InSeGAN: A Generative Approach to Segmenting Identical Instances in Depth Images —Supplementary Material—

Anoop Cherian[1] Gonçalo Dias Pais[2*] Siddarth Jain[1] Tim K. Marks[1] Alan Sullivan[1]
[1]Mitsubishi Electric Research Labs (MERL), Cambridge, MA
[2]Instituto Superior Técnico, University of Lisbon, Portugal
[1]{cherian, sjain, tmarks, sullivan}@merl.com [2]goncalo.pais@tecnico.ulisboa.pt

Summary of Supplementary Material

In this supplementary material, we provide additional qualitative results (examples) and numerical results, as well as algorithmic and experimental details.

1. Explanations and insights into InSeGAN
2. How is the implicit template learned?
3. How to extend the framework for arbitrary number of instances n?
4. Results on the synthetic setting with $n = 10$ cones
5. Qualitative instance segmentation results on real data
6. More on the neural architectures, synthetic data generation, and real robotic data collection setups.
7. Additional qualitative instance segmentation results.

1. InSeGAN: Insights and Why it Works

A curious reader of our work might ask, *How does the network learn to disentangle the depth image into each instance poses and the implicit template? In particular, how does it learn to disentangle the pose of each instance from the depth image into a separate latent vector in $\hat{\mathbf{Z}}$? And, why does the network learn an implicit template model that represents a single instance, rather than multiple instances within a single template?* This is, we believe, because of the way the generator-discriminator pipeline is trained. For example, let us assume for a moment that a single latent noise vector $\mathbf{z}$ controls more than one (or in the extreme, all) of the instances in a depth image. As $\mathbf{z}$ is randomly sampled from a distribution, it is unlikely that only some of the vectors in $\mathbf{Z}$ (the collection of n $\mathbf{z}$ vectors used as input to the generator) would render the instances and some would not, given that aggregation of all the generated instances should match up to the number of instances in the input data—a requirement that the discriminator will eventually learn to verify in the generated images. Further, given that the object appearances are varied, it is perhaps easier for the generator to learn to render the appearance of a single instance than to capture the joint appearance distribution for all instances, which could be very large and diverse.

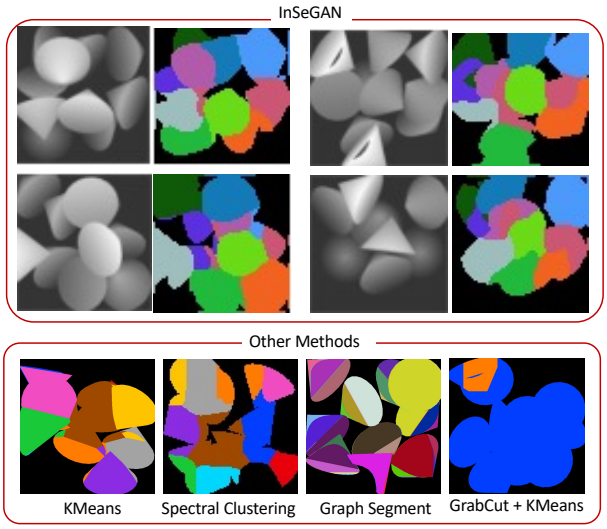


Figure 1. Qualitative results of InSeGAN segmentation on the synthetic 10-cones dataset. We also show example segmentations produced by other methods.

2. How is the Implicit Template Learned?

Note that the template is learned jointly with the rest of the modules. The teplate is implemented as a PyTorch weight tensor and is updated with the backpropagation gradients from the losses. Simply put, when training the setup, all the arrows in Figure 2 gets reversed, thus training the template along with all of the other weights in the network.

3. Using Arbitrary Number of Instances n?

It is straightforward to extend InSeGAN for an arbitrary number of instances n, which we do by using training images with varying numbers of instances. Assuming a max

*Work done as part of MERL internship.

of n instances in the training images, we have InSeGAN sample a random number ($\leq n$) of pose vectors at the input in Fig. 2 (paper). Further, we also add a simple module that predicts the number of instances in the rendered image, which is used to produce that many pose vectors. A loss is enforced that ensures the number of sampled pose vectors and the number of estimated pose vectors (by the instance pose encoder) are the same. The rest of the pipeline stays the same. At test time, the input depth image is passed through the instance pose encoder, alongside the number of estimated instances (by the additional module), and each of the produced instance poses are decoded individually to produce the segmentations. We implemented this variant of our scheme and found that the GAN successfully learns to match the new distribution, which is that of depth images with varied instance count and produces instance segmentations for arbitrary number of object instances. In Fig. 2, we show results on the Cone class when we vary the count between 4 and 9. On these data, we achieved 45.1% mIoU.

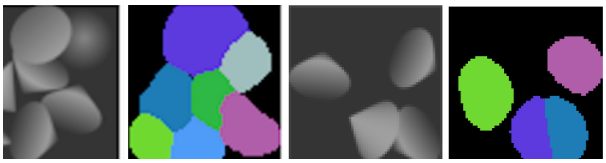


Figure 2. InSeGAN results when we use the same model to learn distributions of images with varying number of instnaces. The results show segmentation visualizations when we used 4–9 instances in each of the depth images.

4. Synthetic Setting with 10 Cones

As introduced in Figure 1 of the main paper, we also explored the scalability of InSeGAN to depth images with more than 5 instances. Similarly to how we produced the synthetic Insta-10 dataset with $n = 5$ instances in each category, we produced an additional dataset using $n = 10$ instances of cones, to explore how well our model handles the more difficult case of depth images with twice as many instances. As in the Insta-10 dataset, all 10 instances were randomly dropped into a bin in sequence using a physics simulator. Similar to each category in Insta-10, we created 10,000 depth images with 10 cones each, of which we used 100 for validation and 100 for testing. We did not use K-Means to select difficult examples for our test set in the 10-instance setting, because the increased number of cones means that every depth image in the set is cluttered and quite challenging. We trained our InSeGAN model with exactly the same setting and hyperparameters (except for the number of instances n). Qualitative results are provided in Figure 1. In Table 1, we quantitatively compare the performance of InSeGAN on this dataset.

Method	mIoU
KMeans	0.302
Spectral Clustering	0.324
Superpixels [9]	0.398
GrabCut+KMeans	0.021
InSeGAN	**0.501**

Table 1. Numerical comparison of InSeGAN vs. other methods on challenging dataset with $n = 10$ cones in each image.

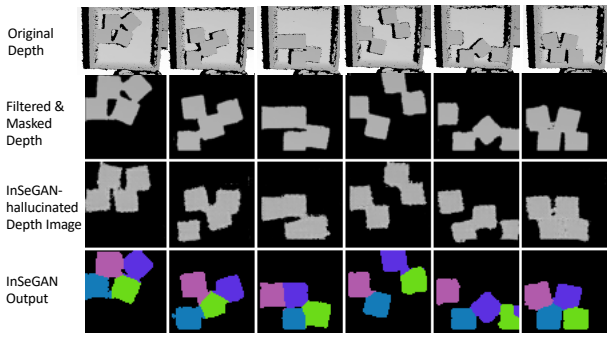


Figure 3. Qualitative results on instance segmentation of real data. We show the original depth images collected using a robot (first row), the output of a masking and filtering step we do to clean up the inputs (second row), the depth images hallucinated/rendered by InSeGAN (third row), and the segmentations (fourth row).

5. Qualitative Results on Real Data

In Figure 3, we show qualitative results of instance segmentation on real data. We also highlight the preprocessing steps we follow to apply InSeGAN to this dataset, which are necessary because the input depth images are very noisy (e.g., jitter/spurious noise in the depth sensor, and hole-filling that the sensor algorithm implicitly applies). These steps often alter the object shape, for example, merging two adjacent instances to appear as a single large object. To use these depth images in our setup, we first masked out the surrounding region (everything outside of the bin). This is possible because the bin is always at the same location. After applying the mask, we thresholded the z values in the depth image to only show z values greater than half the height of a block instances. This provided a relatively clean depth image, reducing the jitter and other artifacts (see Figure 3). Next, we applied InSeGAN to these preprocessed images. The qualitative and quantitative results (in the main paper) show that InSeGAN is very successful in segmentation on these images (85% mIoU).

6. Data Collection Setup

In this section, we detail our synthetic and real-world data collection setups.

6.1. Physics Simulator and Depth Image Generation

As described in the main paper, we use the NVIDIA PhysX physics simulator[1] to create our Insta-10 dataset. A screenshot of this simulator software setup is shown in Figure 5. Specifically, the simulation consists of a virtual bin of a suitable size and depth (depending on the size of the object) into which virtual instances (Cones in the figure, for example) are dropped sequentially from random locations above the bin. Next, an overhead simulator depth camera captures the depth image associated with the instances. A snapshot of the instance segmentation of the five objects is shown in the figure (right). The simulator automatically takes care of avoiding intersecting objects (because it is physically impossible) and accounts for occlusions. It takes approximately 2 seconds to generate one depth image using this setup with 5 identical object instances.

6.2. Robotic Collection of Real-World Depth Images

We first describe our robotic experiment system, then explain how we use it to collect more than 3,000 real-world depth images for testing our approach. Our experiments are carried out on a Fetch robot [10] equipped with a 7-degree-of-freedom (7-DOF) arm, and we use ROS [7] as our development system. The Fetch robotic arm is equipped with a stock two-fingered parallel gripper. Mounted above the box is a downward-pointing Intel RealSense Depth Camera (D435), which consists of depth sensors, RGB sensor, and infrared projector. The camera, which is attached to a Noga magnetic base, provides a depth stream output with resolution up to 1280×720 resolution of the scene with which the Fetch robot interacts. For trajectory planning, we use the Expansive Space Tree (EST) planner [6]. Note that during the experiments, human involvement is limited to switching on the robot, configuring the planner, and placing the objects in the box arbitrarily. Apart from this initialization, our robotic pipeline has no human involvement in the process of data collection.

The data collection setup is depicted in Fig. 4. The workspace is first set up with a box with plain background, and the box is fitted with a handle that is grasped by the Fetch robotic arm. Four identical wooden blocks are placed inside the box in random pose configurations. A single instance of a trial proceeds as follows: A depth image of the box is captured by the depth camera and recorded to a disk. The robot then initiates the trajectory planner, and the robotic arm executes a motion trajectory to tilt-shake the box randomly in the clockwise or anticlockwise direction such that the motion is collision free, then returns the box to its original location. The degree of tilt shake is also randomized (up to a specified maximum to prevent the blocks falling out of box). Multiple trials are executed in succession for the robot to autonomously record the dataset, with four cycles per minute.

[1] https://developer.nvidia.com/physx-sdk

Figure 4. Robotic data collection system used to acquire real-world depth images.

7. Network Architectures

In this section, we will detail the neural architectures of the three modules in InSeGAN: (i) the Encoder, (ii) the Discriminator, and (iii) the Generator.

Generator: In Fig. 6, we provide the detailed architecture of our InSeGAN Generator. It has five submodules: (i) A pose decoder, which takes n random noise vectors $\mathbf{z}_i \in \mathbb{R}^{128} \sim \mathrm{N}(0, \mathbf{I}_{128})$, where $n = 5$ in our setup, and produces 6-D vectors that are assumed to be axis-angle representations of rotations and translations [12] (three dimensions for rotation and three for translation). Each 6-D vector is then transformed into a rotation matrix and a translation vectors, to produce an element in the special Euclidean group $\big(\mathrm{SE}(3)\big)$. (ii) A 3D implicit template generation module, which takes a $4 \times 4 \times 4 \times 64$ dimensional tensor (representing an implicit 3D template of the object) as input, then up-samples in 3D using ResNet blocks and 3D instance normalization layers to produce a $16 \times 16 \times 16 \times 16$ feature maps. (iii) A spatial transformer network (STN) [2], which takes as input the 3D implicit template and the geometric transform for every instance, then transforms the template, resamples it, and produces a transformed feature map of the same size as its input. (iv) A single-instance feature generator module, which reshapes the transformed template fea-

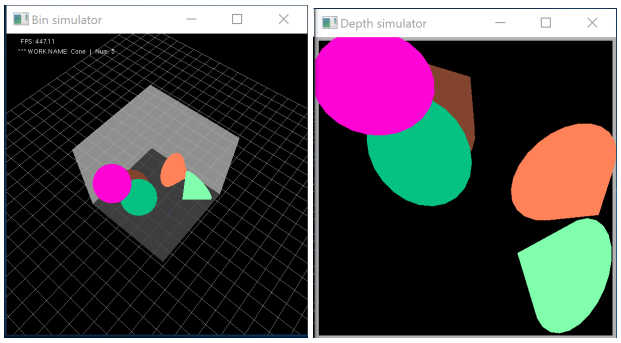


Figure 5. An illustration of the physics simulator that we use to render our synthetic dataset, Insta-10. *Left:* the simulated bin into which the identical objects (e.g., Cone) are dropped. *Right:* The ground-truth instance segmentation masks for each of the instances. We use the depth images associated with these instances for training InSeGAN, so that at inference time, segmentation masks are recovered. The ground-truth instance segmentation masks are not used for training—they are only used for testing our unsupervised method.

ture and produces single-instance 2D feature maps (each of size $16 \times 16 \times 128$). (v) A depth renderer module that takes an average pool over the n feature maps representing the n instances, and renders a multiple-instance depth image from the pooled feature map.

The 3D implicit template loosely follows the architecture of a HoloGAN [5], but differs in that we do not use any stochastic modules (via MLP) that were critical in their framework to produce stochastic components in the generated images (RGB images, in their case). We found that using noise vectors as in HoloGAN failed in our setup, causing us to lose the ability to disentangle instances.

Encoder and Discriminator: In Fig. 7, we show the neural network used in our Encoder and our Discriminator. They loosely follow similar architectures, except that the Discriminator takes a 64×64 depth image (either generated or from the real examples) as input and produces a scalar score, while the encoder takes a generated depth image and produces the n pose instance vectors as output. We use 128-D noise vectors when generating the images, and thus the Encoder is expected to produce 128-D features as output (one 128-D feature for each instance). Both the Discriminator and the Encoder use 2D convolutions, leaky ReLU activations, and 2D instance normalization [8] modules.

7.1. Implementation Details and Training Setup

Our InSeGAN modules are implemented in PyTorch. As alluded to above, we generate 224×224 depth images using our simulator; however, we use 64×64 images in our InSeGAN pipeline. To this end, each 224×224 image is rescaled to 64×64 and normalized using mean subtraction and normalization by the variance. For training, we use horizontal and vertical image flips for data augmentations. We do not use any other augmentation scheme.

7.2. Evaluation Details

For our evaluations, we use the mean IoU (mIoU) metric between the ground truth instance segments and the predicted segmentations. Specifically, for each ground truth segment, we find the predicted segment that is most overlapping with this segment, and compute their intersection-over-union (IoU); we then use every segment's IoU to compute the mean IoU over all segments.

Training: We train our modules for 1000 epochs using a single GPU; each epoch takes approximately 30 seconds on the $\sim$10,000 training samples for each object. We use the Adam optimizer, with a learning rate of 2×10^{-4}, and $\beta_1 = 0.5$. We use 128-D noise samples from a Normal distribution for the noise vectors, and a batch size of 128 samples.

8. Additional Ablative Studies

In this section, we extend the ablative studies presented in the main paper with additional results, and analyze and substantiate the importance of each choice in InSeGAN.

Is 3D Generator Important? An important choice that we made in InSeGAN is the use of a 3D generator instead of a 2D generator. For comparison, we use a standard 2D image-based generator typically used in conditional GANs [4]. Specifically, for the 2D generator, we replace the 3D modules in InSeGAN (i.e., the 3D implicit template, the pose encoder, and the STN) by 2D convolutions and upsampling layers, similar to those used in the encoder and the discriminator. We perform two experiments to analyze and substantiate our choice: (i) to evaluate the training stability and convergence, and (ii) to evaluate the performance of instance segmentation on the various objects. In Figs. 8, we plot the convergence of the 2D and 3D GANs on three objects from our Insta-10 dataset, namely Obj01, Cone, and Connector. We make three observations from these results: (i) 3D GAN is significantly faster than 2D GAN in convergence, (ii) 3D GAN is more stable, and (iii) 3D GAN leads to better mIoU for instance segmentation. In Table 1 of the main paper, we provide comparisons of the 3D and 2D GANs on all the objects in the Insta-10 dataset. Our results show that our 3D generator is significantly better than a 2D generator on a majority of the data classes.

Do We Need All Training Samples? In Fig. 9(a), we plot the performance against increasing the number of data samples. That is, we use a random subset of the 10K depth images and evaluate it on our test set. We used subsets with 500, 1000, 3000, 7000, and the full 9800 samples. In Fig. 9(a), we plot this performance. As is clear more training data is useful, although this increment appears to be dependent on the object class. In Fig. 11, we show qualita-

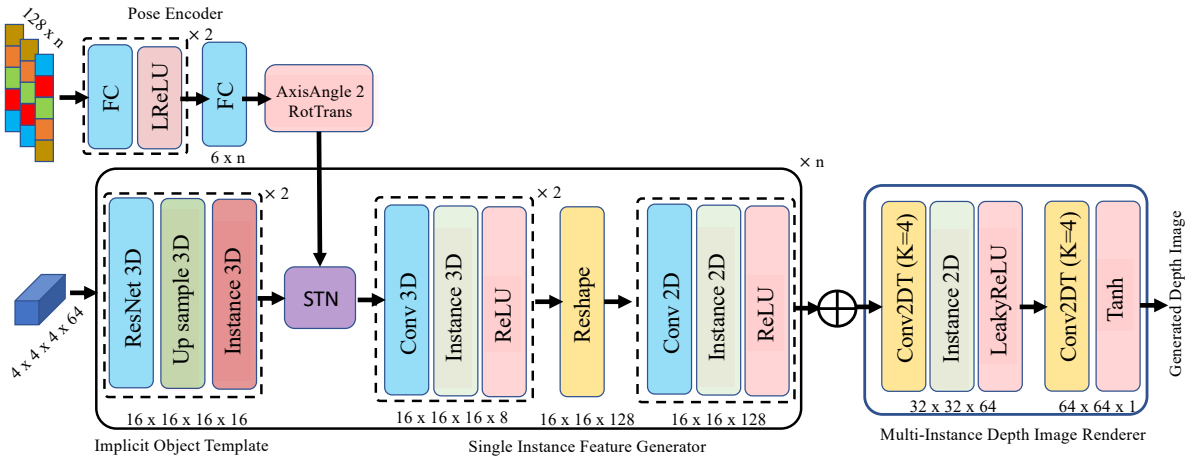
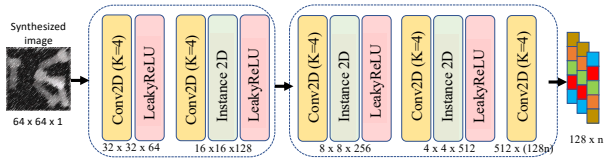
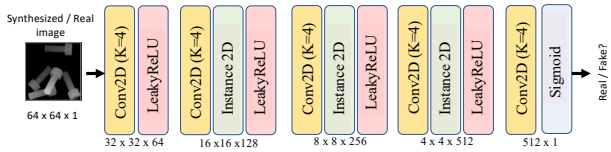


Figure 6. Detailed architecture of InSeGAN generator.



(a) Encoder



(b) Discriminator

Figure 7. (a) depicts detailed architecture of our Encoder module, and (b) shows our Discriminator module.

tive results of instance segmentations obtained for different training set sizes to gain insights into what the performances reported in Fig. 9(a) can be interpreted as. The results show that beyond about 3000 samples, our method seems to start producing qualitatively reasonable instance segmentations, albeit with more data mIoU performance improves.

Number of Instances/Disentanglement? A key question about our framework is whether the algorithm really needs to know the exact number of instances in order to do well at inference, if the model is trained for a fixed number of instances? What happens if we only have a rough estimate? In this section, we empirically answer this question. In Fig. 9(b), we plot the performance against increasing the number of instances used in InSeGAN; i.e., we increase n from 1 to 7 for the number of noise vectors we sample for the generator. Recall that all our ground-truth depth images consist of 5 instances. The plots in Fig. 9(b) for two objects (Bolt and Obj01) shows that InSeGAN performs reasonably well when the number of instances is close to the ground-truth number. In Fig. 12, we plot qualitative results from these choices. Interestingly, we find that using $n = 1$ completely fails to capturing the shapes of the objects, while $n = 4$ learns a two-sided bolt, and $n = 5$ seems to capture the shape perfectly. While $n > 5$ seems to show some improvements, it is not consistent across the data classes. Overall, it looks like a rough estimate of the number of instances is sufficient to achieve reasonable instance segmentation performance.

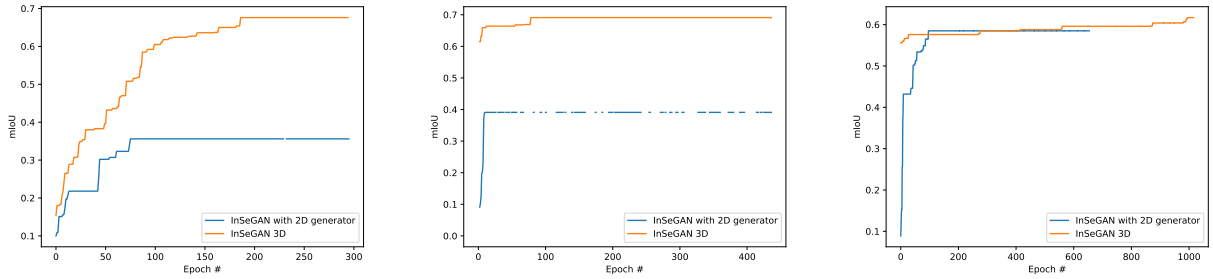
Effect of Noise in the Depth Images? In Table 2, we added Gaussian noise $N(0, \sigma)$ to each pixel in the synthetic depth images input to the algorithm for $\sigma = 0.1, 0.2, 0.5$, and pixels depth values in the range $[-1, 1]$. We find that InSeGAN's performance on noisy depth images is still much better than the performance of K-Means on the noise-free images.

σ	KMeans	No noise	0.1	0.2	0.5
Bolt	0.18	0.424	0.352	0.326	0.318
obj01	0.2	0.686	0.662	0.643	0.421

Table 2. mIOU for for different noise levels in the depth images.

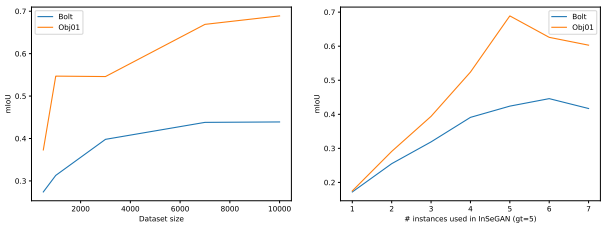
8.1. Qualitative Comparisons

In Fig. 13, we compare qualitative results from InSeGAN with those from other methods. For spectral clustering, we used an automatic bandwidth selection scheme in the nearest neighbor kernel construction. For Wu et al. [11], we use their 1-channel variant, as the 2-channel variant turned out to be very expensive – it is 32x slower than 1-channel. That said, we did explore the performance of 2-channels on our Bolt class, but did not see any significant performance differences to using 1-channel. We also show comparisons to another recent state of the art method, IODINE [1]. For all of the prior works, we used code provided by the respective authors, and only changed the file path to our dataset. They were trained until convergence



(a) Obj01 (b) Cone (c) Connector

Figure 8. Convergence plots for three objects comparing InSeGAN with 3D modules (i.e., using pose encoder, 3D instance template, and STN), shown in orange, with a version in which the 3D modules are replaced by a 2D GAN (i.e., replacing the 3D modules by 2D convolutions and upsampling layers, similar to the encoder and discriminator in reverse). In the figures, we plot mIoU versus the number of training epochs. As is clear, using a 3D GAN leads to better performance and more stable convergence. Note that in the Cone (middle plot), the 2D generator is unstable and often diverges—we reset the optimizer when this happens. This is captured by the discontinuities in the blue plot. In contrast, using the 3D generator leads to very stable training of the generator and discriminator.



(a) Datasize (b) Instances

Figure 9. (a) IoU vs. increasing dataset size. (b) IoU vs. increasing number of instances used in InSeGAN (n), where the ground-truth number of instances used to generate the data was always $n = 5$. Results are shown for two object categories from Insta-10: bolt (blue) and Obj01 (orange).

Figure 10. Results using Slot Attention [3].

(that is, until no change in the objective was found). As is clear from Fig. 13, InSeGAN produces more reasonable segmentations than other methods. We found that IODINE completely fails on our dataset. In contrast, InSeGAN, via modeling the 3D shape of the objects, leads to significant benefits in challenging segmentation settings. In Fig. 10, we show qualitative results using the recent Slot Attention method [3] for the cone class with 5 and 10 instances.

8.2. Qualitative Results

In Figure. 14, we show several more qualitative results for each of the 10 object classes in Insta-10.

References

[1] Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matt Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019. 5, 8

[2] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2017–2025, 2015. 3

[3] Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. In *NeurIPS*, 2020. 6

[4] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 4

[5] Thu Nguyen-Phuoc, Chuan Li, Lucas Theis, Christian Richardt, and Yong-Liang Yang. Hologan: Unsupervised learning of 3d representations from natural images. In *IEEE Int'l Conf. Computer Vision (ICCV)*, pages 7588–7597, 2019. 4

[6] Jeff M Phillips, Nazareth Bedrossian, and Lydia E Kavraki. Guided expansive spaces trees: A search strategy for motion- and cost-constrained state spaces. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3968–3973, 2004. 3

[7] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009. 3

[8] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 4

[9] Xing Wei, Qingxiong Yang, Yihong Gong, Narendra Ahuja, and Ming-Hsuan Yang. Superpixel hierarchy. *IEEE Transactions on Image Processing*, 27(10):4838–4849, 2018. 2

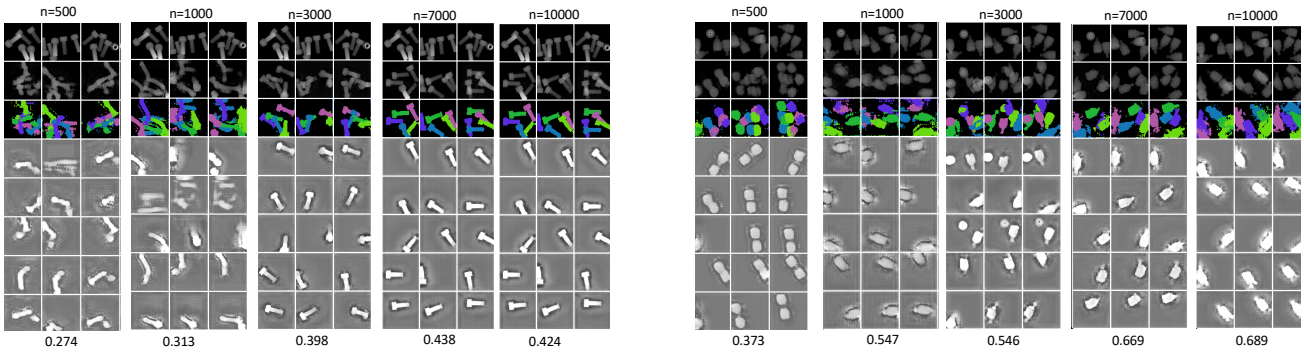


Figure 11. Qualitative instance segmentation results using various data training sizes, for two object classes: Bolt (left) and Obj01 (right). *First row:* input depth image; *second row:* hallucinated depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN. The mIoU on the full test set is shown at the bottom.

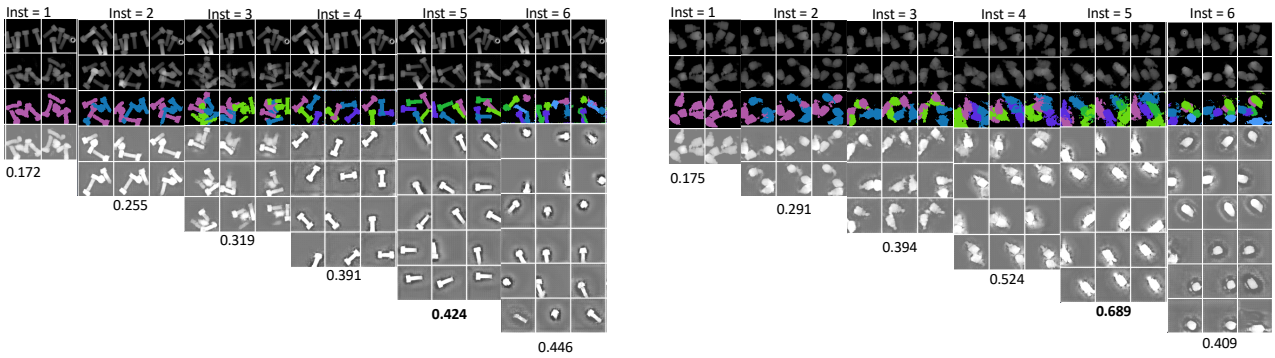


Figure 12. Qualitative instance segmentation results when the number of instances used in InSeGAN is increased, with a fixed ground-truth number of instances ($n = 5$ instances). *First row:* input depth image; *second row:* hallucinated depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN. The mIoU on the full test set is shown at the bottom.

[10] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch and freight: Standard platforms for service robot applications. 3

[11] Yuanwei Wu, Tim Marks, Anoop Cherian, Siheng Chen, Chen Feng, Guanghui Wang, and Alan Sullivan. Unsupervised joint 3d object model learning and 6d pose estimation for depth-based instance segmentation. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019. 5

[12] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5745–5753, 2019. 3

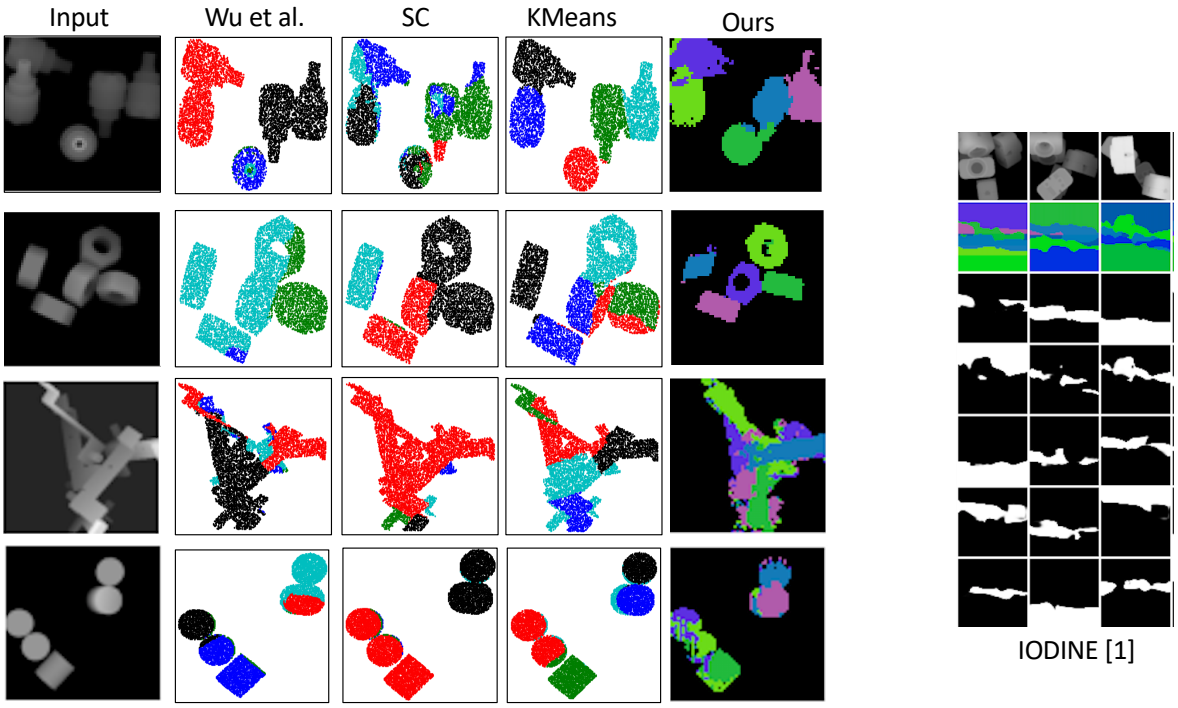


Figure 13. Qualitative comparisons of results from InSeGAN against other methods. On the right, we show a sample result from a segmentation from the competitive method IODINE [1] (using their code on our data).)

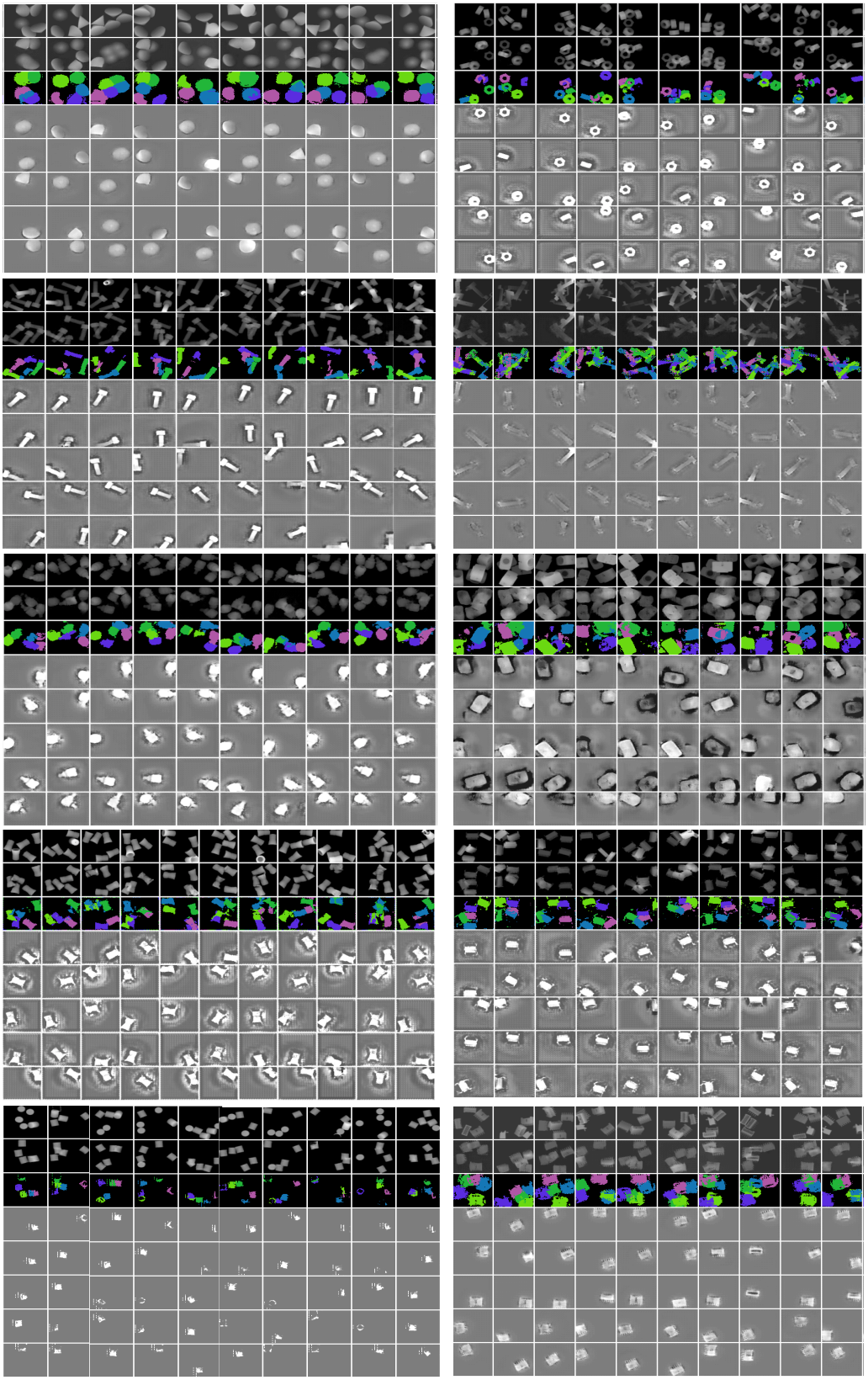


Figure 14. Qualitative results using InSeGAN on the 10 object classes in Insta-10. We show 10 segmentation results for each class. *First row:* input depth image; *second row:* hallucinated (reconstructed) depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN.