Detection and Recognition of Dinning Table Objects Using RGB-D Camera for Robotic Applications

Jakob Baumgartner
Faculty of Electrical Engineering,
University of Ljubljana
Tržaška 25, 1000 Ljubljana, Slovenia
jakob.baumgartner@fe.uni-lj.si

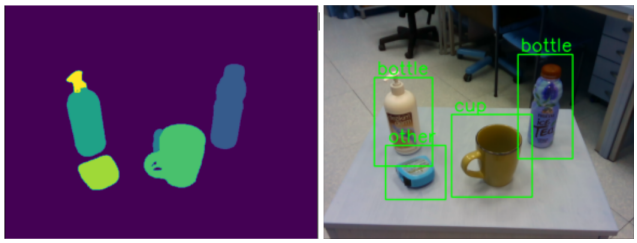

Figure 1. An example result of our segmented, classificator.

Abstract

In this paper we present a pipeline for 3D segmentation and classification of dining table objects. The created pipeline is intended to be used in real robot applications where we work in an unstructured environment. The algorithm created is able to correctly segment previously seen or unseen objects that might appear on a dining table and classify known common objects. The input to our pipeline is a single RGBD image. We depth and color information to segment every object on the table. Than we capture cutouts of detected objects and run classification using only RGB information. We test proposed pipeline on images captures using Intel RealSense d435i camera.

1. Introduction

RGBD (red, green, blue, depth) sensors [16] use a combination of traditional RGB (red, green, blue) cameras that capture colour information about an object and depth sensors that measure the distance between the sensor and the object. In the context of a dining table, RGBD sensors can be used to allow the robot to recognise and classify a variety of objects, such as dishes, glasses, utensils and food. By combining the colour information from the RGB cameras with the depth information from the depth sensors, the robot is able to create a detailed 3D model of the objects on the table so that it can recognise their shape, size and orientation.

RGBD object detection and classification algorithms have a number of advantages over traditional object detection algorithms that rely solely on colour information[1]. One advantage of using RGBD information is that the algorithm can better handle occlusion and clutter in the scene. Occlusion occurs when an object blocks the view of another object, and clutter refers to the presence of multiple objects in the scene that may be difficult to distinguish. By using depth information, the algorithm can better understand the spatial relationships between objects and more accurately detect and classify objects even when they are partially occluded or surrounded by clutter. Example of occlusion on a dining table is when a plate is partially hidden behind a glass. Another benefit of using RGBD information is that it enables the algorithm to better distinguish between objects that are similar in colour but different in shape or size. This can be particularly useful when there are multiple objects of the same colour in the scene, as the algorithm can use the depth information to distinguish them based on their shape and size. In our case it can help us distinguish between colourful tablecloths as a background and objects sitting on them.

However, the use of RGBD object recognition is still less common than the use of RGB object recognition. One reason for this is that RGB sensors are more widespread and less expensive than RGBD sensors. Another reason is the greater difficulty in labelling and especially segmenting such data sets due to the added dimensionality of data. As a result, there are far fewer and less comprehensive RGBD datasets than RGB datasets. For this reason, researchers are trying to train new algorithms on simulated datasets. Such datasets exhibit perfect segmentation, but there is an open problem in transferring the trained models from simulation to working with real data. While synthetic depth trained

algorithms usually transfers quite well to real world, simulated RGB images lack the textures and lighting conditions of the real world, so algorithms trained on synthetic RGB data do not transfer well to the real world. This has changed to some extent with the advent of commercial sensors such as Microsoft Kinect and Intel Real-Sense. In addition, algorithms that use RGBD data often require more computing power and resources to analyse the data, which can make them difficult to implement in real-time applications or on devices with limited resources.

In this paper, we aim to create an image processing pipeline that could be used on a robot in the hospitality industry to segment objects on a dining table. We can call this task partially structured because it contains a number of known and unknown objects. A typical input image contains both known objects, such as glasses and plates, and unknown objects, such as items left behind by customers in a restaurant. In order for our robot to clear the table after the guests, it must be able to recognise each object on the table so that it can remove it from the table.

2. Related Work

In recent years, there has been significant progress in the field of unseen object instance segmentation, which aims to detect and segment instances of objects that have not been seen during training. This is a challenging task that requires the ability to generalize to new objects while still accurately segmenting instances.

Several methods have been proposed to address this problem, including Mask R-CNN by He et al. (2017) [] [7], which extends the Faster R-CNN framework to predict instance masks in addition to object bounding boxes. Panoptic FPN by Kirillov et al. (2019) [10] also builds upon the FPN architecture to generate panoptic segmentation, including both object instances and stuff segments. YOLACT by Bolya et al. (2019) [4] uses a fully-convolutional model, that trades some of the segmentation performance for being able to run real-time. SolO-instance by Wang et al. (2020) [17] uses a set of prototypes to represent object instances, which are then refined in a manner that allows for instance-specific predictions. PointRend by Kirillov et al. (2020) [11] uses a neural network module that performs point-based segmentation predictions at adaptively selected locations based on an iterative subdivision algorithm. PointRend achieves significantly better segmentation results on several popular datasets compared to previous approaches. PolarMask by Xie et al. (2020) [21] introduces an anchor-box free and single-shot instance segmentation method that can be used as a mask prediction module for instance segmentation. PointGroup by Jiang et al. (2020) [9] presents a new end to-end bottom-up architecture, specifically focused on better grouping the points by exploring the void space between objects.

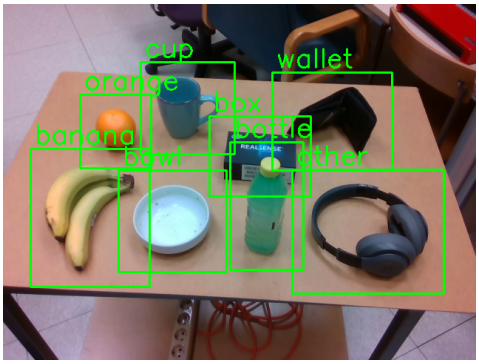

Figure 2. We have trained our classifier to seperate objects into one of the eight different categories.

The following methods focus specifically on the segmentation of unseen instances. In the absence of sufficiently large datasets for segmenting instances in cluttered environments, these approaches use synthetic data and generalise the algorithms so that they are applicable to the detection of real-world data [19]. These methods are most comparable to our method of choice, which is also trained on synthetic data. Being state-of-the-art, these methods offer similar performance to the method we chose for segmentation, while working at the same speed or slower. UCN by Xiang et al. (2021) [18] uses RGB-D feature embeddings from synthetic data to perform instance segmentation. MSMFormer by Lu et al. (2022) [12] uses a new transformer architecture that simulates the von Mises-Fisher (vMF) mean shift clustering algorithm.

These methods demonstrate the progress that has been made in unseen object instance segmentation and highlight the potential for future developments in this field.

3. Methods

Our pipeline consists of two separate components, as can be seen in the figure 3. Together, they allow us to solve the problem of instance segmentation and classification of objects on a dinner table in a real-world environment. The input to our pipeline is a single RGB-D image. Our method returns a list of segmented instances with segmentation masks and the positions of the objects. Our method also returns classifications for a few trained object categories.

The first component in our pipeline is an Unseen Object Instance Segmentator (UOIS) [20]. Due to the unstructured nature of our task, it is infeasible and impractical to model every possible object in our environment. For a robot to successfully manipulate objects on a dining table, it must be able to recognise and segment instances of previously seen and unseen objects [19]. This component takes as input RGB and point cloud data and provides segmentation masks and positions for each known and unknown object

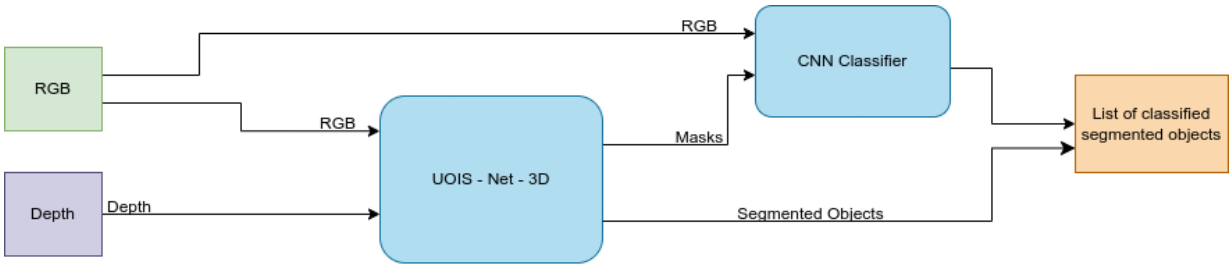

Figure 3. Full proposed pipeline.

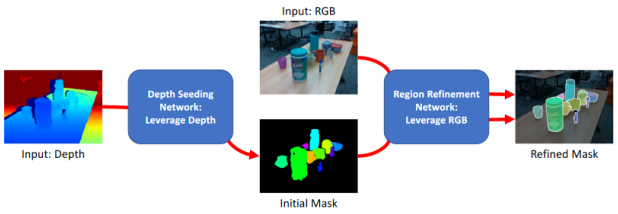

Figure 4. UOIS-Net-3D consists of two neural networks, that process Depth and RGB data seperately, to produce instance segmentation masks.

on the dining table. It is able to segment objects even if they are cluttered together or partially occluded. Before we can use this stage, we need to backproject Depth image back onto the point cloud data [5], for this we need to know the intrinsic matrix of the camera used.

The second component of our pipeline is a CNN classifier. UOIS provides instance segmentation masks. With these, we cut out individual objects and then perform a classification for the common objects on a dining table. These detections can be passed on to the robot so that it can separate different objects on the dinner table such as plates, cups, bottles and fruits from other objects and the rubbish left on the table.

3.1. Unseen Object Instance Segmentator

To divide cluttered objects on the dining table into individual instances, we use Unseen Object Instance Segmentator (UOIS-Net-3D) [20]. The algorithm takes as input a single RGBD image and outputs object instance segmentation masks for all previously seen or unseen objects. The output of the algorithm does not contain specific categories of objects, only background, table and object instance.

The algorithm consists of separately trained neural networks and multiple stages. A simple representation of the algorithm can be seen in the figure 4.

3.1.1 Method

Depth Seeding Network - DSN The first stage of the instance segmentation network takes as input an organised point cloud D with XYZ coordinates. The reason why we use depth data first is because of the better generalisation of neural networks trained on simulated depth data to real-world scenarios, compared to networks trained on synthetic RGB data. The organised point cloud is passed through the DSN network, which returns a segmentation mask for each pixel for the given image and the 3D offsets to the object centres. $V' \in \Re^{H \times W \times 3}$.

Initial Mask Processing Module - IMP In this phase of the algorithm, we apply some basic image techniques to the initial segmentation mask. First, we apply an opening operation (erosion+dilation) to each mask instance to remove salt/pepper noise. Then we apply a closing operation (dilation+erosion) to remove small holes in the mask. Finally, we remove minor unconnected components of each mask.

Region Refinement Network - RRN At this stage, we use another neural network to refine the segmentation mask obtained from the depth image with additional RGB information. The input of the RRN is a 4-channel cropped image consisting of a concatenated cropped RGB and a single object mask. We use a U-Net [14] network architecture. During the training process, we first train the DSN network and then use the output segmentations for RRN training. During training, we first use the segmentation masks before input to the RRN. This allows us to simulate the noise and errors that occur when using data from a real depth sensor but are not seen on the clear synthetic images. For pertrubation, we use augmentation techniques such as translation, rotation, addition, cutting, morphological operations and random elipses. For an example of a segmentation result, see image 1.

3.2. Classifier

The final component of our pipeline is an image classification network. We use segmentation masks obtained from UOIS to cut out each object from the RGB image and then use the ResNet [8] convolutional model to classify each object into one of the categories listed. The categories are: ”banana”, ”bottle”, ”bowl”, ”box”, ”cup”, ”orange”,

”other”, ”wallet”. The idea behind this is that a robot working in the hospitality industry can divide the objects into a section for laundry and a section for the bin / other objects.

ResNet (short for Residual Network) is a specific type of neural network that was introduced[8] in 2015 by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun in their paper “Deep Residual Learning for Image Recognition”. It is an innovative neural network that stacks residual blocks on top of each other to form a network.

ResNet is a deep neural network that has been widely used in computer vision tasks such as image classification, object detection, and semantic segmentation . It is a type of convolutional neural network (CNN) that uses residual connections to help with training deeper networks . Residual connections are used to skip over some layers in the network and allow the gradient to flow more easily during backpropagation . This helps to prevent the vanishing gradient problem that can occur when training very deep networks .

ResNet has achieved state-of-the-art results in a wide range of computer vision tasks, including image classification, object detection, and semantic segmentation. It has also been adapted for use in other domains, such as speech recognition and natural language processing.

For our pipeline we used a varient of ResNet with 152 layers.

4. Datasets

The ResNet classifier uses RGB data, while the UOIS segmentator uses RGBD data. Ideally, we would train both networks on the same dataset, using only RGB components to train ResNet. RGBD datasets are relatively rare, as they require specialised hardware and software to acquire, and take a lot of work to annotate. Consequently as far as we know, there are currently no available datasets with which we could train both of our networks. For this reason, we train our segmentator on an RGBD Tabletop Object Dataset (TOD) and ResNet on selected categories of a combination of ImageNet categories and added images that were web screpped.

4.1. Classifier

To train our classifier, we used selected categories of images from the ImageNet dataset. ImageNet consists of more than 20,000 categories, from which we selected seven categories of classified objects. The categories are: ”banana”, ”bottle”, ”bowl”, ”box”, ”cup”, ”orange”, ”wallet”, which represent everyday objects that we would expect to find on a dining table. We also mixed some of the images from the other categories into the a ”other” category. While ImageNet is an important database that is often used as a benchmark for classification tasks, it does contain some mislabeled images as well as images that are labeled correctly but are not a good representation of the category. A large portion of the dataset does not contain bounding boxes. Another problem with the dataset is the unequal representation of the categories, i.e., the unequal number of images per category. To prepare our dataset, we created a script that only collects images with bounding boxes added. The objects are cropped from the images using the information from the bounding boxes and resized to 200x200 pixels. To expand the collection of images, we created a partially automated web scrapper and collected hundreds more images for our categories. Each collected image was manually checked for suitability before being added to the collection. We wanted to make sure, that had equal representation ob categories as well as good examples of images. One exemption is the ”Other” category, we gave it more images, to make it more prominent. Exact number of object in each category can be seen in table 1. Before training we split dataset randomly into training (75%), validation (15%) and testing (10%) categories.

4.2. Segmentator

The segmentator method was trained on a synthetic dataset called the Tabletop Object Dataset (TOD). The dataset consists of 40,000 synthetic scenes. Each scene consists of a SUNCG [15] home environment. In this environment is a ShapeNet [6] model of a table with 5 to 25 ShapeNet [6] models of objects randomly placed or stacked on the table. For each object, the constalation dataset contains seven different views. Due to the limitations of the simulator used (PyBullet), RGB textures do not look photorealistic. The depth data, while clear compared to data from an actual commercial sensor, is closer to a realistic representation.A synthetic dataset was used as there is no dataset collected and annotated in the real world that meets the requirements for training such an application.

Category	ImageNet	Scrapped	Total
Banana	666	100	766
Bottle	721	81	802
Bowl	585	181	766
Box	508	0	508
Cup	618	299	917
Orange	680	80	760
Other	1119	0	1119
Wallet	581	0	581

Table 1. Data representation in dataset used to train classifier.

5. Experiments

In order for our pipeline to be useful for robotic applications, it must be able to work with a high success rate in each part of the pipeline and execute at a reasonable speed.

In the experimental part of this article, we therefore focus on testing parts of the pipeline individually and in conjunction with others.

To perform the experiments, we have created the pipeline of algorithms described. We use PyTorch [13] implementations [3] [2] of each of the methods described above.

5.1. Hardware

To deploy our algorithm, we would need adequate hardware. GPUs, are designed particularly for parallel processing and are well-suited for this task, as they can perform multiple mathematical operations simultaneously. We utilized cloud GPUs from Google Colab tool for testing, using the Tesla T4 with 16GB of memory and 13GB of system RAM.

6. Results

6.1. Time Trial

We have imagined an application that is intended to operate on a robot that moves in a changing environment. Consequently, our pipeline must be capable of operating in real-time, with as many frames per second as possible. In this part of the experiments, we separately tested our segmentator and classifier, and then together to determine how many images per second our pipeline is capable of processing.

To test the effect of the number of captured objects on the speed of algorithm execution, we captured 20 images with one, two, three, four, or five objects on the surface of the table. From table 6.2 and figure 5, we can see that both the segmentator and classifier are dependent on the number of objects in the image. As we perform classification for each segmented object sequentially, we can observe a linear dependence of the classification time on the number of objects.

Num.	Segmentator	Classificator	Pipeline
1-object	0.243	0.557	0.800
2-objects	0.294	1.176	1.470
3-objects	0.323	1.413	1.736
4-objects	0.346	1.871	2.217
5-objects	0.436	2.234	2.670

Table 2. Table of execution times of our pipeline in seconds [s].

6.2. Classification

To visualize results of our classifier we tested it on a testing part of our ImageNet+Scrapped dataset. We collected results in table 6.2 and in the confusion matrix 6. With achieved an average precision of 97%. The biggest classification error is with classification of bowl, that is 95% right.

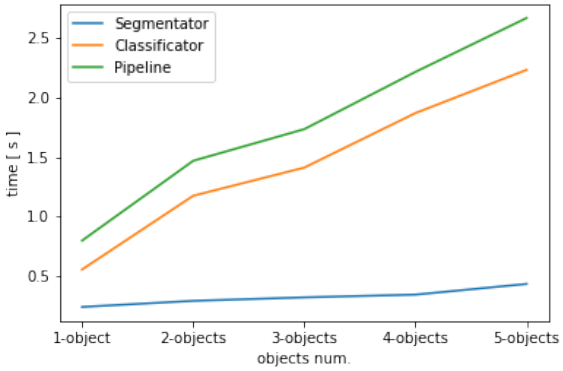

Figure 5. Plot displaying relation between number of objects and calculation time.

This is due to the fact, that classifier doesn't get information about size of the objects. As such it can sometimes mix cups and bowls, if the cup handle isn't clearly visible. What we can also see is that we picked a good group of objects to represent the "other" category. We correctly classified every object that didn't belong to any other category.

$$Recall = \frac{TP}{(TP + FN)} \tag{1}$$

$$Precision = \frac{TP}{(TP + FP)} \tag{2}$$

$$Accuracy = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{3}$$

$$F1Score = 2 * (\frac{Precision * Recall}{(Precision + Recall)}) \tag{4}$$

Object	Precision	Recall	Accuracy	F1 Score
Bowl	0.95	0.96	0.98	0.95
Cup	0.91	0.95	0.97	0.93
Other	0.99	0.99	1.0	0.99
Wallet	1.0	0.97	0.99	0.98
Box	0.96	1.0	0.99	0.98
Banana	1.0	0.95	0.99	0.97
Bottle	0.99	0.98	0.99	0.98
Orange	0.99	0.99	0.99	0.99
Average:	**0.97**	**0.97**	**0.99**	**0.97**

Table 3. Table of calculations of precision, recall, accuracy and f1-score for every category.

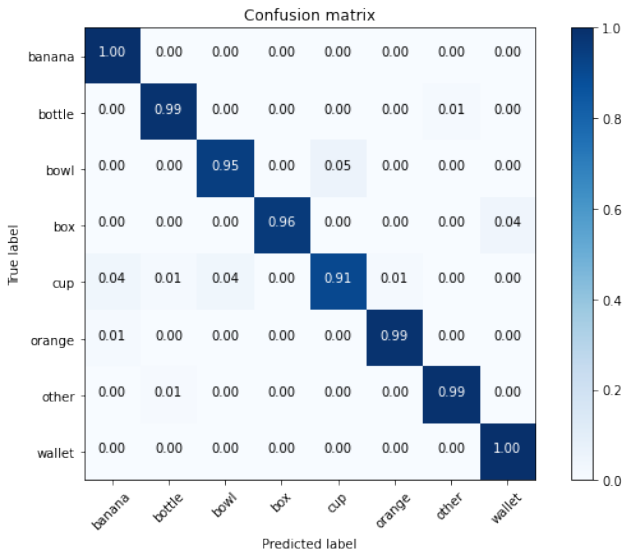

Figure 6. Plot of confusion matrix.

6.3. Segmentation

For our segmentator we used weights from already pretrained checkpoint. This is due to the lack of any better dataset than the one it was trained on in the original paper. Still we present test results on OCID and OSD datasets and for comparison results of two other state of the art segmentation models.

Used segmentation method is supposed to be object invariant, that is to be able to segment previously unseen objects. In our testing this appeared to be true. It appears however that the method is also sensitive to the angle at which the objects are captured. If the image was captured too horizontally, the model wasn't able to segment objects. It also struggled with small and transparent objects. This is mostly due to the limitations of commercial depth cameras, as they have too low resolution for detection of small objects and integrated algorithms make mistakes in depth estimations of transparent surfaces. Probably as a consequence of limited training dataset objects and generalization, it can mistakenly categorize complex objects as multiple different segments (image 7). Segmentator is also very sensitive to over or under exposure and they way scene is lighted.

Method	Precision	Recall	F-score
UOIS-Net-3D	86.5	86.6	86.4
Mask R-CNN	79.2	78.6	78.0
PointGroup	81.6	80.1	80.1

Table 4. Table of UOIS segmentation method compared against state of the art methods, we are using OCID dataset and overlap measure.

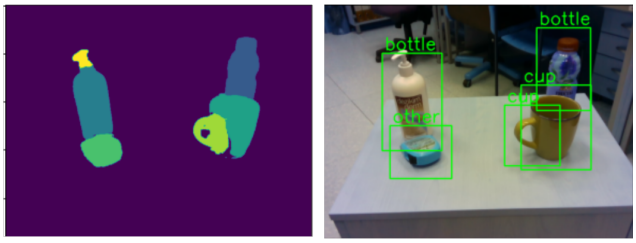

Figure 7. Image of bad segmentation.

Method	Precision	Recall	F-score
UOIS-Net-3D	85.7	82.5	83.3
Mask R-CNN	74.0	74.6	74.1
PointGroup	79.5	78.2	78.8

Table 5. Table of UOIS segmentation method compared against state of the art methods, we are using OSD dataset and overlap measure.

7. Conclusion

We developed a pipeline that segments all objects on the table surface based on information from an RGBD camera and then classifies the objects into known categories or remaining objects.

After testing, we can conclude that such pipeline is not optimal for use on a real robot. The main problem is the use of a depth camera on which used algorith relies heavily. Because of that we cannot detect small objects (such as keys and cutlery) that appear on dining surfaces. In addition, the depth camera does not capture data correctly about transparent materials. As a result, we cannot detect glass cups either, which are common in catering. Finally, changing lighting has a strong influence on the depth detection, which limits use of algorithm in daylight.

We also trained a classifier on a combination of ImageNet and web-scraped photos. The classifier works nearly perfectly.

The algorithm as we developed it is therefore not very suitable for use on a real robot. A more suitable approach might be one that would segment and classify using RGB images and use depth images only for additional information to the robot when choosing how to grasp objects.

There are a few pros for using depth segmentation. Algorithm is resistant on variable backgrounds. Also when depth image is captured correctly it can give us good base for objects detection.

References

[1] *RGB-D Image Analysis and Processing*. Advances in Computer Vision and Pattern Recognition. Springer International Publishing, Cham, 2019.

[2] Resnet pytorch. https://pytorch.org/hub/pytorch_vision_resnet/, 2021.

[3] Unseen object instance segmentation for robotic environments. https://github.com/chrisdxie/uois, 2021.

[4] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. Yolact: Real-time instance segmentation. (arXiv:1904.02689), Oct 2019. arXiv:1904.02689 [cs].

[5] Gazi Erkan Bostanci, Nadia Kanwal, and Adrian Clark. Augmented reality applications for cultural heritage using kinect. *Human-centric Computing and Information Sciences*, 5:1–18, Jul 2015.

[6] Angel Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. Shapenet: An information-rich 3d model repository. Dec 2015.

[7] Michael Danielczuk, Matthew Matl, Saurabh Gupta, Andrew Li, Andrew Lee, Jeffrey Mahler, and Ken Goldberg. Segmenting unknown 3d objects from real depth images using mask r-cnn trained on synthetic data. (arXiv:1809.05825), Mar 2019. 19 citations (Semantic Scholar/arXiv) [2023-01-30] arXiv:1809.05825 [cs].

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. (arXiv:1512.03385), Dec 2015. arXiv:1512.03385 [cs].

[9] Li Jiang, Hengshuang Zhao, Shaoshuai Shi, Shu Liu, Chi-Wing Fu, and Jiaya Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. (arXiv:2004.01658), Apr 2020. 158 citations (Semantic Scholar/arXiv) [2023-01-28] arXiv:2004.01658 [cs].

[10] Alexander Kirillov, Ross Girshick, Kaiming He, and Piotr Dollar. Panoptic feature pyramid networks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 6392–6401, Long Beach, CA, USA, Jun 2019. IEEE.

[11] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. Pointrend: Image segmentation as rendering. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, page 9796–9805, Seattle, WA, USA, Jun 2020. IEEE.

[12] Yangxiao Lu, Yuqiao Chen, Nicholas Ruozzi, and Yu Xiang. Mean shift mask transformer for unseen object instance segmentation. (arXiv:2211.11679), Nov 2022. 0 citations (Semantic Scholar/arXiv) [2023-01-31] arXiv:2211.11679 [cs] version: 1.

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. (arXiv:1912.01703), Dec 2019. arXiv:1912.01703 [cs, stat].

[14] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[15] Shuran Song, Fisher Yu, Andy Zeng, Angel X. Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. (arXiv:1611.08974), Nov 2016. arXiv:1611.08974 [cs].

[16] Kyriaki A. Tychola, Ioannis Tsimperidis, and George A. Papakostas. On 3d reconstruction using rgb-d cameras. *Digital*, 2(33):401–421, Sep 2022.

[17] Xinlong Wang, Tao Kong, Chunhua Shen, Yuning Jiang, and Lei Li. Solo: Segmenting objects by locations. (arXiv:1912.04488), Jul 2020. arXiv:1912.04488 [cs].

[18] Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. (arXiv:2007.15157), Mar 2021. 47 citations (Semantic Scholar/arXiv) [2023-01-28] arXiv:2007.15157 [cs].

[19] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. (arXiv:1907.13236), Jul 2020. 58 citations (Semantic Scholar/arXiv) [2023-01-30] arXiv:1907.13236 [cs].

[20] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. (arXiv:2007.08073), Oct 2021. 48 citations (Semantic Scholar/arXiv) [2023-01-28] arXiv:2007.08073 [cs].

[21] Enze Xie, Peize Sun, Xiaoge Song, Wenhai Wang, Ding Liang, Chunhua Shen, and Ping Luo. Polarmask: Single shot instance segmentation with polar representation. (arXiv:1909.13226), Feb 2020.