FewSOL: A Dataset for Few-Shot Object Learning in Robotic Environments

Jishnu Jaykumar P[1] **Yu-Wei Chao**[2] **Yu Xiang**[1]
[1]The University of Texas at Dallas [2]NVIDIA
{jishnu.p,yu.xiang}@utdallas.edu ychao@nvidia.com

Abstract: We introduce the Few-Shot Object Learning (FewSOL) dataset for object recognition with a few images per object. We captured 336 real-world objects with 9 RGB-D images per object from different views. Object segmentation masks, object poses and object attributes are provided. In addition, synthetic images generated using 330 3D object models are used to augment the dataset. We investigated (i) few-shot object classification and (ii) joint object segmentation and few-shot classification with the state-of-the-art methods for few-shot learning and meta-learning using our dataset. The evaluation results show that there is still a large margin to be improved for few-shot object classification in robotic environments. Our dataset can be used to study a set of few-shot object recognition problems such as classification, detection and segmentation, shape reconstruction, pose estimation, keypoint correspondences and attribute recognition. The dataset and code are available at https://irvlutd.github.io/FewSOL.

Keywords: Robot Perception, Object Recognition, Few-Shot Learning, Meta-Learning, Dataset Construction, Sim-to-Real

1 Introduction

For robots to work in human environments, they will encounter various objects in our daily lives. How can we build models to enable robots to recognize all kinds of objects and eventually manipulate these objects? In the robotics community, model-based object recognition has been the focus, where 3D models of objects are built and used for recognition. For example, the YCB Object and Model Set [1] has significantly benefited 6D object pose estimation research and manipulation research. The limitation of model-based object recognition is that it is difficult to obtain a large number of 3D models for many objects in the real world. The 3D scanning techniques are expensive and certain objects such as reflective objects and transparent objects cannot be reconstructed well. Another paradigm for object recognition focuses on recognizing object categories such as bowls, mugs and bottles. Most datasets for object category recognition only contain dozens of categories. For instance, the MSCOCO dataset for object detection and segmentation [2] has 80 categories. The NOCS dataset for object category pose estimation [3] only has 6 categories. While large-scale datasets collected from the Internet such as ImageNet [4] and Visual Genome [5] contain large numbers of object categories, these datasets are useful for learning visual representations. The Internet images are not very suitable to learn object representations for robot manipulation, where we care about detailed properties of objects such as 3D shape, 6D pose or visual correspondences.

In order to overcome the limitations of model-based approaches and category-based approaches for object recognition in robot perception, in this work, we introduce a new dataset to facilitate object recognition in robotic environments. We are motivated by few-shot learning and meta-learning in the literature [6]. Our aspiration is that, if robots can recognize objects from a few exemplar images, it is possible to scale up the number of objects a robot can recognize. Because collecting a few images per object is a much easier process compared to building a 3D model of an object. In addition, models trained in the meta-learning setting [7] can generalize to new objects without re-training.

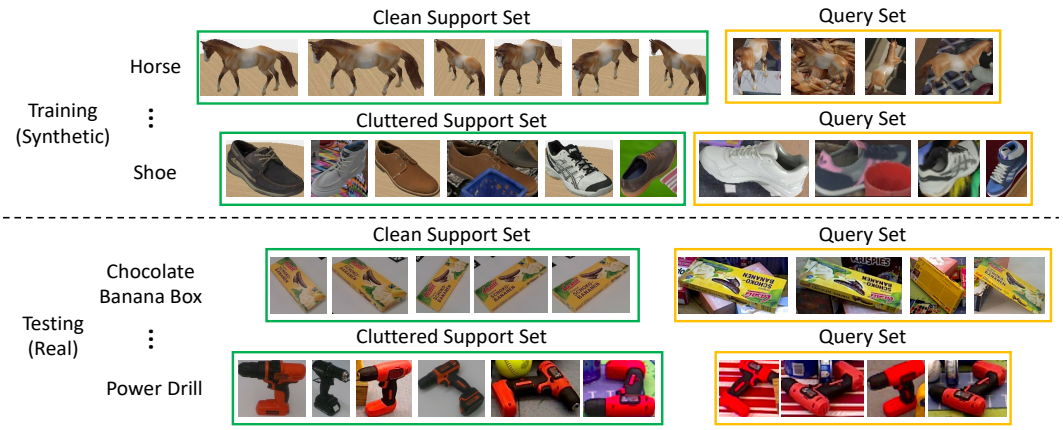


Figure 1: Examples of support sets and query sets from our dataset. Clean support sets only contain images with single objects in clean background, while cluttered support sets have images with multiple objects and different background.

In our Few-Shot Object Learning (FewSOL) dataset, we have collected images for 336 objects in the real world. For each object, we collected 9 RGB-D images from different views, i.e., 9 shots per object. We provide ground truth segmentation masks and 6D object poses of these objects, where the object poses are computed using AR tags. In addition, we employed Amazon Mechanical Turk (MTurk) to collect annotations of these objects including object names, object categories, materials, function and colors. For each object, we collected annotations from 5 MTurkers and then merged their answers. In this way, we can account for how different people name these objects in our dataset. Based on the collected object names, we have defined 198 classes for these 336 objects. In few-shot learning or meta-learning settings, we can think of these images as support sets. Our goal is to apply learned models to cluttered scenes. Therefore, we include the images from the Object Clutter Indoor Dataset (OCID) [8] in our dataset. Segmentation masks of objects are provided in the OCID dataset. We manually annotated class names of these objects, and found that they belong to 52 classes in our object categories. These images can be used as query sets.

To further expand the scale of our dataset, we also generate synthetic data to complement the data from the real world . We selected 330 3D object models from the Google Scanned Objects dataset [9] and used the PyBullet simulator to compose synthetic scenes of these objects and render synthetic RGB-D images from the scenes. Similar to the real-world data, we first put each 3D model onto a table and generate 9 views. The benefit of using synthetic data is that we can generate cluttered scenes with these objects and obtain annotations for all the objects such as segmentation masks. We generate 40,000 cluttered scenes and rendered 7 views per scene.

In this paper, we use our dataset to study two problems: few-shot object classification and joint object segmentation and few-shot classification. For few-show classification, we follow the protocol proposed in the Meta-Dataset [10] which constructs episodes for training and testing. Each episode consists of multiple support sets and query sets with different numbers of classes and images per class. Fig. 1 illustrates some support sets and query sets from our dataset. We reserve the cluttered images from the OCID dataset for testing, and limit training on synthetic data only. In this way, we can investigate sim-to-real transfer for few-shot learning with our dataset. We use the ground truth segmentation masks to crop the objects for classification. For joint object segmentation and few-shot classification, we first apply an object segmentation method and then use the predicted masks to crop the objects for classification. Therefore, segmentation errors can be accounted for in the classification accuracy. We have evaluated state-of-the-art methods for few-shot learning and meta-learning on these two settings using our dataset. These results can be used for future comparison.

To the best of our knowledge, our dataset is the first large-scale dataset for few-shot object learning. Although we only investigated few-shot classification in this paper, our dataset can be used to study a set of important problems for few-shot object recognition such as detection and segmentation, shape reconstruction, pose estimation, keypoint correspondences and attribute recognition.

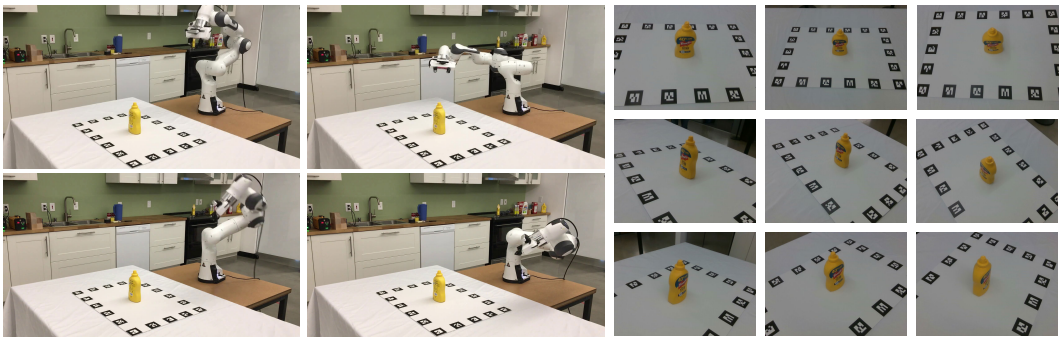


Figure 2: (a) Our data capture system with a Franka Emika Panda arm. (b) 9 images of a mustard bottle from different views captured in our dataset.

2 Related Work

Few-Shot Learning and Meta-Learning. In the context of image classification, few-shot learning indicates using a few images per class. The problem is usually formulated as "N-way, k-shot", i.e., N classes with k images per class. The end goal of few-shot learning is to learn a model on a set of training classes $\mathcal{C}_{train}$ that can generalize to novel classes $\mathcal{C}_{test}$ in testing. Each class has a support set and a query set. While the ground truth labels of both the support set and the query set for a class in $\mathcal{C}_{train}$ are available for learning, for a testing class in $\mathcal{C}_{test}$, only labels of the support set are available. Non-episodic approaches using all the data in $\mathcal{C}_{train}$ for training such as k-NN and its 'Finetuned' variants [11, 12, 13, 14]. These methods focus on learning feature representations using neural networks that can be used in $\mathcal{C}_{test}$. Episodic approaches are considered to be meta-learners. An episode in training or testing consists of a subset of classes with support and query sets. Learning is performed by minimizing the loss on the query sets of the training episode. Representative episodic approaches include Prototypical Networks [15], Matching Networks [16], Relation Networks [17], Model Agnostic Meta-Learning (MAML) [7], Proto-MAML [10] and CrossTransformers [18]. We evaluated majority of these state-of-the-art few-shot learning methods on FewSOL.

Datasets for Few-Shot Learning. The Omniglot [19] and the mini-ImageNet [16] are widely used for evaluating few-shot learning methods in the literature. Recently, the Meta-Dataset [10] was introduced for benchmarking few-shot learning and meta-learning methods. Meta-Dataset leverages data from 10 different datasets: ILSVRC-2012 (ImageNet [20]), Omniglot [19], Aircraft [21], CUB-200-2011 (Birds [22]), Describable Textures [23], Quick Draw [24], Fungi [25], VGG Flower [26], Traffic Signs [27] and MSCOCO [2]. As we can see, these data do not include daily objects for robot manipulation. Our dataset complements existing datasets for few-shot learning by explicitly addressing robotic applications.

3 Dataset Construction

In this work, our goal is to contribute a large-scale dataset of household objects in the few-shot learning setting that can be useful for various robotic applications, as well as studying few-shot object recognition. Our dataset consists of both real-world images and synthetic images. We describe our dataset construction in this section.

3.1 Data Capture in the Real World

For each object in the real world, we capture multiple exemplar images of the object. We automate this process using a Franka Emika Panda arm as shown in Fig. 2(a). An Intel RealSense D415 camera is mounted onto the Panda arm gripper. It can capture both RGB images and depth images. We specify 9 waypoints of the camera pose and utilize motion planning of the arm to move the camera to these poses. In this way, for each object, we can automatically capture 9 RGB-D images from 9 different views (Fig. 2(b), 3 poses on the left, 3 poses in the front and 3 poses on the right).

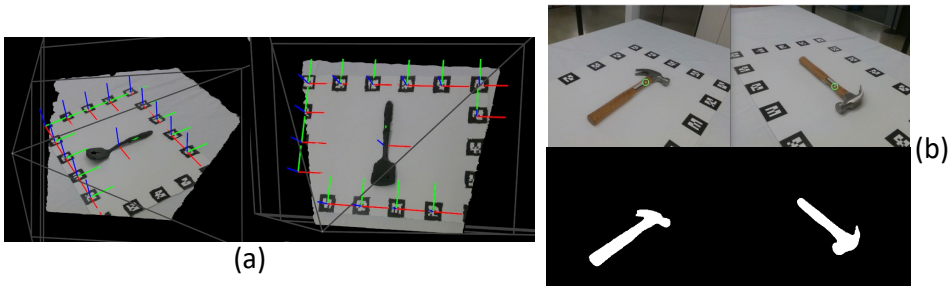


Figure 3: (a) Object poses from AR tags (b) Pixel correspondences using computed object poses and the segmentation masks of the objects.

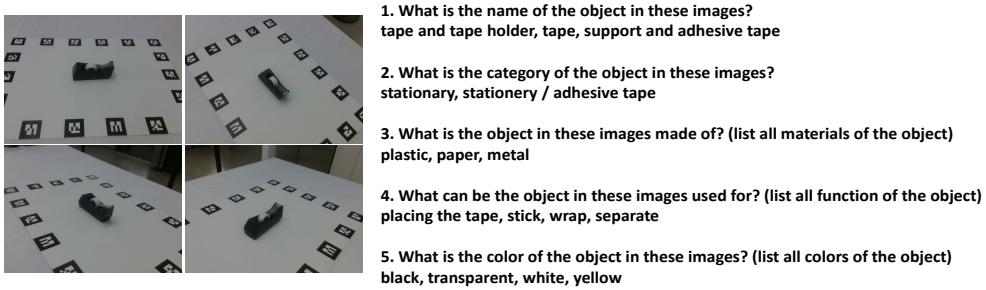


Figure 4: Our Amazon Mechanical Turk questionnaire for object annotation.

In order to accurately estimate the camera poses of these 9 images with respect to the object, we have designed a marker board with 18 AR tags. During data capture, we place the object in the center of the board. For each captured image, we use the detected AR tag poses in the image to compute the camera pose with respect to the center of the marker board and treat this pose as the estimated object pose of the image. Because there are noises in the AR tag poses, we use the RANSAC algorithm to estimate the object pose in this process. Fig. 3(a) shows the AR tag poses and the computed object poses with the point clouds from the RealSense camera. Using the object poses and the point clouds, we can also compute pixel correspondences between images as shown in Fig. 3(b). Consequently, we obtain 9 RGB-D images with their estimated object poses for each object. We have captured 336 objects in total. These include various daily objects from grocery stores and tools.

3.2 Data Annotation

After capturing these objects, our next step is to provide annotations of these objects. First, we generate the segmentation masks of these captured objects. Instead of manually segmenting these objects, we utilize the unseen object instance segmentation method proposed in [28] for segmentation. The trained network in [28] cannot successfully segment all the objects in the beginning. Therefore, we bootstrap the network by finetuning it on our dataset. After applying the network on all the data, we manually select accurate segmentation for finetuning, and then apply the finetuned network to unsuccessful images. We iterate this process until all the objects can be segmented by the network. Fig. 3(b) shows two examples of the generated segmentation masks.

Second, we provide object class labels and additional attributes for these objects. We leverage Amazon Mechanical Turk (MTurk) for this annotation. In this way, we can gather how lay people name the classes and attributes of the objects, which can be useful to deploy object recognition systems to real-world robotic scenarios. We designed 5 questions for each object and ask MTurkers to answer these questions. For each object, we gather answers from 5 different MTurkers and merge their answers. Fig. 4 illustrates an example with the questions and the merged answers. These annotations can used to recognize detailed attributes of objects. Based on these annotations from MTurk, we define 198 object classes for these 336 captured objects. Since 9 images are captured for each object, on average, each class has around 15 images in our dataset.

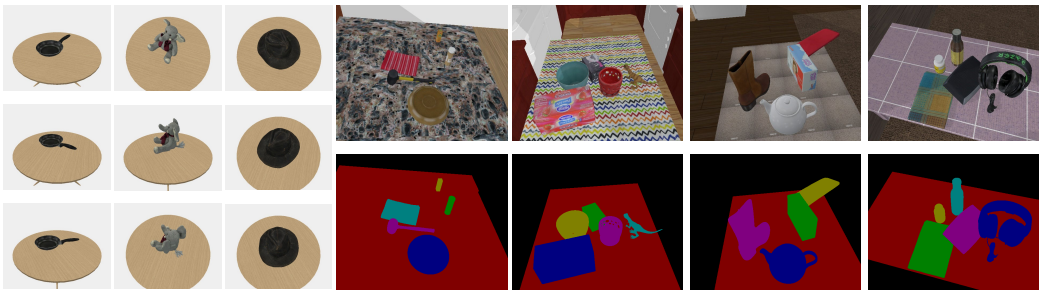


Figure 5: (a) Synthetic objects with clean background. (b) Synthetic objects in cluttered scenes.

3.3 Synthetic Data Generation

Leveraging synthetic data for learning has been successful in various robotic problems such as object segmentation [28], grasping [29] and control policy learning [30], since one can generate large-scale synthetic data with ground truth annotations automatically. In our dataset, we also leverage synthetic images for few-shot object learning. To do so, we selected 330 3D object models from Google Scanned Objects [9]. We use these 3D models to generate two types of data. First, similar to our real-world data capture, we place each object onto a table in the PyBullet simulator and generate 9 RGB-D images of each object from 9 different views. Fig 5(a) shows some examples of these multi-view images. Second, we generate cluttered scenes using these objects as shown in Fig 5(b). Thanks to the simulator, we can obtain object segmentation masks of these images effortlessly, while collecting cluttered scenes in the real world with segmentation annotations is time-consuming. We generated 40,000 scenes of these objects on tabletops and rendered 7 RGB-D images per scene. In order to obtain class names of these 330 objects, we also employ MTurk to collect annotations of these objects as described in Sec. 3.2. Eventually, we define 125 classes for these synthetic objects. Please see the supplementary materials for more details about our dataset including an example of the data capture process and examples of all the objects in the dataset.

3.4 Joint Object Segmentation and Few-Shot Classification

In real-world robotic applications, objects usually appear in cluttered scenes. Therefore, we need to separate an object from other objects and the background and then classify it. For our dataset, we target at the problem of joint object segmentation and few-shot classification. The problem is illustrated in Fig. 6. Given an image of a cluttered scene, the task is to segment objects in the scene and classify each object in few-shot learning settings. In order to evaluate the performance on joint segmentation and few-shot classification on real-world images, we leverage the Object Clutter Indoor Dataset (OCID) proposed in [8]. This dataset was originally proposed for object segmentation. It provides segmentation masks of objects in the dataset. We manually annotate objects in the OCID dataset with class labels. We found that there are 2300 objects in the OCID dataset after filtering a few bad segmentation annotations. These objects belong to 52 classes in our dataset. Objects in OCID can be partially occluded which makes the few-shot object classification challenging.

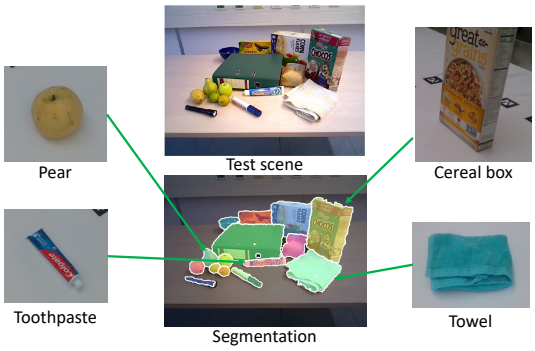


Figure 6: Illustration of the joint object segmentation and few-shot classification problem with an image from the OCID dataset [8].

3.5 Training and Testing

Our goal in building this dataset is to develop perception models that can classify objects in cluttered scenes with a few examples per class. Therefore, we reserve the objects in the OCID dataset for

testing. These are real-world objects in cluttered scenes (see Fig. 6). For training, we use the synthetic images rendered using Google Scanned Objects. The reasons for using synthetic data for training are: i) it is easy to generate cluttered scenes with ground truth annotations; ii) we can study sim-to-real transfer with our dataset.

In few-shot learning or meta-learning settings, both training and testing have a support set and a query set. During training, the labels of the support set and the query set are available. So we can use the labels on the query set to compute the training loss. During testing, labels of the support set are provided and the goal is to infer labels of the query set. Testing classes can be different from the training classes. When using our dataset, we consider two types of support sets: *clean support sets* and *cluttered support sets*. Clean support sets only consist of images of clean background without occlusions, while cluttered support sets contain images with different background and occlusions. Fig. 1 illustrates these two types of support sets and their query sets. Since query images are from cluttered scenes, training with clean support sets is more challenging. However, if a method can work well with clean support sets, it requires less annotations, i.e., no annotation from cluttered scenes is needed. For example, given a novel object, we can just collect a few images of the object in a clean background in order to recognize it. Therefore, we encourage models that can perform well with clean support sets in our dataset.

4 Benchmarking Experiments

In this section, we evaluate the state-of-the-art methods for few-shot learning and meta-learning on our dataset and analyze their performance.

4.1 Few-Shot Object Classification

First, we follow the training and testing procedure designed in Meta-Dataset [10] and evaluated the following methods on our dataset: k-NN baseline that classifies each query example to the class of its closest support example, Finetune baseline that trains a classifier on top of the feature embedding using the support set of a test episode, Prototypical Networks [15], Matching Networks [16], first-order Model Agnostic Meta-Learning (fo-MAML) [7], first-order Proto-MAML (fo-Proto-MAML) introduced in [10] and CrossTransformers (CTX) [18]. CTX uses an attention mechanism to compute a 'query-aligned' prototype for each class, and then use these prototypes as in Prototypical Networks. [18] also introduces using SimCLR [31] as training episodes by treating every image as its own class for self-supervised learning (CTX+SimCLR). Details about the training and testing setup can be found in the supplementary materials. 95% confidence intervals for the few-shot classification accuracy of these methods are presented in Table 1.

Training of these methods is performed on 112 classes of our synthetic dataset, and testing is conducted on 52 classes in the OCID dataset [8] and 13 validation classes in the synthetic dataset. The backbones of these methods are resnet-34 except for the Finetune baseline (resnet-18 due to GPU memory limit). As in Meta-Dataset [10], we compare with and without pre-training for the backbone network, where pre-training initializes the backbone weights as the k-NN Baseline model trained on ImageNet. We also use either clean support sets or cluttered support sets during training. The choice of pre-training and support set generates four training settings as shown in Table 1. For testing episodes, we evaluate on both clean support sets and cluttered support sets as well.

For the results in Table 1, we have the following observations. i) Using pre-training is beneficial. Most classification accuracies are improved with pre-training. ii) The performance on the synthetic classes is much better than the performance on the real classes. We can see the sim-to-real gap clearly. iii) Using cluttered support sets can achieve better performance than using clean support sets. Because query sets contain different background and occlusions. However, obtaining annotations for cluttered support sets in the real world is expensive. Methods using clean support sets in testing are encouraged. iv) Among the 52 classes in OCID, we separately tested on 41 unseen classes, i.e., novel classes not presented in training, and 11 seen classes. Overall, the performance on seen classes is better. v) Among these evaluated methods, CrossTransformers [18] achieves the best

Method	OCID (Real) [8]						Google (Synthetic) [9]	
	All (52 classes)		Unseen (41 classes)		Seen (11 classes)		Unseen (13 classes)	
	Cluttered S	Clean S	Cluttered S	Clean S	Cluttered S	Clean S	Cluttered S	Clean S
Training setting: clean support set without pre-training								
k-NN [10]	52.92 ± 1.08	16.19 ± 0.74	55.12 ± 1.08	16.73 ± 0.62	55.90 ± 1.08	31.94 ± 0.89	83.43 ± 0.76	79.91 ± 0.79
Finetune [10]	**57.96 ± 1.08**	28.99 ± 0.92	**62.45 ± 1.13**	**33.14 ± 0.91**	58.49 ± 1.01	36.46 ± 1.04	63.36 ± 1.75	66.60 ± 1.19
ProtoNet [15]	45.02 ± 0.89	16.18 ± 0.74	50.00 ± 0.91	17.20 ± 0.72	48.90 ± 0.88	32.73 ± 0.89	71.62 ± 0.98	72.63 ± 0.79
MatchingNet [16]	51.58 ± 1.08	19.40 ± 0.72	58.24 ± 1.04	21.70 ± 0.76	53.99 ± 1.02	33.10 ± 0.94	76.26 ± 0.82	77.26 ± 0.76
fo-MAML [7]	24.24 ± 1.17	17.01 ± 1.00	30.29 ± 1.33	19.09 ± 0.86	41.82 ± 0.95	41.82 ± 0.95	51.31 ± 1.70	59.54 ± 0.96
fo-Proto-MAML [10]	49.57 ± 1.00	20.06 ± 0.79	55.96 ± 1.04	22.51 ± 0.73	55.32 ± 1.03	33.45 ± 0.95	68.70 ± 1.42	81.69 ± 0.80
CTX [18]	53.17 ± 1.05	18.49 ± 0.84	55.81 ± 0.99	20.60 ± 0.94	56.77 ± 0.98	35.41 ± 0.93	**86.46 ± 0.70**	**88.08 ± 0.63**
CTX+SimCLR [18]	53.87 ± 1.03	**30.31 ± 1.00**	56.56 ± 0.97	30.43 ± 0.93	**64.90 ± 0.98**	**53.70 ± 1.18**	85.15 ± 0.69	83.94 ± 0.65
Training setting: cluttered support set without pre-training								
k-NN [10]	58.78 ± 1.04	21.72 ± 0.84	61.56 ± 1.13	22.17 ± 0.78	**66.33 ± 1.02**	41.76 ± 1.00	86.94 ± 0.67	80.99 ± 0.69
Finetune [10]	58.63 ± 1.12	29.94 ± 0.90	63.18 ± 1.06	32.71 ± 0.89	59.28 ± 1.07	39.71 ± 1.01	66.36 ± 1.77	65.02 ± 1.24
ProtoNet [15]	42.56 ± 0.88	15.17 ± 0.82	47.60 ± 0.87	16.23 ± 0.77	48.93 ± 0.89	33.69 ± 1.02	71.21 ± 0.97	66.76 ± 0.87
MatchingNet [16]	52.94 ± 1.09	17.98 ± 0.77	56.20 ± 1.05	19.52 ± 0.73	54.07 ± 1.03	31.18 ± 0.93	78.51 ± 0.82	72.25 ± 0.87
fo-MAML [7]	43.92 ± 1.07	17.26 ± 0.87	49.21 ± 1.03	18.80 ± 0.80	51.94 ± 1.00	28.91 ± 0.92	70.78 ± 0.90	66.54 ± 0.88
fo-Proto-MAML [10]	51.00 ± 1.02	17.35 ± 0.75	55.46 ± 1.06	19.59 ± 0.74	56.90 ± 1.06	31.99 ± 0.91	76.78 ± 1.10	77.36 ± 0.83
CTX [18]	49.96 ± 1.04	18.43 ± 0.74	53.91 ± 1.02	20.82 ± 0.87	57.97 ± 0.98	36.93 ± 1.03	**92.45 ± 0.46**	89.82 ± 0.58
CTX+SimCLR [18]	**60.83 ± 1.06**	**31.67 ± 0.97**	**63.80 ± 1.09**	**33.34 ± 0.99**	66.25 ± 1.01	**51.62 ± 1.10**	89.58 ± 0.57	**88.99 ± 0.57**
Training setting: clean support set with pre-training								
k-NN [10]	59.34 ± 1.10	23.40 ± 0.85	63.02 ± 1.07	24.98 ± 0.86	66.24 ± 1.01	39.36 ± 1.01	89.18 ± 0.59	85.77 ± 0.69
Finetune [10]	**59.77 ± 1.08**	32.15 ± 0.90	**64.01 ± 1.08**	35.54 ± 0.90	58.30 ± 1.09	37.72 ± 1.04	66.09 ± 1.72	69.85 ± 1.09
ProtoNet [15]	57.54 ± 1.06	**34.47 ± 1.00**	61.25 ± 1.11	**37.06 ± 1.01**	65.90 ± 1.04	51.07 ± 1.04	74.37 ± 1.12	81.38 ± 0.67
MatchingNet [16]	53.81 ± 1.02	26.33 ± 0.94	57.77 ± 0.93	28.05 ± 1.00	61.83 ± 0.97	45.81 ± 1.07	65.40 ± 1.57	85.18 ± 0.70
fo-MAML [7]	44.92 ± 1.20	15.71 ± 0.77	51.67 ± 1.13	17.74 ± 0.77	56.02 ± 1.04	30.78 ± 0.95	70.91 ± 1.08	73.86 ± 0.88
fo-Proto-MAML [10]	57.09 ± 1.04	27.01 ± 0.94	60.29 ± 1.02	28.69 ± 0.88	66.75 ± 1.04	44.39 ± 1.10	77.16 ± 1.10	88.14 ± 0.68
CTX [18]	56.65 ± 1.02	29.06 ± 0.99	60.33 ± 1.02	29.96 ± 0.94	65.47 ± 1.04	45.48 ± 1.12	**90.66 ± 0.63**	**92.72 ± 0.45**
CTX+SimCLR [18]	57.47 ± 1.03	31.29 ± 0.98	59.32 ± 0.98	31.31 ± 0.93	**67.73 ± 0.91**	**53.67 ± 1.14**	81.76 ± 0.74	82.76 ± 0.74
Training setting: cluttered support set with pre-training								
k-NN [10]	60.63 ± 1.10	23.09 ± 0.87	62.54 ± 1.12	23.97 ± 0.77	65.16 ± 1.04	40.82 ± 1.04	89.21 ± 0.55	84.49 ± 0.74
Finetune [10]	60.11 ± 1.12	31.23 ± 0.95	62.58 ± 1.10	33.22 ± 0.94	58.89 ± 1.03	36.48 ± 1.06	66.49 ± 1.73	68.31 ± 1.15
ProtoNet [15]	59.02 ± 1.00	31.86 ± 1.02	61.56 ± 1.09	34.12 ± 1.06	66.47 ± 0.94	48.80 ± 1.12	79.49 ± 0.94	79.19 ± 0.78
MatchingNet [16]	62.35 ± 1.06	28.50 ± 0.93	**65.41 ± 1.06**	30.16 ± 0.94	70.50 ± 0.99	44.01 ± 1.03	85.44 ± 0.63	84.10 ± 0.70
fo-MAML [7]	56.04 ± 1.08	20.64 ± 0.76	58.01 ± 1.12	21.24 ± 0.81	58.81 ± 1.12	32.38 ± 0.96	79.12 ± 0.95	71.88 ± 1.00
fo-Proto-MAML [10]	60.98 ± 1.01	28.32 ± 0.91	63.18 ± 1.02	29.08 ± 0.93	71.44 ± 1.00	46.98 ± 1.07	89.21 ± 0.70	86.70 ± 0.71
CTX [18]	56.29 ± 0.93	26.93 ± 0.91	58.52 ± 0.92	27.40 ± 0.95	63.00 ± 1.01	44.11 ± 1.06	94.51 ± 0.39	**92.06 ± 0.48**
CTX+SimCLR [18]	**62.70 ± 1.07**	**38.56 ± 1.12**	64.86 ± 1.09	**38.22 ± 1.07**	**73.11 ± 0.93**	**61.47 ± 1.11**	**89.23 ± 0.60**	90.01 ± 0.49

Table 1: Benchmarking results on few-shot object classification in terms of 95% confidence intervals for classification accuracy with *episodic testing* consisting of 600 episodes as in Meta-Dataset [10]

performance. When using clean support sets, CTX+SimCLR has a large margin compared to other methods, highlighting the importance of self-supervised representation learning in SimCLR [31].

4.2 Joint Object Segmentation and Few-Shot Classification

In this experiment, we conduct non-episodic testing on all the 2,300 objects among the 52 classes in the OCID dataset. When cropping objects from the original images for classification, we tested using ground truth masks and using the predicted masks from [28]. When using predicted masks, we need to assign a mask to each object. This is achieved by the Hungarian method with pairwise F-measure that computes a matching between predicted masks and ground truth objects. These cropped objects construct the query set. We use the real-world objects that we captured on a tabletop as the clean support sets. We present top-1 and top-5 classification accuracies of the evaluated methods in Table 2. The methods are trained on the 112 classes of our synthetic dataset with pre-training. If an object cannot be segmented by a segmentation method, i.e., no assigned mask for the Hungarian matching, we consider this object as a misclassification. In this way, the classification accuracy also accounts for the segmentation performance, and using ground truth segmentation masks focuses on evaluating the classification performance only. For Table 2, we can see that: i) The top-1 accuracy is around 25% for the best method, which indicates that there is still a large margin to be improved in this setting. The difficulties lie in using synthetic images for training and using clean support sets during testing. ii) Classification accuracies of seen classes are much higher than unseen classes. iii) Overall, CTX+SimCLR achieves the best performance when training with cluttered support sets, while Prototypical Network is better when training with clean support sets.

4.3 Qualitative Results in the Real World

In this experiment, our goal is to build a few-shot classification model that works the best on real-world perception systems. So we train CTX+SimCLR [18] with all the real data and the synthetic data in our dataset, and then test the trained model in our lab. Cluttered support sets are used for training. Fig. 7 shows one testing image and the few-shot classification results. RGB-D images are

Method	OCID (Real) [8]					
	Use GT segmentation			Use segmentation from [28]		
	All (52) Clean S	Unseen (41) Clean S	Seen (11) Clean S	All (52) Clean S	Unseen (41) Clean S	Seen (11) Clean S
Training setting: clean support set with pre-training (top-1, top-5)						
k-NN [10]	14.65, 25.22	15.33, 24.41	41.03, 72.65	12.70, 23.22	13.70, 22.59	36.75, 67.95
Finetune [10]	22.26, 50.17	**26.41**, 58.20	31.62, 80.34	21.30, 48.57	**24.34**, 53.94	35.47, 67.38
ProtoNet [15]	**25.17**, **57.30**	25.22, **58.45**	51.99, **94.73**	**22.96**, **51.96**	22.65, **54.32**	49.86, **87.75**
MatchingNet [16]	17.39, 48.35	14.64, 50.06	51.85, 90.31	15.78, 45.13	13.08, 46.93	49.15, 84.47
fo-MAML [7]	11.43, 31.48	11.58, 34.73	36.89, 69.94	10.91, 29.17	10.01, 32.35	31.77, 63.68
fo-Proto-MAML [10]	14.35, 28.96	5.63, 40.61	45.58, 71.51	13.39, 26.96	5.51, 37.73	41.74, 67.24
CTX [18]	17.48, 46.57	18.21, 49.81	51.85, 87.75	15.70, 43.83	16.90, 46.31	47.86, 81.34
CTX+SimCLR [18]	18.57, 50.30	20.46, 51.06	**57.55**, 93.16	16.48, 46.17	17.71, 47.12	**52.14**, 85.75
Training setting: cluttered support set with pre-training (top-1, top-5)						
k-NN [10]	13.70, 23.83	15.33, 24.28	47.72, 72.79	13.26, 23.22	14.14, 22.90	44.73, 68.66
Finetune [10]	22.17, 53.35	24.34, 55.63	31.91, 71.51	18.26, 44.22	20.65, 52.00	36.04, 69.52
ProtoNet [15]	21.35, 50.57	22.34, 51.31	51.99, 90.46	18.61, 47.22	18.21, 48.12	45.44, 85.33
MatchingNet [16]	17.52, 50.96	17.77, 52.32	49.43, 88.18	16.52, 46.52	15.58, 48.81	43.45, 82.76
fo-MAML [7]	16.48, 38.52	13.70, 39.49	37.46, 77.07	15.35, 35.04	11.08, 34.36	40.31, 69.94
fo-Proto-MAML [10]	11.04, 28.70	4.01, 38.67	43.73, 72.65	9.91, 26.35	3.57, 35.79	40.46, 68.09
CTX [18]	19.00, 45.48	17.71, 44.74	51.85, 88.75	17.13, 42.22	16.08, 42.12	47.15, 83.19
CTX+SimCLR [18]	**24.61**, **62.39**	**25.16**, **63.52**	**65.81**, **96.30**	**22.17**, **57.43**	**23.28**, **57.57**	**59.12**, **88.32**

Table 2: Benchmarking results on joint object segmentation and few-shot classification in terms of top-1 and top-5 classification accuracy with *non-episodic testing* on the OCID dataset [8]

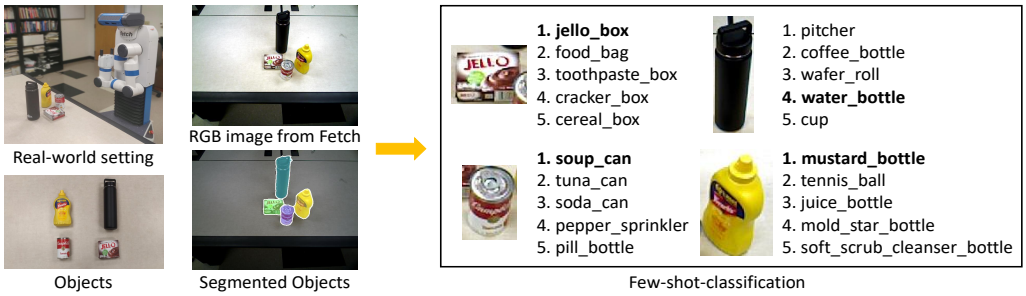


Figure 7: Qualitative classification results with top-5 predictions from our real-world testing.

collected from a Fetch mobile manipulator and we used [28] for object segmentation. We tested on 32 objects with 4 objects in an image. The model achieves 28.13% top-1 accuracy and 56.25% top-5 accuracy. Please see the supplementary materials for these classification results. The low top-1 accuracy indicates the difficulty of the few-shot object classification problem in the real world. We hope that our dataset can be used to build better models for this problem.

5 Conclusion and Future Work

We introduce the Few-Shot Object Learning (FewSOL) dataset for few-shot object recognition. Different from existing datasets for few-shot learning, our dataset contains daily objects such as personal items, tools and fruits. We provide RGB-D images, object segmentation masks, object poses and object attribute annotations in the dataset. We hope the dataset can facilitate progress on robot object perception. If a robot can recognize all the object classes in the dataset (198 classes of real objects), this will help lots of robotic applications such as manipulation, object retrieval, object grounding, task planning, and so on. We demonstrated using our dataset for few-shot classification and joint segmentation and few-shot classification in this paper. From the experimental results, we see a need to improve the few-shot recognition performance for real-world robotic applications. In the future, we plan to study how to leverage depth data and multi-view information to improve few-shot object classification. We also plan to study few-shot object representation learning for shape reconstruction, object pose estimation and object attribute recognition using the FewSOL dataset.

Acknowledgments

This work was supported in part by the DARPA Perceptually-enabled Task Guidance (PTG) Program under contract number HR00112220005.

References

[1] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar. Benchmarking in manipulation research: The YCB object and model set and benchmarking protocols. *arXiv preprint arXiv:1502.03143*, 2015.

[2] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision (ECCV)*, pages 740–755. Springer, 2014.

[3] H. Wang, S. Sridhar, J. Huang, J. Valentin, S. Song, and L. J. Guibas. Normalized object coordinate space for category-level 6D object pose and size estimation. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2642–2651, 2019.

[4] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255. Ieee, 2009.

[5] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, et al. Visual genome: Connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision (IJCV)*, 123(1):32–73, 2017.

[6] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni. Generalizing from a few examples: A survey on few-shot learning. *ACM computing surveys (CSUR)*, 53(3):1–34, 2020.

[7] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning (ICML)*, pages 1126–1135. PMLR, 2017.

[8] M. Suchi, T. Patten, D. Fischinger, and M. Vincze. Easylabel: A semi-automatic pixel-wise object annotation tool for creating robotic rgb-d datasets. In *International Conference on Robotics and Automation (ICRA)*, pages 6678–6684. IEEE, 2019.

[9] L. Downs, A. Francis, N. Koenig, B. Kinman, R. Hickman, K. Reymann, T. B. McHugh, and V. Vanhoucke. Google scanned objects: A high-quality dataset of 3D scanned household items. *arXiv preprint arXiv:2204.11918*, 2022.

[10] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, et al. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[11] S. Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4367–4375, 2018.

[12] H. Qi, M. Brown, and D. G. Lowe. Low-shot learning with imprinted weights. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5822–5830, 2018.

[13] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

[14] Y. Tian, Y. Wang, D. Krishnan, J. B. Tenenbaum, and P. Isola. Rethinking few-shot image classification: a good embedding is all you need? In *European Conference on Computer Vision (ECCV)*, pages 266–282. Springer, 2020.

[15] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 30, 2017.

[16] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. *Advances in Neural Information Processing Systems (NeurIPS)*, 29, 2016.

[17] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1199–1208, 2018.

[18] C. Doersch, A. Gupta, and A. Zisserman. Crosstransformers: spatially-aware few-shot transfer. *Advances in Neural Information Processing Systems (NeurIPS)*, 33:21981–21993, 2020.

[19] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[20] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[21] S. Maji, E. Rahtu, J. Kannala, M. Blaschko, and A. Vedaldi. Fine-grained visual classification of aircraft. *arXiv preprint arXiv:1306.5151*, 2013.

[22] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The caltech-ucsd birds-200-2011 dataset. 2011.

[23] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3606–3613, 2014.

[24] J. Jongejan, H. Rowley, T. Kawashima, J. Kim, , and N. Fox-Gieg. The quick, draw! – a.i. experiment, quickdraw.withgoogle.com. 2016.

[25] M. Sulc, L. Picek, J. Matas, T. Jeppesen, and J. Heilmann-Clausen. Fungi recognition: A practical use case. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 2316–2324, 2020.

[26] M.-E. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.

[27] S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, and C. Igel. Detection of traffic signs in real-world images: The german traffic sign detection benchmark. In *The International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2013.

[28] Y. Xiang, C. Xie, A. Mousavian, and D. Fox. Learning RGB-D feature embeddings for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, pages 461–470. PMLR, 2021.

[29] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. A. Ojea, and K. Goldberg. Dex-net 2.0: Deep learning to plan robust grasps with synthetic point clouds and analytic grasp metrics. *arXiv preprint arXiv:1703.09312*, 2017.

[30] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the sim-to-real loop: Adapting simulation randomization with real world experience. In *International Conference on Robotics and Automation (ICRA)*, pages 8973–8979. IEEE, 2019.

[31] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International Conference on Machine Learning (ICML)*, pages 1597–1607. PMLR, 2020.

APPENDIX

Contents

A Definitions

A **clean support set** (Clean S) consists of single objects with clean background. A **cluttered support set** (Cluttered S) consists of single objects or objects from a collection of objects in a cluttered scene with different backgrounds and occlusions. An **episodic testing** indicates using a number of episodes in testing where each episode consists of several support and query sets from different classes. A **non-episodic testing** indicates the classification of all the test images in the test set without using episodes. These images can be considered to be a whole query set.

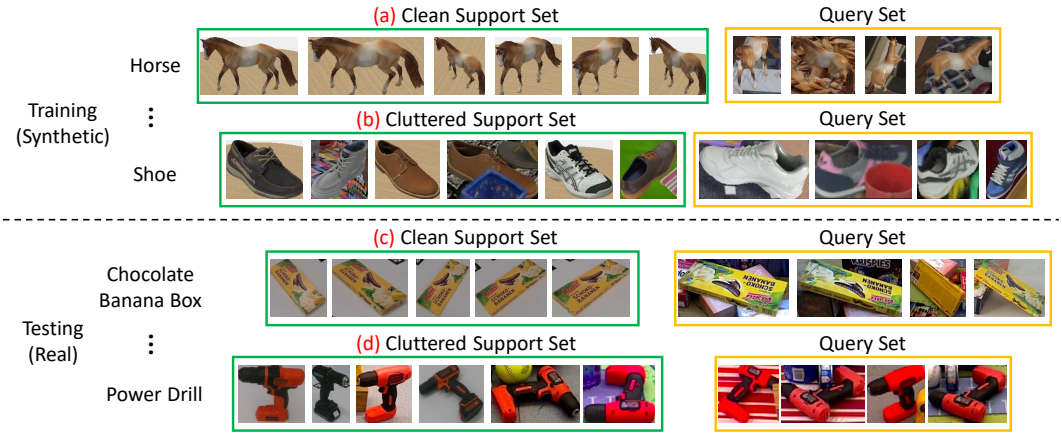


Figure 8: {a,c} and {b,d} are examples of clean and cluttered support sets of training and testing episodes, respectively.

B Data and Evaluation Metrics

B.1 Data Preparation Pipeline

We follow the same data preparation pipeline devised in META-DATASET [10]. Raw FewSOL dataset is converted to the TFRecord[1] format. The data pipeline utilizes TFRecords to form episodes. The data pipeline randomly shuffles and selects entries from TFRecords to form the train and test episodes which are then used to train and test the respective models mentioned in Section C.1.

B.2 Dataset Split

Train and Validation Sets. For few-shot classification and joint segmentation and few-shot classification, training data comprised 125 synthetic object classes. The training data is split into train and validation sets based on a 90:10 ratio of classes. The classes were first sorted in descending order based on the number of support examples. The top 90% classes, i.e., 112 classes form the train set, and the last 10%, i.e., 13 classes form the validation set. This would allow using more variety of support examples for training and validation using fewer support examples which is an apt case for the real world as well. With this setup, it is clear that train and validation classes are disjoint.

Test Sets. For few shot classification, as shown in Table 3, four variants of the test data were generated: (i) **All** comprises of 52 test classes, (ii) **Unseen** comprises of 41 test classes (disjoint from training classes), (iii) **Seen** comprises of 11 test classes (common with training classes) and (iv) **Unseen (Synthetic)** comprises

Variant	#Classes
All	52
Unseen	41
Seen	11
Unseen (Synthetic)	13

Table 3: Test data variants.

[1] https://www.tensorflow.org/tutorials/load_data/tfrecord

of 13 test classes (common with the validation set portion of the training data). For joint segmentation and few shot classification, only {All, Unseen, Seen} variants were used.

The Seen variant has 11 real object classes that coincide with synthetic object classes, hence a need to keep all 11 synthetic object classes in the train split came up. This is useful for studying the effect on these classes during episodic testing of the Seen variant. This need was catered by transferring any of the 11 synthetic object classes common with the Seen variant test classes from validation split to train split. The void of the transferred classes in the validation split was filled by the bottom classes of the initially sorted train split so that classes with fewer support samples stay in the validation split and the train split gets classes with more support samples which may aid in training the models. Fortunately, with the current setup of $112/13$ after sorting only one class had to be swapped. The last train split class **product_box** was swapped with **sponge** class from validation split. After all these steps, validation classes were product_box, eagle, wood_tower, cream_box, cake_pan, screwdriver, food_can, bottle, paper_roll, honey_dipper, racoon, hard_drive, and rubber_band. Furthermore, the last step of image filtration discussed in Section B.3 was performed, i.e., all images with either width or height less than 15 pixels were discarded as it may cause the problem shown in Figure 9.

For the real-world setup, all real and synthetic data from the FewSOL dataset were used for training. Hence, the training data comprised 323 (125+198) classes. This forms a mixture of synthetic and real object classes. Since only 52 out of 198 classes had query sets, the query sets for the remaining 146 classes were generated by splitting the available support set images into two halves. One half was considered as the support set while the other formed the query set. A 90:10 split yielded 291 train and 32 validation classes. While these 32 validation classes have real images, there are similar 32 synthetic classes in 291 train classes. This is a good setup as the Sim-to-Real transfer effect could be utilized here. Thus, in this way train and validation classes are disjoint.

B.3 Data preprocessing

Discard Low Resolution Images. As shown in Figure 9, low-resolution images, i.e., images having very small width or height when resized to 84x84x3 or 126x126x3 yields blurred images with no useful feature to learn. Hence, we discard images with width or height less than 15 pixels to avoid including noises while training and testing (except during real-world testing with 198 classes and joint object segmentation and few shot classification training and testing). The threshold for 15 pixels has been chosen with the heuristic that it is very small and going less than that may not help. Also, increasing this threshold might discard a lot of images that could be useful for training and testing.

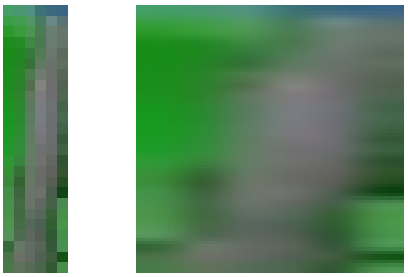


Figure 9: The left (low resolution and blurred) image shows an object after cropping from a cluttered scene while the right image shows the same when resized to 84x84x3 size. This may not help during training as all the images are reshaped to 84x84x3 or 126x126x3 before being fed into any model for few-shot learning.

Support Set Oversampling. This step is required for few shot classification only. As discussed in Section B.1, the data pipeline randomly shuffles and selects entries from TFRecords to form episodes. This affects FewSOL dataset as it has fewer support samples w.r.t. query (refer Table 6 and 7). Hence, while performing the filtration discussed in Section C.9, either no or fewer support images might be left. This is a case of set imbalance. Thus with the standard data pipeline of META-DATASET [10], the solution was to oversample the support images to match the cardinality of the query set. This oversampling only happens at the TFRecords level and not on the raw images. This gives a fair chance to the support samples to be selected alongside the query samples. Oversampling is done as $N_{ic} = \lceil Q_c/S_c \rceil$, where N_{ic} is the number of times to oversample the i^{th}

Backbone	Accuracy	Image Size
four_layer_convnet	25.30%	84
four_layer_convnet	31.17%	126
resnet	33.20%	84
resnet	29.96%	126
resnet34	**34.44%**	84
resnet34	**33.76%**	126
resnet34 (max-stride=16)	**35.14%**	126

Table 4: Ablation study of image size w.r.t. few-shot classification accuracy for selected backbones using k-NN [10] as the learner model. Here, k-NN [10] is trained on ImageNet [4]. k-NN [10] was selected following the suggestion from META-DATASET [10] that some of the best meta-learning models are initialized from the weights of a batch baseline.

support image in class C, S_c and Q_c are the number of available support and query images in class C, respectively.

B.4 Evaluation Metrics

For few shot classification similar to META-DATASET [10], we report 95% confidence intervals for classification accuracy with episodic testing consisting of 600 episodes. This is reasonable as variable "N-way K-shot" setup is used to construct test episodes. For joint object segmentation and few shot classification along with real-world testing, top-1 and top-5 classification accuracy are reported on all query data while utilizing all support data.

C Experiment Details

C.1 Models

We purposefully restricted our experiments to a selected set of few-shot and meta-learning methods from the growing list of META-DATASET [10], viz., k-NN [10], k-NN-Finetuned [10], Prototypical Networks [15], Matching Networks [16], Model Agnostic Meta-Learning (MAML) [7], Proto-MAML [10], CrossTransformers (CTX) [18] and CTX+SimCLR [18] variant due to the combination explosion of the experiments, hardware constraints and stability of META-DATASET [10] codebase. We think that these methods are a good representative sample to begin with.

C.2 Embedding Networks (Backbones)

Each model described in Section C.1 needs a backbone to learn useful features. META-DATASET [10] offers a variety of backbone options: four layer convolutional network (four_layer_convnet), resnet18 (resnet), wide_resnet, and resnet34. All models use resnet34 backbone architecture except k-NN-Finetuned [10] which uses resnet due to GPU memory limit. Resnet34 was mainly chosen due to its superior performance during pretraining using ImageNet [4]. Resnet with image_size=84 gave on par performance w.r.t. Resnet34, thus it was the next substitute for k-NN-Finetuned [10]. CTX uses resnet34 with an extra setting of max_stride=16.

C.3 Backbone Pretraining

As in META-DATASET [10], we pretrain the backbones mentioned in Table 4 using ImageNet [4] for 50K updates. These backbones were selected based on the suggestions from META-DATASET [10]. An ablation study on the effect of image size on few shot classification accuracy of selected backbones is shown in Table 4. The accuracy reported is the mean component of the 95% confidence interval. Confidence interval is used mainly due to the variable "N-way K-shot" setup. It is clear from Table 4 that resnet34 performs better. Thus it was selected as the backbone. Since CTX+SimCLR [18] performs better at higher resolution, we chose to go ahead with the im-

Model	Backbone	Max Stride	Image Size	Num Training Updates
k-NN [10]	resnet34	-	126	75K
k-NN-Finetuned [10]	resnet	-	84	75K
ProtoNet [15]	resnet34	-	126	75K
MatchingNet [16]	resnet34	-	126	75K
fo-MAML [7]	resnet34	-	126	75K
fo-Proto-MAML [10]	resnet34	-	126	75K
CTX [18]	resnet34	16	126	100K
CTX+SimCLR [18]	resnet34	16	126	400K

Table 5: Details of hyperparameters for training and testing. Num Training Updates denotes the maximum number of updates that training process can perform.

age_size=126. We tried using image_size=224 but were constrained due to the GPU memory limit. To maintain uniformity and perform comparison across various methods, resnet34 was used as the backbone with image_size=126. The GPU memory limit in case of k-NN-Finetuned [10] made the option of going ahead with resnet as the backbone with image_size=84 as it performed on par with resnet34 (refer Table 4). The images are resized to (image_size x image_size) before the forward pass similar to META-DATASET [10].

C.4 Hyperparameters

The best hyperparameters reported for each model in META-DATASET [10] have been used for our experiments. The changes to the default hyperparameters related to backbones are discussed in Section C.3. Following are the changes from our end: (i) Model training involving resnet34 backbone have learning_rate set as 0.001052178216688174, (ii) Number of validation episodes are set to 60 (default 600) taking into consideration the minimum need for constructing a confidence interval and the exceeding time for validation, (iii) EpisodeDescriptionConfig.max_support_size_contrib_per_class is set to 9 instead of default 500, (iv) EpisodeDescriptionConfig.max_ways_upper_bound is set to 52 and 198 for few shot classification and joint segmentation and few shot classification respectively.

C.5 Training Details

We have two training setups: (a) using 125 synthetic classes and (b) using all classes of the FewSOL dataset, i.e. 323 classes. The training setup (a) is further divided into two more variants based on the type of support set used for training: (i) using clean support sets and (ii) using cluttered support sets. Figure 8 shows the clean and cluttered support sets for few shot classification training and testing. Each support set setup is further divided into 2 types based on (i) using pretrained backbones and (ii) using backbones initialized randomly. Details of the backbones are mentioned in Section C.2. We use the same protocol as in META-DATASET [10] for episode composition. We use the variable "N-way K-shot" setup for training. Training setup (a) is used to train models mentioned in Section C.1.

Training setup (b) uses all synthetic and real data from the FewSOL dataset to train CTX+SimCLR [18]. Only cluttered support sets are used for training this particular model which would be used for real-world testing. Refer Section D for more details.

C.6 Selection of the Best Trained Checkpoints

As stated in META-DATASET [10], actual early stopping is not performed, in that training is not stopped early according to validation performance. Instead, checkpoints are recorded every 500^{th} update during training, and validation error is saved at these times. The checkpoint corresponding to the least validation error is chosen as the best one. This procedure is used for selecting the best-trained model checkpoints in all the experiments.

C.7 Testing Details

For the few shot classification experiment, we use the variable "N-way K-shot" setup as in META-DATASET [10] for constructing test episodes. We call this 'episodic testing'. Testing was done on all the variants using the models trained on 4 training setups discussed in Section C.5. For joint object segmentation and few shot classification experiment, we used all the query images from {All, Unseen, Seen} variants. Refer Section B.2. We call this the 'non-episodic' setup. Few shot classification and joint object segmentation and few shot classification testing share the same trained model checkpoints. They only differ in the type of testing. The former uses episodic whereas the latter uses non-episodic testing.

For real-world testing, we have setup an environment where a Fetch mobile manipulator is facing a desk containing various real-world objects mentioned in Section 11. We have 8 setups each containing 4 different objects. Fetch takes an image containing the objects on the desk then [28] is used for object segmentation. The object masks are then used to crop the detected objects. The cropped object images become the query set whereas the support set of 198 real classes forms the support set of the new real-world test set. This new real-world test set is tested using the model trained using training setup (b) (refer Section C.5).

C.8 Experimental Hardware

All experiments were conducted on two NVIDIA RTX A5000 24GB GPUs. A Fetch mobile manipulator[2] was used for real-world testing.

C.9 Changes introduced in META-DATASET [10] code

All the images are first converted into tfrecords format. The default tfrecords are a collection of example protocol buffer[3] strings. Each example string consists of two elements, (i) image and (ii) label in bytes format. Meta-Dataset leverages data from 10 different datasets: ILSVRC-2012 (ImageNet [20]), Omniglot [19], Aircraft [21], CUB-200-2011 (Birds [22]), Describable Textures [23], Quick Draw [24], Fungi [25], VGG Flower [26], Traffic Signs [27] and MSCOCO [2]. As we can see, unlike the FewSOL dataset, these data are not divided into support and query sets specifically. Hence, we added a new key to each example string namely **set** which could take one value from {*support, query, ''*}. '' denotes that the particular example can belong to either support or query. This ensures the existing 10 datasets of META-DATASET [10] are compatible with the new protocol buffer structure. The default META-DATASET [10] pipeline shuffles and randomly selects a set of example strings which are then again randomly divided into support and query sets. This is reasonable as there is no bifurcation of support and query sets in the existing META-DATASET [10] datasets. But FewSOL dataset has a clear distinction between support and query and thus a need to select support and query data from the respective category was necessary. For this purpose, a perform_filtration feature is introduced. If set to boolean 'True', the pipeline filters the support and query sets so that the resulting data belongs to the respective categories. If set to boolean 'False', the standard META-DATASET [10] data pipeline is run.

Figure 10: Fetch mobile manipulator facing the objects from Set-1 (Figure 11a) on a table

[2] https://fetchrobotics.com/fetch-mobile-manipulator
[3] https://developers.google.com/protocol-buffers

D Qualitative Results in the Real World

In this experiment, our goal is to build a few-shot classification model that works the best on real-world perception systems. So we train CTX+SimCLR [18] with all real and synthetic data from the FewSOL dataset and then test the trained model in our lab. Cluttered support sets are used for training. This setup is selected due to the superior performance of CTX+SimCLR [18] with cluttered support set training during joint object segmentation and few shot classification experiment. Figure 10 shows fetch mobile manipulator facing objects from Set-1 (Figure 11a) on a table. RGB-D images are collected from a Fetch mobile manipulator and [28] is used for object segmentation. We tested on 32 objects with 4 objects in an image (refer Figure 11).

D.1 Real World Objects

(a) Set-1; mustard_bottle, water_bottle, jello_box, soup_can

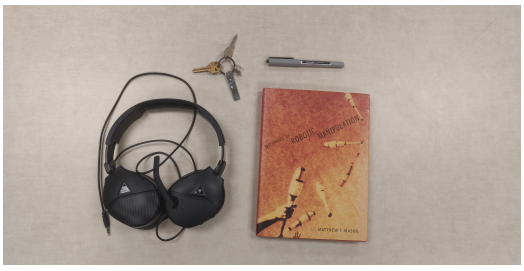
(b) Set-2; soft_scrub_cleanser_bottle, tennis_ball, ball, cracker_box

(c) Set-3; cup, jello_box, meat_can, clock

(d) Set-4; tuna_can, air_duster_can, marker, knife

(e) Set-5; keyboard, game_controller, hand_sanitizer, mouse

(f) Set-6; wood_block, folder, sticky_notes, stapler

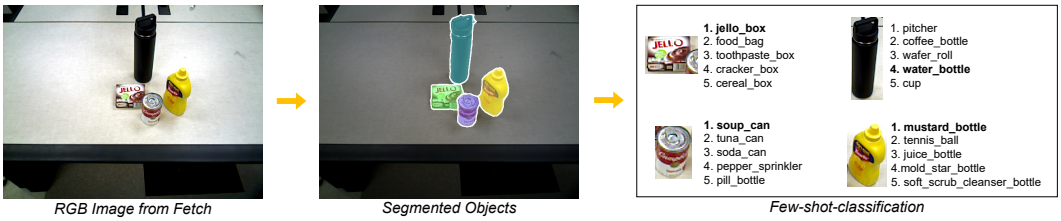


(g) Set-7; key, pen, book, headphone

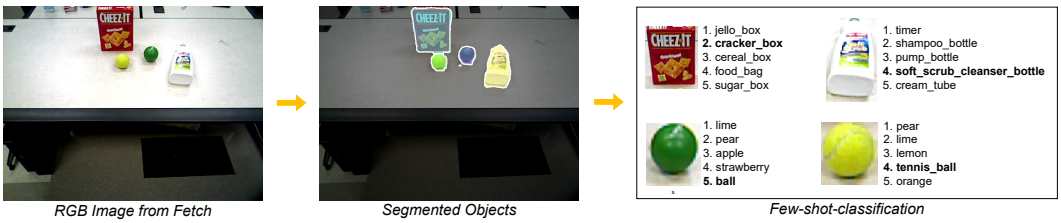
(h) Set-8; mug, charger, cellphone, spoon

Figure 11: 8 sets, each containing 4 different real world objects

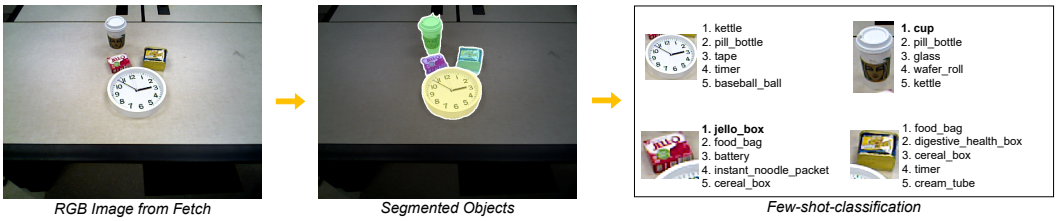
D.2 Joint Object Segmentation and Few-Shot Classification



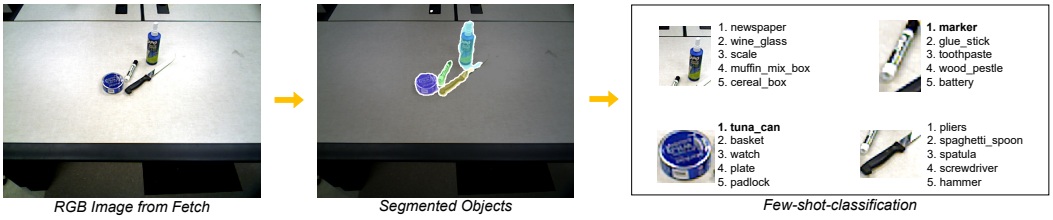
(a) Set-1



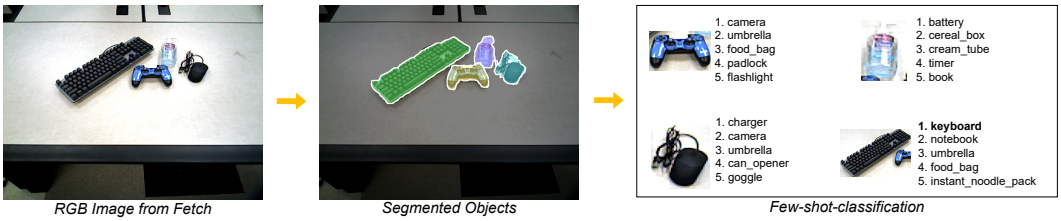
(b) Set-2



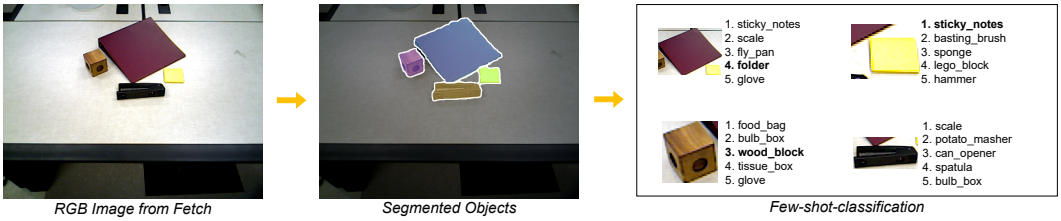
(c) Set-3



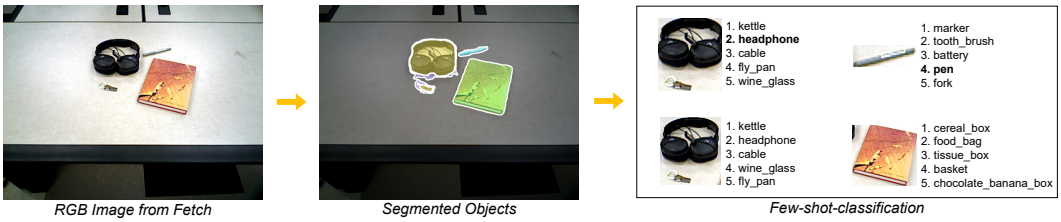
(d) Set-4



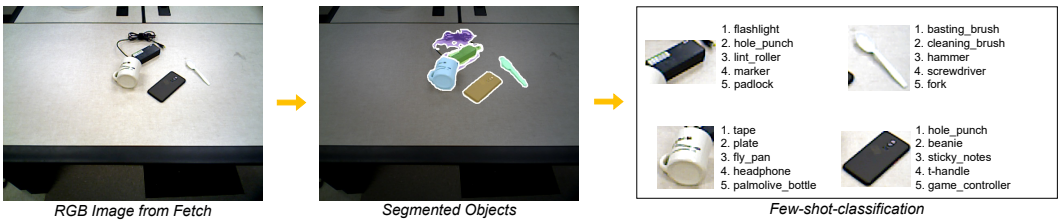
(e) Set-5



(f) Set-6



(g) Set-7

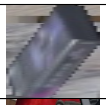
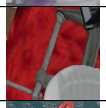
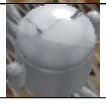


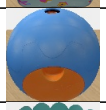
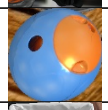
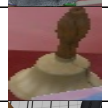
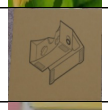
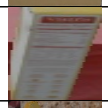
(h) Set-8

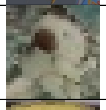
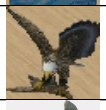
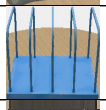
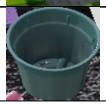
Figure 12: Joint object segmentation and few shot classification on real world objects shown in Figure:11

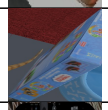
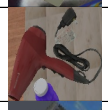
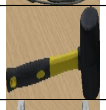
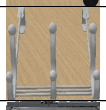
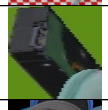
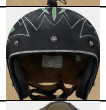
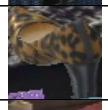
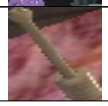
E Lists of Object Classes

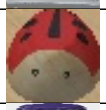
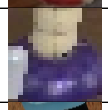
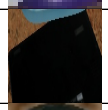
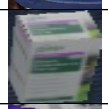
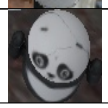
E.1 Synthetic Data

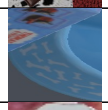
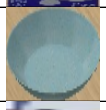
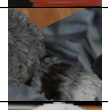
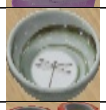
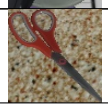
#	Class	#Clean Images	#Cluttered Images	Clean Sample	Cluttered Sample
1	CD	36	15806		
2	C_clamp	9	3745		
3	airplane	9	3704		
4	alarm_clock	9	3985		
5	android	9	3852		

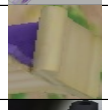
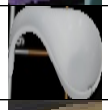
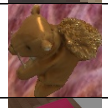
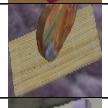
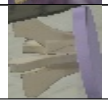
6	backpack	36	13118		
7	ball	9	3684		
8	basket	72	28429		
9	bear	18	6890		
10	bell	9	3549		
11	boot	18	7559		
12	bottle	9	4058		
13	bowl	171	73417		
14	bulb_box	9	4089		
15	butter_dish	9	3847		
16	cable	27	11673		
17	cake_pan	9	4194		
18	can_opener	27	12493		
19	candy_bar	9	3796		
20	cardboard_box	9	3887		
21	cereal_box	18	7865		
22	chair	9	4208		
23	coffee_box	18	8173		

24	coffee_can	27	12128		
25	coffee_marker	9	3253		
26	cooker	9	3360		
27	cow	27	11076		
28	cream_box	9	4382		
29	cream_tube	18	7924		
30	cup	18	8231		
31	dinosaur	54	24143		
32	dog	18	7674		
33	doll	9	4037		
34	dragon	9	3780		
35	drink_box	45	18692		
36	eagle	9	3895		
37	elephant	18	7422		
38	file_sorter	9	3947		
39	flash_drive	9	3086		
40	flashlight	9	3990		
41	flower_pot	90	40153		

42	flower_stand	9	3907		
43	fly_pan	9	3847		
44	food_can	9	3943		
45	game_card_box	63	27039		
46	game_controller	27	11991		
47	game_tape	27	11666		
48	garden_swing	9	4385		
49	gift_box	9	4155		
50	hair_dryer	9	3869		
51	hair_straightener	18	7248		
52	hammer	9	3836		
53	hanger	9	3956		
54	hard_drive	9	3924		
55	hat	54	18716		
56	headphone	9	4212		
57	helmet	9	3425		
58	high_heel_shoe	18	7921		
59	honey_dipper	9	3771		

60	horse	9	4174		
61	jug	9	3925		
62	kettle	9	4131		
63	keyboard	36	14118		
64	ladybug	9	3377		
65	lamp	9	3942		
66	laptop	18	6163		
67	lens	9	3994		
68	letter_holder	9	4026		
69	lion	9	4181		
70	lunch_bag	81	34509		
71	lunch_box	9	3910		
72	medicine_box	9	3951		
73	milk_bottle	9	4128		
74	mouse	36	16515		
75	mouse_pad	9	3812		
76	mug	36	16705		
77	panda	9	4030		

78	paper_roll	9	4022		
79	pencil_case	27	12198		
80	pencil_holder	9	3887		
81	pet_dish	36	15117		
82	pill_bottle	90	39322		
83	pill_box	9	3798		
84	plate	81	34261		
85	portable_drive	18	8180		
86	pot	18	8250		
87	product_box	216	95600		
88	racoon	9	3875		
89	ramekin	18	7634		
90	refrigerator	9	4049		
91	rhino	9	4058		
92	rubber_band	9	3500		
93	sanitary_pads	9	3511		
94	saucer	9	3992		
95	scissors	9	3586		

96	screwdriver	9	3949		
97	shark	18	7312		
98	sheep	9	4093		
99	shoe	117	50700		
100	sink	9	4064		
101	slipper	63	28479		
102	smartphone_printer	9	4028		
103	snack_box	126	56524		
104	sofa	9	4060		
105	spatula	9	3917		
106	speaker	18	8103		
107	sponge	9	3805		
108	spoon	9	4079		
109	sprinkler	9	3846		
110	squirrel	9	3861		
111	storage_box	18	6472		
112	sushi_mat	9	3729		
113	table	9	3980		

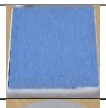
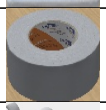
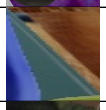
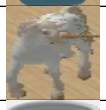
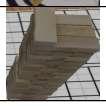
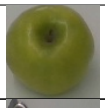
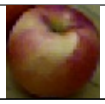
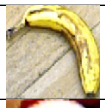
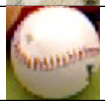
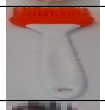
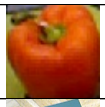
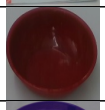
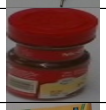
114	tablet	9	3742		
115	tape	18	8108		
116	teapot	9	4058		
117	toaster	18	7034		
118	towel	180	76769		
119	trash_bin	9	3665		
120	tray	27	11222		
121	unicon	9	4047		
122	utensil_holder	9	3889		
123	whale	9	3413		
124	wood_box	9	3778		
125	wood_tower	9	4034		

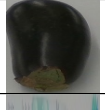
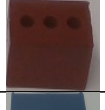
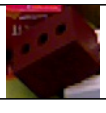
Table 6: 125 object classes of the 330 3D models from the Google Scanned Objects [9]

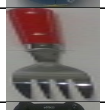
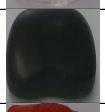
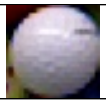
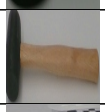
E.2 Real-World Data

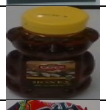
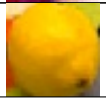
#	Class	#Our Images	#OCID Images	Our Sample	OCID Sample
1	adapter_cable	9	0		-
2	air_duster_can	9	0		-
3	air_purifying_bag	9	0		-
4	aluminum_foil_box	9	0		-

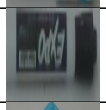
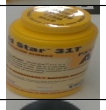
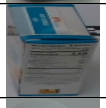
5	apple	27	68		
6	arm_filter_holder	9	0		-
7	avocado	9	0		-
8	baking_tray	27	0		-
9	ball	36	112		
10	ball_of_wool	9	0		-
11	banana	18	54		
12	baseball_ball	18	34		
13	basket	9	0		-
14	basting_brush	9	0		-
15	battery	9	0		-
16	beanie	18	0		-
17	bell_pepper	18	12		
18	binder	18	26		
19	book	9	0		-
20	bowl	45	12		
21	bucket	9	0		-
22	bulb_box	18	36		

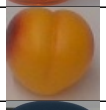
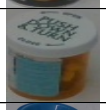
23	cabbage	9	0		-
24	cable	27	0		-
25	camera	9	0		-
26	can_opener	9	0		-
27	carrot	9	0		-
28	carrying_case	9	0		-
29	cellphone	18	0		-
30	cereal_box	36	252		
31	charger	9	0		-
32	chili_powder_bottle	9	0		-
33	chocolate_banana_box	9	38		
34	clamp	18	14		
35	cleaning_brush	18	0		-
36	cling_wrap_box	9	0		-
37	clock	9	0		-
38	coffee_bottle	9	0		-
39	coffee_can	18	18		
40	comb	18	0		-

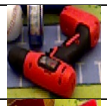
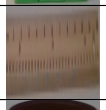
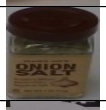
41	cracker_box	18	22		
42	cream_jar	9	0		-
43	cream_tube	27	0		-
44	cucumber	9	0		-
45	cup	27	0		-
46	digestive_health_box	9	0		-
47	donut_sticks_box	9	0		-
48	drano_dual_force_bottle	9	0		-
49	egg	9	0		-
50	egg_box	9	0		-
51	eggplant	9	0		-
52	electric_toothbrush	9	0		-
53	fig_bar_box	9	0		-
54	flash_drive	9	0		-
55	flashlight	18	18		
56	fly_pan	9	0		-
57	foam_brick	9	18		
58	folder	9	0		-

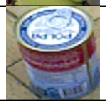
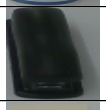
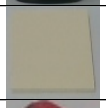
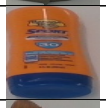
59	food_bag	27	102		
60	food_storage_container	27	0		-
61	football	9	18		
62	fork	45	0		-
63	game_controller	9	0		-
64	garlic	9	0		-
65	ginger	9	0		-
66	glass	27	0		-
67	glasses_box	9	0		-
68	glove	18	0		-
69	glue_bottle	18	0		-
70	glue_stick	27	38		
71	goggle	27	0		-
72	golf_ball	9	18		
73	goo_gone_bottle	9	0		-
74	hair_brush	9	0		-
75	hammer	27	0		-
76	hand_sanitizer	9	0		-

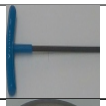
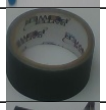
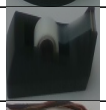
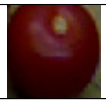
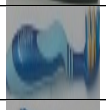
77	handy_pot	27	0		-
78	headphone	27	0		-
79	hole_punch	9	0		-
80	honey_bottle	9	0		-
81	instant_noodle_packet	9	26		
82	jar	9	0		-
83	jello_box	18	52		
84	juice_bottle	18	0		-
85	juice_pouch	9	0		-
86	kettle	9	0		-
87	key	9	0		-
88	keyboard	18	48		
89	kiwi	9	0		-
90	knife	54	0		-
91	label_marker	9	0		-
92	ladle	36	0		-
93	lego_block	18	22		
94	lemon	27	40		

95	lighter	9	0		-
96	lime	18	21		
97	lint_roller	9	0		-
98	marker	27	104		
99	mask	9	0		-
100	meat_can	9	0		-
101	milk_box	18	0		-
102	milk_frothing_pitcher	9	0		-
103	milk_jug	9	0		-
104	mms_package	9	0		-
105	mold_star_bottle	9	0		-
106	mouse	18	0		-
107	muffin_mix_box	9	0		-
108	mug	45	44		
109	mustard_bottle	9	16		
110	newspaper	9	0		-
111	notebook	9	0		-
112	onion	18	0		-

113	orange	18	68		
114	oven_glove	9	0		-
115	padlock	9	0		-
116	palmolive_bottle	9	0		-
117	peach	9	48		
118	peanut_jar	9	0		-
119	pear	18	31		
120	peeler	9	0		-
121	pen	9	0		-
122	pencil	9	0		-
123	pepper_sprinkler	9	0		-
124	pill_bottle	9	0		-
125	pitcher	9	14		
126	pizza_cutter	9	0		-
127	plant_vase	9	0		-
128	plate	45	0		-
129	pliers	18	0		-
130	potato	9	46		

131	potato_masher	9	0		-
132	potato_starch_package	9	0		-
133	power_drill	27	16		
134	pringles_can	9	24		
135	protein_bar	9	0		-
136	pump_bottle	9	0		-
137	remote_controller	18	0		-
138	rinse_aid_bottle	18	0		-
139	root_beer_bottle	9	0		-
140	rubiks_cube	9	16		
141	ruler	9	0		-
142	salt_bottle	18	0		-
143	salt_can	9	0		-
144	scale	9	0		-
145	scissors	27	0		-
146	scoop	9	0		-
147	screwdriver	18	0		-
148	shampoo_bottle	27	92		

149	shaver	9	0		-
150	shoe	9	0		-
151	slipper	18	0		-
152	soda_can	45	76		
153	soft_scrub_cleanser_bottle	9	16		
154	soup_can	36	60		
155	spaghetti_spoon	9	0		-
156	spatula	36	0		-
157	sponge	18	80		
158	spoon	36	0		-
159	spray_bottle	36	0		-
160	stapler	9	28		
161	sticky_notes	9	0		-
162	strawberry	9	0		-
163	sugar_box	9	20		
164	sugar_package	9	0		-
165	sunscreen_bottle	9	0		-
166	sweet_potato	9	0		-

167	t-handle	9	0		-
168	tape	18	0		-
169	tape_dispenser	18	0		-
170	tape_measure	27	0		-
171	tea_tin	9	0		-
172	tennis_ball	9	14		
173	thermometer_box	9	0		-
174	timer	9	22		
175	tissue_box	18	128		
176	tofu_package	18	0		-
177	tomato	18	20		
178	tong	27	0		-
179	toothbrush	9	0		-
180	toothpaste	36	38		
181	toothpaste_box	9	0		-
182	towel	36	60		
183	travel_pillow	9	0		-
184	trowel	9	0		-

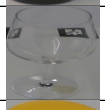
185	tuna_can	27	54		
186	umbrella	18	0		-
187	wafer_roll_can	18	0		-
188	watch	18	0		-
189	water_bottle	36	0		-
190	watermelon	9	0		-
191	wine_bottle	9	0		-
192	wine_glass	9	0		-
193	wipes_bottle	9	0		-
194	wood_block	18	24		
195	wood_box	9	22		
196	wood_pestle	9	0		-
197	wood_plate	9	0		-
198	wrench	9	0		-

Table 7: 198 object classes of the 336 objects we captured in our FewSOL dataset. 52 classes have examples from the OCID dataset [8]