Supplementary Materials for: RICE: Refining Instance Masks in Cluttered Environments with Graph Neural Networks

Christopher Xie[1] **Arsalan Mousavian**[2] **Yu Xiang**[2] **Dieter Fox**[1,2]
[1]University of Washington [2]NVIDIA
{chrisxie,fox}@cs.washington.edu {amousavian,yux}@nvidia.com

1 Sampling Operation Networks details

1.1 Architecture Details

Node Encoder The Node Encoder consists of three separate CNN encoders, which consume RGB features (output by Resnet50+FPN [1, 2]), a backprojected XYZ point cloud (from a depth map and known camera intrinsics), and the mask, respectively. Each CNN encoder has 3 blocks of 2 3x3 convolutions followed by a 2x2 max pooling for resolution reduction. Lastly, there is a 7^{th} convolution layer. Each convolution is immediately followed by a GroupNorm layer [3] and ReLU. The resulting features are 2x2 averaged pooled, flattened, then put through an MLP with 2 hidden layers of dimension 1024 and 512 and an output dimension of 128 (except for SplitNet, which directly consumes the fully convolutional output after the 7^{th} conv layer).

SplitNet SplitNet builds off of the multi-stream CNN encoders from the Node Encoder. Given the outputs after the 7^{th} conv layer of the Node Enocder, these are then concatenated and passed through another conv layer for fusion. Next, this fused output is passed to a U-Net [4] style decoder. Skip connections are fed from the multi-stream CNN encoders to the decoder. Essentially, the overall architecture (including the Node Encoder) is a multi-stream encoder-decoder UNet architecture. This is similar to Y-Net [5], except that it has three streams instead of two. Weights of the Node Encoder (multi-stream encoders) are shared with DeleteNet.

DeleteNet DeleteNet also builds off of the Node Encoder (off of the MLP outputs). In particular, for each node, it computes the difference between the Node Encoder output $\mathbf{v}_i - \mathbf{v}_{bg}$, where $\mathbf{v}_{bg}$ is the Node Encoder output of the background node. This difference is then passed through an MLP with 2 hidden layers of dimension 512 and output dimension of 1. The score is passed through a sigmoid to be in the range $[0, 1]$. Weights of the Node Encoder are shared with SplitNet.

1.2 Sampling a Split from SplitNet

We provide pseudocode of how to sample a split of mask $S \in \{0, 1\}^{h \times w}$ given the output of SplitNet, which is a pixel-dense probability map $p \in [0, 1]^{h \times w}$ in Algorithm 1. At a high level, we essentially compute a probability distribution over the contour of S by seeing which pixels on the contour is close to split-able boundaries given by p (more weight is given to split-able boundaries that are large components, since it is likely to find a path through that boundary). Then, start and end points are sampled and the lowest cost (where cost is 1 - p) is computed, scored and returned.

2 Segmentation Graph Scoring Network Details

A high-level illustration of SGS-Net can be found in Figure 4 of the main paper. The initial node features $\mathbf{v}_i^{(0)}$ are given by the Node Encoder, and we obtain initial edge features $\mathbf{e}_{ij}^{(0)}$ by running the Node Encoder on all neighboring union masks S_{ij}. Then, we run multiple Residual GraphNet Layers (RGLs). Our Residual GraphNet Layer is an adaptation of a GraphNet Layer [6] with residual

5th Conference on Robot Learning (CoRL 2021), London, UK.

Algorithm 1 Sampling a Split

Require: Segmentation mask $S \in \{0,1\}^{h \times w}$, SplitNet output $p \in [0,1]^{h \times w}$, boundary threshold ν.

1: Compute contour of S.
2: Threshold p by ν, and compute the connected components. Create an image $\tilde{p} \in \mathbb{Z}^{h \times w}$ where $\tilde{p}_i$ is the size (in pixels) of the component at p_i for pixel i.
3: Compute contour probabilities for each contour pixel by weighted average of $\tilde{p}$ with Gaussian weights.
4: Sample start and end points on contour from contour probabilities.
5: Compute highest probability path from start to end through p, resulting in trajectory $\tau = \{(u_t, v_t)\}_{t=1}^{L}$.
6: Compute score $s_\tau = \frac{1}{L_i} \sum_t p_i[u_t, v_t]$.
7: **return** τ, s_τ

connections. Our RGL first applies an edge update:

$$\mathbf{e}_{ij}^{(l+1)} = \mathbf{e}_{ij}^{(l)} + \phi_e^{(l)} \left(\mathbf{v}_i^{(l)}, \mathbf{v}_j^{(l)}, \mathbf{e}_{ij}^{(l)} \right), \tag{1}$$

where $\phi_e^{(l)}$ is an MLP, and l describes the layer depth. This is followed by a node update:

$$\mathcal{E}_i^{(l)} = \left\{ \phi_{v_1}^{(l)} \left(\mathbf{e}_{ij}^{(l+1)}, \mathbf{v}_j^{(l)} \right) : (i,j) \in E \right\} \tag{2}$$

$$\mathbf{v}_i^{(l+1)} = \mathbf{v}_i^{(l)} + \phi_{v_2}^{(l)} \left(\overline{\mathcal{E}_i^{(l)}}, \mathbf{v}_i^{(l)} \right), \tag{3}$$

where $\overline{A}$ is the mean of all elements in the set A, and $\phi_{v_1}^{(l)}, \phi_{v_2}^{(l)}$ are MLPs. We additionaly apply ReLUs after the residual connections. After passing through L levels of RGLs, we end up with the set of node and edge feature vectors $\mathcal{V} = \left\{ \mathbf{v}_i^{(L)} \right\}$, $\mathcal{E} = \left\{ \mathbf{e}_{ij}^{(L)} \right\}$. We pass these through an output layer that aggregates these features:

$$s_G = \sigma \left(\phi_o \left(\overline{\mathcal{V}}, \overline{\mathcal{E}} \right) \right) \in [0,1], \tag{4}$$

where ϕ_o is yet another MLP and s_G is the predicted graph score for segmentation graph G, and σ is the sigmoid function.

3 Implementation Details

Our Node Encoder is shared amongst all of the networks, including SplitNet, DeleteNet, and SGS-Net. We first jointly train the SO-Nets for 200k iterations, with one segmentation graph per network per iteration (the batch sizes is the number of instances in the segmentation graph) so that the Node Encoder contains useful information for both operations. Next, we hold the Node Encoder fixed while we train SGS-Net for 100k iterations.

To generate training examples for SGS-Net, we take an initial segmentation and perturb it with the four proposed sampling operations and compute their ground truth scores. However, we do not use the SO-Nets here, instead we perform the perturbation of the initial segmentation by randomly splitting masks with sampled lines, merging neighboring masks, deleting masks, and adding masks in the same fashion as Xie et al. [7]. Modality tuning is performed during training of the SO-Nets, and held fixed during SGS-Net training. We use the outputs of Xie et al. [8] as initial segmentations, but we can similarly use ground truth labels. To generate training examples for SplitNet, given an initial segmentation, we randomly choose neighboring masks to merge. Lastly, to generate training examples for DeleteNet, given an initial segmentation, we randomly add masks in the same fashion as Xie et al. [7].

All images have resolution $H = 480, W = 640$. For our networks, we crop and resize the image, depth, and masks to $h = w = 64$. All training procedures use Adam [9] with an initial learning rate of 1e-4. We use $K = 3$ sample tree expansion iterations with a branching factor $B = 3$. Max nodes and edges are set to $m_n = 100, m_e = 300$ during training, and $m_n = 350, m_e = 1750$ during

Algorithm 2 RICE

Require: Initial instance segmentation S, RGB image I, organized point cloud D.
Build G_S with NodeEncoder applied to I, D, S
Initialize $T = \{G_S\}$
for $k \in [K]$ **do**
  for $G \in T$.leaves() **do**
    for $b \in [B]$ **do**
      Randomly choose a sampling operation and apply it to G to get candidate graph G'
      Apply SGS-Net to obtain $s_{G'}, s_G$
      if $s_{G'} > s_G$ **then**
        Add G' to T as a child of G
      end if
      if T.exceeds_budget(m_n, m_e) **then**
        Return T
      end if
    end for
  end for
end for
return Highest scoring graph in T and/or contour uncertainties

inference. Undirected edges are handled by including both (i, j) and (j, i) as directed edges in the graph. For each segmentation graph, edges are connected between nodes if their corresponding masks are within 10 pixels in set distance. Additionally, when sampling a candidate graph, we first randomly choose a sampling operation, compute all possible perturbation scores (e.g. split scores s_τ for each mask), and randomly select 3 of these perturbations that have a score of 0.7 or higher. This gives the opportunity to explore more segmentations within the allotted budget. All experiments are trained and evaluated on a single NVIDIA RTX2080ti GPU.

4 Pseudocode for Building the Sample Tree

We provide pseudocode for building the sample tree in Algorithm 2.

5 Example Sample Tree

In Figure 4, we show an example of a sample tree with branch factor $B = 2$ and $K = 2$ expansion iterations. We also visualize the ground truth score ($.8F + .2F@.75$) and the predicted score from SGS-Net. Note that the SGS-Net scores improve as the graph node gets further away from the root. In this particular example, SGS-Net would return the bottom left graph, which is also the most accurate graph.

6 Additional Experimental Results

6.1 Modality Tuning

Inspired by Aytar et al. [12], we perform an ablation to study how to best generalize from our non-photorealistic dataset TOD to real-world data. Starting from a ResNet50 pre-trained on COCO [13], we ablate over tuning the `conv1` layer, the `conv2_1`, `conv2_2`, `conv2_3` bottleneck building blocks [1], or keeping ResNet fixed. The idea is that by fixing the rest of the layers, we can encourage ResNet to learn the high level representation it has learned on COCO, on our simulated dataset. Thus, our SO-Nets and SGS-Net will learn to consume this high-level representation to provide their predictions. Then, during inference in the real-world, we resort back to the pre-trained ResNet to extract that representation from real images. In Figure 1, we show the results of our experiment. Interestingly, only modality-tuning `conv1` leads a small dip in performance compared to no tuning, while the optimal tuning for our scenario is to tune `conv1` and `conv2_1`. For the rest of this section, all networks will have been trained with this optimal setting.

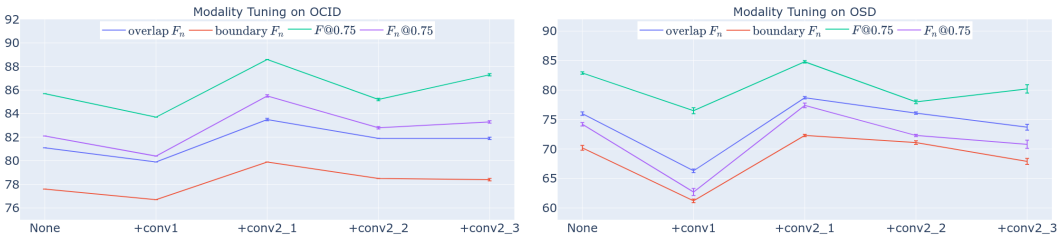


Figure 1: Modality tuning on OCID [10] and OSD [11] shows that tuning up to conv2_1 when training on simulated data generalizes best to real data. Note that standard deviation bars are shown, but are very tight and difficult to see.

Initial Instance Segmentation Method	**Split** only		**Merge** only	
	F@.75	F_n@0.75	F@.75	F_n@0.75
UCN [14]	92.0 (100%)	87.2 (64.7%)	88.8 (-23.1%)	87.5 (73.5%)
UOIS-Net-3D [8]	88.5 (101%)	84.6 (91.6%)	78.4 (2.1%)	77.4 (4.8%)
Mask R-CNN [15]	79.7 (60.3%)	75.8 (82.0%)	66.0 (0.0%)	64.6 (1.4%)
PointGroup [16]	82.4 (86.1%)	77.4 (87.7%)	61.2 (1.6%)	60.7 (2.1%)
	Delete only		**Add** only	
	F@.75	F_n@0.75	F@.75	F_n@0.75
UCN [14]	89.1 (-11.5%)	86.9 (55.9%)	89.5 (3.8%)	86.8 (52.9%)
UOIS-Net-3D [8]	78.2 (0.1%)	77.3 (3.6%)	78.3 (1.1%)	76.8 (-2.4%)
Mask R-CNN [15]	66.0 (0.0%)	64.8 (2.9%)	69.1 (13.6%)	67.0 (18.7%)
PointGroup [16]	61.0 (0.8%)	60.6 (1.5%)	64.0 (12.7%)	62.9 (13.3%)

Table 1: Sampling Operation Ablation on OCID [10]. We omit standard deviations as they are all less than 0.0005. We show results on F@.75 and F_n@.75. In parentheses, we show relative gain compared to the full RICE method (with all sampling operations).

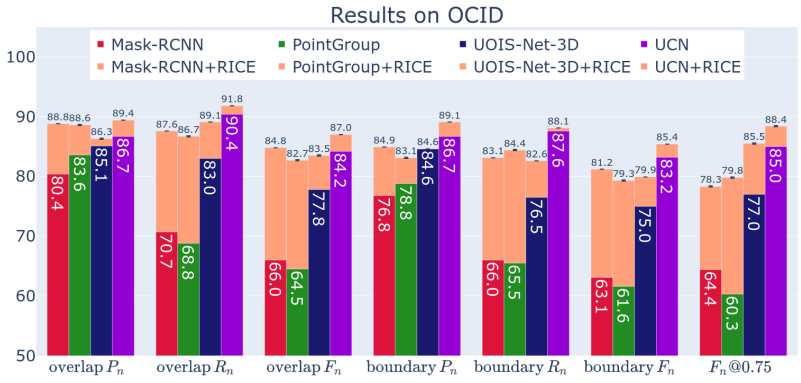
6.2 Evaluating the Usefulness of Each Sampling Operation

We provide an ablation experiment where we test the efficacy of each sampling operation. In particular, we test RICE while only using one sampling operation at a time, so that the sample tree is built only by a particular operation, e.g. splitting. This allows us to determine which of the sampling operations is most helpful in comparison to the initial instance segmentation method (e.g. UOIS-Net-3D [8], UCN [14]). We run these experiments on the larger OCID [10] dataset.

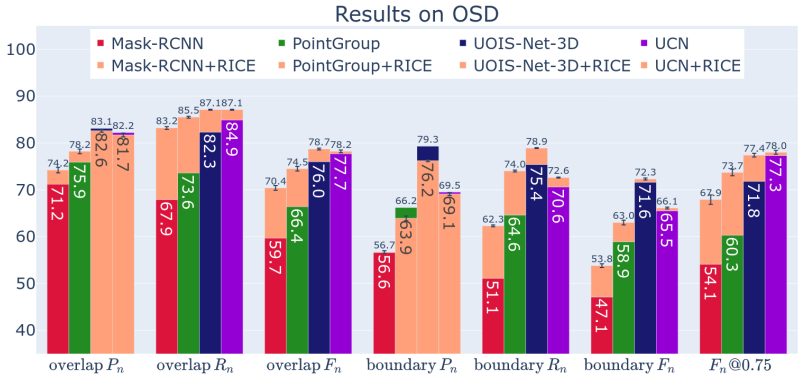
We test **Split** only, **Merge** only, and **Delete** only, and **Add** only. In Table 1, we show the results for F@.75 and F_n@.75 metrics using UCN [14], UOIS-Net-3D [8], Mask R-CNN [15], and PointGroup [16] as initial instance segmentation methods. We also show the percentage of increased performance with respect to the increased performance when using all sampling operations in parentheses. Clearly, we see that the split operation alone results in most of the performance gain compared to the full RICE method. This indicates that all four initial instance segmentation methods tend to under-segment, which is a common failure case in densely cluttered environments. UCN gains a lot from merging; the reason for this is that a common failure case from their pixel-clustering procedure is that the boundaries of the objects tend to be clustered as a separate object which results in over-segmentation. Merging can easily solve this issue. It turns out that the need to add and delete masks is relatively rare given the initial segmentation methods, which leads to smaller performance gains when considering these operations. UCN, Mask R-CNN and PointGroup benefit the most from adding, which suggests that they are either predicting false negatives.

6.3 Full Results on P/R/F

We provide full results on all Object Size Normalized (OSN) metrics for OCID [10] and OSD [11] in Figures 2a and 2b. For OCID, we can see increases in performance across all metrics in light orange. When RICE underperforms the initial segmentation, we color the bar underneath as light orange. On



(a) Results on OCID [10] using OSN metrics.



(b) Results on OSD [11] using OSN metrics.

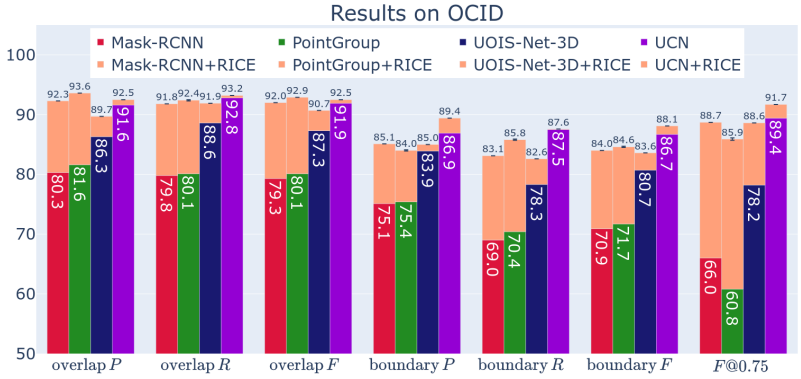
OSD, we can see that overlap P_n and boundary P_n are slightly worse than the initial method, while the recall R_n and F_n measures are still higher.

Figures 3a and 3b show Overlap and Boundary P/R/F measures on OCID and OSD. They show similar results to the OSN measures, but the numbers are higher. This suggests that common mistakes for SOTA methods more commonly occur in smaller objects (investigations of failure cases from Xie et al. [8], Xiang et al. [14] suggest this is the case). These numbers are directly comparable with previously published works.

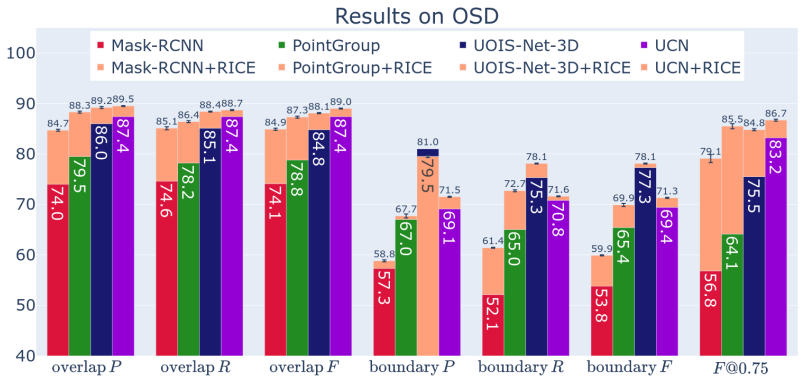
7 More Details for: Guiding a Manipulator with Contour Uncertainties for Efficient Scene Understanding

To extract contour uncertainties, we exploit the fact that RICE is a stochastic algorithm by design. Each leaf is essentially a sampled trajectory of states (segmentations) and actions (perturbations) from the initial segmentation S. These trajectories may have explored different parts of the state space (e.g. perturbed different object masks in the scene). We compute the standard deviation of the contours of each leaf graph in order to provide contour uncertainty estimates, which are shown in red in Figure 8 of the main paper and the Supplemental video. Additionally, we visualize the confident contours (which are present in each leaf graph) in green.

The contour uncertainties depict certain segmentations that not all of the trajectories explored. It reflects which objects RICE is not as confident about. If a mask S_i in the initial segmentation is split the same way in all leaf graphs, then RICE is confident that S_i should be split, and there will be no uncertainty. However, if S_i is only split in some of the leaf graphs, then RICE is not as confident



(a) Results on OCID [10] using Overlap and Boundary P/R/F metrics as defined by Xie et al. [7]. Note that the UCN [14] numbers are slightly different than in the published paper, due to the authors finding a bug in the reporting code. This is the case for Xie et al. [8] as well.



(b) Results on OSD [11] using Overlap and Boundary P/R/F metrics as defined by Xie et al. [7].

Algorithm 3 Guiding a Manipulator with RICE

1: **repeat**
2: Get S from initial instance segmentation method (e.g. UCN [14])
3: Run RICE to get best masks and contour uncertainty
4: **if** Contour Uncertainty is present **then**
5: Sample a grasp with Sundermeyer et al. [17] from the uncertain masks, and execute it
6: **end if**
7: **until** No Contour Uncertainty

about whether S_i truly represents more than 1 object, and an interaction is required to resolve such uncertainty. For example, in Trial 2 in the Supplemental video, the cup and the bowl it is on top of (far left) are constantly under-segmented together by UCN [14]. RICE splits it correctly each time, and there is no uncertainty about splitting that mask, as evidenced by the uncertainty contours.

We provide a short description of the grasping algorithm with pseudocode in Algorithm 3.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

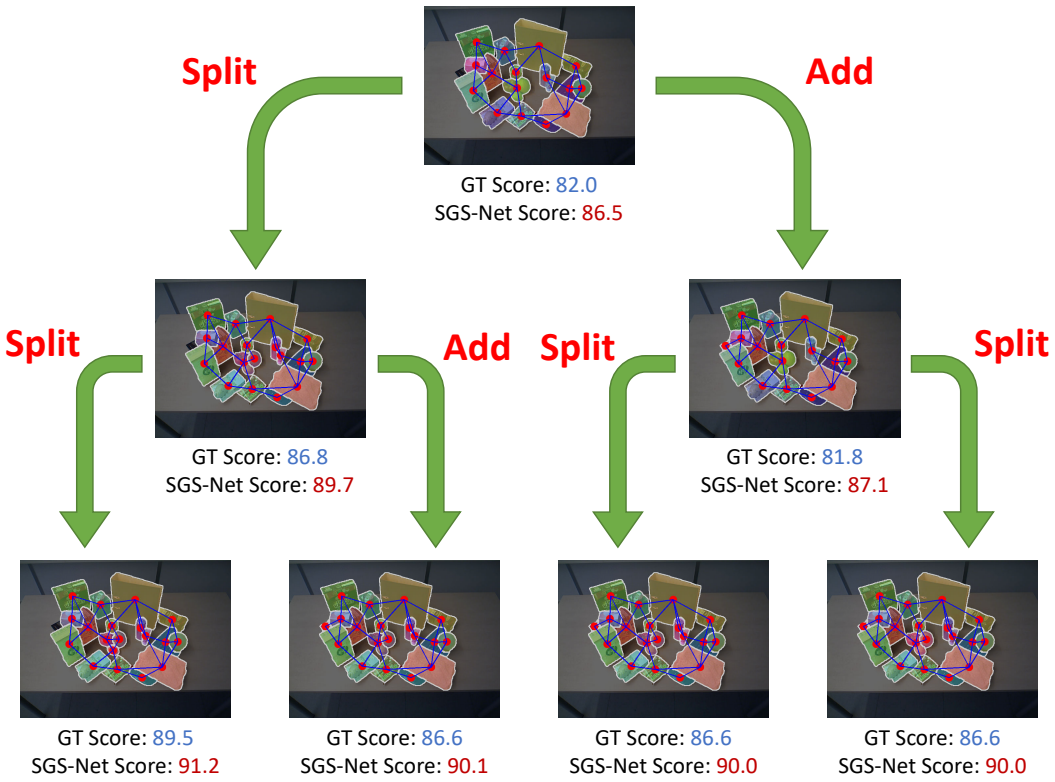


Figure 4: Example of a sample tree. Ground Truth and SGS-Net scores are shown, along with the chosen sampling operations. In this example, all leaves improve upon the initial segmentation graph, with the highest ranking graph also being the closest to the ground truth segmentation. Very similar splits and adds are investigated in the leaf trajectories.

[2] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[3] Y. Wu and K. He. Group normalization. In *European Conference on Computer Vision (ECCV)*, 2018.

[4] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, 2015.

[5] C. Xie, Y. Xiang, Z. Harchaoui, and D. Fox. Object discovery in videos as foreground motion clustering. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[6] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[7] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. The best of both modes: Separately leveraging rgb and depth for unseen object instance segmentation. In *Conference on Robot Learning (CoRL)*, 2019.

[8] C. Xie, Y. Xiang, A. Mousavian, and D. Fox. Unseen object instance segmentation for robotic environments. *IEEE Transactions on Robotics (T-RO)*, 2021.

[9] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, (ICLR)*, 2015.

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

[11] A. Richtsfeld, T. Mörwald, J. Prankl, M. Zillich, and M. Vincze. Segmentation of unknown objects in indoor environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012.

[12] Y. Aytar, L. Castrejon, C. Vondrick, H. Pirsiavash, and A. Torralba. Cross-modal scene networks. *IEEE transactions on pattern analysis and machine intelligence*, 40(10):2303–2314, 2017.

[13] 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 Computer Vision (ECCV)*, 2014.

[14] 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)*, 2020.

[15] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[16] L. Jiang, H. Zhao, S. Shi, S. Liu, C.-W. Fu, and J. Jia. Pointgroup: Dual-set point grouping for 3d instance segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[17] M. Sundermeyer, A. Mousavian, R. Triebel, and F. Dieter. Contact-graspnet: Efficient 6-dof grasp generation in clutteredscenes. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2021.