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

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

Abstract

In this paper, we present InSeGAN*, an unsupervised 3D generative adversarial network (GAN) for segmenting (nearly) identical instances of rigid objects in depth images. Using an analysis-by-synthesis approach, we design a novel GAN architecture to synthesize a multiple-instance depth image with independent control over each instance. InSeGAN takes in a set of code vectors (e.g., random noise vectors), each encoding the 3D pose of an object that is represented by a learned implicit object template. The generator has two distinct modules. The first module, the instance feature generator, uses each encoded pose to transform the implicit template into a feature map representation of each object instance. The second module, the depth image renderer, aggregates all of the single-instance feature maps output by the first module and generates a multiple-instance depth image. A discriminator distinguishes the generated multiple-instance depth images from the distribution of true depth images.*

To use our model for instance segmentation, we propose an instance pose encoder that learns to take in a generated depth image and reproduce the pose code vectors for all of the object instances. To evaluate our approach, we introduce a new synthetic dataset, "Insta-10", consisting of 100,000 depth images, each with 5 instances of an object from one of 10 classes. Our experiments on Insta-10, as well as on real-world noisy depth images, show that InSeGAN achieves state-of-the-art performance, often outperforming prior methods by large margins.

1. Introduction

Identifying (nearly) identical instances of objects is a problem that is ubiquitous in daily life. For example, when taking a paperclip from a container, choosing an apple from a box, or removing a book from a library shelf, humans subconsciously solve this problem because we have an understanding of what the individual instances are. However, when robots are deployed for such a *picking* task, they need to be able to identify the instances for planning their grasp and approach [30, 2]. Such a problem is commonplace in large manufacturing, industrial, and agricultural contexts [48, 47, 45, 16, 20]. Examples include an industrial robot picking parts from a bin, a warehouse robot picking and placing packages into a delivery truck, or even a fruit-picking robot picking identical fruits in a supermarket. In these scenarios, the robot's owners often have no access to a 3D model of the object to be picked, and annotating individual instances for training can be costly, inconvenient, and unscalable. However, they may have access to a large number of unlabeled images each containing multiple instances of the object, such as depth images of boxes as they

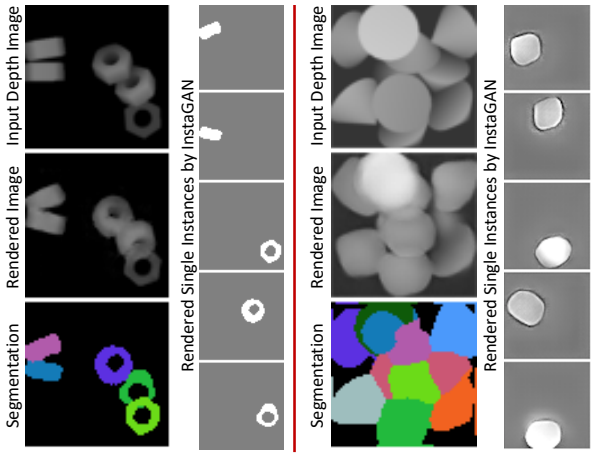


Figure 1. Segmentations and single instances disentangled by InSeGAN on two multiple-instance depth images (*Left:* Nut with 5 instances; *Right:* Cone with 10 instances—challenging). InSeGAN needs only unlabelled multiple-instance depth images for training. For each input image, the hallucinated depth image ("rendered image") and the single instances disentangled from the depth image ("rendered single instances") are shown. We use depth pooling (Z-buffering) and thresholding to produce instance segmentation ("segmentation") from the generated single instances. Note that our method learns the shape of the object automatically.

*Work done as part of an internship at MERL.

travel on a conveyor belt from production to a packaging section. Our goal in this paper is to build an unsupervised instance segmentation algorithm using unlabeled depth images, each containing multiple identical instances of a 3D object.

Our problem setting is very different from the instance segmentation setups that are typically considered, such as that of Mask-RCNN [12], 3D point cloud segmentation [28], scene understanding [9], and others [10, 21, 25]. While these methods usually consider segmenting instances from cluttered backgrounds, our backgrounds are usually simple; however, the foreground instances can be heavily (self-)occluded or may vary drastically in appearance across their poses (see Fig. 1 for example). Prior methods to solve our instance segmentation problem use 3D CAD models [18], fit the 3D instances using primitive shapes [11], or use classical image-matching techniques to identify the instances [4, 35]. More recently, some have attempted to solve this task using deep learning approaches. For example, in Wu et al. [46], a 3D rendering framework is presented that is trained to infer the segmentation masks; however, their losses are prone to local minima. In the recent IODINE [9], MONET [5], and Slot Attention [29] deep models, the focus is on RGB scene decomposition, and may not generalize to segmenting foreground instances from each other.

In this paper, we present a general unsupervised framework for instance segmentation in depth images, which we call *InSeGAN*. Our model is inspired by a key observation made in several recent works (e.g., [23, 33]) that random noise that is systematically injected into a generative adversarial network (GAN) can control various attributes of the generated images. A natural question then is whether we can generate an image with a specific number of instances feeding in the respective number of random noise vectors. If so, then instance segmentation could be reduced to simply decoding a test image into several noise vectors, each of which generates its respective instance. InSeGAN implements this idea using a combination of a 3D GAN and an image encoder within an *analysis-by-synthesis* framework, illustrated in Fig. 2. The training data consist of an unlabeled collection of depth images, each image consisting of n instances of a rigid object. InSeGAN learns an implicit 3D representation of the object shape and a pose decoder that maps random noise vectors to 3D rigid transformations. The generator has two stages. In the first stage, the decoded 3D transformation is applied to the implicit object template, which an *instance feature generator* converts into a feature-map representation of a single object instance. After the first stage generates n such instance representations from n random noise vectors, the second stage aggregates these instance representations and feeds them into a *depth image renderer* to produce synthetic depth images that are similar in distribution to the training images, as enforced via a discriminator. To achieve instance segmentation, we train an encoder that takes as input a generated multiple-instance depth image and encodes it into a latent space in which it must match the random noise vectors that originally generated the images in the GAN stream, thus closing the generation cycle. At inference time, a given depth image first goes through the encoder to get its set of single-instance latent vectors; these are then fed into the GAN to synthesize each instance (each image segment) individually. Results on two example test images are shown in Fig. 1.

While the task of instance segmentation has been approached in various contexts, there is no existing dataset that encompasses this task in the context we are after in this paper. For example, images in standard datasets such as MSCOCO [27] and CityScapes [6] contain objects of several different classes and background, which may not belong to a common latent space. We introduce a new dataset, dubbed "Insta-10," consisting of 10 object classes and 10,000 depth images per class. Each image was rendered using a physics engine that simulated a bin into which 5 instances of an object are randomly dropped, resulting in arbitrary poses of the objects in the rendered depth images. The instances can have significant occlusions and size variations (due to varying distances from the camera), making the task very challenging. We use this dataset to compare our scheme with closely related methods. We also apply our instance segmentation approach to a real-world dataset of blocks in noisy depth images. Our results show that InSeGAN outperforms all of the prior methods by a significant margin on most of the object classes.

We now summarize this paper's primary contributions:

- We propose InSeGAN, a 3D GAN that learns to generate multiple-instance depth images from sets of random noise vectors in an unsupervised manner.
- We propose a two-stage generator structure for InSeGAN, in which the first stage generates a feature map representation of each instance, and the second aggregates these single-instance feature maps and renders a multiple-instance depth image.
- To enable segmentation, we propose an instance pose encoder that encodes a multiple-instance depth image into a set of latent vectors that would generate it. To train this encoder, we introduce novel cycle-consistency losses.
- We have created a new large-scale and challenging dataset, *Insta-10*, which we are making public to advance research on this topic.
- Our experiments on synthetic and real datasets demonstrate that InSeGAN achieves state-of-the-art performance. On the Insta-10 dataset, InSeGAN shows a relative improvement of nearly 35% against the recent method of Wu et al. [46] and nearly 9.3% against Locatello et al. [29].

2. Related Work

In this section, we review some of the closely related approaches to our method.

Multiple Objects and Instance Segmentation: In IODINE [9], a variational generative model is proposed for instance segmentation of RGB images using an iterative refinement of latent vectors to characterize the object instances, similar to an expectation maximization (EM) algorithm. Their key idea is to use a fixed number of latent vectors to describe the scene and iteratively infer the association of these vectors to the instances, an approach that can be unstable for complicated scenes (such as the depth images we consider in our dataset). In Slot Attention [29], abstract scene components, called slots, are learned for each instance in an unsupervised manner, but they do not account for the 3D structure of the scene or the instances. In Liao et al. [26] and O3V-voxel [13], multiple object instances are created in an adversarial setting through image composition. Both of these methods produce a 3D feature latent space—the former a 2D primitive of the 3D object and the latter a 3D voxel representation—for each object instance. Using a fixed number of instances, [26] composes the scene by projecting the primitive to create depth and alpha maps. In [13], the authors propose a scheme to generate a video sequence to extract the multiple instance images. They follow a framework similar to [5, 9], where the initial image is generated from a sequence of real images, through an encoder. However, they generate a feature voxel representation for each object. At each time instance, each object is rendered and they are composed together.

There are prior approaches that tackle the multiple instance segmentation problem for 2D and 3D images in a *supervised* manner. Most of these methods, e.g., [12, 36], first extract Regions of Interest (RoI) from the input, subsequently classifying the object in each selected region. Mask RCNN [12] expands Faster RCNN [38] by creating a new segmentation branch to classify per-pixel object segments. DeepMask [36] learns those RoIs and their underlying masks, which are then passed through the Fast RCNN [8] for classification. Along similar lines, point cloud segmentation has been explored in several recent works. For example, [50, 42] propose a 2D architecture. GsPN [50] proposes a network to generate shapes with their specific segmentations and bounding boxes. SGPN [42] generates a similarity matrix and group proposals to create independent clusters for classification. In contrast to these popular methods for instance segmentation, we differ in that we approach the problem from an unsupervised perspective.

3D Disentanglement: Several recent works have proposed approaches for disentangling 3D attributes using deep learning via implicit or explicit representations. Deep Voxels [40] proposes a synthesis approach to learning an implicit 3D representation of the object. The method learns to synthesize novel perspectives of an object from a learned voxel feature volume. From these voxels, one may create an explicit 3D model of the object. However, their model is not generative and requires camera parameters. HoloGAN [33] proposes a generative method that creates an implicit 3D volume of single instances. It first learns a 3D representation, which it transforms using a target pose, then projects to 2D features and renders to a final image. Our method is inspired by HoloGAN, but we go beyond it by deriving a scheme for disentangling the object instances. Another related work is PlatonicGAN [14], which creates a 3D representation of an object while generating different unseen views via adversarial learning. However, as in HoloGAN, this method is limited to a single rotated object.

The prior work that is most similar to ours is Wu et al. [46], which proposes to disentangle object instances and their 6D poses in an unsupervised manner, concurrently learning an explicit 3D point cloud template of the object. While our objective is similar, our proposed framework is completely different. The framework of [46] requires explicit modeling of point occlusions and computes point cloud alignments using Chamfer distance, which make the scheme computationally expensive. We avoid these challenges by using depth images, and we introduce a discriminator that implicitly learns these steps efficiently.

3. Proposed Method

Let $\mathcal{X}$ be a given dataset, where each $\mathbf{x} \in \mathcal{X}$ is a depth image consisting of n instances of a rigid object. Note that the same rigid object is depicted in all of the images in $\mathcal{X}$. To simplify the notation, we will use $\mathcal{X}$ to also characterize the distribution of $\mathbf{x}$. Further, for clarity of presentation, we assume that n is known and fixed for $\mathcal{X}$, however note that it is straightforward to extend InSeGAN for an arbitrary number of instances by using training images with varying numbers of instances (see Supplementary materials for details). Our goal in InSeGAN is to learn a model only from $\mathcal{X}$ (without any labels) such that at test time, when given a depth image $\mathbf{x}$, the learned model outputs the segmentation masks associated with each instance in the depth image. In the next section, we provide a brief overview of the InSeGAN architecture, followed by a detailed look into each of its components.

3.1. InSeGAN Overview

The basic architecture of InSeGAN follows a standard generative adversarial framework, however with several non-trivial twists. It consists of a generator module G that—instead of taking a single noise vector as input (as in a typical GAN)—takes n noise vectors, $\{\mathbf{z}_1, \mathbf{z}_2, \cdots, \mathbf{z}_n\}$, each $\mathbf{z} \in \mathbb{R}^d \sim \mathrm{N}(\mathbf{0}, \mathbf{I}_d)$, and generates a multiple-instance depth image as output. Thus, $G : \mathbb{R}^{d \times n} \rightarrow \hat{\mathcal{X}}$, where $\hat{\mathcal{X}}$ is used to signify the distribution of the generated depth images, with the limit $\hat{\mathcal{X}} \rightarrow \mathcal{X}$ when G is well-trained. We

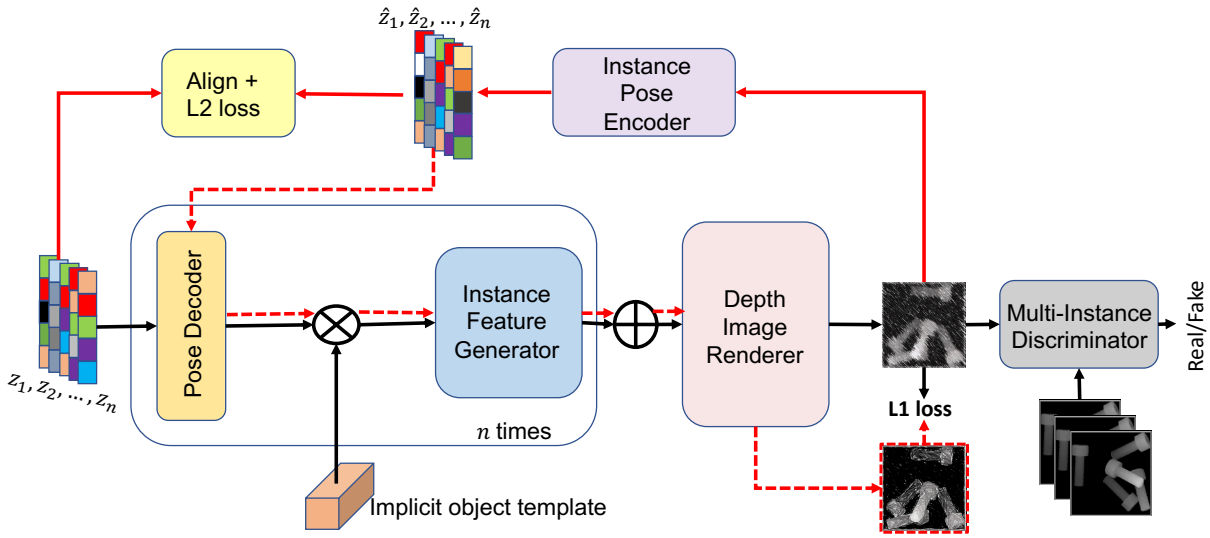


Figure 2. A schematic illustration of the training scheme in InSeGAN. There are three distinct control flows in the framework, as denoted by the black, solid red, and dashed red arrows. The black arrows capture the generative process producing a multiple instance depth image, while the solid red arrows depict the scheme to encode a generated depth image to its instances. The dashed red arrows depict the control flow to train the Instance Encoder via using the encoded latent vectors to re-create the already-generated image.

denote the set of noise vectors by the matrix $\mathbf{Z} \in \mathbb{R}^{d \times n}$ and the distribution of $\mathbf{Z}$ as $\mathcal{Z} = \{\mathrm{N}(\mathbf{0}, \mathbf{I}_d)\}^n$. Next, a discriminator module D is trained to distinguish whether its input is an image generated by G or a sample from the data distribution $\mathcal{X}$. The modules G and D are trained in a min-max adversarial game so that G learns to generate images that can fool D, while D in turn learns to distinguish whether its inputs are real or fake; the optimum occurs when D cannot recognize whether its input is from G or $\mathcal{X}$. Apart from the generator and discriminator modules, we also have an instance pose encoder module, E, that is key to achieving instance segmentation. Specifically, $E : \hat{\mathcal{X}} \rightarrow \mathbb{R}^{d \times n}$ takes as input a generated depth image, and learns to output vectors that match the latent noise vectors that generated the input depth image. The essence of InSeGAN is to have the generator G produce depth images for which the instance segments are implicitly known (through $\mathbf{Z}$), so that E can be trained on them to learn to disentangle the instances. In the limit as $\hat{\mathcal{X}} \rightarrow \mathcal{X}$, as guided by the discriminator D, the encoder E will eventually learn to do instance segmentation on real images from $\mathcal{X}$. An overview of the InSeGAN training pipeline is shown in Fig. 2. Next, we will describe each of the modules in detail.

3.2. InSeGAN Generator

The key to InSeGAN is to have the generator G accomplish two tasks jointly: (i) to produce depth images $\hat{\mathbf{x}}$ that match the input image distribution $\mathcal{X}$, and (ii) to identify each object instance in the generated image $\hat{\mathbf{x}}$. To this end, we note that *sans* the other instances, each instance is an independent depth rendering of an object in an arbitrary 3D pose. A multiple-instance depth image may be generated by merging the individual instances, followed by depth-based inter-object occlusion reasoning.

Motivated by the above insight, we propose to separate the generator G into two distinct modules: (i) an *instance feature generator* that generates feature maps for single object instances, and (ii) a *depth image renderer* module that aggregates the single-instance feature maps and renders the multiple-instance depth image. As the instances are assumed to be of the same object, we propose to sample each noise vector $\mathbf{z} \in \mathbf{Z}$ from the same latent distribution, $\mathbf{z} \sim \mathrm{N}(\mathbf{0}, \mathbf{I}_d)$. Further, our system learns an implicit 3D object model (template) that, when geometrically transformed, produces the varied appearances of the instances.

Our first step in the generator pipeline is to produce 6-DOF (6 degrees of freedom) 3D rigid geometric transforms that can be applied to the implicit object template to produce a transformed implicit model representing each instance. To this end, each noise vector $\mathbf{z} \in \mathbf{Z}$ is converted to an element of the special Euclidean group $\big(\mathrm{SE}(3)\big)$ using a *pose decoder* module (see Fig. 2), which is a fully connected neural network that is denoted $G_p : \mathbb{R}^d \rightarrow \mathbb{R}^6$. Given a noise vector $\mathbf{z}$, G_p produces a corresponding axis-angle representation; this is next converted to an element in the Special Euclidean group, $\mathrm{SE}(3)$. We denote this operator by $\Lambda : \mathbb{R}^6 \rightarrow \mathrm{SO}(3) \times \mathbb{R}^3$, i.e., Λ produces a rotation matrix $R \in \mathrm{SO}(3)$ (the special orthogonal group) and a translation vector $t \in \mathbb{R}^3$. A natural question in this context is why we do not sample the transformation matrix directly (as in, e.g., HoloGAN [33]). This is because, as will be clear shortly, we need to match the output of the instance pose encoder module E with the pose representations of the instances, and having a Euclidean embedding

for these representations offers computationally more efficient similarity measures than directly using a rotation matrix (or axis-angle) parameterization of the underlying nonlinear geometric manifold [17, 51].

Next, we use the transformation matrix thus created, i.e., $\Lambda(G_p(\mathbf{z}))$, to geometrically transform an implicit shape tensor $T \in \mathbb{R}^{h\times h\times h\times k}$ (we use h=4, k=128); this parameter tensor is shared by all the instances and will, when trained (with the other modules in the pipeline), implicitly capture the shape and appearance of the object. Similar to HoloGAN [33], we use a Spatial Transformer Network (STN) [19] to apply the geometric transform to this implicit template. The transformed T is reshaped to $\mathbb{R}^{kh\times h\times h}$ and projected from 3D to 2D using a single-instance projection module, G_s, to output $\hat{\mathbf{x}}_f \in \mathbb{R}^{c\times h\times h}$, which captures the feature map representation of an instance. The above steps can be formally written as:

$$\mathcal{F}(\mathbf{z}) := G_s\Big(\mathrm{STN}\big(\Lambda\left(G_p\left(\mathbf{z}\right)\right), T\big)\Big). \tag{1}$$

Next, we propose to combine these feature maps by average-pooling them, then render a multiple-instance depth image using a rendering module G_r, as follows:

$$\hat{\mathbf{x}} = G(\mathbf{Z}) := G_r(\bar{\mathcal{F}}) \text{ where } \bar{\mathcal{F}} = \frac{1}{|\mathbf{Z}|}\sum_{\mathbf{z}\in\mathbf{Z}} \mathcal{F}(\mathbf{z}), \tag{2}$$

where $\hat{\mathbf{x}}$ denotes a depth image generated by G. This generative control flow is depicted using black arrows in Fig. 2.

3.3. InSeGAN Discriminator

As in standard GANs, the task of the discriminator D is to decide whether its input comes from the natural distribution of multiple-instance depth images that produced the training set (i.e., $\mathcal{X}$) or is synthesized by our generator G (i.e., $\hat{\mathcal{X}}$). Following standard architectures, D consists of several 2D convolution, instance normalization, and LeakyRELU layers, and outputs a classification score in $[0, 1]$. The objectives for training the discriminator and generator, respectively, are to minimize the following losses:

$$\begin{aligned}\mathcal{L}_D &:= -\mathbb{E}_{\mathbf{x}\sim\mathcal{X}} \log(D(\mathbf{x})) - \mathbb{E}_{\mathbf{Z}\sim\mathcal{Z}} \log\left(1 - D(G(\mathbf{Z}))\right),\\ \mathcal{L}_G &:= -\mathbb{E}_{\mathbf{Z}\sim\mathcal{Z}} \log D(G(\mathbf{Z})).\end{aligned} \tag{3}$$

The task for our discriminator is significantly different from that in prior works, as it must learn to: (i) count whether the number of rendered instances matches the number of instances in the data distribution, (ii) verify whether the rendered 3D posed objects obtained via transforming the *still-being-learned* object template T capture the individual appearances (which are also being learned) of the instances, and (iii) whether the layout of the rendered image is similar to the compositions of the instances in the training depth images. Fortunately, with access to a suitable dataset, D can automatically achieve these desired behaviors when adversarially trained with the generator.

3.4. InSeGAN Instance Pose Encoder

We now introduce our *instance pose encoder* module, E, which is the key to instance segmentation. The task of this module is to take as input a multiple-instance depth image $\hat{\mathbf{x}}$, produced by G, and reconstruct each of the noise vectors in $\mathbf{Z}$ (encoding the instance poses) that were used to generate $\hat{\mathbf{x}}$. The encoder outputs $\hat{\mathbf{Z}}$, a set of latent vectors. Indeed, as $\hat{\mathbf{x}}$ is produced by aggregating n independently sampled instance appearances of the object, inverting the process amounts to disentangling $\hat{\mathbf{x}}$ into its respective instances. Thus, when the generator is trained well, i.e., $\hat{\mathbf{x}} \approx \mathbf{x}$, we will eventually learn to disentangle each instance in a ground truth image. While this idea is conceptually simple, implementing it practically is not straightforward. There are four main difficulties: (a) the input $\mathbf{Z}$ to the generator and the output $\hat{\mathbf{Z}}$ of E are unordered sets, which need to be aligned before comparing them; (b) the average pooling operator in (2) aggregates several feature maps into one—an operation that loses the distinctiveness of each of the instance feature maps; (c) the depth renderer G_r may remove occluded parts of the instances, thus posing ambiguities when mapping them back to the noise vectors; and (d) the pose encoder G_p projects its noise input to the space of rigid body transforms, an operation that is inherently low-rank and nonlinear. We tackle these challenges via imposing losses on the encoder so that it learns to invert each module in the generator. We decompose the encoder as $E = G_s^{-1} \circ G_r^{-1}$, consisting of: (i) an image derenderer G_r^{-1} that takes a depth image and produces feature maps, and (ii) an instance decoder G_s^{-1} that takes the feature maps from G_r^{-1} and produces $\hat{\mathbf{Z}}$.

Alignment and Reconstruction: To tackle our first difficulty, (a), we propose to align the sets $\mathbf{Z}$ and $\hat{\mathbf{Z}}$ before computing a reconstruction loss on them. Specifically, we seek to find an alignment matrix $\pi \in \Pi(\mathbf{Z}, \hat{\mathbf{Z}})$, where Π denotes the set of all such alignments (i.e., permutations) on its inputs, such that the reconstruction loss is minimized:

$$\mathcal{L}_E^a = \left\|\mathbf{Z} - \pi^*(\hat{\mathbf{Z}})\right\|^2, \text{ where } \pi^* = \underset{\pi\in\Pi(\mathbf{Z},\hat{\mathbf{Z}})}{\arg\min}\ \mathrm{OT}(\pi, \mathbf{D}(\mathbf{Z}, \hat{\mathbf{Z}})), \tag{4}$$

where $\mathbf{D}$ denotes the pairwise distances between the columns in $\mathbf{Z}$ and $\hat{\mathbf{Z}}$, and OT is some suitable matching scheme. We use a general purpose optimal transport (IPOT [49]) scheme to implement the alignment, which returns a permutation matrix π^* that is used to align the matrices before comparing them using the ℓ_2 distance.[1] We show this encoder control flow using solid red arrows in Fig. 2.

Intermediate Reconstruction: To tackle difficulties (b) and (c) in the encoder design, which involve E learning to

[1] We may also use a Hungarian matching scheme [22] to implement OT if the number of data instances is small, which is usually significantly faster than optimal transport methods. Note: our experiments suggest that a *greedy* way to align is not useful—see Section 4.1.

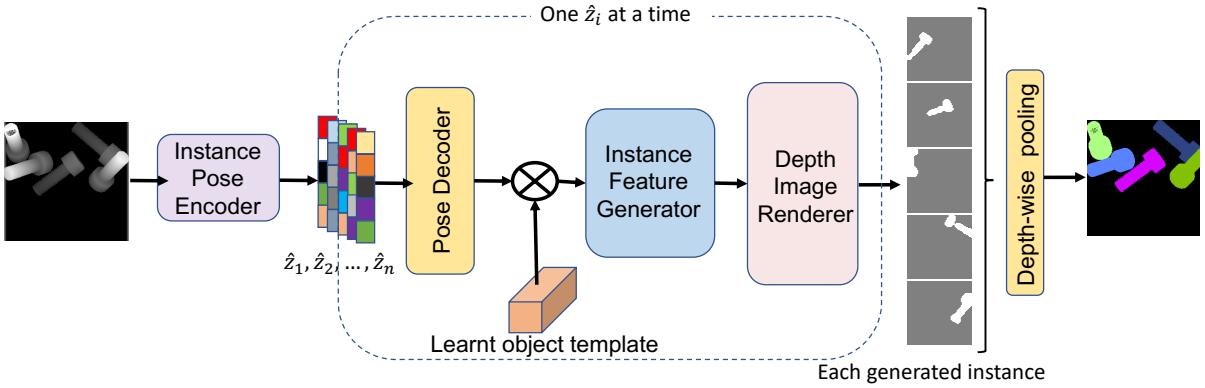


Figure 3. InSeGAN inference pipeline (see Sec. 3.5 for details).

invert the depth renderer, we use the output from the derenderer sub-module G_r^{-1} in E. Specifically, G_r^{-1} is forced to reconstruct the average-pooled feature map $\bar{\mathcal{F}}$ in (2). Let us denote this loss by $\mathcal{L}_E^i = \left\| \bar{\mathcal{F}} - G_r^{-1}(\hat{\mathbf{x}}) \right\|^2$.

Pose Decoding: Although one could apply the above intermediate feature decoding strategy even to the pose decoder G_p, it would not be very efficient to compare its output $\Lambda(G_p(\hat{\mathbf{Z}}))$ to the rigid transforms produced during the generative process. This is because the geometric matrix that Λ produces involves a rotation matrix, and thus optimizing would ideally require Riemannian optimization methods in the space of SO(3) [1], which is not well suited for standard optimization schemes such as Adam [24]. Furthermore, there may be several different geometric transformations that could achieve the same output [51]. To avoid this technicality, we propose to learn the rigid transform indirectly, by avoiding exact reconstruction of the transform and instead asking it to have the desired outcome in the generative process. Specifically, we propose to use the $\hat{\mathbf{Z}}$ produced by the encoder, and use it as a noise matrix to produce a depth image $G(\hat{\mathbf{Z}})$; this depth image is then compared with the depth image generated in the previous pass using $\mathbf{Z}$. The following loss, $\mathcal{L}_E^p$, captures this idea:

$$\mathcal{L}_E^p = \|G(\mathbf{Z}) - G(E(\hat{\mathbf{x}}))\|_1 . \tag{5}$$

The above control flow is illustrated in Fig. 2 by the dashed red arrows that go from noise vectors $\hat{\mathbf{z}}$ through the pose decoder and over to the depth renderer, i.e., the output of G.

Encoder Loss: We combine the above three losses when training the parameters of the encoder module (see the Supplementary Material for details on its architecture):

$$\mathcal{L}_E = \mathcal{L}_E^a + \lambda_1 \mathcal{L}_E^i + \lambda_2 \mathcal{L}_E^p, \tag{6}$$

where the λ's provide weights to each type of loss.[2] When backpropagating the gradients on the encoder losses, we fix the generator parameters, as otherwise they will co-adapt with the encoder parameters, making training unstable.

[2]We use $\lambda_1 = \lambda_2 = 1$ in all our experiments.

Learning the Implicit Object Template: The template is implemented as a weight tensor, learned via backpropagation gradients from the above loss. During training, backpropagation reverses all of the arrows in Fig. 2.

3.5. InSeGAN Inference

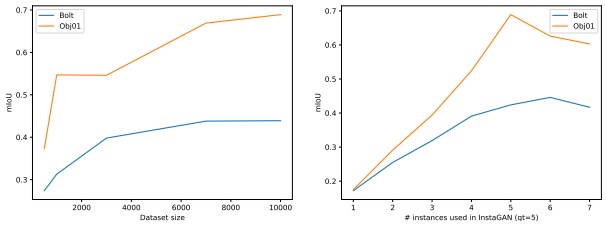
At inference time, we assume to be given only a depth image consisting of multiple instances of the rigid object; our goal is to segment the instances and render each instance separately, while producing an instance segmentation on the input. To this end, our inference pipeline resembles the generative process, but with some important differences as illustrated in Fig. 3. Specifically, for inference we input the multiple-instance depth image to the instance pose encoder module E, which produces a set of latent vectors $\hat{\mathbf{Z}}$. Each $\hat{\mathbf{z}} \in \hat{\mathbf{Z}}$ is input individually into the trained single-instance generator G_s, the output of which is rendered using G_r to form a single-instance depth image that corresponds to $\hat{\mathbf{z}}$. We emphasize that in the inference phase, the depth image renderer sits within the single-instance generation phase—this contrasts with the training setting, in which the renderer takes as input the aggregated feature tensor $\bar{\mathcal{F}}$. Once the single instances are rendered, as shown in Fig. 3, we use a depth-wise max pooling on these instance depth images for inter-instance occlusion reasoning, followed by thresholding (and applying basic image filters to) the single instances. Thresholding removes any bias introduced during depth rendering. To produce the pixel-wise segmentation, we use the index of the generated instance that is selected for a given pixel.

3.6. Training Pipeline

We train our full framework, including the InSeGAN generator G, discriminator D, and Encoder E, by minimizing the sum of all the losses given by:

$$\mathcal{L} = \mathcal{L}_D + \mathcal{L}_E + \mathcal{L}_G. \tag{7}$$

The gradients for the various modules are computed using PyTorch autograd. We use Adam for training all our models, with a learning rate of 0.0002, $\beta_1 = 0.5$, and $\beta_2 = 0.99$.



(a) Datasize (b) Instances

Figure 4. mIoU versus (a) training dataset size, (b) number of instances n in model (ground truth has 5 instances).

4. Experiments and Results

In this section, we present experiments demonstrating the empirical benefits of InSeGAN on the task of instance segmentation. We will first introduce our new synthetic dataset, *Insta-10*, on which most of our experiments are based. We then introduce a real-world dataset that we collected to evaluate the application of our method on (naturally noisy) depth images of real objects.

Insta-10 Dataset: While there are several real-world datasets used for instance segmentation, such as MSCOCO [27], and CityScapes [6], they typically involve background objects, and other *stuff* that are unrelated to the objects to be segmented. In addition, datasets such as CLEVR [21] are proposed for visual reasoning tasks, and thus may not fully analyze the segmentation quality. To fill this gap, we introduce Insta-10, a large-scale dataset consisting of depth images of multiple instances of a CAD object model. We remove color and texture from the instances, to analyze the segmentation performance under the difficult condition in which there are minimal attributes other than shape. This is inspired by the observation that most industrial objects do not usually have textures [15], in addition to the intuition that sometimes RGB could distract a shape-based segmenter.

To create the dataset, we used 10 CAD object models (3 from the T-less dataset [15] and 7 from our own library). We use the PhysX physics simulator[3] to simulate sequentially dropping objects into a bin, producing synthetic multiple-instance depth images. We used 5 instances of the same object in each depth image, yielding substantial inter-instance occlusion, and we selected the bin width so that instance segmentation was challenging but not too hard (even for humans). In addition to the depth images, we provide the point clouds associated with each image and ground truth instance segmentation masks; these masks are used for only evaluation, not training. We collected 10K images per object, for a total of 100K depth images in the entire dataset, each with dimension 224×224. Sample images are in Figs. 21 and 20.

Real-World Depth Images Using a Robot: Apart from the synthetic Insta-10 dataset, we also analyze the adaptability of our scheme to practical settings. For this experiment, we used a box containing 4 identical wooden blocks (see Fig. 6), of which depth images were captured using an Intel RealSense Depth Camera (D435). To produce multiple diverse images consisting of varied configurations of the blocks, we programmed a Fetch robot [44] to shake the box between images. We collected 3,000 depth images using this setup, of which we hand-annotated 62 images that we reserved for evaluation. The depth images from this setting are very noisy, and as a result, often the shapes of the objects do not appear to be identical.

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

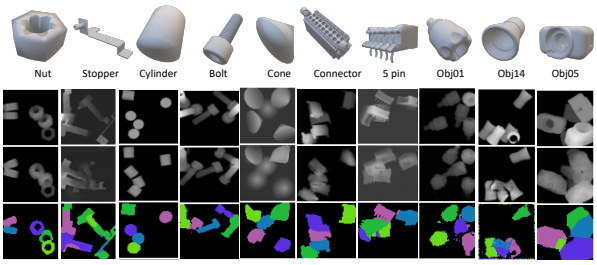


Figure 5. Qualitative results on Insta-10 objects. First row: CAD models used to produce Insta-10. Second row: the input depth images. Third row: rendered depth image by InSeGAN. Fourth row: the predicted segmentations by InSeGAN.

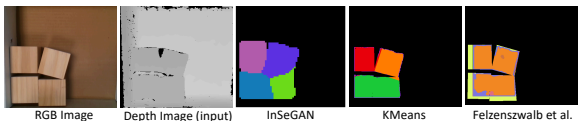


Figure 6. Qualitative results on real data. We show the RGB image, the noisy depth input, and the segmentations produced.

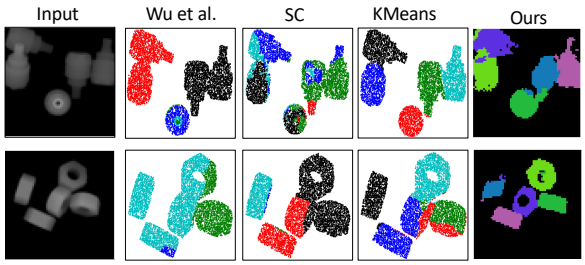


Figure 7. Qualitative comparisons against other methods.

Evaluation Metric and Experimental Setting: To evaluate our scheme, we use mean intersection-over-union (mIoU), a standard metric for semantic segmentation. For training and evaluation, we split the data subsets associated with each class into a training, validation, and test set. In the Insta-10 dataset, we use 100 randomly selected images for validation in each class. For the test set, we selected 100 images on which KMeans fails, thereby avoiding segmentations that are trivial for standard methods.

Performance Analysis: On the Insta-10 dataset, we compare our method on both non-deep and deep learning methods. The non-deep methods include classic segmentation

Method	Nut	Stop.	Cyl.	Bolt	Cone	Conn.	5-pin	Obj01	Obj14	Obj05	Avg mIoU
Non-Deep Learning Methods											
K-Means	0.64	0.297	0.7	0.18	0.35	0.554	0.628	0.208	0.496	0.59	0.464
Spectral Clustering [32]	0.56	0.36	0.54	0.22	0.41	0.56	0.58	0.25	0.47	0.57	0.452
GrabCut [39]+KMeans	0.572	0.232	0.572	0.472	0.231	0.519	0.497	0.597	0.557	**0.605**	0.486
GraphCut [3]	0.569	0.1	0.589	0.447	0.12	0.476	0.12	0.597	0.540	0.511	0.373
Deep Learning Methods											
Wu et al. [46]	0.45	0.28	0.57	0.27	0.33	0.38	0.43	0.23	0.44	0.57	0.385
IODINE [9]	0.026	0.059	0.019	0.040	0.089	0.032	0.034	0.058	0.053	0.118	0.053
Slot Attn. [29]	0.375	0.276	0.535	0.43	**0.68**	**0.662**	0.628	0.655	**0.622**	0.481	0.535
InSeGAN (2D) (ours)	0.215	**0.365**	0.258	0.524	0.435	0.585	0.628	0.365	0.286	0.532	0.419
InSeGAN (3D) (ours)	**0.773**	0.301	**0.760**	**0.539**	0.47	0.655	**0.642**	**0.686**	0.591	0.483	**0.590**

Table 1. Mean IoU (mIoU) between the segmentation masks predicted by each method and the ground-truth masks.

Generator Loss	Bolt	Obj01
$\mathcal{L}_E^a$ (OT) + $\mathcal{L}_E^i + \mathcal{L}_E^p$	**0.424**	**0.686**
$\mathcal{L}_E^a$ (greedy) + $+\mathcal{L}_E^i + \mathcal{L}_E^p$	0.383	0.664
$\mathcal{L}_E^a$ (OT) + $\mathcal{L}_E^i$	0.312	0.360
$\mathcal{L}_E^a$ (OT)	0.303	0.402

Table 2. Ablation study on the various losses used in InSeGAN generator and the mIoU achieved on two classes.

Method	mIoU
KMeans	0.797
Spectral Clustering	0.668
Graph Segmentation [7]	0.436
InSeGAN	**0.857**

Table 3. Results on real-world data collected using a robot.

algorithms [32, 3, 39]. The deep learning comparisons include: (i) Wu et al. [46], which is most similar to ours; (ii) IODINE [9], which was proposed for scene decomposition rather than instance segmentation; and (iii) Slot Attention [29]. We use the public code for (ii) and (iii), using their default hyper-parameters. In Table 1, we show these results. We find that for most object classes (6/10), InSeGAN outperforms all other methods.

On the Stopper class, which is the most difficult, InSeGAN outperforms all other methods except for spectral clustering. Overall, InSeGAN demonstrates a relative improvement of 9.3% over the best-performing previous method (averaged across all 10 classes). We found that the recent method of IODINE [9] fails on our images, perhaps because it is designed for scene decomposition tasks. From Table 3, we see that our method generalizes to real data as well. In Fig. 21, we show several qualitative results produced by InSeGAN. More results are provided in the Supplementary Material.

4.1. Ablation Studies

In this section, we analyze each component in our design, empirically justifying its importance.

Is the 3D Generator Important? To answer this question, we replace the 3D modules in InSeGAN (3D implicit template, pose encoder, and STN) by 2D convolutions and upsampling layers, similar to those used in the encoder and discriminator. In Table 1, we provide comparisons of the 3D and 2D GANs on the Insta-10 dataset. Results show that our 3D generator is significantly better than a 2D generator.

Are all the losses important? There are three losses in the InSeGAN generator: (i) $\mathcal{L}_E^a$, the alignment loss, (ii) $\mathcal{L}_E^i$ on the intermediate feature maps, and (iii) $\mathcal{L}_E^p$ between the generated depth image and the re-generated depth image. For (i), we compare a *greedy* choice for alignment vs. using optimal transport. We provide ablative studies on two object classes, Bolt and Obj01. As is clear from Table 2, using a greedy alignment leads to lower performance. Further, we find that using $\mathcal{L}_E^p$ is empirically very important, yielding 10–20% performance improvement. Our analysis confirms the importance of all of the losses used in our architecture.

Do we need all training samples? In Fig. 16(a), we plot the performance versus increasing the number of data samples; i.e., we train on a random subset of the 10K depth images in the training set. Clearly more training data is useful, but this increment appears to be dependent on the object class.

Wrong number of instances? In Fig. 16(b), we plot the performance versus increasing the number of instances used in InSeGAN; i.e., we increase n from 1 to 7. This is a mismatch from the true number of instances (5 in every depth image). The plot shows that InSeGAN performs reasonably well when the number of instances is close to the ground truth. In the Supplementary Material, we show how to handle an unknown number of instances n in each image.

5. Conclusion

In this paper, we presented InSeGAN, a novel 3D GAN to solve unsupervised instance segmentation. We find that by pairing the discriminator with a carefully designed generator, the model can reconstruct individual object instances even under clutter and severe occlusions. We introduce a new large-scale dataset, which we are making publicly available, to empirically analyze our approach. Our method demonstrates state-of-the-art results, generalizing well to real-world images.

References

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 6

[2] Marcos Alonso, Alberto Izaguirre, and Manuel Graña. Current research trends in robot grasping and bin picking. In *The 13th International Conference on Soft Computing Models in Industrial and Environmental Applications*, pages 367–376. Springer, 2018. 1

[3] Yuri Boykov and Vladimir Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE transactions on pattern analysis and machine intelligence*, 26(9):1124–1137, 2004. 8

[4] Dirk Buchholz. *Bin-picking: new approaches for a classical problem*, volume 44. Springer, 2015. 2

[5] Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019. 2, 3

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3213–3223, 2016. 2, 7

[7] Pedro F Felzenszwalb and Daniel P Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004. 8

[8] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. 3

[9] 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. 2, 3, 8, 16, 17

[10] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey on instance segmentation: state of the art. *International Journal of Multimedia Information Retrieval*, pages 1–19, 2020. 2

[11] Kensuke Harada, Kazuyuki Nagata, Tokuo Tsuji, Natsuki Yamanobe, Akira Nakamura, and Yoshihiro Kawai. Probabilistic approach for object bin picking approximated by cylinders. In *2013 IEEE International Conference on Robotics and Automation*, pages 3742–3747. IEEE, 2013. 2

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2961–2969, 2017. 2, 3

[13] Paul Henderson and Christoph H Lampert. Unsupervised object-centric video generation and decomposition in 3d. *Advances in Neural Information Processing Systems (NIPS)*, 33, 2020. 3

[14] Philipp Henzler, Niloy J Mitra, and Tobias Ritschel. Escaping plato's cave: 3d shape from adversarial rendering. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 9984–9993, 2019. 3

[15] Tomáš Hodan, Pavel Haluza, Štepán Obdržálek, Jiri Matas, Manolis Lourakis, and Xenophon Zabulis. T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE, 2017. 7

[16] Yinlin Hu, Joachim Hugonot, Pascal Fua, and Mathieu Salzmann. Segmentation-driven 6d object pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3385–3394, 2019. 1

[17] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009. 5

[18] Katsushi Ikeuchi. Generating an interpretation tree from a cad model for 3d-object recognition in bin-picking tasks. *International Journal of Computer Vision*, 1(2):145–165, 1987. 2

[19] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 2017–2025, 2015. 5, 13

[20] Omid Hosseini Jafari, Siva Karthik Mustikovela, Karl Pertsch, Eric Brachmann, and Carsten Rother. ipose: instance-aware 6d pose estimation of partly occluded objects. In *Asian Conference on Computer Vision*, pages 477–492. Springer, 2018. 1

[21] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2901–2910, 2017. 2, 7

[22] Roy Jonker and Ton Volgenant. Improving the hungarian assignment algorithm. *Operations Research Letters*, 5(4):171–175, 1986. 5

[23] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4401–4410, 2019. 2

[24] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 6

[25] Victor Lempitsky and Andrew Zisserman. Learning to count objects in images. In *Advances in neural information processing systems*, pages 1324–1332, 2010. 2

[26] Yiyi Liao, Katja Schwarz, Lars Mescheder, and Andreas Geiger. Towards unsupervised learning of generative models for 3d controllable image synthesis. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 5871–5880, 2020. 3

[27] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European Conf. Computer Vision (ECCV)*, pages 740–755. Springer, 2014. 2, 7

[28] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768, 2018. 2

[29] 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. 2, 3, 8, 15, 16

[30] Jeffrey Mahler and Ken Goldberg. Learning deep policies for robot bin picking by simulating robust grasping sequences. In *Conference on robot learning*, pages 515–524. PMLR, 2017. 1

[31] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014. 14

[32] Andrew Y Ng, Michael I Jordan, Yair Weiss, et al. On spectral clustering: Analysis and an algorithm. *Advances in neural information processing systems*, 2:849–856, 2002. 8

[33] 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. 2, 3, 4, 5, 13

[34] 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. 12

[35] Paolo Piccinini, Andrea Prati, and Rita Cucchiara. Sift-based segmentation of multiple instances of low-textured objects. *International Journal of Computer Theory and Engineering*, 5(1):41–46, 2013. 2

[36] Pedro OO Pinheiro, Ronan Collobert, and Piotr Dollár. Learning to segment object candidates. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1990–1998, 2015. 3

[37] 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. 12

[38] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *NIPS*, pages 91–99, 2015. 3

[39] Carsten Rother, Vladimir Kolmogorov, and Andrew Blake. Grabcut: Interactive foreground extraction using iterated graph cuts. *ACM transactions on graphics (TOG)*, 23(3):309–314, 2004. 8

[40] Vincent Sitzmann, Justus Thies, Felix Heide, Matthias Nießner, Gordon Wetzstein, and Michael Zollhofer. Deepvoxels: Learning persistent 3d feature embeddings. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2437–2446, 2019. 3

[41] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. 13

[42] Weiyue Wang, Ronald Yu, Qiangui Huang, and Ulrich Neumann. Sgpn: Similarity group proposal network for 3d point cloud instance segmentation. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2569–2578, 2018. 3

[43] Xing Wei, Qingxiong Yang, Yihong Gong, Narendra Ahuja, and Ming-Hsuan Yang. Superpixel hierarchy. *IEEE Transactions on Image Processing*, 27(10):4838–4849, 2018. 12

[44] Melonee Wise, Michael Ferguson, Derek King, Eric Diehr, and David Dymesich. Fetch and freight: Standard platforms for service robot applications. 7, 12

[45] Yongxiang Wu, Yili Fu, and Shuguo Wang. Deep instance segmentation and 6d object pose estimation in cluttered scenes for robotic autonomous grasping. *Industrial Robot: the international journal of robotics research and application*, 2020. 1

[46] 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. 2, 3, 8, 16

[47] Yu Xiang, Christopher Xie, Arsalan Mousavian, and Dieter Fox. Learning rgb-d feature embeddings for unseen object instance segmentation. *arXiv preprint arXiv:2007.15157*, 2020. 1

[48] Christopher Xie, Yu Xiang, Arsalan Mousavian, and Dieter Fox. Unseen object instance segmentation for robotic environments. *arXiv preprint arXiv:2007.08073*, 2020. 1

[49] Yujia Xie, Xiangfeng Wang, Ruijia Wang, and Hongyuan Zha. A fast proximal point method for computing exact wasserstein distance. In *Uncertainty in Artificial Intelligence*, pages 433–453. PMLR, 2020. 5

[50] Li Yi, Wang Zhao, He Wang, Minhyuk Sung, and Leonidas J Guibas. Gspn: Generative shape proposal network for 3d instance segmentation in point cloud. In *IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 3947–3956, 2019. 3

[51] 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. 5, 6, 13

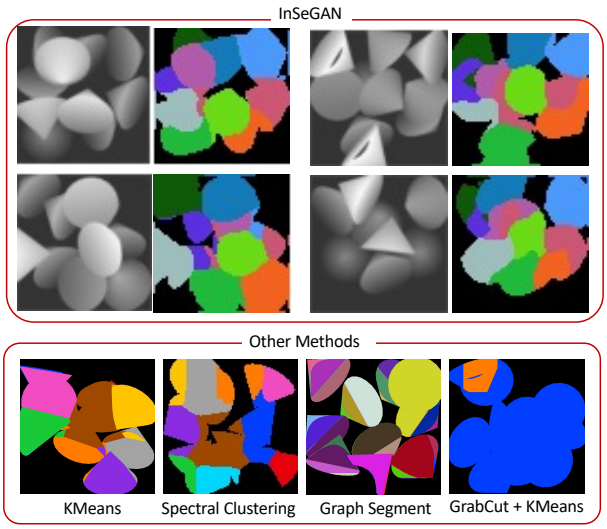


Figure 8. Qualitative results of InSeGAN segmentation on the synthetic 10-cones dataset. We also show example segmentations produced by other methods.

A. 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.

B. 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.

C. 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 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. 9, we show results on the Cone class when we vary the count between 4 and 9. On these data, we achieved 45.1% mIoU.

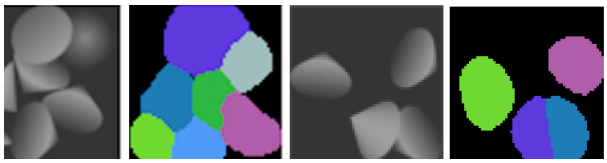


Figure 9. InSeGAN results when we use the same model to learn distributions of images with varying number of instnaces. The results show segmentation visualizations when we used 4–9 instances in each of the depth images.

D. 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

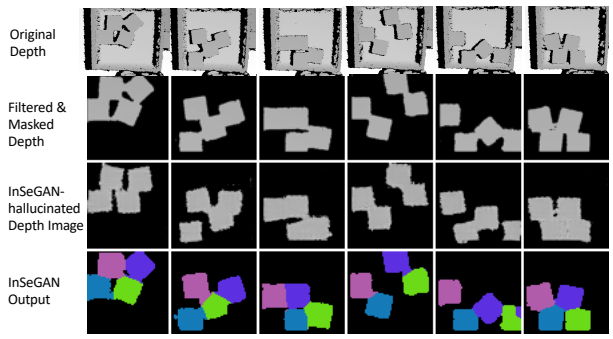


Figure 10. Qualitative results on instance segmentation of real data. We show the original depth images collected using a robot (first row), the output of a masking and filtering step we do to clean up the inputs (second row), the depth images hallucinated/rendered by InSeGAN (third row), and the segmentations (fourth row).

number of instances n). Qualitative results are provided in Figure 8. In Table 4, we quantitatively compare the performance of InSeGAN on this dataset.

Method	mIoU
KMeans	0.302
Spectral Clustering	0.324
Superpixels [43]	0.398
GrabCut+KMeans	0.021
InSeGAN	**0.501**

Table 4. Numerical comparison of InSeGAN vs. other methods on challenging dataset with $n = 10$ cones in each image.

E. Qualitative Results on Real Data

In Figure 10, 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 10). 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).

F. Data Collection Setup

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

F.1. Physics Simulator and Depth Image Generation

As described in the main paper, we use the NVIDIA PhysX physics simulator[4] to create our Insta-10 dataset. A screenshot of this simulator software setup is shown in Figure 12. 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.

F.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 [44] equipped with a 7-degree-of-freedom (7-DOF) arm, and we use ROS [37] 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 $\times$ 720 resolution of the scene with which the Fetch robot interacts. For trajectory planning, we use the Expansive Space Tree (EST) planner [34]. 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. 11. 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

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

Figure 11. Robotic data collection system used to acquire real-world depth images.

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.

G. 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. 13, 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 [51] (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) [19], 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 feature 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.

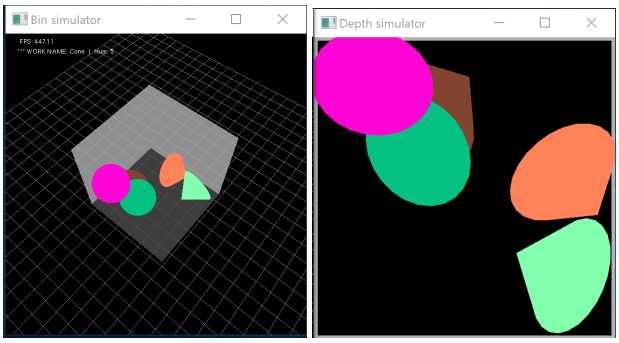


Figure 12. An illustration of the physics simulator that we use to render our synthetic dataset, Insta-10. *Left:* the simulated bin into which the identical objects (e.g., Cone) are dropped. *Right:* The ground-truth instance segmentation masks for each of the instances. We use the depth images associated 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.

The 3D implicit template loosely follows the architecture of a HoloGAN [33], 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. 14, 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 [41] modules.

G.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

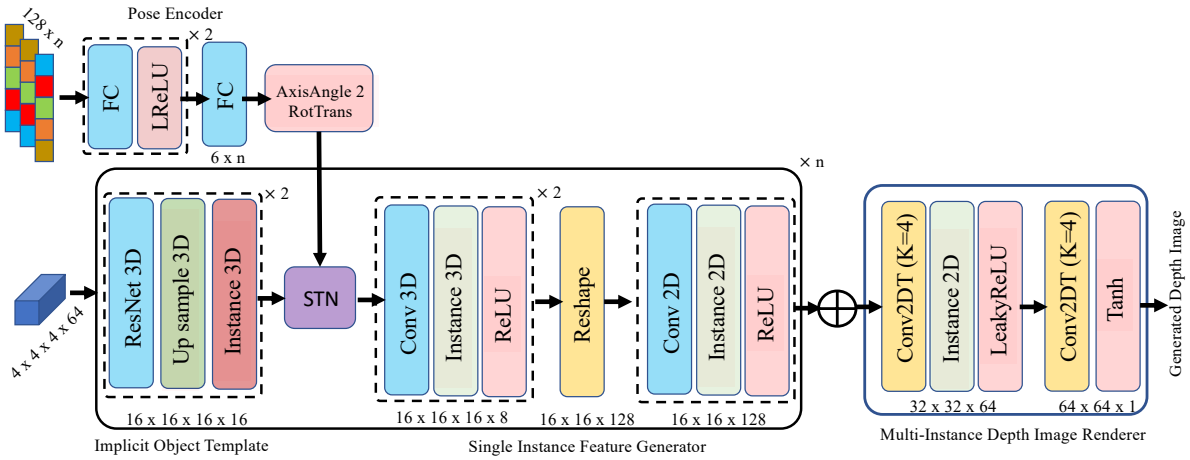
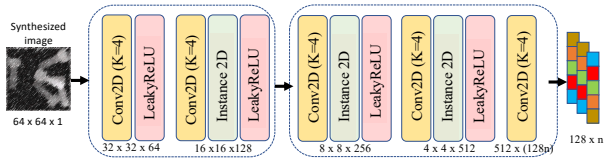
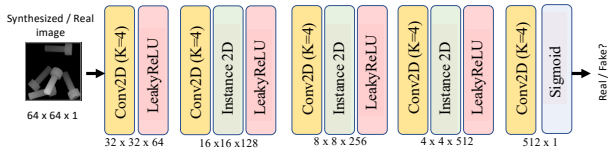


Figure 13. Detailed architecture of InSeGAN generator.



(a) Encoder



(b) Discriminator

Figure 14. (a) depicts detailed architecture of our Encoder module, and (b) shows our Discriminator module.

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.

G.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.

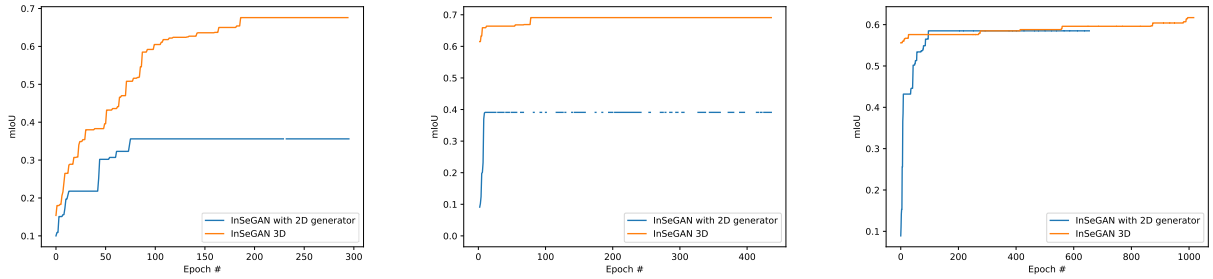
Training: We train our modules for 1000 epochs using a single GPU; each epoch takes approximately 30 seconds on the ∼10,000 training samples for each object. We use the Adam optimizer, with a learning rate of 2×10^{-4}, and $\beta_1 = 0.5$. We use 128-D noise samples from a Normal distribution for the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

H. 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.

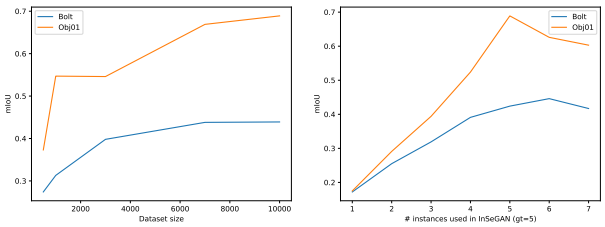
Is 3D Generator Important? An important choice that we made in InSeGAN is the use of a 3D generator instead of a 2D generator. For comparison, we use a standard 2D image-based generator typically used in conditional GANs [31]. Specifically, for the 2D generator, we replace the 3D modules in InSeGAN (i.e., 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. 15, 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. 16(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. 16(a), we plot this performance. As is clear more



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

Figure 15. 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 16. (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).

training data is useful, although this increment appears to be dependent on the object class. In Fig. 18, we show qualitative results of instance segmentations obtained for different training set sizes to gain insights into what the performances reported in Fig. 16(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.

Figure 17. Results using Slot Attention [29].

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. 16(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. 16(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. 19, 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 5, 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 5. mIOU for for different noise levels in the depth images.

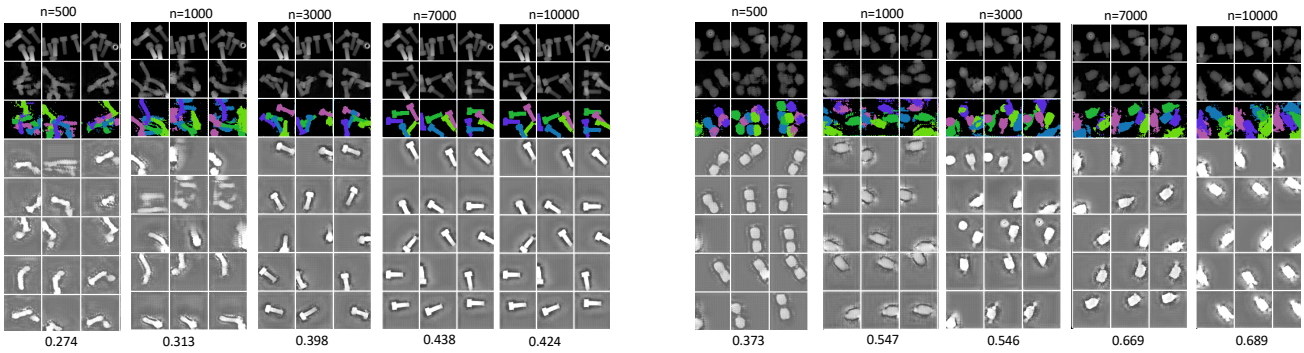


Figure 18. Qualitative instance segmentation results using various data training sizes, for two object classes: Bolt (left) and Obj01 (right). *First row:* input depth image; *second row:* hallucinated depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN. The mIoU on the full test set is shown at the bottom.

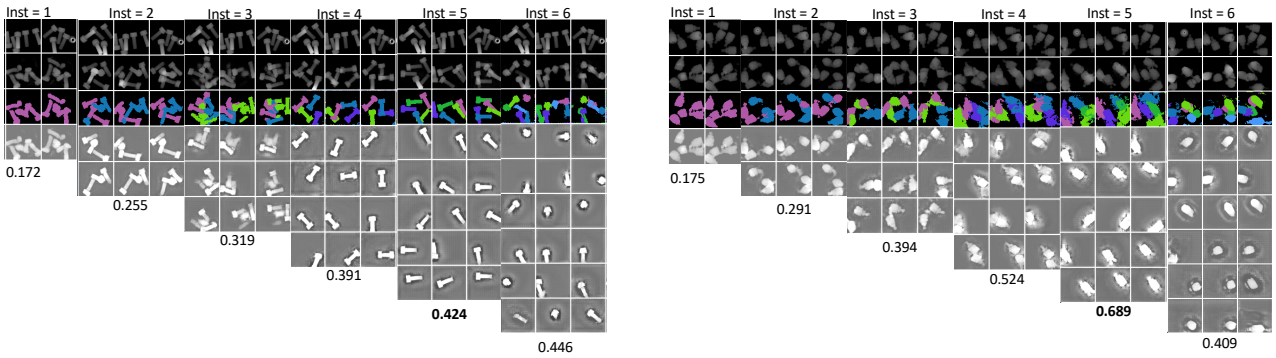


Figure 19. Qualitative instance segmentation results when the number of instances used in InSeGAN is increased, with a fixed ground-truth number of instances ($n = 5$ instances). *First row:* input depth image; *second row:* hallucinated depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN. The mIoU on the full test set is shown at the bottom.

H.1. Qualitative Comparisons

In Fig. 20, 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. [46], 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 [9]. 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 (that is, until no change in the objective was found). As is clear from Fig. 20, InSeGAN produces more reasonable segmentations than other methods. In contrast, InSeGAN, via modeling the 3D shape of the objects, leads to significant benefits in challenging segmentation settings. In Fig. 17, we show qualitative results using the recent Slot Attention method [29] for the cone class with 5 and 10 instances.

H.2. Qualitative Results

In Figure. 21, we show several more qualitative results for each of the 10 object classes in Insta-10.

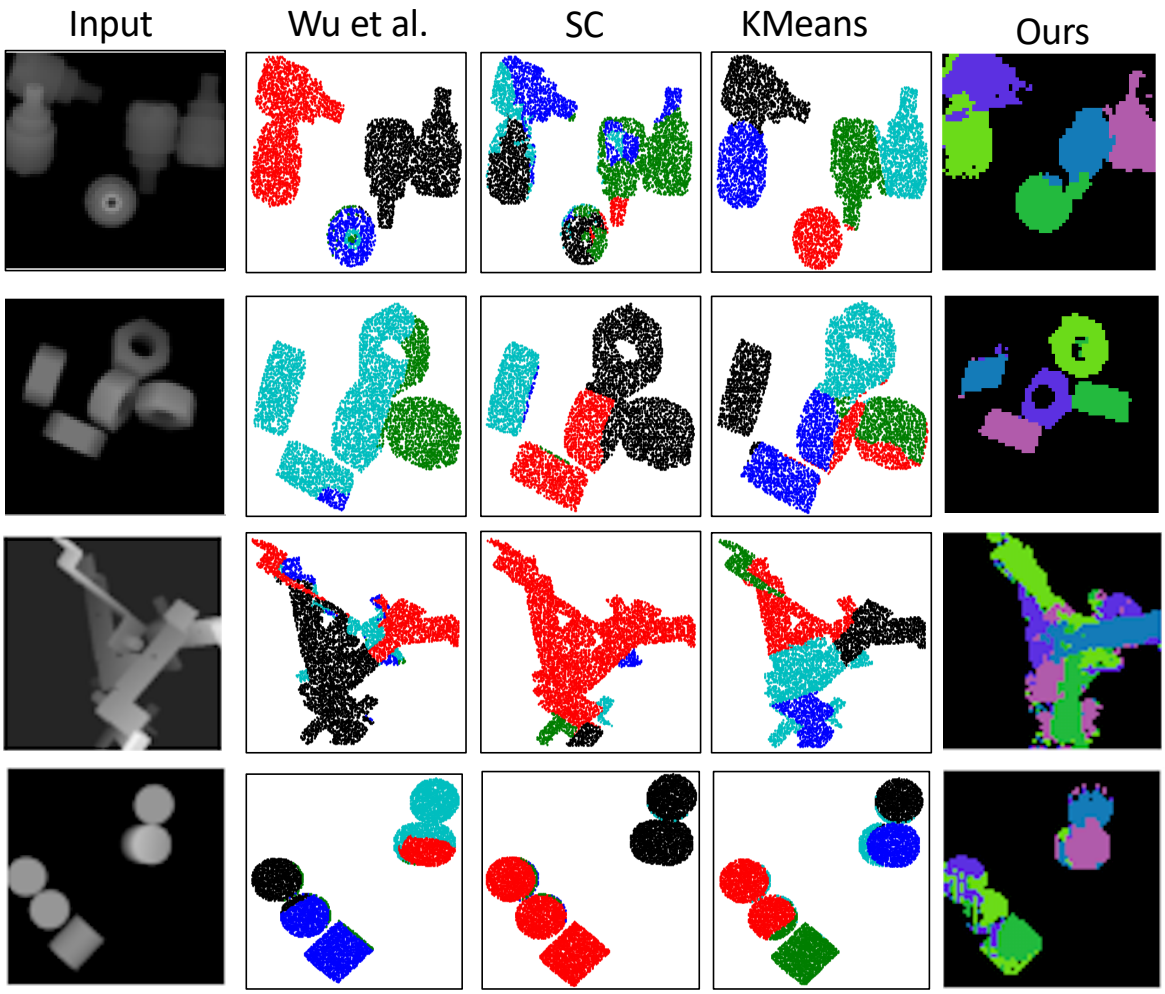


Figure 20. Qualitative comparisons of results from InSeGAN against other methods. On the right, we show a sample result from a segmentation from the competitive method IODINE [9] (using their code on our data).)

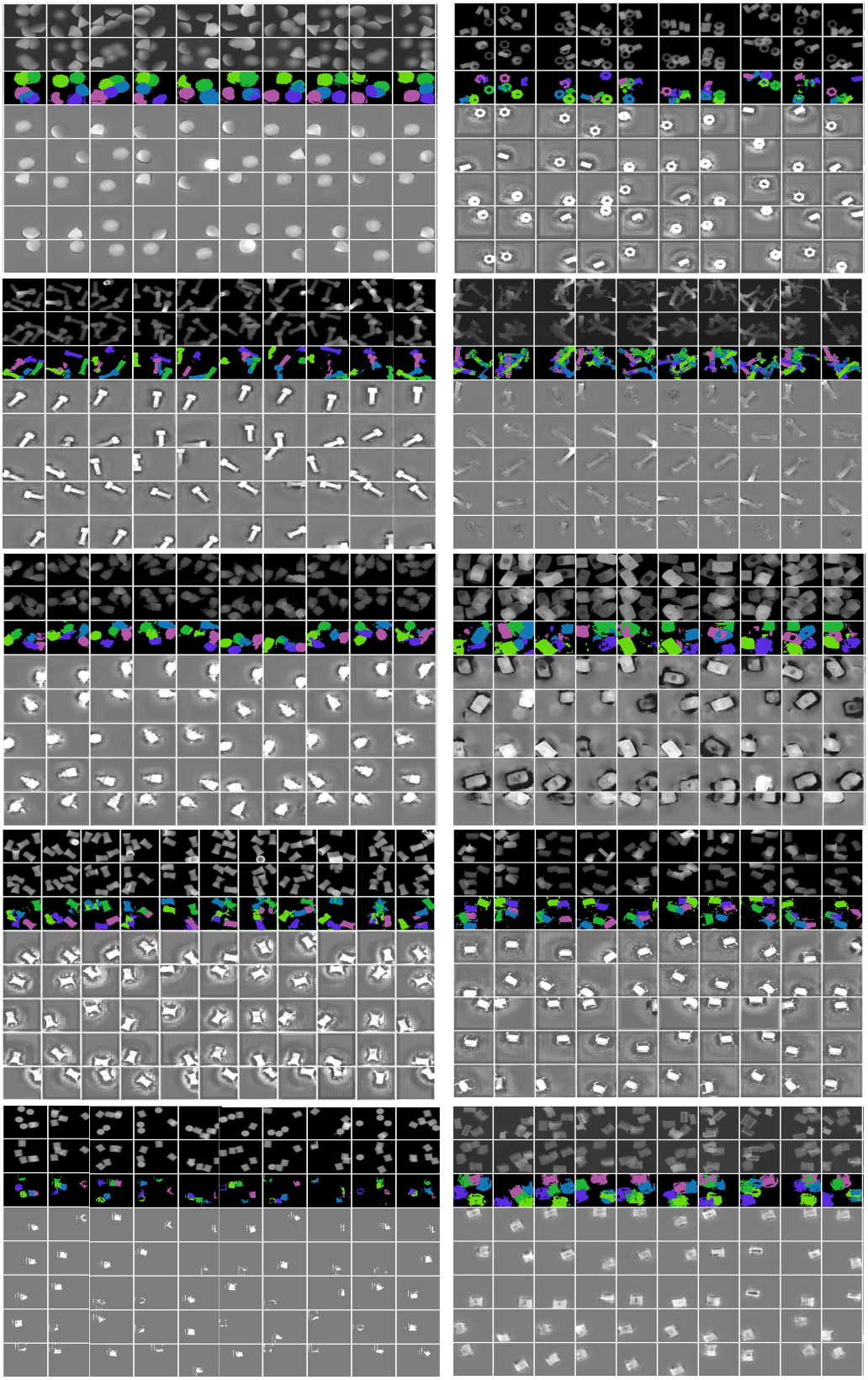


Figure 21. Qualitative results using InSeGAN on the 10 object classes in Insta-10. We show 10 segmentation results for each class. *First row:* input depth image; *second row:* hallucinated (reconstructed) depth image by InSeGAN; *third row:* inferred instance segmentation; *fourth row onwards:* the single instances hallucinated by InSeGAN.