INFERNO: INFERRING OBJECT-CENTRIC 3D SCENE REPRESENTATIONS WITHOUT SUPERVISION

Anonymous authors
Paper under double-blind review

ABSTRACT

We propose INFERNO, a method to infer object-centric representations of visual scenes without annotations. Our method decomposes a scene into multiple objects, with each object having a structured representation that disentangles its shape, appearance and pose. Each object representation defines a localized neural radiance field used to generate 2D views of the scene through differentiable rendering. Our model is subsequently trained by minimizing a reconstruction loss between inputs and corresponding rendered scenes. We empirically show that INFERNO discovers objects in a scene without supervision. We also validate the interpretability of the learned representations by manipulating inferred scenes and showing the corresponding effect in the rendered output. Finally, we demonstrate the usefulness of our 3D object representations in a visual reasoning task using the CATER dataset.

1 INTRODUCTION

Inferring objects and their geometry in a scene is a fundamental ability of biological visual systems (Kahneman et al., 1992; Roelfsema et al., 1998). Replicating this ability in machines is a promising step for several applications such as object manipulation, navigation or forecasting. Recent works (Locatello et al., 2020; Burgess et al., 2019) have shown that neural networks can learn object-centric representations from low-level features. They learn to recognize the objects in a visual scene from a singe image without supervision. However, most of those approaches consider 2D representations and ignore the underlying 3D geometry of scenes. On the other hand, Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) combine differentiable renderers with gradient-based optimization to learn high-fidelity 3D scene reconstructions. NeRFs have been subsequently used in 3D-aware generative models, including compositional scene models (Niemeyer & Geiger, 2021).

In this work, we leverage these advances in object-centric representation learning and 3D modelling and propose INFERNO, a model which infers a structured representation of objects and their poses from a single image. Each object is represented by latent variables characterizing its appearance, together with an explicit representation of their poses (translation, scale and rotation). The object representations are then decoded using implicit functions that are localized in the scene according to the objects poses and combined together to generate a 2D output view. Our model does not need supervision and instead is fitted by minimizing a reconstruction loss, akin to an autoencoder.

Disentangling the object appearance and pose in a scene representation allows for scene manipulations. We demonstrate that INFERNO learns interpretable object poses, which we can modify and render to alter the pose of an object in a scene. We also validate that our approach learns meaningful representations for object discovery and visual reasoning. More specifically, we show that our approach is competitive on the CLEVR6 object discovery benchmark (Greff et al., 2019) as well as for the snitch localization visual reasoning task of CATER (Girdhar & Ramanan, 2019).

In summary, our contributions are the following:

- We propose a model able to infer and render 3D scene representations composed of multiple objects, each of them modeled by an implicit function and explicitly localized in the scene.
- We show that the learned representations are interpretable and amenable to manipulations.
- We demonstrate that the inferred representations are useful for downstream tasks by showing competitive performance in object discovery and reasoning tasks.

2 METHOD

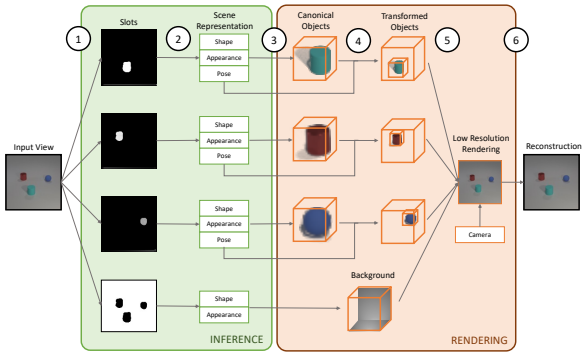

Figure 1: **Model Overview:** We propose INFERNO, a model that infers and renders object-centric 3D scene representations. **1** Our model first decomposes an input observation into multiple object slots. **2** For each slot we infer a structured 3D representation. **3** The shape and appearance determine canonical objects rendered through NeRFs. **4** Objects are transformed and located in the overall scene according to their pose. **5** We combine objects and background and render a low-resolution scene given a camera location. **6** (Optionally) The output of rendering is scaled up to full resolution.

We propose INFERNO (Infer NeRF Objects). The goal of our method is to infer object-centric 3D scene representations from single 2D views. Given an image $x \in \mathbb{R}^{H \times W \times 3}$, we learn an inference function f_θ that maps images to scene representations $s = f_\theta(x) = (o_1, o_2, ..., o_K, o_{bg}, c)$. Scenes are composed of K objects o_i, a background object o_{bg} and a camera location c.

Each object is composed of three tensors $o_i = (o_i^{shape}, o_i^{app}, o_i^{pose})$. The object shape $o_i^{shape} \in \mathbb{R}^{D_{shape}}$ and object appearance $o^{app} \in \mathbb{R}^{D_{shape}}$ are tensors that respectively describe the shape occupancy and color of an object with an implicit function. The object location $o_i^{pose} \in \mathbb{R}^{4\times4}$ is an affine matrix that describes the object pose (i.e. scale, translation and rotation) in the scene. The background object $o^{bg} = (bg^{shape}, bg^{app})$ only models shape and color, and its location is fixed, encompassing the *back-of-scene* cube. We also define a camera matrix $c \in \mathbb{R}^{3\times4}$, that determines the location of the scene camera and defines a 2D projection of the 3D scene.

To optimize our inference function we minimize a reconstruction loss over a dataset, similar to an auto-encoder. We define a rendering function g_γ that takes as input a scene representation and generates a 2D view of that scene $\hat{x} = g_\gamma(f_\theta(x))$. We assume a isotropic Gaussian likelihood model with unit covariance and optimize the probability of the data under our model, which is equivalent to minimizing the mean squared error of inputs and reconstructions: We describe our inference mechanism, rendering pipeline and implementation in the Appendix.

3 RELATED WORK

3.1 3D SHAPE REPRESENTATIONS

There are different ways to represent 3D geometry such as voxels or meshes (Rematas & Ferrari, 2020; Gkioxari et al., 2019). Voxel-based methods have trouble scaling up to high resolutions as the size of a voxel representation scales cubically with the resolution. Recently, the use of functions that implicitly model 3D volumes has gained popularity (Park et al., 2019; Mescheder et al., 2019; Sitzmann et al., 2020b;a). Implicit representations have better scaling properties, as usually the output resolution does not directly affect the dimensionality of the learned function. NeRFs (Mildenhall et al., 2020) generate scenes by learning a function that outputs the occupancy and color of points in a scene when viewed from a particular direction. NeRFs have obtained superior reconstructions compared to other implict methods, and our model uses NeRFs to represent multiple objects and the background of a scene. Most methods using NeRFs represent scenes monolithically as a single entity. GIRAFFE (Niemeyer & Geiger, 2021) is a GAN-based method that represents multiple objects in a scene with NeRFs. Our model uses a rendering pipeline inspired by GIRAFFE. However, we

focus on recovering scene representations from existing images, while GIRAFFE does not have an inference mechanism. ObjSURF (Stelzner et al., 2021) and UORF (Yu et al., 2021) infer scene representations composed of multiple objects, each object represented with a differently instantiated NeRF. Different from our work, they focus on novel view generation and do not explicitly infer the pose of the different objects in the scene. Furthermore, both methods require multiple scene views and their associated ground-truth camera locations, and ObjSURF requires depth annotations.

3.2 Object-Centric Scene Models

Recently there is a line of work on object-centric generative models of scenes (Kosiorek et al., 2021; Burgess et al., 2019; Locatello et al., 2020). These models learn to generate scenes as a composition of multiple objects and a background. MONet (Burgess et al., 2019) implements a multi-object VAE that segments object sequentially by infering latents corresponding over parts of the scene not yet attended to iteratively. Slot-Attention (Locatello et al., 2020) maps a set of entities, called slots, to image features through multiple rounds of soft attention. The slots compete among themselves to attend to features, making each slot attend to a region of the input image. Our model uses a variant of Slot Attention to decide on which part of a 2D view should each object in our scene attend to, but we infer 3D-aware representations for each object. Object-centric scene models have also been implemented as world models, with the goal of simulating dynamics (Lin et al., 2020). By decomposing the scene into objects, these methods can simulate dynamics at an object level, which are usually simpler and shared among objects. Contrary to most previous approaches which segment 2D shapes, our method infers object-centric scene representations in 3D space.

4 Experiments

In this section we showcase the capabilities of INFERNO with three main experiments. First, we demonstrate the interpretability of the scene representations through manipulating scenes and verifying the corresponding effects in the rendered outputs. Then we show that it learns to identify and segment the objects in a scene without supervision. Finally, we highlight the usefulness of such representations for downstream tasks by applying our model on the CATER snitch localization task.

4.1 Training Setup

We train our models for 400K iterations using the Adam optimizer Kingma & Ba (2014) and 1×10^{-4} learning rate. We use a batch size of 32 and one nVidia V100 GPU. We use learning rate warmup (Goyal et al., 2017) to avoid optimization issues with Slot Attention and we use a weight decay rate of 1×10^{-6}. We use three iterations of slot attention during training and evaluation. We remove the neural upscaler and render at full resolution after 100K iterations. We set the number of objects in a scene as the maximum possible number of objects in a dataset, i.e. 5 objects for CLEVR2345 and 10 for CATER. Refer to Appendix B for more details on the experimental setup.

4.2 Scene Inference

Table 1: **Reconstruction error on CLEVR2345** We compare INFERNO to an autoencoder baseline with a single NeRF object (NeRF-AE). Our model produces better reconstructions under all metrics and allows for object identity/pose manipulations.

Model	MSE (↓)	PSNR (↑)	SSIM (↑)	LPIPS (↓)
NeRF-AE	5.14×10^{-4}	44.89	59.24 %	168.9×10^{-3}
Ours	1.22×10^{-4}	52.07	72.4 %	18.93×10^{-3}

In this section we demonstrate the properties of our scene representation. Our model infers object-centric scene representations from single 2D views. These representations disentangle the appearance and pose of objects, which allows for semantic manipulations of the scene not possible otherwise. These manipulations can be validated by rendering the modified scene representations. Additionally, we verify that decomposing the scene into multiple objects leads to better reconstructions.

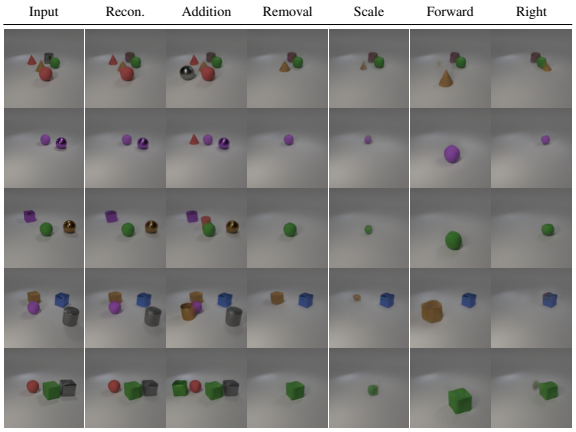

Figure 2: **Manipulated scenes on CLEVR2345**

Table 2: **FID on CLEVR2345**

Model	FID (↓)
GIRAFFE	37.7
Ours - Reconstruction	23.5
Ours - Remove Object	42.4
Ours - Add Object	27.2
Ours - Swap Object	**23.7**

First, we verify the quality of INFERNO's generations by comparing them two baselines: i) a version of our model that does not consider multiple objects, and ii) the GAN method of GIRAFFE (Niemeyer & Geiger, 2021). We perform this comparison on the CLEVR-2345 dataset introduced by GIRAFFE, which contains CLEVR images with 2 to 5 objects. We compare reconstruction using MSE, PSNR and SSIM and we compare populations of generations using the Frechet Inception Distance (FID).

In INFERNO, we generate novel scenes by inferring representations for ground-truth images and then manipulating them. To compare to GIRAFFE, we manipulate scenes by adding additional objects and swapping object shapes and appearances across scenes. Manipulations are described in more detail in the Appendix. We highlight that GIRAFFE is an unconditional model, while INFERNO generates novel scenes conditioned on existing ones. We also investigate different interpretable manipulations of scene representations and visualize the effects in the corresponding output renderings. We also conduct this experiment on the CLEVR-2345 dataset. We validate that our model is able to render out-of-distribution scenes not corresponding to training examples, such as scenes having 1 or 6 objects, and verify that the pose manipulations have semantically coherent effects.

Table 1 shows the reconstruction metrics obtained by our model and baseline on the CLEVR2345 dataset. Note that GIRAFFE is a GAN-method that does not have an inference mechanism, and therefore it cannot reconstruct scenes. We observe that the NeRF-AE baseline obtains higher reconstruction error than our regular model, as our object-centric method can make better use of its capacity. In Table 2 we compare INFERNO with GIRAFFE using the FID metric. Our model reconstructions have better FID than the generations of the GIRAFFE. Additionally, our model can perform inference and manipulations on existing scenes. We use that capability to generate novel scenes by manipulating existing ones. Our model is able to generate novel scenes with additional objects or with altered object shapes and appearances, with better FID than GIRAFFE. In Figure 2 we show examples of the manipulations possible with our model. Given a scene representation, we can add, remove or rearrange objects, change their locations or change the object scales. While some of these manipulations can be performed with 2D object-centric models, modifications to the scale and location of the objects are hard to implement without explicitly modeling 3D object poses.

4.3 Object Discovery

Unsupervised object discovery consists in segmenting the objects in a scene without using annotations. We test our model on CLEVR6, a variant of CLEVR with scenes of up to 6 objects annotated with 2D object masks. We choose this dataset to compare to previous work in unsupervised object discovery. Note that this setup evaluates 2D segmentation masks, although our model naturally provides 3D segmentations. We evaluate the quality of the segmentations using the Adjusted Random Index (ARI) metric (Rand, 1971), which is a measure of clustering similarity. In particular, we consider objects as different clusters and compare the cluster assignment of each pixel in the image to its prediction.

Table 3 reports ARI for different models. INFERNO is competitive with state-of-the-art methods, surpassing MONet and having slightly lower ARI than Slot-Attention and IODINE. However, our

Table 3: **Object Discovery on CLEVR6** Despite inferring 3D object poses, our model is competitive with state-of-the-art 2D object discovery methods.

Model	ARI % (↑)
Slot-Attention	98.8 ± 0.3
IODINE	98.8 ± 0.0
MONet	96.2 ± 0.6
Slot MLP	60.4 ± 6.6
Ours	96.7 ± 0.2

Table 4: **Snitch Localization on CATER**. Our model outperforms the R3D models and is competitive with the state-of-the-art specialized models, without being specifically designed for this task.

	Model	Top-1	Top-5
Specialized	R3D LSTM	60.2	81.8
	Hopper	73.2	93.8
	Aloe (w/out SSL loss)	60.1	-
	Aloe	**74.0**	**94.0**
Fine-tune	Slot-Attention	59.1	88.0
	Ours (w/out pretraining)	2.91	12.9
	Ours (w/out SSL loss)	69.17	87.68
	Ours	**71.7**	**88.9**

model learns to segment objects in 3D, while the other baselines extract 2D segmentation masks. A visual example of the object masks infered by our model can be found in Figure 3 in the Appendix.

4.4 Snitch Localization

The goal of this experiment is to show that the representation learned by our model is useful for the snitch localization task from CATER (Girdhar & Ramanan, 2019). We follow the experimental setup of Ding et al. (2020). First, we train INFERNO to reconstruct images from the CATER dataset. Then we discard its rendering pipeline, and use the scene representations as input to a 12-layer transformer to predict the final snitch position. Each object in our representation is an input token to the transformer. The last output of the transformer is fed to a MLP head that predicts the logits for the 36 possible output positions. We minimize the sum of the cross-entropy and a L1 loss between the predicted and the true snitch final position. Following (Ding et al., 2020), we optionally use an auxiliary SSL loss after pretraining. During training, we sample 40 frames from a video and predict the snitch location from these frames. At test time, we randomly sample 10 temporal crops of 40 frames each and average the model predictions to compute the final probabilities.

Table 4 reports CATER Top-1 and Top-5 accuracies for different methods. We first compare our model pretrained to reconstruct images on CATER with a randomly initialized encoder. Using a randomly initialized encoder does not perform well, suggesting that the pretraining rather than the encoder architecture is key to learn useful representations for the task. We also observe that the additional SSL loss slightly improves the performance of our model. We next compare our approach to methods designed for CATER including Aloe (Ding et al., 2020), R3D (Girdhar & Ramanan, 2019), and Hopper (Zhou et al., 2021). Our model outperforms the R3D model, indicating that our representation is useful for visual reasoning tasks. Our model outperforms Aloe, an object-centric method using 2D object representation, when both methods do not use additional SSL loss. Aloe benefits more from the use of an additional SSL loss. Overall, INFERNO achieves performances close to the state-of-art specialized approaches. We also evaluate the performance of a slot-attention baseline (Locatello et al., 2020) in Table 4. The slot-attention baseline first pretrains a slot-attention encoder, with a similar architecture than our model, by reconstructing CATER frames using a mask decoder, and then fine-tunes the encoder as in our setup. Our model significantly outperforms the slot-attention model, supporting the advantage of using 3D-aware representations to solve CATER.

5 Conclusions and Future Work

We propose INFERNO, a model for inferring object-centric 3D scene representations. Our model is able to discover objects in a scene without annotations, and the inferred scene representations are interpretable and amenable to manipulations. Further, the scene representation is useful for visual reasoning downstream tasks such as the snitch localization task in CATER. While our model can render novel scenes from modified scene representations, currently it has issues inferring the hidden sides of objects or scaling to more complex datasets, which we will address in future work.

REFERENCES

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.

David Ding, Felix Hill, Adam Santoro, and Matt Botvinick. Object-based attention for spatio-temporal reasoning: Outperforming neuro-symbolic models with flexible distributed architectures. *arXiv preprint arXiv:2012.08508*, 2020.

Vincent Dumoulin, Ethan Perez, Nathan Schucher, Florian Strub, Harm de Vries, Aaron Courville, and Yoshua Bengio. Feature-wise transformations. *Distill*, 3(7):e11, 2018.

Rohit Girdhar and Deva Ramanan. Cater: A diagnostic dataset for compositional actions and temporal reasoning. *arXiv preprint arXiv:1910.04744*, 2019.

Georgia Gkioxari, Jitendra Malik, and Justin Johnson. Mesh r-cnn. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 9785–9795, 2019.

Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv preprint arXiv:1706.02677*, 2017.

Klaus Greff, Raphaël Lopez Kaufman, Rishabh Kabra, Nick Watters, Christopher Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. In *International Conference on Machine Learning*, pp. 2424–2433. PMLR, 2019.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

Daniel Kahneman, Anne Treisman, and Brian J Gibbs. The reviewing of object files: Object-specific integration of information. *Cognitive psychology*, 24(2):175–219, 1992.

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

Adam R Kosiorek, Heiko Strathmann, Daniel Zoran, Pol Moreno, Rosalia Schneider, Soňa Mokrá, and Danilo J Rezende. Nerf-vae: A geometry aware 3d scene generative model. *arXiv preprint arXiv:2104.00587*, 2021.

Zhixuan Lin, Yi-Fu Wu, Skand Peri, Bofeng Fu, Jindong Jiang, and Sungjin Ahn. Improving generative imagination in object-centric world models. In *International Conference on Machine Learning*, pp. 6140–6149. PMLR, 2020.

Francesco Locatello, Dirk Weissenborn, Thomas Unterthiner, Aravindh Mahendran, Georg Heigold, Jakob Uszkoreit, Alexey Dosovitskiy, and Thomas Kipf. Object-centric learning with slot attention. *arXiv preprint arXiv:2006.15055*, 2020.

Lars Mescheder, Michael Oechsle, Michael Niemeyer, Sebastian Nowozin, and Andreas Geiger. Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

Ben Mildenhall, Pratul P Srinivasan, Matthew Tancik, Jonathan T Barron, Ravi Ramamoorthi, and Ren Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. In *European conference on computer vision*, pp. 405–421. Springer, 2020.

Michael Niemeyer and Andreas Geiger. Giraffe: Representing scenes as compositional generative neural feature fields. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11453–11464, 2021.

Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 165–174, 2019.

William M Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical association*, 66(336):846–850, 1971.

Konstantinos Rematas and Vittorio Ferrari. Neural voxel renderer: Learning an accurate and controllable rendering tool. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5417–5427, 2020.

Pieter R Roelfsema, Victor AF Lamme, and Henk Spekreijse. Object-based attention in the primary visual cortex of the macaque monkey. *Nature*, 395(6700):376–381, 1998.

Katja Schwarz, Yiyi Liao, Michael Niemeyer, and Andreas Geiger. Graf: Generative radiance fields for 3d-aware image synthesis. *arXiv preprint arXiv:2007.02442*, 2020.

Vincent Sitzmann, Eric R Chan, Richard Tucker, Noah Snavely, and Gordon Wetzstein. Metasdf: Meta-learning signed distance functions. *arXiv preprint arXiv:2006.09662*, 2020a.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, 33, 2020b.

Karl Stelzner, Kristian Kersting, and Adam R Kosiorek. Decomposing 3d scenes into objects via unsupervised volume segmentation. *arXiv preprint arXiv:2104.01148*, 2021.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

Hong-Xing Yu, Leonidas J Guibas, and Jiajun Wu. Unsupervised discovery of object radiance fields. *arXiv preprint arXiv:2107.07905*, 2021.

Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 586–595, 2018.

Honglu Zhou, Asim Kadav, Farley Lai, Alexandru Niculescu-Mizil, Martin Renqiang Min, Mubbasir Kapadia, and Hans Peter Graf. Hopper: Multi-hop transformer for spatiotemporal reasoning. *arXiv preprint arXiv:2103.10574*, 2021.

A ADDITIONAL MODEL DETAILS

Our model is composed of inference and rendering pipelines, with five main modules: encoder, slot attention, slot to object mapping, NeRF decoder and neural network upscaler. In this section we provide additional details about each part of the model as well as these modules and describe their architecture.

A.1 RENDERING PIPELINE

We represent objects as Neural Radiance Fields (NeRFs) (Mildenhall et al., 2020) with a similar setup as that of GIRAFFE (Niemeyer & Geiger, 2021). A NeRF is a function g_τ that defines a 3D shape implicitly. It takes as input a 3D location $\mathbf{l} = (x, y, z)$ and a 2D viewing direction $\mathbf{d} = (\psi, \phi)$ and outputs an occupancy value σ and a color value $\mathbf{a} = (r, g, b)$. NeRFs are usually implemented using fully connected neural networks. Additionally, the inputs are usually embedded into a higher-dimensional space using positional encodings γ that embed locations and viewing directions into higher dimensional spaces $\mathbb{R}^{P_l}$ and $\mathbb{R}^{P_d}$, respectively.

$$\begin{aligned} g_\tau : \mathbb{R}^{P_l} \times \mathbb{R}^{P_d} &\rightarrow \mathbb{R}^+ \times \mathbb{R}^3; \\ (\gamma(\mathbf{l}), \gamma(\mathbf{d})) &\rightarrow (\sigma, \mathbf{a}) \end{aligned} \tag{1}$$

To represent multiple shapes with the same NeRF function, we can augment it with latent variables that determine which shape is being modeled (Schwarz et al., 2020). NeRFs are usually augmented with two random variables: one random variable $\mu \in \mathbb{R}^{D_{shape}}$ defines the shape of the entity being modeled, while $\upsilon \in \mathbb{R}^{D_{app}}$ models its appearance. In practice, this specialization is enforced by making the occupancy output a function of only the shape latent, while the color output is conditioned on the appearance latent.

$$\begin{aligned} g'_\tau : \mathbb{R}^{P_l} \times \mathbb{R}^{P_d} \times \mathbb{R}^{D_{shape}} \times \mathbb{R}^{D_{app}} &\rightarrow \mathbb{R}^+ \times \mathbb{R}^3; \\ (\gamma(\mathbf{l}), \gamma(\mathbf{d}), \mu, \upsilon) &\rightarrow (\sigma, \mathbf{a}) \end{aligned} \tag{2}$$

In INFERNO, we share a single parametrization of a NeRF function across all objects. Each object specific shape and appearance is defined by the shape and appearance latent variables, which correspond to the object attributes o^{shape}, o^{app}. The background is defined as another NeRF with separate parameters. The background NeRF also has shape and appearance latent variables to model different backgrounds.

The pose of an object o_i in the scene is determined by the affine transformation matrix o_i^{pose}. We denote the coordinate system of the NeRF function of an object as the object space, and the coordinate system of the scene (and the background NeRF) as scene space. Given an object pose, we can convert points from the scene space to the object space by applying the o^{pose} transformation matrix on those points, and we can transform points from object space to scene space by computing the inverse of the object pose matrix.

To render a scene, we cast rays from each pixel in the 2D plane defined by a given camera to the 3D scene. We evaluate NeRFs at different points along a given ray, and integrate their occupancy and color outputs to determine pixel values. Rays might traverse multiple object NeRFs in addition to the background NeRF. To determine the occupancy and color of points described by multiple NeRFs, we first query each NeRF at those particular points. To query the object NeRFs, we first need to transform the points from scene space to the particular object space. Then, we compose the results of each NeRF with a pooling function C, which in our case is a weighted average:

$$C(l, \mathbf{d}) = (\sigma = \sum_{i=1}^{N} \sigma_i, \frac{1}{\sigma} \sum_{i=1}^{N} \sigma_i \mathbf{a}_i) \tag{3}$$

Since rendering with NeRFs as originally proposed is computationally expensive, at the beginning of training we render output views at a fixed low resolution. Low resolution scenes are then upscaled

to the desired output resolution using a convolutional neural network, keeping the entire rendering pipeline differentiable. Additionally, low resolution scenes are rendered with additional channels, allowing for more detailed upscalings beyond those possible when just rendering low resolution RGB outputs. After some iterations and once the model is capable of reconstructing the input view, we remove the neural network upscaler and render with NeRFs at full resolution. This second stage of training has slower iteration times and higher memory requirements.

A.2 Inference Mechanism

Given the rendering pipeline, the goal of our method is to infer representations that reconstruct a given scene. Our inference mechanism computes image features through a neural network encoder and then extracts K object and a background slots. These slots are then mapped to our structured scene representation through learned neural networks.

To extract image features for each object and background slots, we use Slot Attention (Locatello et al., 2020). Slot Attention is a mechanism that maps a set of K entities, called slots, to image features without annotations. It extracts image features $I \in \mathbb{R}^{H \times W \times D}$ from a given input using a resolution-preserving convolutional encoder. These features are then attended to by a set of K randomly sampled slots $\pi_j \sim \mathcal{N}(\mu, \sigma)$ of dimension D, where μ and σ are learnable parameters. We denote by π the matrix concatenating all the sampled slots. Slots attend spatial chunks of the input features I through soft-attention $u = TQ^T$, where $T = k(I)$ and $Q = q(\pi)$ are the embeddings of the inputs and slots respectively. The attention weights are normalized through a softmax operating on the slots axis, which makes slots compete among themselves and discourages multiple slots from attending the same input region $w = softmax(u)$. The weighted average of I according to the attention weights is then computed and fed to a GRU network, to update each slot value: $\pi_j = GRU(w_j * I, \pi_j)\ \forall j \in 1, K$. Multiple rounds of soft attention are performed to iteratively refine the slots. For more details about Slot-Attention, please refer to (Locatello et al., 2020).

Object slots are unstructured tensors that result from aggregating image features. We map these slots to our structured scene representation through small fully-connected networks that operate on individual objects. More concretely, we map object slots to their 3D pose in the scene through a 2-layer MLP. To infer the object shape and appearance tensors we also use 2-layer MLPs, but we make them conditional on the predicted object pose through conditional normalization (Dumoulin et al., 2018).

A.3 Model Architecture

Table 5: **Encoder Neural Network**

Layer Type	Size	Normalization	Activation	Other details
Conv 5×5	64	-	ReLU	Stride 1 Pad. 1
Conv 5×5	64	-	ReLU	Stride 1 Pad. 1
Conv 5×5	64	-	ReLU	Stride 1 Pad. 1
Conv 5×5	64	-	ReLU	Stride 1 Pad. 1

Encoder The goal of the encoder is to extract image features. We use an encoder with no downsampling, as it is typically used with Slot Attention. Details about the encoder architecture are described in Table 5.

Slot Attention We employ Slot Attention to map image features to object slots. Slots are sampled randomly from a Gaussian distribution with learned parameters. We use different distributions for the background slot and the object slots. During training we employ three iterations of slot attention to refine the image features to slot assignments.

Slot to Object Net The background and object slots are mapped to scene parameters using a series of MLP. For each object slot, we first map the object to its pose parameters. We use a 2-layer MLP with LayerNorm to map a slot to its pose. The size of the hidden dimension is the same than the output

Table 6: **Slot Attention Neural Network**

Name	Size	Description
Positional emb.	64	Additive embedding, same size as CNN input features
Flatten	-	Flattens the spatial dimensions of CNN features
QKV MLP	128	Linear layers that map slots and input features to the same dimension
LayerNorm	128	Normalizes the slots/inputs
MLP + GRU	128	The output of soft-attention goes through a linear layer + GRU

Table 7: **Slot to Object MLP details**

MLP Name	Size	Act and Norm.	Description
Obj Pose	7	ReLU, LayerNorm	Slot to translation, scale and rotation
Obj Shape/App.	128	ReLU, CondLayerNorm	Slot to shape and appearance
BG Shape/App.	128	ReLU, LayerNorm	Background slot to its shape/app, fixed pose.

dimension size. We parametrize object pose as a 7-dimensional tensor. We use three dimensions for the object location along each axis, three dimensions for the scale of the object and a single dimension to express a rotation of the object along the X axis. These parameters are then mapped to their corresponding 4×4 affine transformation matrix for each object. Note that in practice we are not modeling rotations in the experimental section.

Once we have inferred the object poses, we infer object shapes and appearances. These are inferred individually for each object using a common 2-layer MLP. The MLPs are conditional on the object pose using Conditional Layer Normalization, that makes the learned parameters of LayerNorm be a function of a condition. Specifically, we map the 7-dimensional pose tensor to LayerNorm parameters with a single linear layer with no activation or normalization. Shapes and appearances are defined by the output of the MLPs, which produce two 128-dimensional tensors.

For the background object we only infer shape and appearance, and define its pose to be that of the scene cube. The shape and appearance of the background are inferred through another 2-layer MLP with ReLU and LayerNorm.

Note that our scene representation also admits a camera pose. In our experiments we fix the camera location to look at the scene from a standard location (centered and 33 degrees above the Z plane). Other concurrent approaches (Yu et al., 2021; Stelzner et al., 2021) use ground-truth camera locations to generate novel views, while we focus on recovering scene representations without the use of ground-truth annotations.

Table 8: **Details about the NeRF MLPs used**

MLP Name	Layers	Size	Description
Obj MLP	8	64	ReLU activation, no norm. Skip connection with layer 4.
BG MLP	4	16	ReLU activation , no norm.

NeRF MLPs With the object shape and appearance tensors we can render them following GRAF (Schwarz et al., 2020). We use one NeRF for the objects and one NeRF MLP for the background. The details about each NeRF architecture can be found in Table 8. Note that, to render objects according to their pose, we query their NeRF MLP in a canonical object space by transforming input coordinates in scene space to object space using the object pose. To reduce the computational complexity of rendering with many NeRFs, we render scenes at a fixed resolution of 16x16. Instead of rendering RGB pixels, we render feature images with 128 channels. The output of the rendering is then upscaled and mapped to RGB views with a neural network upscaler.

Table 9: **Neural Upscaler architecture**

Layer	Size	Activation	Normalization	Other
Conv 3×3	64	ReLU	Instance	Stride=1 Pad=1
Upsample	-	-	-	Nearest Neighbors
Conv 3×3	64	ReLU	Instance	Stride=1 Pad=1
Upsample	-	-	-	Nearest Neighbors
(Only 128px) Conv 3×3	64	ReLU	Instance	Stride=1 Pad=1
(Only 128px) Upsample	-	-	-	Nearest Neighbors
Conv 3×3	3	-	-	Stride=1 Pad=1

Neural Upscaler The neural upscaler takes the low resolution output of the NeRF MLPs and upscales it to the full output resolution. Additionally, it maps the rendered image to RGB space. This module is implemented using a convolutional neural network. We always render the NeRF output at 16px when using a neural network upscaler. Consequently, we add additional layers to the neural upscaler depending on the desired output resolution. For most experiments we use a resolution of 64px, while for the Scene Inference experiments on CLEVR2345 we use a resolution of 128px. The architecture of the neural upscaler can be found in Table 9. After some training iterations and once the model correctly reconstructs its inputs, we remove the neural upscaler and render scenes using NeRFs at full resolution.

B Experiment Details

B.1 Scene Manipulation

Dataset For generating manipulated scenes we consider the CLEVR2345 dataset (Niemeyer & Geiger, 2021). Images in the CLEVR2345 dataset contain from 2 to 5 objects. We use the original train and test splits. Images are resized to 128x128 pixels and RGB values are normalized in the [0, 1] range.

Training We use a batch size of 128 and train our model for 400k iterations. We use Adam with a learning rate of 1×10^{-4} and weight decay 1×10^{-6}. We use 5 objects and rely on the model to not use additional slots if the scene shows less than 5 objects. We use the neural upscaler with additional layers to upscale to 128px. We remove the neural upscaler and render at full resolution after 100K iterations. Additionally, for this experiment we use an additional LPIPS loss. We use the LPIPS metric computed by an AlexNet network, and we add this loss to our regular MSE loss. We weight the LPIPS loss by a factor of 100, so that it has a comparable order of magnitude to the MSE loss.

Manipulations Manipulations are done as follows:

- *Substraction:* We randomly delete up to two of the object slots.
- *Addition:* We randomly add an object slot from another scene.
- *Scale:* We reduce the scale (in all XYZ axis) of one of the objects in the substraction scene.
- *Forward:* We manipulate the pose vector of one object in the substraction scene and move it forward on the Z axis.
- *Right:* We manipulate the pose vector of one object in the substraction scene and move it forward on the X axis.

Additionally, we consider the *Swap* transformation for Table 1. This transformations modifies a scene by replacing the object shape and appearance vectors with those of an object from another scene.

Metrics To compare reconstructions we use Mean-Squared Error, Peak Signal-to-Noise Ratio (PSNR), Structural Similarity (SSIM) and the LPIPS metrics.

MSE measure the average squared difference between pixel values.

$$\text{MSE}(x, x') = \frac{1}{N} \sum_{N} (x - x')^2 \tag{4}$$

PSNR is a metric commonly used in signal processing.

$$\text{PSNR}(x, x') = -10 \log_{10}(\text{MSE}(x, x')) \tag{5}$$

SSIM (Wang et al., 2004) provides scores more aligned with human perception, specially under the presence of image noise. Scores are computed convolutionally by applying a kernel over images, which are then contrasted.

LPIPS (Zhang et al., 2018) computes differences in neural network activations for two images. It is a perceptual metric that has been shown to have higher correlation to human perception than other metrics not based on neural networks.

To compare populations of generated images we use the Frechet Inception Distance (Heusel et al., 2017). The Frechet Inception Distance embeds images into a neural network space and then fits a Gaussian distribution to the generated and ground-truth activation statistics. The score is obtained by then computing the Frechet distance between the two. Note that other metrics such as Inception Score are not applicable for the CLEVR2345 since there are no well-defined classes.

B.2 Object Discovery

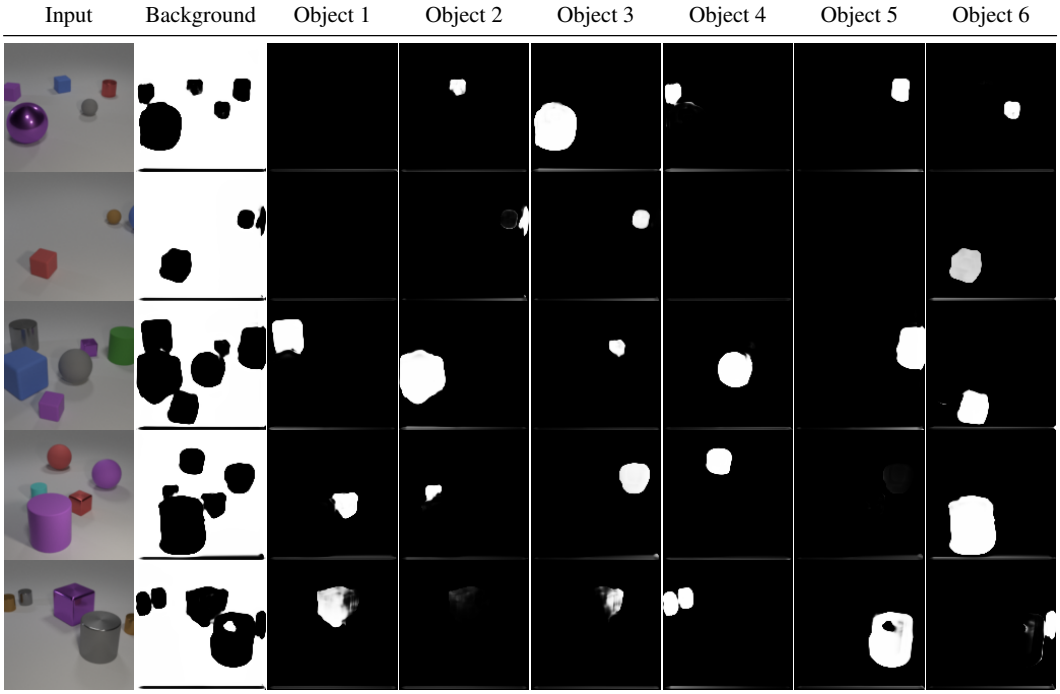

Figure 3: **Object Discovery on CLEVR6**: INFERNO identifies the different objects in a scene without supervision. For each input image, we show which regions of the input are attended by each object as well as the background. We include an example of a failed segmentation in the last row, where one object slot (4) is trying to represent multiple objects at the same time.

Dataset For object discovery we consider the CLEVR6 dataset. We use the original CLEVR6 dataset and extract the images from TFRecord files available at this URL. We use the original training/test split, using the first 70% images for training and the remaining ones for test. We take a crop between pixels [29, 221] and [64, 256], for the height and width respectively, and then resize the crop to 64px. We normalize the value of the images between [0, 1]. To generate the CLEVR6 dataset, we keep only those images that have at maximum 6 objects according to the annotation files.

Training We use a batch size of 128 and train our model for 400k iterations. We use Adam with a learning rate of 1×10^{-4} and weight decay 1×10^{-6}. We use 6 objects and rely on the model to not use additional slots when needed.

Metrics We follow previous work (Greff et al., 2019) and use the Adjusted Rand Index (ARI) (Rand, 1971) to evaluate cluster assignments in object discovery. ARI scores range from 0 (random assigment) to 1 (perfect match). As in previous works, we do not consider a segmentation mask for the background.

Additional Visualizations An additional visualization of the object masks learned by our model can be seen in 3. We can see that the model learns to segment different objects and properly discards object slots when the number of objects in the scene is lower than needed. We also include an example that our model fails to segment properly - multiple objects are segmented by the same object slot, and a single object is represented in multiple parts by different slots.

B.3 Visual Reasoning on CATER

For the visual reasoning task, we consider the CATER dataset and uses 5K videos that do not have camera motion. All the videos are resized to a 64x64 resolution.

B.3.1 Pretraining

We first pretrain our INFERNO to reconstruct individual frames from the CATER dataset. We bootstrap a model trained on CLEVR6 for object discovery for 400k iterations and train it on the CATER dataset for an additional 100k iterations to speed-up training, as the iteration time of a model with 10 objects is larger. We use a batch size of 128 and we use Adam with a learning rate of 1×10^{-4} and weight decay 1×10^{-6}. When training on CLEVR6 we use 6 object slots, while when training on CATER we use 10 object slots.

B.3.2 Finetuning

After pretraining, we finetune the INFERNO encoder to the supervised task of snitch localization. We discard the rendering pipeline of our model, and instead feed the inferred object slots representations to a transformer that aims at predicting the final position of the snitch.

To predict the snitch, we consider a 12 layers transformer with the hidden dimension of 128 which takes the slot representation as input. The transformer treats each object as input element. A learned positional embedding is added each slots representation based on their frames index, i.e. the position of the objects is the same within a frame. The final output to the transformer is given to a 1 layer MLP head with an hidden dimension of 128. It outputs 36 logits that correspond to possible snitch location. We minimize the sum of the cross-entropy between the predicted position and the true target and a the l1 loss between the prediction and target.

We optionally use a SSL loss similar to Ding et al. (2020). The SSL loss randomly masks one object per-frame and tries to predict its representation at the corresponding output. The model minimizes the L2 distance between the predicted object representation and the observed one. The SSL loss is only backpropagated through the transformer and not the encoder. We weight the SSL loss by a factor of $1.0e-3$.

We use an Adam optimizer to minimize the loss. The initial learning rate is set to the $1.0e-4$ and gradually decreased to $1.0e-6$ using a cosine learning rate decay. Similarly, we use a initial weight decay of $1.0e-5$ that we increases to $1.0e-3$ using a cosine schedule. Our model is finetuned for 500 epochs. We don't make use of learning rate decay.

During training, we randomly samples 40 frames from a video and predict the snitch localization from this one crop. At test time, we randomly sample 10 temporal crop of 40 frames each and average the prediction over the 10 crops as our final prediction.

Figure 4: **Additional reconstructions on CLEVR2345.**

C Additional Visualizations

We have included additional manipulations of one scene in GIF format in the supplementary material. We show: i) each object rendered individually, ii) object identity swaps with other scenes, iii) object translations along one axis, iv) translations in diagonal (two axis), and v) objects moving in a circle.

We also include more examples of reconstructions and scene manipulations where we add and remove objects in Figure 4 and Figure 5, respectively.

D Novel View Synthesis

In this section we synthesize novel views of a scene by modifying the camera pose. For each example we show the input image, our reconstruction, then two images as a result of moving the camera $\pm 15°$

Figure 5: **Additional object additions and removals on CLEVR2345.** For each scene, we show images with one randomly added object, and with 1-3 random objects removed. Some of these images show out-of-distribution samples with a number of objects not seen during training (2-5 objects).

in the azimuth axis, and two images as a result of zooming in the scene. This experiment is shown in Figure 6. While our model is trained without ground-truth camera poses and with single views of scenes, it is able to generalize to small camera pose modifications and render novel views of a scene.

E NeRF output with Neural Upscaler

In this section we show the raw outputs of the NeRF function. These outputs show scene views rendered at low resolution, which are then upscaled with a neural network. While these generations have reduced details due to their resolution, they clearly show the different objects and their location in the scene. We show these visualizations in Figure 7. NeRFs are rendered at low resolution to ease the computational costs, as the time and memory requirements of rendering with a NeRF are linearly

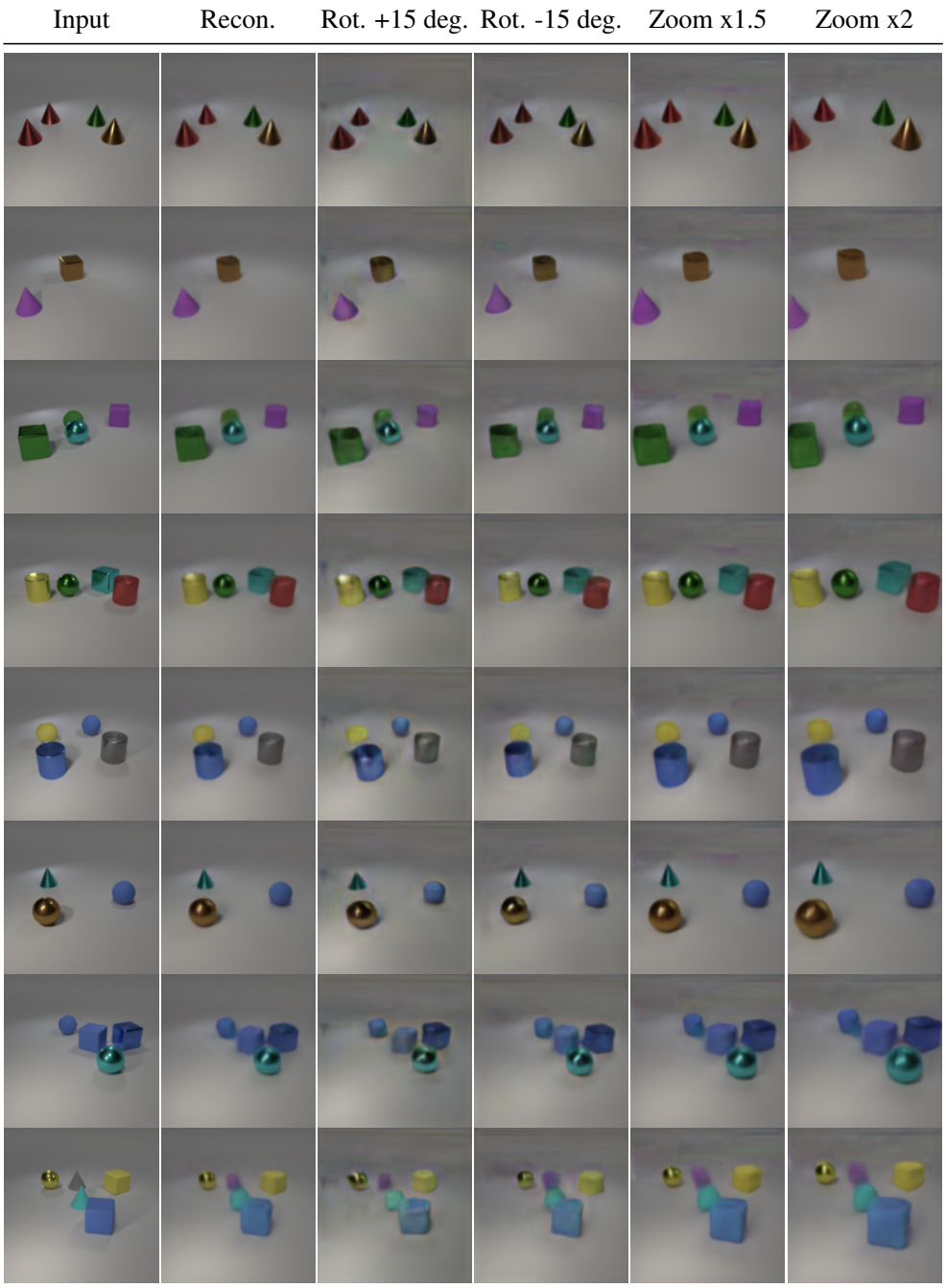

Figure 6: **Novel view synthesis on CLEVR2345.** For each scene, we move the camera $\pm 15^\circ$ on the azimuth axis. Additionally, we zoom in the scene twice. While our model is trained with a fixed default camera pose and single scene views, it is able to generalize to small camera pose modifications and render novel views of a scene.

correlated with the number of casted rays/pixels in the output image. After we bootstrap the model with the neural upscaler, we then remove it and render with NeRFs at full resolution.

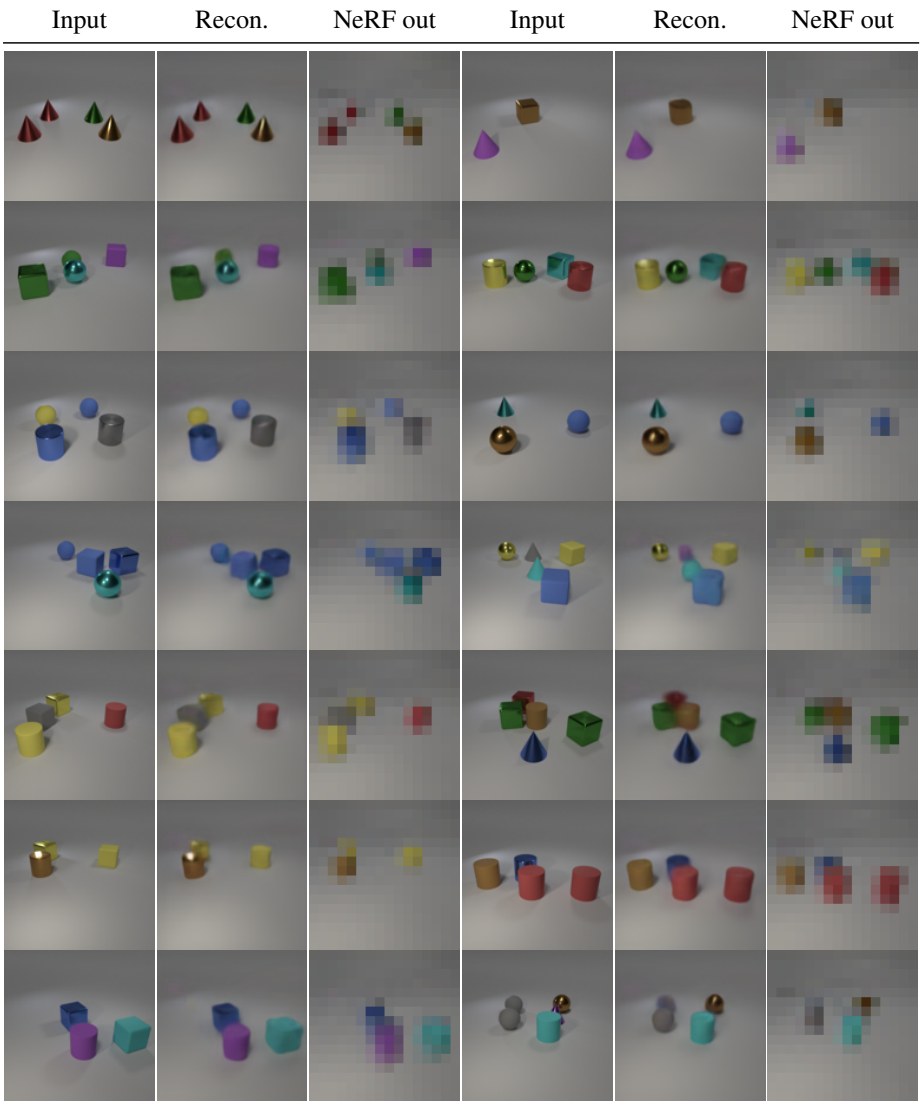

Figure 7: **NeRF outputs on CLEVR2345.** For each input scene we show the reconstructed image as well as each the low resolution output of the NeRF function. This output is then upscaled with a neural network to obtain the reconstructed scene. While NeRF output lack full detail, they correctly depict each individual object and their position in the scene.

F SLOT VISUALIZATION

In this section we show the rendering of individual slots. Given an input scene, we first infer its representation. Then, for each object in the representation, we render it individually against a black background. Examples from this visualization are shown in Figure 8. We observe that, in general, each object slot is rendering a part of the scene corresponding to a single object instance. Additionally, for some objects their slot captures parts of its shading.

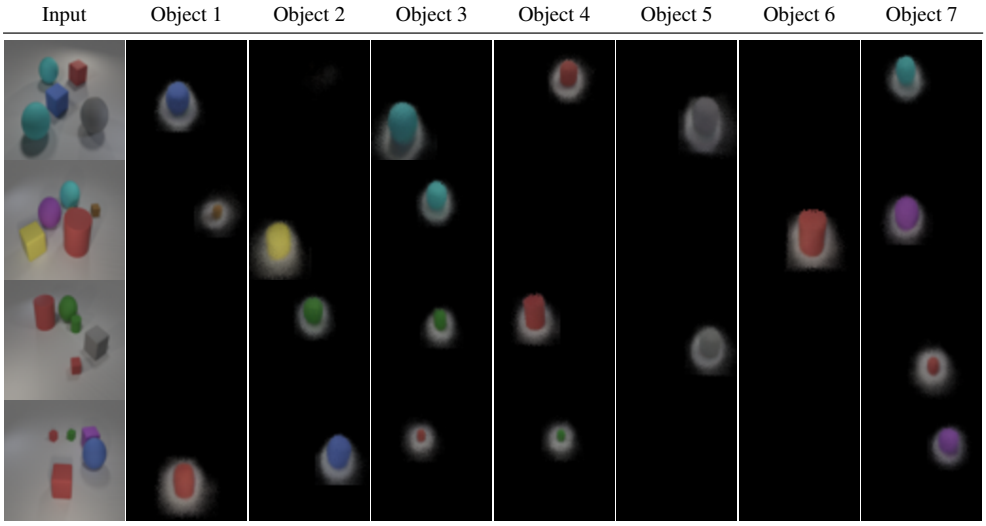

Figure 8: **Object Slots rendered individually on CATER:** For each input scene we show the reconstructed image as well as each individual object slot rendered against a black background. We observe that each object slot is rendering a part of the scene corresponding to a single object instance.