes to broader communities. While our dataset enables InsDet research to move forward, similar to other works, directly applying algorithms brought by our dataset is risky in real-world applications.

Limitations. We note several limitations in our current work. First, while our work uses normal cameras to collect datasets, we expect to use better and cheaper hardware (e.g., depth camera and IMU) for data collection. Second, while the cut-paste-learn method we adopt does not consider geometric cues when synthesizing training images, we hope to incorporate such information to generate better and more realistic training images, e.g., pasting instances only on up-surfaces like tables, desks, and floors. Third, while SAM+DINOv2$_f$ performs the best, this method is time-consuming (see a run-time study in the supplement); real-world applications should consider real-time requirements.

Future work. In view of the above limitations, the future work includes: (1) Exploring high-resolution images for more precise detection on *hard* situations, e.g., one can combine proposals generated from multi-scale and multi-resolution images. (2) Developing faster algorithms, e.g., one can use multi-scale detectors to attend to regions of interest for progressive detection. (3) Bridging end-to-end fast models and powerful yet slow pretrained models, e.g., one can train lightweight adaptors atop pretrained models for better InsDet.

6 Conclusion

We explore the problem of Instance Detection (InsDet) by introducing a new dataset consisting of high-resolution images and formulating a realistic unified protocol. We revisit representative InsDet methods in the cut-paste-learn framework and design a non-learned method by leveraging publicly-available pretrained models. Extensive experiments show that the non-learned method significantly outperforms end-to-end InsDet models. Yet, the non-learned method is slow because running large pretrained models takes more time than end-to-end trained models. Moreover, all methods struggle in hard situations (e.g., in front of heavy occlusions and a high level of clutter in the scene). This shows that our dataset serves as a challenging venue for the community to study InsDet.

References

[1] Phil Ammirato, Patrick Poirson, Eunbyung Park, Jana Kosecka, and Alexander C. Berg. A dataset for developing and benchmarking active vision. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.

[2] Phil Ammirato, Cheng-Yang Fu, Mykhailo Shvets, Jana Kosecka, and Alexander C Berg. Target driven instance detection. *arXiv:1803.04610*, 2018.

[3] Siddharth Ancha, Junyu Nan, and David Held. Combining deep learning and verification for precise object instance detection. *arXiv:1912.12270*, 2019.

[4] Richard Bormann, Xinjie Wang, Markus Völk, Kilian Kleeberger, and Jochen Lindermayr. Real-time instance detection with fast incremental learning. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.

[5] Eric Brachmann, Alexander Krull, Frank Michel, Stefan Gumhold, Jamie Shotton, and Carsten Rother. Learning 6d object pose estimation using 3d object coordinates. In *ECCV*, 2014.

[6] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *CVPR*, 2018.

[7] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.

[8] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[9] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[10] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[11] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[12] Debidatta Dwibedi, Ishan Misra, and Martial Hebert. Cut, paste and learn: Surprisingly easy synthesis for instance detection. In *ICCV*, 2017.

[13] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2009.

[14] David Gale and Lloyd S Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, 1962.

[15] Georgios Georgakis, Md. Alimoor Reza, Arsalan Mousavian, Phi Hung Le, and Jana Kosecka. Multiview rgb-d dataset for object instance detection. *International Conference on 3D Vision (3DV)*, 2016.

[16] Georgios Georgakis, Md Alimoor Reza, Arsalan Mousavian, Phi-Hung Le, and Jana Kovsecká. Multiview rgb-d dataset for object instance detection. In *International Conference on 3D Vision (3DV)*, 2016.

[17] Georgios Georgakis, Arsalan Mousavian, Alexander C Berg, and Jana Kosecka. Synthesizing training data for object detection in indoor scenes. *Robotics: Science and Systems (RSS)*, 2017.

[18] Ross Girshick. Fast r-cnn. In *ICCV*, 2015.

[19] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[20] Stefan Hinterstoisser, Cedric Cagniart, Slobodan Ilic, Peter Sturm, Nassir Navab, Pascal Fua, and Vincent Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE transactions on pattern analysis and machine intelligence*, 34(5):876–888, 2011.

[21] Stefan Hinterstoißer, Vincent Lepetit, Slobodan Ilic, Stefan Holzer, Gary R. Bradski, Kurt Konolige, and Nassir Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian Conference on Computer Vision*, 2012.

[22] Tomávs Hodavn, Vibhav Vineet, Ran Gal, Emanuel Shalev, Jon Hanzelka, Treb Connell, Pedro Urbina, Sudipta N Sinha, and Brian Guenter. Photorealistic image synthesis for object instance detection. In *IEEE international conference on image processing (ICIP)*, 2019.

[23] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *ICCV*, 2017.

[24] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv:2304.02643*, 2023.

[25] Ikki Kishida, Hong Chen, Masaki Baba, Jiren Jin, Ayako Amma, and Hideki Nakayama. Object recognition with continual open set domain adaptation for home robot. In *WACV*, 2021.

[26] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view rgb-d object dataset. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2011.

[27] Kevin Lai, Liefeng Bo, and Dieter Fox. Unsupervised feature learning for 3d scene labeling. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[28] Hengduo Li, Bharat Singh, Mahyar Najibi, Zuxuan Wu, and Larry S Davis. An analysis of pre-training on object detection. *arXiv:1904.05871*, 2019.

[29] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[30] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *ICCV*, 2017.

[31] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *ECCV*, 2016.

[32] David G McVitie and Leslie B Wilson. The stable marriage problem. *Communications of the ACM*, 14(7):486–490, 1971.

[33] Jean-Philippe Mercier, Mathieu Garon, Philippe Giguere, and Jean-Francois Lalonde. Deep template-based object instance detection. In *WACV*, 2021.

[34] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv:2304.07193*, 2023.

[35] A Quadros, James Patrick Underwood, and Bertrand Douillard. An occlusion-aware feature for range images. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[36] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv:1804.02767*, 2018.

[37] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *CVPR*, 2016.

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, 2015.

[39] Colin Rennie, Rahul Shome, Kostas E Bekris, and Alberto F De Souza. A dataset for improved rgbd-based object detection and pose estimation for warehouse pick-and-place. *IEEE Robotics and Automation Letters*, 1(2):1179–1185, 2016.

[40] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. " grabcut" interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004.

[41] Neil Savage et al. Robots rise to meet the challenge of caring for old people. *Nature*, 601(7893): 8–10, 2022.

[42] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *CVPR*, 2016.

[43] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *CVPR Workshops*, 2014.

[44] Arjun Singh, James Sha, Karthik S. Narayan, Tudor Achim, and P. Abbeel. Bigbird: A large-scale 3d database of object instances. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[45] Zhi Tian, Hao Chen, Xinlong Wang, Yuliang Liu, and Chunhua Shen. AdelaiDet: A toolbox for instance-level recognition tasks. `https://git.io/adelaidet`, 2019.

[46] Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. Fcos: Fully convolutional one-stage object detection. In *ICCV*, 2019.

[47] Chien-Yao Wang, Alexey Bochkovskiy, and Hong-Yuan Mark Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. *arXiv:2207.02696*, 2022.

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

[49] Hao Zhang, Feng Li, Shilong Liu, Lei Zhang, Hang Su, Jun Zhu, Lionel Ni, and Harry Shum. Dino: Detr with improved denoising anchor boxes for end-to-end object detection. In *International Conference on Learning Representations*, 2022.

[50] Xingyi Zhou, Dequan Wang, and Philipp Krähenbühl. Objects as points. In *arXiv:1904.07850*, 2019.

[51] Xingyi Zhou, Vladlen Koltun, and Philipp Krähenbühl. Probabilistic two-stage detection. In *arXiv:2103.07461*, 2021.

[52] Xizhou Zhu, Weijie Su, Lewei Lu, Bin Li, Xiaogang Wang, and Jifeng Dai. Deformable detr: Deformable transformers for end-to-end object detection. *arXiv:2010.04159*, 2020.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section 3.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See the last paragraph in Section 1.
   (b) Did you describe the limitations of your work? [Yes] See the second paragraph in Section 5.3
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] See the first paragraph in Section 5.3
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments (e.g. for benchmarks)...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We are constructing a website for this work, and will release open-source code.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Implementation details in Section 5. We (will) release open-source code for further details and reproduction.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [N/A]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 5.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See implementations in Section 5
   (b) Did you mention the license of the assets? [No] We use multiple open-source GitHub repositories which have different licenes but are free to use for non-commercial and research perposes.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]