Multi-Object Representation Learning with Iterative Variational Inference

Klaus Greff [1][2]   **Raphaël Lopez Kaufman** [3]   **Rishabh Kabra** [3]   **Nick Watters** [3]   **Chris Burgess** [3]   **Daniel Zoran** [3]
Loic Matthey [3]   **Matthew Botvinick** [3]   **Alexander Lerchner** [3]

Abstract

Human perception is structured around objects which form the basis for our higher-level cognition and impressive systematic generalization abilities. Yet most work on representation learning focuses on feature learning without even considering multiple objects, or treats segmentation as an (often supervised) preprocessing step. Instead, we argue for the importance of learning to segment and represent objects *jointly*. We demonstrate that, starting from the simple assumption that a scene is composed of multiple entities, it is possible to learn to segment images into interpretable objects with disentangled representations. Our method learns – without supervision – to inpaint occluded parts, and extrapolates to scenes with more objects and to unseen objects with novel feature combinations. We also show that, due to the use of iterative variational inference, our system is able to learn multi-modal posteriors for ambiguous inputs and extends naturally to sequences.

1. Introduction

Learning good representations of complex visual scenes is a challenging problem for artificial intelligence that is far from solved. Recent breakthroughs in unsupervised representation learning (Higgins et al., 2017a; Makhzani et al., 2015; Chen et al., 2016) tend to focus on data where a single object of interest is placed in front of some background (e.g. dSprites, 3D Chairs, CelebA). Yet in general, visual scenes contain a variable number of objects arranged in various spatial configurations, and often with partial occlusions (e.g., CLEVR, Johnson et al. 2017; see Figure 1). This motivates the question: what forms a good representation of a scene with multiple objects? In line with recent advances (Burgess et al., 2019; van Steenkiste et al., 2018; Eslami et al., 2016),
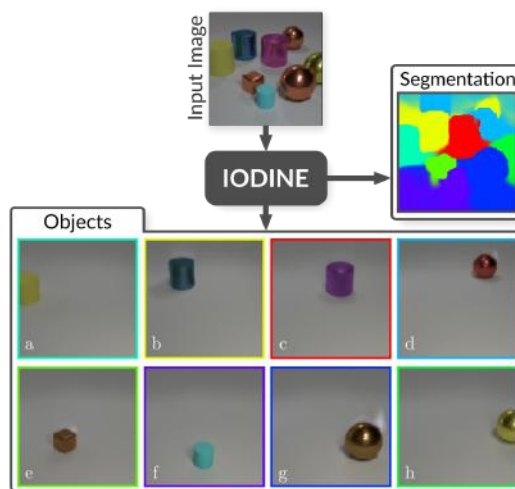

Figure 1. Object decomposition of an image from the CLEVR dataset by IODINE. The model is able to decompose the image into separate objects in an unsupervised manner, inpainting occluded objects in the process (see slots (d), (e) and (h)).

we maintain that discovery of objects in a scene should be considered a crucial aspect of representation learning, rather than treated as a separate problem.

We approach the problem from a spatial mixture model perspective (Greff et al., 2017) and use amortized iterative refinement (Marino et al., 2018b) of latent object representations within a variational framework (Rezende et al., 2014; Kingma & Welling, 2013). We encode our basic intuition about the existence of objects into the structure of our model, which simultaneously facilitates their discovery and efficient representation in a fully data-driven, unsupervised manner. We name the resulting architecture **IODINE** (short for Iterative Object Decomposition Inference NEtwork).

IODINE can segment complex scenes and learn disentangled object features without supervision on datasets like CLEVR, Objects Room (Burgess et al., 2019), and Tetris (see Appendix B). We show systematic generalization to more objects than included in the training regime, as well as objects formed with unseen feature combinations. This highlights the benefits of multi-object representation learning by comparison to a VAE's single-slot representations. We also justify how the sampling used in iterative refinement lends to resolving multi-modal and multi-stable decomposition.
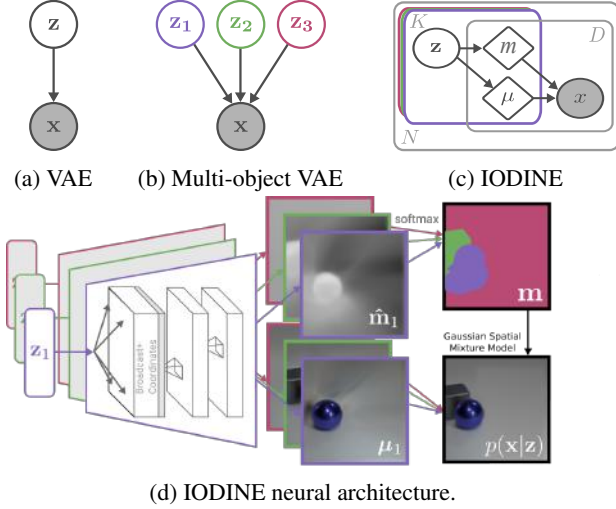
---

[1]The Swiss AI lab IDSIA, Lugano, Switzerland [2]Work done at DeepMind [3]DeepMind, London, UK. Correspondence to: Klaus Greff <klaus.greff@startmail.com>.

(a) VAE   (b) Multi-object VAE   (c) IODINE

(d) IODINE neural architecture.

Figure 2. Generative model illustrations. (a) A regular VAE decoder. (b) A hypothetical multi-object VAE decoder that recomposes the scene from three objects. (c) IODINE's multi-object decoder showing latent vectors (denoted z) corresponding to K objects refined over N iterations from images of dimension D. The deterministic pixel-wise means and masks are denoted μ and m respectively. (d) The neural architecture of the IODINE's multi-object spatial mixture decoder.

2. Method

We first express multi-object representation learning within the framework of generative modelling (Section 2.1). Then, building upon the successful Variational AutoEncoder framework (VAEs; Rezende et al. 2014; Kingma & Welling 2013), we leverage variational inference to jointly learn both the generative and inference model (Section 2.2). There we also discuss the particular challenges that arise for inference in a multi-object context and show how they can be solved using iterative amortization. Finally, in Section 2.3 we bring all elements together and show how the complete system can be trained end-to-end.

2.1. Multi-Object Representations

Flat vector representations as used by standard VAEs are inadequate for capturing the combinatorial object structure that many datasets exhibit. To achieve the kind of systematic generalization that is so natural for humans, we propose employing a *multi-slot* representation where each slot shares the underlying representation format, and each would ideally describe an independent part of the input. Consider the example in Figure 1: by construction, the scene consists of 8 objects, each with its own properties such as shape, size, position, color and material. To split objects, a flat representation would have to represent each object using separate feature dimensions. But this neglects the simple and (to us) trivial fact that they are interchangeable objects with common properties.

Generative Model   We represent each scene with K latent object representations $z_k \in \mathbb{R}^M$ that collaborate to generate the input image $x \in \mathbb{R}^D$ (c.f. Figure 2b). The z_k are assumed to be independent and their generative mechanism is shared such that any ordering of them produces the same image (i.e. entailing permutation invariance). Objects distinguished in this way can easily be compared, reused and recombined, thus facilitating combinatorial generalization.

The image x is modeled with a spatial Gaussian mixture model where each mixing component (slot) corresponds to a single object. That means each object vector z_k is decoded into a pixel-wise mean μ_{ik} (the appearance of the object) and a pixel-wise assignment $m_{ik} = p(C = k|z_k)$ (the segmentation mask; c.f. Figure 2c). Assuming that the pixels i are independent conditioned on z, the likelihood thus becomes:

$$p(\mathbf{x}|\mathbf{z}) = \prod_{i=1}^{D} \sum_{k=1}^{K} m_{ik} \mathcal{N}(x_i; \mu_{ik}, \sigma^2), \qquad (1)$$

where we use a global fixed variance σ^2 for all pixels.

Decoder Structure   Our decoder network structure directly reflects the structure of the generative model. See Figure 2d for an illustration. Each object latent z_k is decoded separately into pixel-wise means μ_k and mask-logits $\hat{m}_k$, which we then normalize using a softmax operation applied across slots such that the masks m_k for each pixel sum to 1. Together, μ and m parameterize the spatial mixture distribution as defined in Equation (1). For the network architecture we use a broadcast decoder (Watters et al., 2019), which spatially replicates the latent vector z_k, appends two coordinate channels (ranging from -1 to 1 horizontally and vertically), and applies a series of size-preserving convolutional layers. This structure encourages disentangling the position across the image from other features such as color or texture, and generally supports disentangling. All slots k share weights to ensure a common format, and are independently decoded, up until the mask normalization.

2.2. Inference

Similar to VAEs, we use amortized variational inference to get an approximate posterior $q_\lambda(\mathbf{z}|\mathbf{x})$ parameterized as a Gaussian with parameters $\boldsymbol{\lambda} = \{\boldsymbol{\mu_z}, \boldsymbol{\sigma_z}\}$. However, our object-oriented generative model poses a few specific challenges for the inference process: Firstly, being a (spatial) mixture model, we need to infer both the components (i.e. object appearance) and the mixing (i.e. object segmentation). This type of problem is well known, for example in clustering and image segmentation, and is traditionally tackled as an iterative procedure, because there are no efficient direct solutions. A related second problem is that any slot can, in principle, explain any pixel. Once a pixel is explained
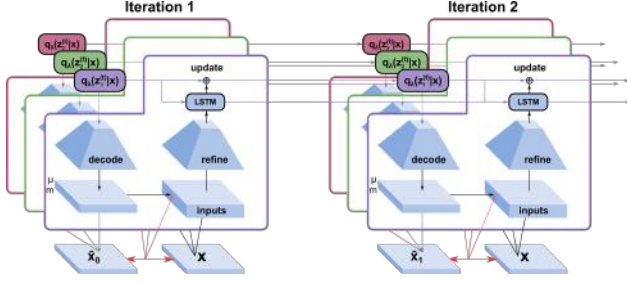
Figure 3. Illustration of the iterative inference procedure.

by one of the slots, the others don't need to account for it anymore. This explaining-away property complicates the inference by strongly coupling it across the individual slots. Finally, slot permutation invariance induces a multimodal posterior with at least one mode per slot permutation. This is problematic, since our approximate posterior $q_\lambda(\mathbf{z}|\mathbf{x})$ is parameterized as a unimodal distribution. For all the above reasons, the standard feed-forward VAE inference model is inadequate for our case, so we consider a more powerful method for inference.

Iterative Inference The basic idea of iterative inference is to start with an arbitrary guess for the posterior parameters $\boldsymbol{\lambda}$, and then iteratively refine them using the input and samples from the current posterior estimate. We build on the framework of iterative amortized inference (Marino et al., 2018b), which uses a trained refinement network f_ϕ. Unlike Marino et al., we consider only additive updates to the posterior and use several salient auxiliary inputs $\mathbf{a}$ to the refinement network (instead of just $\nabla_\lambda \mathcal{L}$). We update the posterior of the K slots independently and in parallel (indicated by $\overset{k}{\leftarrow}$ and $\overset{k}{\sim}$), as follows:

$$\mathbf{z}_k^{(t)} \overset{k}{\sim} q_\lambda(\mathbf{z}_k^{(t)}|\mathbf{x}) \tag{2}$$

$$\boldsymbol{\lambda}_k^{(t+1)} \overset{k}{\leftarrow} \boldsymbol{\lambda}_k^{(t)} + f_\phi(\mathbf{z}_k^{(t)}, \mathbf{x}, \mathbf{a}_k), \tag{3}$$

Thus the only place where the slots interact are at the input level. As refinement network f_ϕ we use a convolutional network followed by an LSTM (see Appendix C for details). Instead of amortizing the posterior directly (as in a regular VAE encoder), the refinement network can be thought of as amortizing the gradient of the posterior (Marino et al., 2018a). The alternating updates to $q_\lambda(\mathbf{z}|\mathbf{x})$ and $p(\mathbf{x}|\mathbf{z})$ are also akin to message passing.

Inputs For each slot k we feed a set of auxiliary inputs $\mathbf{a}_k$ to the refinement network f_ϕ which then computes an update for the posterior $\boldsymbol{\lambda}_k$. Crucially, we include gradient information about the ELBO in the inputs, as it conveys information about what is not yet explained by other slots. Omitting the superscript (t) for clarity, the auxiliary inputs $\mathbf{a}_k$ are (see Appendix C for details): image

Algorithm 1 IODINE Pseudocode.

Input: image $\mathbf{x}$, hyperparamters K, T, σ^2
Input: trainable parameters $\boldsymbol{\lambda}^{(1)}, \boldsymbol{\theta}, \boldsymbol{\phi}$
Initialize: $\mathbf{h}_k^{(1)} \overset{k}{\leftarrow} \mathbf{0}$
for $t = 1$ **to** T **do**
  $\mathbf{z}_k^{(t)} \overset{k}{\sim} q_\lambda(\mathbf{z}_k^{(t)}|\mathbf{x})$ // Sample
  $\boldsymbol{\mu}_k^{(t)}, \hat{\boldsymbol{m}}_k^{(t)} \overset{k}{\leftarrow} g_\theta(\mathbf{z}_k^{(t)})$ // Decode
  $\boldsymbol{m}^{(t)} \leftarrow \text{softmax}_k(\hat{\boldsymbol{m}}_k^{(t)})$ // Masks
  $p(\mathbf{x}|\mathbf{z}^{(t)}) \leftarrow \sum_k \boldsymbol{m}_k^{(t)} \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k^{(t)}, \sigma^2)$ // Likelihood
  $\mathcal{L}^{(t)} \leftarrow D_{KL}(q_\lambda(\mathbf{z}^{(t)}|\mathbf{x})||p(\mathbf{z})) - \log p(\mathbf{x}|\mathbf{z}^{(t)})$
  $\mathbf{a_k} \overset{k}{\leftarrow} \text{inputs}(\mathbf{x}, \mathbf{z}_k^{(t)}, \boldsymbol{\lambda}_k^{(t)})$ // Inputs
  $\boldsymbol{\lambda}_k^{(t+1)}, \mathbf{h}^{(t+1)} \overset{k}{\leftarrow} f_\phi(\mathbf{a}_k, \mathbf{h}_k^{(t)})$ // Refinement
end for

x, means $\boldsymbol{\mu}_k$, masks $\mathbf{m}_k$, mask-logits $\hat{\boldsymbol{m}}_k$, mean gradient $\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$, mask gradient $\nabla_{\mathbf{m}_k}\mathcal{L}$, posterior gradient $\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$, posterior mask $p(\mathbf{m}_k|\mathbf{x}, \boldsymbol{\mu}) = \frac{p(x|\mu_k)}{\sum_j p(x|\mu_j)}$, pixelwise likelihood $p(\mathbf{x}|\mathbf{z})$, leave-one-out likelihood $p(\mathbf{x}|\mathbf{z}_{i\neq k})$, and two coordinate channels like in the decoder.

With the exception of $\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$, these are all image-sized and cheap to compute, so we feed them as additional input-channels into the refinement network. The approximate gradient $\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$ is computed using the reparameterization trick by a backward pass through the generator network. This is computationally quite expensive, but we found that this information helps to significantly improve training of the refinement network. This input is the same size as the posterior $\boldsymbol{\lambda}_k$ and is fed to the LSTM part of the refinement network. Like Marino et al. (2018b) we found it beneficial to normalize the gradient-based inputs with LayerNorm (Ba et al., 2016). See Section 4.3 for an ablation study.

2.3. Training

We train the parameters of the decoder ($\boldsymbol{\theta}$), of the refinement network ($\boldsymbol{\phi}$), and of the initial posterior ($\boldsymbol{\lambda}^{(1)}$) by gradient descent through the unrolled iterations. In principle, it is enough to minimize the final negative ELBO $\mathcal{L}^T$, but we found it beneficial to use a weighted sum which also includes earlier terms:

$$\mathcal{L}_{\text{total}} = \sum_{t=1}^{T} \frac{t}{T}\mathcal{L}^{(t)}. \tag{4}$$

Each refinement step of IODINE uses gradient information to optimize the posterior $\boldsymbol{\lambda}$. Unfortunately, backpropagating through this process leads to numerical instabilities connected to double derivatives like $\nabla_\Theta \nabla_z \mathcal{L}$. We found that this problem can be mitigated by dropping the double derivative terms, i.e. stopping the gradients from backpropagating through the gradient-inputs $\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$, $\nabla_{\mathbf{m}_k}\mathcal{L}$, and $\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$ (see Appendix C for details).

3. Related Work

Representation learning (Bengio et al., 2013) has received much attention and has seen several recent breakthroughs. This includes disentangled representations through the use of β-VAEs (Higgins et al., 2017a), adversarial autoencoders (Makhzani et al., 2015), Factor VAEs (Kim & Mnih, 2018), and improved generalization through non-euclidean embeddings (Nickel & Kiela, 2017). However, most advances have focused on the feature-level structure of representations, and do not address the issue of representing multiple, potentially repeating objects, which we tackle here.

Another line of work is concerned with obtaining segmentations of images, usually without considering representation learning. This has led to impressive results on real-world images, however, many approaches (such as "semantic segmentation" or object detection) rely on supervised signals (Girshick, 2015; He et al., 2017; Redmon & Farhadi, 2018), while others require hand-engineered features (Shi & Malik, 2000; Felzenszwalb & Huttenlocher, 2004). In contrast, as we learn to both segment and represent, our method can perform inpainting (Figure 1) and deal with ambiguity (Figure 10), going beyond what most methods relying on feature engineering are currently able to do.

Works tackling the full problem of scene representation are rarer. Probabilistic programming based approaches, like stroke-based character generation (Lake et al., 2015) or 3D indoor scene rendering (Pero et al., 2012), have produced appealing results, but require carefully engineered generative models, which are typically not fully learned from data. Work on end-to-end models has shown promise in using autoregressive inference or generative approaches (Eslami et al., 2016; Gregor et al., 2015), including the recent MONet (Burgess et al., 2019). Few methods can achieve similar comparable with the complexity of the scenes we consider here, apart from MONet. Section 4.1 shows a preliminary comparison between MONet and IODINE, and we discuss their relationship further in Appendix A.3.

Two other methods related to ours are Neural Expectation Maximization (Greff et al., 2017) (along with its sequential and relational extensions (van Steenkiste et al., 2018)) and Tagger (Greff et al., 2016). NEM uses recurrent neural networks to amortize expectation maximization for a spatial mixture model. However, NEM variants fail to cope with colored scenes, as we note in our comparison in Section 4.1. Tagger also uses iterative inference to segment and represent images based on a denoising training objective. We disregard Tagger for our comparison, because (1) its use of a Ladder network means that there is no bottleneck and thus no explicit object representations, and (2) without adapting it to a convolutional architecture, it does not scale to larger images (Tagger would require $\approx 600M$ weights for CLEVR).
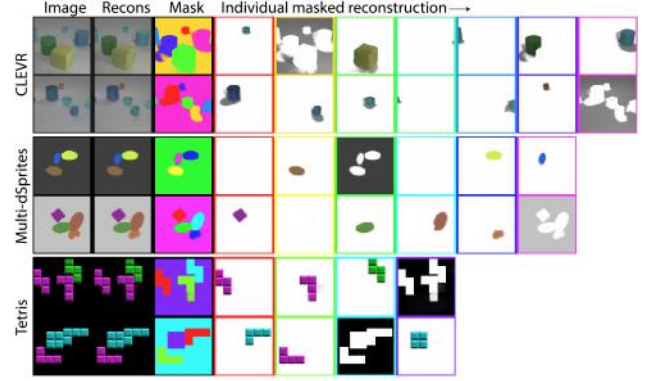

Figure 4. IODINE segmentations and object reconstructions on CLEVR6 (top), Multi-dSprites (middle), and Tetris (bottom). The individual masked reconstruction slots represent objects separately (along with their shadow on CLEVR). Border colours are matched to the segmentation mask on the left.
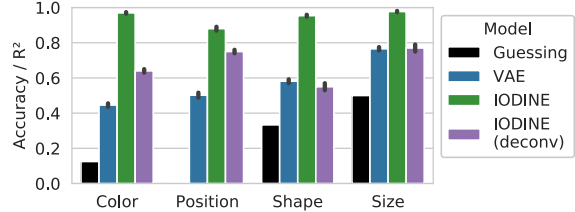

Figure 5. Prediction accuracy / R^2 score for the factor regression on CLEVR6. Position is continuous; the rest are categorical with 8 colors, 3 shapes, and 2 sizes. IODINE (deconv) does not use spatial broadcasting in the decoder (see Section 4.3).

4. Results

We evaluate our model on three main datasets: 1) CLEVR (Johnson et al., 2017) and a variant CLEVR6 which uses only scenes with up to 6 objects, 2) a multi-object version of the dSprites dataset (Matthey et al., 2017), and 3) a dataset of multiple "Tetris"-like pieces that we created. In all cases we train the system using the Adam optimizer (Kingma & Ba, 2015) to minimize the negative ELBO for 10^6 updates. We varied several hyperparameters, including: number of slots, dimensionality of $\mathbf{z}_k$, number of inference iterations, number of convolutional layers and their filter sizes, batch size, and learning rate. For details of the models and hyperparameters refer to Appendix C and the code for all experiments can be found on GitHub.

4.1. Decomposition

IODINE can provide a readily interpretable segmentation of the data, as seen in Figure 4. These examples clearly demonstrate the models ability to segmenting out the same objects which were used to generate the dataset, despite never having received supervision to do so. To quantify segmentation quality, we measure the similarity between ground-truth (instance) segmentations and our predicted object masks using the Adjusted Rand Index (ARI; Rand 1971;
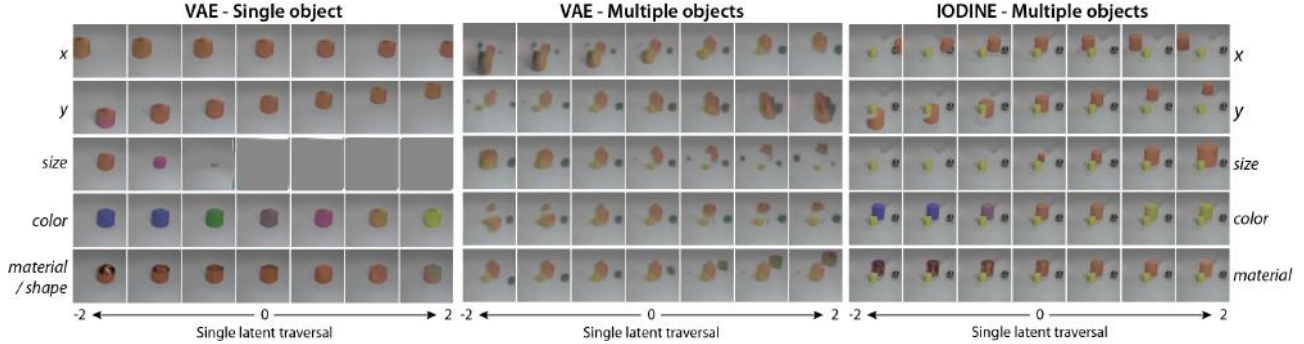
Figure 6. Disentanglement in regular VAEs vs IODINE. Rows indicate traversals of single latents, annotated by our interpretation of their effects. (Left) When a VAE is trained on single-object scenes it can disentangle meaningful factors of variation. (Center) When the same VAE is trained on multi-object scenes, the latents entangle across both factors and objects. (Right) In contrast, traversals of individual latents in IODINE vary individual factors of single objects, here the orange cylinder. Thus, the architectural bias for discovering multiple entities in a common format enables not only the discovery of objects, but also facilitates disentangling of their features.

Hubert & Arabie 1985). ARI is a measure of clustering similarity that ranges from 0 (chance) to 1 (perfect clustering) and can handle arbitrary permutations of the clusters. We apply it as a measure of instance segmentation quality by treating each foreground pixel (ignoring the background) as one point and its segmentation as cluster assignment. As shown in Table 1, IODINE achieves almost perfect ARI scores of around 0.99 for CLEVR6, and Tetris as well as a relatively good score of 0.77 for Multi-dSprites. The lower scores on Multi-dSprites are largely because IODINE struggles to produce sharp boundaries for the sprites, and we are uncertain as to the reasons for this behaviour.

We compare with MONet (Burgess et al., 2019), following the CLEVR model implementation described in the paper except using fewer (7) slots and different standard deviations for the decoder distribution (0.06 and 0.1 for σ_{bg} and σ_{fg}, respectively), which gave better scores. With this, MONet obtained a similar ARI score (0.96) as IODINE on CLEVR6, and on Multi-dSprites it performed significantly better with a score of 0.90 (using the unmodified model). We also attempted to compare ARI scores to Neural Expectation Maximization, but neither Relational-NEM nor the simpler RNN-NEM variant could cope well with colored images. As a result, we could only compare with those methods on a binarized version of Multi-dSprites and the Shapes dataset. These scores are summarized in Table 1.

4.2. Representation Quality

Information Content The object-reconstructions in Figure 4 show that their representations contain all the information about the object. But in what format, and how usable is it? To answer this question we associate each ground-truth object with its corresponding $\mathbf{z}_k$ based on the segmentation masks. We then train a single-layer network to predict ground-truth factors for each object. Note that this predictor is trained *after* IODINE has finished training (i.e. no

	IODINE	R-NEM	MONet
CLEVR6	0.988 ± 0.000	$*$	0.962 ± 0.006
M-dSprites	0.767 ± 0.056	$*$	0.904 ± 0.008
M-dSprites bin.	0.648 ± 0.172	0.685 ± 0.017	
Shapes	0.910 ± 0.119	0.776 ± 0.019	
Tetris	0.992 ± 0.004	$*$	

Table 1. Summary of IODINE's segmentation performance in terms of ARI (mean $\pm$ stddev across five seeds) versus baseline models. For each independent run, we computed the ARI score over 320 images, using only foreground pixels. We then picked the best hyperparameter combination for each model according to the mean ARI score over five random seeds.

supervised fine-tuning). It tells us if a linear mapping is sufficient to extract information like color, position, shape or size of an object from its latent representation, and gives an important indication about the usefulness of the representation. Results in Figure 5 clearly show that a linear mapping is sufficient to extract relevant information about these object attributes from the latent representation to high accuracy. This result is in contrast with the scene representations learned by a standard VAE. Here even training the factor-predictor is difficult, as there is no obvious way to align objects with features. To make this comparison, we chose a canonical ordering of the objects based on their size, material, shape, and position (with decreasing precedence). The precedence of features was intended as a heuristic to maximize the predictability of the ordering. We then trained a linear network to predict the concatenated features of the canonically ordered objects from the latent scene representation. As the results in Figure 5 indicate, the information is present, but in a much less explicit/usable state.

Disentanglement Disentanglement is another important desirable property of representations (Bengio et al., 2013) that captures how well learned features separate and correspond to individual, interpretable factors of variation in the data. While its precise definition is still highly debated (Hig-
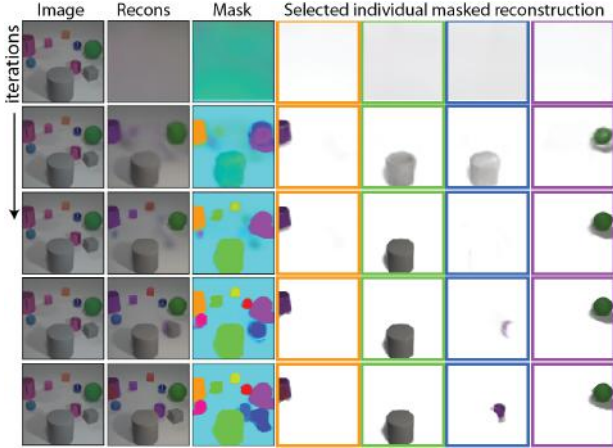
Figure 7. IODINE's iterative inference process and generalization capabilities. Rows indicate steps of iterative inference, refining reconstructions and segmentations when moving down the figure. Of particular interest is the *explaining away* effect visible between slots 2 and 3, where they settle on different objects despite both starting with the large cylinder. The model was only trained with $K = 7$ slots on 3-6 objects (excluding green spheres), and yet is able to generalize to $K = 11$ slots (only 4 are shown, see Figure 19 in the appendix for a full version) on a scene with 9 objects, including the never seen before green sphere (last column).

gins et al., 2018; Eastwood & Williams, 2018; Ridgeway & Mozer, 2018; Locatello et al., 2018), the concept of disentanglement has generated a lot of interest recently. Good disentanglement is believed to lead to both better generalization and more interpretable features (Lake et al., 2016; Higgins et al., 2017b). Interestingly, for these desirable advantages to bear out, disentangled features seem to be most useful for properties of single objects, such as color, position, shape, etc. It is much less clear how to operationalize this in order to create disentangled representations of entire scenes with variable numbers of objects. And indeed, if we train a VAE that can successfully disentangle features of a single-object dataset, we find that that its representation becomes highly entangled on a multi-object dataset, (see Figure 6 left vs middle). IODINE, on the other hand, successfully learns disentangled representations, because it is able to first decompose the scene and then represent individual objects (Figure 6 right). In Figure 6 we show traversals of the most important features (selected by KL) of a standard VAE vs IODINE. While the standard VAE clearly entangles many properties even across multiple objects, IODINE is able to neatly separate them.

Generalization Finally, we can ask directly: Does the system generalize to novel scenes in a systematic way? Specifically, does it generalize to scenes with more or fewer objects than ever encountered during training? Slots are exchangeable by design, so we can freely vary the number of slots during test-time (more on this in Section 4.3).
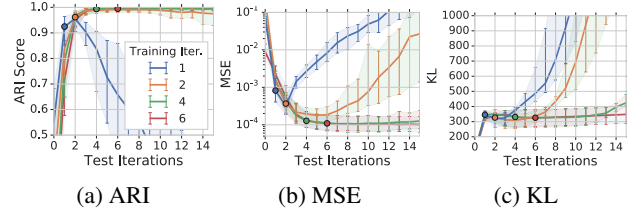

Figure 8. The effect of varying the number of iterations, for both training and at test time. (a) Median ARI score, (b) MSE and (c) KL over test-iterations, for models trained with different numbers of iterations on CLEVR6. The region beyond the filled dots thus shows test-time generalization behavior. Shaded region from 25th to 75th percentile.

So in Figure 7 we qualitatively show the performance of a system that was trained with $K = 7$ on up to 6 objects, but evaluated with $K = 11$ on 9 objects. In Figure 9a the orange boxes show, that, even quantitatively, the segmentation performance decreases little when generalizing to more objects.

A more extreme form of generalization involves handling unseen feature combinations. To test this we trained our system on a subset of CLEVR that does not contain green spheres (though it does contain spheres and other green objects). And then we tested what the system does when confronted with a green sphere. In Figure 7 it can be seen that IODINE is still able to represent green spheres, despite never having seen this combination during training.

4.3. Robustness & Ablation

Iterations The number of iterations is one of the central hyperparameters to our approach. To investigate its impact, we trained four models with 1, 2, 4 and 6 iterations on CLEVR6, and evaluated them all using 15 iterations (c.f. Figure 8). The first thing to note is that the inference converges very quickly within the first 3-5 iterations after which neither the segmentation nor reconstruction change much. The second important finding is that the system is very stable for much longer than the number of iterations it was trained with. The model even further improves the segmentation and reconstruction when it is run for more iterations, though it eventually starts to diverge after about two to three times the number of training iterations as can be seen with the blue and orange curves in Figure 8.

Slots The other central parameter of IODINE is the number of slots K, as it controls the maximum number of objects the system can separate. It is important to distinguish varying K for training vs varying it at test-time. As can be seen in Figure 9, if the model was trained with sufficiently many slots to fit all objects ($K = 7$, and $K = 9$), then test-time behavior generalizes very well. Typical behavior (not
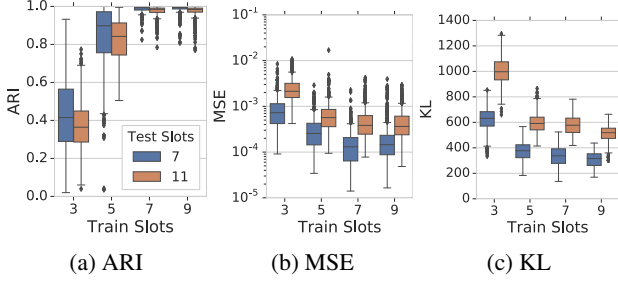
Figure 9. IODINE trained on CLEVR6 with varying numbers of slots (columns). Evaluation of (a) ARI, (b) MSE, and (c) KL with 7 slots on 3-6 Objects (blue) and 11 slots on 3-9 objects (orange).

shown) is to leave excess slots empty, and when confronted with too many objects it will often completely ignore some of them, leaving the other object-representations mostly intact. Given enough slots at test time, such a model can even segment and represent scenes of higher complexity (more objects) than any scene encountered during training (see Figure 7 and the orange boxes in Figure 9). If on the other hand, the model was trained with too few slots ($K = 3$ and $K = 5$), its performance suffers substantially. This happens because, here the only way to reconstruct the entire scene during training is to consistently represent multiple objects per slot. And that leads to the model learning inefficient and entangled representations akin to the VAE in Figure 6 (also apparent from their much higher KL in Figure 9c). Once learned, this sub-optimal strategy cannot be mitigated by increasing the number of slots at test-time as can be seen by their decreased performance in Figure 9a.

Input Ablations   We ablated each of the different inputs to the refinement network described in Section 2.2. Broadly, we found that individually removing an input did not noticeably affect the results (with two exceptions noted below). See Figures 33-40 in the Appendix demonstrating this lack of effect on different terms of the model's loss and the ARI segmentation score on both CLEVR6 and Tetris. A more comprehensive analysis could ablate combinations of inputs and identify synergistic or redundant groups, and thus potentially simplify the model. We didn't pursue this direction since none of the inputs incurs any noticeable computational overhead and at some point during our experimentation each of them contributed towards stable training behavior.

The main exceptions to the above are $\nabla_\lambda \mathcal{L}$ and $\mathbf{x}$. Computing the former requires an entire backward pass through the decoder, and contributes about 20% of the computational cost of the entire model. But we found that it often substantially improves performance and training convergence, which justifies its inclusion. A somewhat surprising finding was that for the Tetris dataset, removing $\mathbf{x}$ from the list of inputs had a pronounced detrimental effect, while for CLEVR it was negligible.
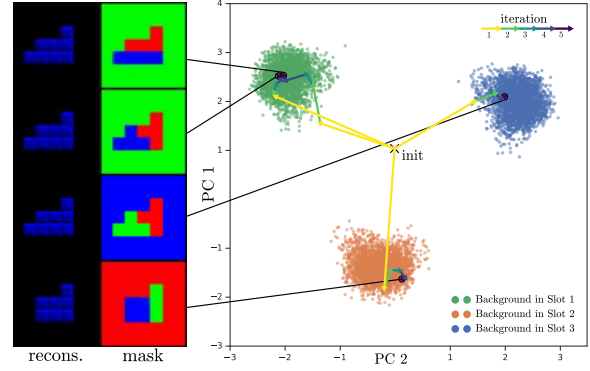

Figure 10. Multi-stability of segmentation when presented with an ambiguous stimulus. **Left:** Depending on the random sampling during iterative refinement, IODINE can produce different permutations of groups (row 2 vs 3), a different decomposition (row 1) or sometimes an invalid segmentation and reconstruction (row 4). **Right:** PCA of the latent space, coloured by which slot corresponds to the background. Paths show the trajectory of the iterative refinement for the four examples on the left.

Broadcast Decoder Ablation   We use the spatial broadcast decoder (Watters et al., 2019) primarily for its significant impact on the disentanglement of the representations, but its continuous spatial representation bias also seems to help decomposition. When replacing it with a deconvolution-based decoder the factor regression scores on CLEVR6 are significantly worse as can be seen in Figure 5. Especially for shape and size it now performs no better than the VAE which uses spatial broadcasting. The foreground-ARI scores also drop significantly (0.67 ± 0.06 down from 0.99) and the model seems less able to specialize slots to single objects (see Figure 23). Note though, that these discrepancies might easily be reduced, since we haven't invested much effort in tuning the architecture of the deconv-based decoder.

4.4. Multi-Modality and Multi-Stability

Standard VAEs are unable to represent multi-modal posteriors, because $q_\lambda(\mathbf{z}|\mathbf{x})$ is parameterized using a unimodal Gaussian distribution. However, as demonstrated in Figure 10, IODINE can actually handle this problem quite well. How is that possible? It turns out that this is an important side-effect of iterative variational inference, that to the best of our knowledge has not been noticed before: The stochasticity at each iteration, which results from sampling $\mathbf{z}$ to approximate the likelihood, implicitly acts as an auxilliary (inference) random variable. This effect compounds over iterations, and is amplified by the slot-structure and the effective message-passing between slots over the course of iterations. In effect the model can implicitly represent multiple modes (if integrated over all ways of sampling $\mathbf{z}$) and thus converge to different modes (see Figure 10 left) depending on these samples. This does not happen in a regular

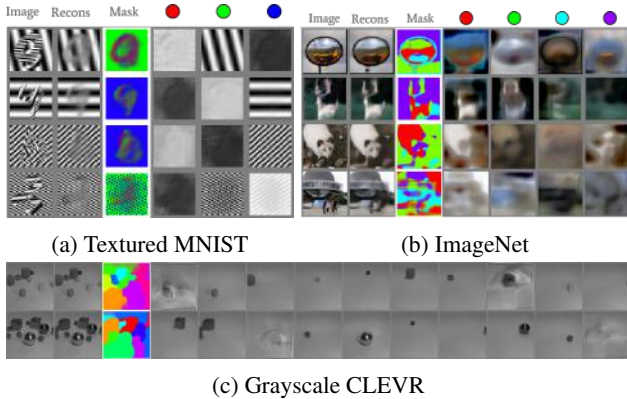(a) Textured MNIST   (b) ImageNet

(c) Grayscale CLEVR

Figure 11. Segmentation challenges a) IODINE did not succeed in capturing the foreground digits in the Textured MNIST dataset. b) IODINE groups ImageNet not into meaningful objects but mostly into regions of similar color. c) On a grayscale version of CLEVR, IODINE still produces the desired groupings.

VAE, where no stochasticity enters the inference process. If we had an exact and deterministic way to compute the likelihood and its gradient, this effect would vanish.

A neat side-effect of this is the ability to elegantly capture ambiguous (aka multi-stable) segmentations such as the ones shown in Figure 10. We presented IODINE with an ambiguous arrangement of Tetris blocks, which has three different yet equally valid "explanations" (given the data distribution). When we evaluate a trained model on this image, we get different segmentations on different evaluations. Some of these correspond to different slot-orderings (1st vs 3rd row). But we also find qualitatively different segmentations (i.e. 3rd vs 4th row) that correspond to different interpretations of the scene. This is an impressive result given that multi-stability is a well-studied, pervasive feature of human perception that is important for handling ambiguity, and that is not modelled by any standard image recognition networks.

5. Discussion and Future Work

We have introduced IODINE, a novel approach for unsupervised representation learning of multi-object scenes, based on amortized iterative refinement of the inferred latent representation. We analyzed IODINE's performance on various datasets, including realistic images containing variable numbers of partially occluded 3D objects, and demonstrated that our method can successfully decompose the scenes into objects and represent each of them in terms of their individual properties such as color, size, and material. IODINE can robustly deal with occlusions by inpainting covered sections, and generalises beyond the training distribution in terms of numerosity and object-property combinations. Furthermore, when applied to scenes with ambiguity in terms of their object decomposition, IODINE can represent – and converge

to – multiple valid solutions given the same input image.

We also probed the limits of our current setup by applying IODINE to the Textured MNIST dataset (Greff et al., 2016) and to ImageNet, testing how it would deal with texture-segmentation and more complex real-world data (Figure 11). Trained on ImageNet data, IODINE segmented mostly by color rather than by objects. This behavior is not unexpected: ImageNet was never designed as a dataset for unsupervised learning, and likely lacks the richness in poses, lighting, sizes, positions and distance variations required to learn object segmentations from scratch. Trained on Textured MNIST, IODINE was able to model the background, but mostly failed to capture the foreground digits. Together these results point to the importance of color as a strong cue for segmentation, especially early in the iterative refinement process. As demonstrated by our results on grayscale CLEVR (Figure 11c) though, color is not a requirement.

Beyond more diverse training data, we want to highlight three other promising directions to scale IODINE to richer real-world data. First, an extension to sequential data is attractive, because temporal data naturally contains rich statistics about objectness both in the movement itself, and in the smooth variations of object factors. IODINE can readily be applied to sequences feeding a new frame at every iteration, and we have done some preliminary experiments described in Appendix A.1. As a nice side-effect, the model automatically maintains the object to slot association, turning it into an unsupervised object tracker.

Physical interaction between objects is another common occurrence in sequential data. IODINE in its current form has limited abilities for modelling dynamics. Even statically placed objects commonly adhere to certain relations between each other, such as cars on streets. IODINE currently assumes objects to be placed independently of each other; relaxing this assumption will be important for modelling physical interactions. Yet there is also a need to balance this with the independence assumption required to split objects, since the system should still be able to segment out a car floating in space. Thus we believe integration with some form of graph network to support relations while preserving slot symmetry is another promising direction.

Finally, object representations have to be useful, such as for supervised tasks, or for agents in reinforcement learning setups. Whatever the task, it should provide important feedback about which objects matter and which are irrelevant. Complex visual scenes can contain an extremely large number of potential objects (think of sand grains on a beach), which can make it unfeasible to represent them all simultaneously. Allowing task-related signals to bias selection, for what and how to decompose, may enable scaling up unsupervised scene representation learning approaches like IODINE to arbitrarily complex scenes.

Acknowledgements

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv:1607. 06450 [cs, stat]*, July 2016.

Bengio, Y., Courville, A., and Vincent, P. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013.

Burgess, C. P., Matthey, L., Watters, N., Kabra, R., Higgins, I., Botvinick, M., and Lerchner, A. MONet: Unsupervised scene decomposition and representation. *arXiv preprint*, January 2019.

Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., and Abbeel, P. InfoGAN: Interpretable representation learning by information maximizing generative adversarial nets. *arXiv:1606.03657 [cs, stat]*, June 2016.

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (ELUs). November 2015.

Eastwood, C. and Williams, C. K. I. A framework for the quantitative evaluation of disentangled representations. *ICLR*, 2018.

Eslami, S. M. A., Heess, N., Weber, T., Tassa, Y., Szepesvari, D., Kavukcuoglu, K., and Hinton, G. E. Attend, infer, repeat: Fast scene understanding with generative models. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3225–3233. Curran Associates, Inc., 2016.

Felzenszwalb, P. F. and Huttenlocher, D. P. Efficient Graph-Based image segmentation. *Int. J. Comput. Vis.*, 59(2): 167–181, September 2004.

Girshick, R. B. Fast R-CNN. *CoRR*, abs/1504.08083, 2015. URL http://arxiv.org/abs/1504.08083.

Greff, K., Rasmus, A., Berglund, M., Hao, T. H., Valpola, H., and Schmidhuber, J. Tagger: Deep unsupervised perceptual grouping. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4484–4492, 2016.

Greff, K., van Steenkiste, S., and Schmidhuber, J. Neural expectation maximization. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6694–6704. Curran Associates, Inc., 2017.

Gregor, K., Danihelka, I., Graves, A., Rezende, D. J., and Wierstra, D. DRAW: A recurrent neural network for image generation. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1462–1471, Lille, France, 2015. JMLR.org.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. B. Mask R-CNN. *CoRR*, abs/1703.06870, 2017. URL http://arxiv.org/abs/1703.06870.

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *In Proceedings of the International Conference on Learning Representations (ICLR)*, 2017a.

Higgins, I., Pal, A., Rusu, A., Matthey, L., Burgess, C., Pritzel, A., Botvinick, M., Blundell, C., and Lerchner, A. DARLA: Improving zero-shot transfer in reinforcement learning. *ICML*, 2017b.

Higgins, I., Amos, D., Pfau, D., Racanière, S., Matthey, L., Rezende, D. J., and Lerchner, A. Towards a definition of disentangled representations. *CoRR*, abs/1812.02230, 2018. URL http://arxiv.org/abs/1812.02230.

Hubert, L. and Arabie, P. Comparing partitions. *J. Classification*, 2(1):193–218, December 1985.

Johnson, J., Hariharan, B., van der Maaten, L., Fei-Fei, L., Zitnick, C. L., and Girshick, R. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 1988–1997. openaccess.thecvf.com, 2017.

Kim, H. and Mnih, A. Disentangling by factorising. February 2018.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. CBLS, 2015.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lake, B. M., Salakhutdinov, R., and Tenenbaum, J. B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, December 2015.

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., and Gershman, S. J. Building machines that learn and think like people. *Behavioral and Brain Sciences*, pp. 1–101, 2016.

Locatello, F., Bauer, S., Lucic, M., Gelly, S., Schölkopf, B., and Bachem, O. Challenging common assumptions in the unsupervised learning of disentangled representations. *arXiv preprint arXiv:1811.12359*, 2018.

Maaten, L. v. d. and Hinton, G. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov): 2579–2605, 2008.

Makhzani, A., Shlens, J., Jaitly, N., Goodfellow, I., and Frey, B. Adversarial autoencoders. November 2015.

Marino, J., Cvitkovic, M., and Yue, Y. A general method for amortizing variational filtering. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 31*, pp. 7868–7879. Curran Associates, Inc., 2018a.

Marino, J., Yue, Y., and Mandt, S. Iterative amortized inference. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3403–3412, Stockholmsmässan, Stockholm Sweden, 2018b. PMLR.

Matthey, L., Higgins, I., Hassabis, D., and Lerchner, A. dsprites: Disentanglement testing sprites dataset. https://github.com/deepmind/dsprites-dataset/, 2017.

Nickel, M. and Kiela, D. Poincaré embeddings for learning hierarchical representations. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30*, pp. 6338–6347. Curran Associates, Inc., 2017.

Pascanu, R., Mikolov, T., and Bengio, Y. Understanding the exploding gradient problem. *CoRR, abs/1211. 5063*, 2012.

Pero, L. D., Bowdish, J., Fried, D., Kermgard, B., Hartley, E., and Barnard, K. Bayesian geometric modeling of indoor scenes. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2719–2726. ieeexplore.ieee.org, June 2012.

Rand, W. M. Objective criteria for the evaluation of clustering methods. *J. Am. Stat. Assoc.*, 66(336):846–850, December 1971.

Redmon, J. and Farhadi, A. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018. URL http://arxiv.org/abs/1804.02767.

Reichert, D. P. and Serre, T. Neuronal Synchrony in Complex-Valued Deep Networks. *arXiv:1312. 6115 [cs, q-bio, stat]*, December 2013.

Rezende, D. J., Mohamed, S., and Wierstra, D. Stochastic backpropagation and approximate inference in deep generative models. In Xing, E. P. and Jebara, T. (eds.), *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pp. 1278–1286, Bejing, China, 2014. PMLR.

Ridgeway, K. and Mozer, M. C. Learning deep disentangled embeddings with the f-statistic loss. *NIPS*, 2018.

Shi, J. and Malik, J. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8): 888–905, August 2000.

van Steenkiste, S., Chang, M., Greff, K., and Schmidhuber, J. Relational neural expectation maximization: Unsupervised discovery of objects and their interactions. In *Proceedings of the International Conference on Learning Representations (ICLR)*, January 2018.

Watters, N., Matthey, L., Burgess, C. P., and Lerchner, A. Spatial broadcast decoder: A simple architecture for learning disentangled representations in vaes. *arXiv*, 1901.07017, 2019.
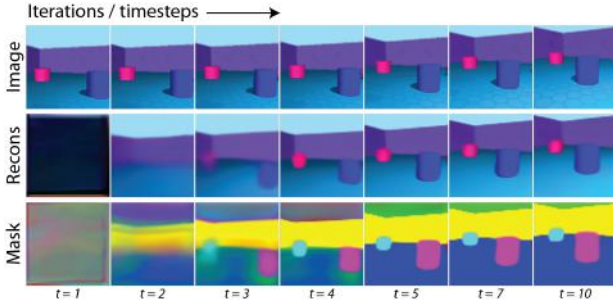
A. Further Discussion

A.1. Sequences

Figure 12. IODINE applied to Objects Room sequences by setting N, the number of refinement iterations, equal to the number of timesteps in the data.

The iterative nature of IODINE lends itself readily to sequential data, by, e.g., feeding a new frame at every iteration, instead of the same input image **x**. This setup corresponds to one iteration per timestep, and using next-step-prediction instead of reconstruction as part of the training objective. An example of this can be seen in Figure 12 where we show a 16 timestep sequence along with reconstructions and masks. When using the model in this way, it automatically maintains the association of object to slot over time (i.e, displaying robust slot stability). Thus, object tracking comes almost for free as a by-product in IODINE. Notice though, that IODINE has to rely on the LSTM that is part of the inference network to model any dynamics. That means none of the dynamics of tracked objects (e.g. velocity) will be part of the object representation.

A.2. Memory Limitations

It is worth pointing out that memory consumption presents an important limiting factor to scaling IODINE. To allow training by backpropagation, each slot and each refinement step require the storage of activations for an entire decoder and refinement network. Memory consumption during training thus scales linearly with both K and T. This is particularly restrictive for sequential data, where the number of steps can grow very large. In our experiments from Appendix A.1, we found that 16 timesteps with a batch-size of 4 was the upper limit on V100 GPUs with 16GB of RAM. Of course this also depends on the size of the input and the size of the network. Note also that at inference time there is no need to keep the activations of previous timesteps, so the dependence on T can be eliminated there.

A.3. Comparison with MONet

The Multi-Object NETwork (MONet; Burgess et al. 2019) is a complementary method for unsupervised object representation learning also developed recently. It learns to sequentially attend to individual objects using a masking network and a VAE. In each step the masking network segments out a yet unexplained part of the image (the next object) which is then fed to the VAE which has to reconstruct that object and the mask. Thus, in contrast to IODINE, MONet uses one iteration per object and doesn't adjust an object once it has been covered.

Both methods focus on the representation learning aspect and both ensure that all objects are encoded in the same format by sharing weights across objects. In our preliminary experiments MONet produced results very similar to IODINE on CLEVR both in terms of segmentation and regarding the quality of object representations, and also learns to inpaint occluded parts of objects.

Since MONet only visits each object once, it is a more lightweight method that requires less computation and memory to train and run. Recurrently iterating over objects also has the benefit that the model can dynamically vary the number of objects, whereas in IODINE the maximum number of objects is a hyperparameter that has to be fixed manually (though it can be changed at test time). The usage of a separate masking network which isn't directly subject to a representational bottleneck likely leads to less regularization for the segmentation mask. This could potentially allow MONet to better deal with complex segmentation shapes. But it also has to use that ability to directly produce masks that respect occlusion, whereas IODINE tends to produce masks for full unoccluded objects and leverages the softmax to resolve overlap. For more complex scenes, we also expect iterative refinement to be advantageous for resolving difficult cases. There, IODINE could start with a rough segmentation and then use the progressively better understanding of the constituent objects for refining the boundaries.

The segmentation process of MONet is deterministic which induces an order on the objects, which might be useful because it naturally prioritizes salient objects. We observed that it typically starts with the background, then processes large frontal objects, and finally smaller or farther away objects. But this approach does break symmetry between objects, and we prefer keeping such a bias out of the object segmentation learning as much as possible.

Another disadvantage of a deterministic segmentation is that it cannot directly deal with ambiguous cases like the one shown in Section 4.4 and Figure 10. The iterative message-passing-like approach of IODINE might also lend itself well for incorporating top-down feedback to bias the segmentation towards one that is useful for a given task. It is less clear how to do that in MONet, though adding a way for conditioning the masking network could potentially serve a similar purpose. Finally the iterative refinement of IODINE naturally extends to sequential data (see Appendix A.1)

which would be less straightforward for MONet.

In summary, it is not at all clear yet which approach will work better and under which circumstances. If the data is sequential or contains ambiguity, IODINE presents a better choice. For other data that is not visually more complex than CLEVR, both methods will likely produce similar results making MONet the simpler and less computationally intensive choice. For more complex data it is unclear yet which approach would be the better choice, and in fact a hybrid approach might be the most promising. Sequentially attending to objects and iterative refinement are not mutually exclusive and might support each other. We consider this a very attractive research direction and are excited to explore its possibilities.

B. Dataset Details

B.1. CLEVR

We regenerated the CLEVR dataset (Johnson et al., 2017) using the authors' open-source code, because we needed ground-truth segmentation masks for evaluation purposes. The dataset contains $70\,000$ images with a resolution of 240×320 pixels, from which we extract a square center crop of 192×192 and scale it to 128×128 pixels. Each scene contains between three and ten objects, characterized in terms of shape (cube, cylinder, or sphere), size (small or large), material (rubber or metal), color (8 different colors), position (continuous), and rotation (continuous).

The subset of images which contain 3-6 objects (inclusive) served as the training set for our experiments; we refer to it as CLEVR6. Unless noted otherwise, we evaluate models on the full CLEVR distribution, containing 3-10 objects. All references to CLEVR refer to the full distribution.

We do not make use of the question answering task. Figure 13 shows a few samples from the dataset.
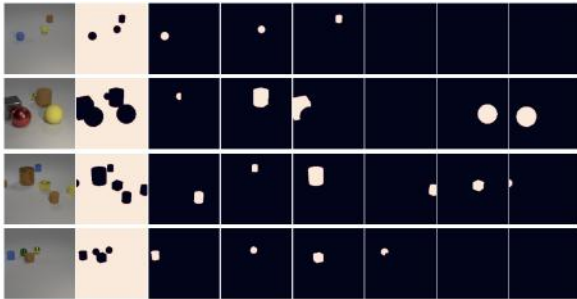
Figure 13. Samples from CLEVR6. The first column is the scene, the second column is the background mask and the following columns are the ground-truth object masks.

B.2. Multi-dSprites

This dataset, based on the dSprites dataset (Matthey et al., 2017), consists of $60\,000$ images with a resolution of 64×64.

Each image contains two to five random sprites, which vary in terms of shape (square, ellipse, or heart), color (uniform saturated colors), scale (continuous), position (continuous), and rotation (continuous). Furthermore the background color is varied in brightness but always remains grayscale. Figure 14 shows a few samples from the dataset.

We also used a binarized version of Multi-dSprites, where the sprites are always white, the background is always black, and each image contains two to three random sprites.

B.3. Tetris

We generated this dataset of $60\,000$ images by placing three random Tetrominoes without overlap in an image of 35×35 pixels. Each Tetromino is composed of four blocks that are each 5×5 pixels. There are a total of 17 different Tetrominoes (counting rotations). We randomly color each Tetromino with one of 6 colors (red, green, blue, cyan, magenta, or yellow). Figure 15 shows a few samples from the dataset.

B.4. Shapes

We use the same shapes dataset as in (Reichert & Serre, 2013). It contains $60\,000$ binary images of size 28×28 each with three random shapes from the set $\{\triangle, \nabla, \square\}$.

B.5. Objects Room

For the preliminary sequential experiments we used a sequential version of the *Objects Room* dataset (Burgess et al., 2019). This dataset consists of 64x64 RGB images of a cubic room, with randomly colored walls, floors and objects randomly scattered around the room. The camera is always positioned on a ring inside the room, always facing towards the centre and oriented vertically in the range $(-25°, 22°)$. There are 3 randomly shaped objects in the room with 1-3 objects visible in any given frame. This version contains sequences of camera-flights for 16 time steps, with the camera
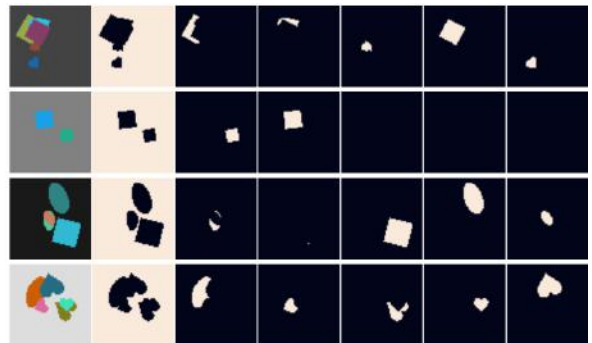

Figure 14. Samples from the Multi-dSprites dataset. The first column is the full image, the second column is the background mask and the following columns are the ground-truth object masks.
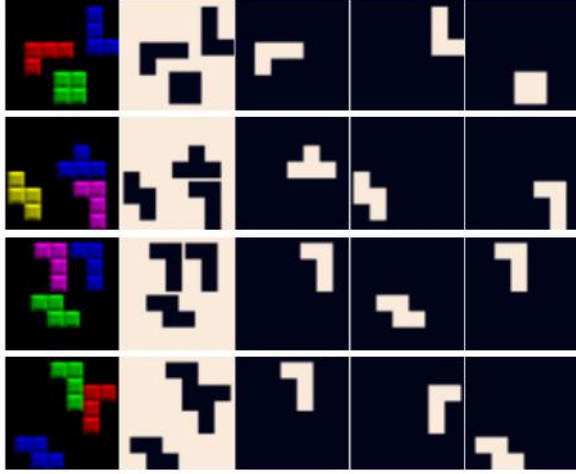
Figure 15. Samples from the Tetris dataset. The first column is the full image, the second column is the background mask and the following columns are the ground-truth object masks.

position and angle (within the above constraints) changing according to a fixed velocity for the entire sequence (with a random velocity sampled for each sequence).

C. Model and Hyperparameter Details

Training Unless otherwise specified all the models are trained with the ADAM optimizer (Kingma & Ba, 2015), with default parameters and a learning rate of 0.0003. We used gradient clipping as recommended by (Pascanu et al., 2012): if the norm of global gradient exceeds 5.0 then the gradient is scaled down to that norm. Note that this is virtually always the case as the gradient norm is typically on the order of 10^5, but we nonetheless found it useful to apply this strategy. We always use $\sigma = 0.1$ for the global scale of the output distribution $p(\mathbf{x}|\mathbf{z}^{(t)}) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k^{(t)}, \sigma^2)$. Finally, batch size was 32 (4×8GPUs).

Initialization of Posterior IODINE iteratively refines an initial posterior $\boldsymbol{\lambda}^{(1)}$ which is independent of the input data. Initially we set this initial value to match the prior (i.e. $q_{\boldsymbol{\lambda}}(\mathbf{z}_k^{(1)}) = \mathcal{N}(\mathbf{0}, \mathbf{1})$). But we found that this poses problems for the model, because of the competing requirements it poses for structuring the latent space w.r.t. the prior: On the one hand, samples from the prior need to be good starting values for iterative refinement. On the other hand, the prior should correspond to the accumulated posterior (KL term). For this reason we decided to simply make the parameters $\boldsymbol{\lambda}^{(1)}$ of the initialization distribution trainable parameters which are optimized alongside the weights of the decoder ($\boldsymbol{\theta}$) and of the refinement network ($\boldsymbol{\phi}$). This lead to faster training, and improved the visual quality of reconstructions from prior samples.

Inputs For all models, we use the following inputs to the refinement network, where LN means Layernorm and SG means stop gradients, and we omit the iteration index $\cdot^{(t)}$ for brevity. The following image-sized inputs are concatenated and fed to the corresponding convolutional network:

Description	Formula	LN	SG	Ch.
image	$\mathbf{x}$			3
means	$\boldsymbol{\mu}$			3
mask	$\boldsymbol{m}_k$			1
mask-logits	$\hat{\boldsymbol{m}}_k$			1
mask posterior	$p(\boldsymbol{m}_k|\mathbf{x}, \boldsymbol{\mu})$			1
gradient of means	$\nabla_{\boldsymbol{\mu}_k}\mathcal{L}$	✓	✓	3
gradient of mask	$\nabla_{\boldsymbol{m}_k}\mathcal{L}$	✓	✓	1
pixelwise likelihood	$p(\mathbf{x}|\mathbf{z})$	✓	✓	1
leave-one-out likelih.	$p(\mathbf{x}|\mathbf{z}_{i\neq k})$	✓	✓	1
coordinate channels				2
			total:	17

The posterior parameters $\boldsymbol{\lambda}$ and their gradients are flat vectors, and as such we concatenate them with the output of the convolutional part of the refinement network and use the result as input to the refinement LSTM:

Description	Formula	LN	SG
gradient of posterior	$\nabla_{\boldsymbol{\lambda}_k}\mathcal{L}$	✓	✓
posterior	$\boldsymbol{\lambda}_k$		

Architecture All layers use the ELU (Clevert et al., 2015) activation function and the Convolutional layers use a stride equal to 1, unless mentioned otherwise. Architecture details for the individual datasets are summarized in the following subsections.

C.1. CLEVR

All models were trained on scenes with 3-6 objects (CLEVR6) with $K = 7$ slots and $T = 5$ iterations, and a latent object dimension of size $\dim(\mathbf{z}_k) = 64$. Training was done on eight V100 GPUs for 1M updates (approx. 1 week wall time). When evaluating on the full CLEVR dataset, we increased the number of slots to $K = 11$. For some of the analysis, we varied T and K as mentioned in the text.

The rest of the architecture and hyperparameters are described in the following.

Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: **z**	64		
Broadcast	*66*		+ coordinates
Conv 3×3	*64*	ELU	
Conv 3×3	*64*	ELU	
Conv 3×3	*64*	ELU	
Conv 3×3	*64*	ELU	
Conv 3×3	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	128	Linear	
LSTM	256	Tanh	
Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$	512		
MLP	256	ELU	
Avg. Pool	64		
Conv 3×3	*64*	ELU	stride 2
Conv 3×3	*64*	ELU	stride 2
Conv 3×3	*64*	ELU	stride 2
Conv 3×3	*64*	ELU	stride 2
Inputs	*17*		

Deconv Decoder used in Section 4.3

Type	Size/*Ch.*	Act. Func.	Comment
Input: **z**	64		
MLP	512	ELU	
MLP	512	ELU	
Reshape	*8*		$8 \times 8 \times 8$
Conv 5×5	*64*	ELU	stride 2
Conv 5×5	*64*	ELU	stride 2
Conv 5×5	*64*	ELU	stride 2
Conv 5×5	*64*	ELU	stride 2
Conv 5×5	*64*	ELU	
Conv 5×5	*4*	Linear	RGB + Mask

C.2. Multi-dSprites

Models were trained with $K = 6$ slots, and used $T = 5$ iterations.

Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: $\boldsymbol{\lambda}$	32		
Broadcast	*34*		+ coordinates
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	32	Linear	
LSTM	128	Tanh	
Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$	192		
MLP	128	ELU	
Avg. Pool	32		
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Inputs	*17*		

C.3. Tetris

Models were trained with $K = 4$ slots, and used $T = 5$ iterations. For Tetris, in contrast to the other models, we did not use an LSTM in the refinement network.

Decoder

Type	Size/*Ch.*	Act. Func.	Comment
Input: $\boldsymbol{\lambda}$	64		
Broadcast	*66*		+ coordinates
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*4*	Linear	RGB + Mask

Refinement Network

Type	Size/*Ch.*	Act. Func.	Comment
MLP	64	Linear	
Concat $[\boldsymbol{\lambda}, \nabla_{\boldsymbol{\lambda}}\mathcal{L}]$	256		
MLP	128	ELU	
Avg. Pool	32		
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Conv 5×5	*32*	ELU	
Inputs	*17*		
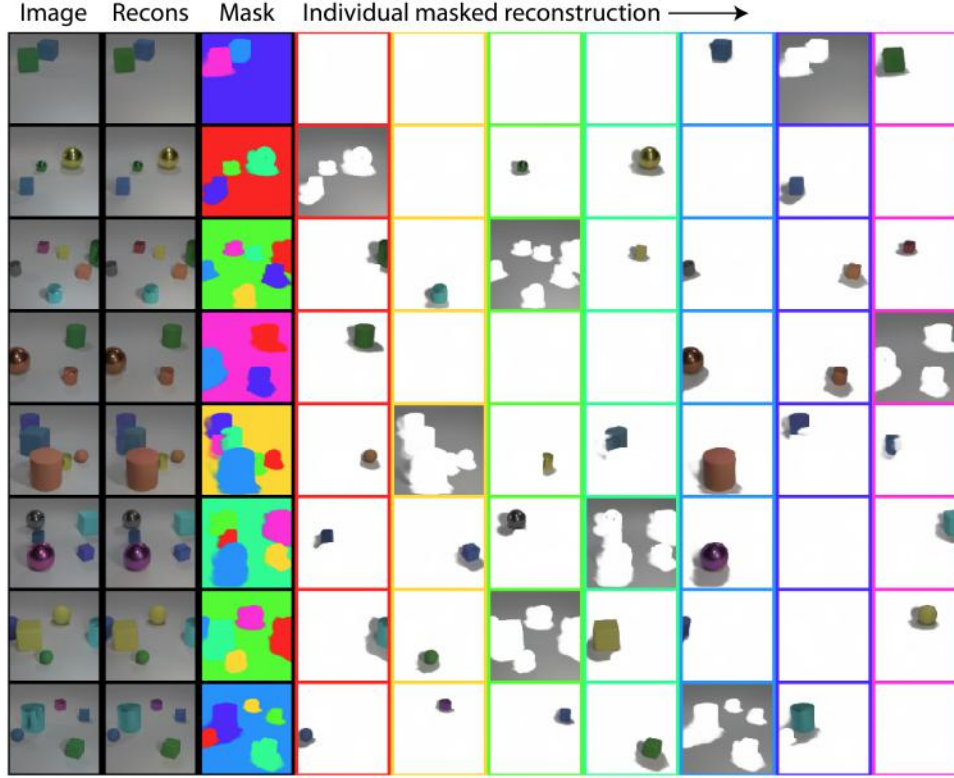
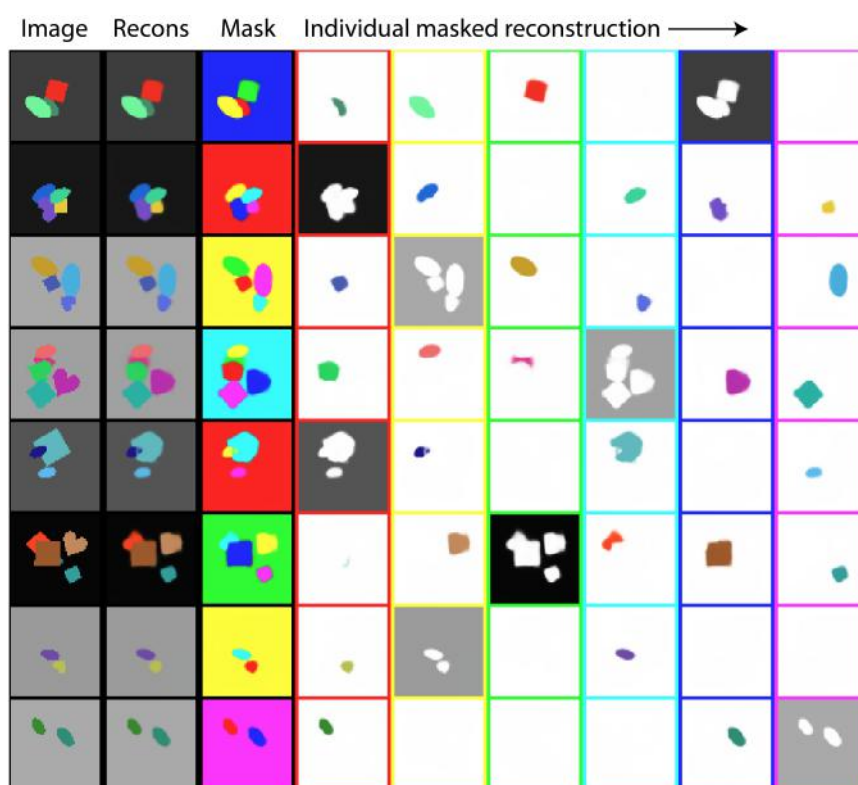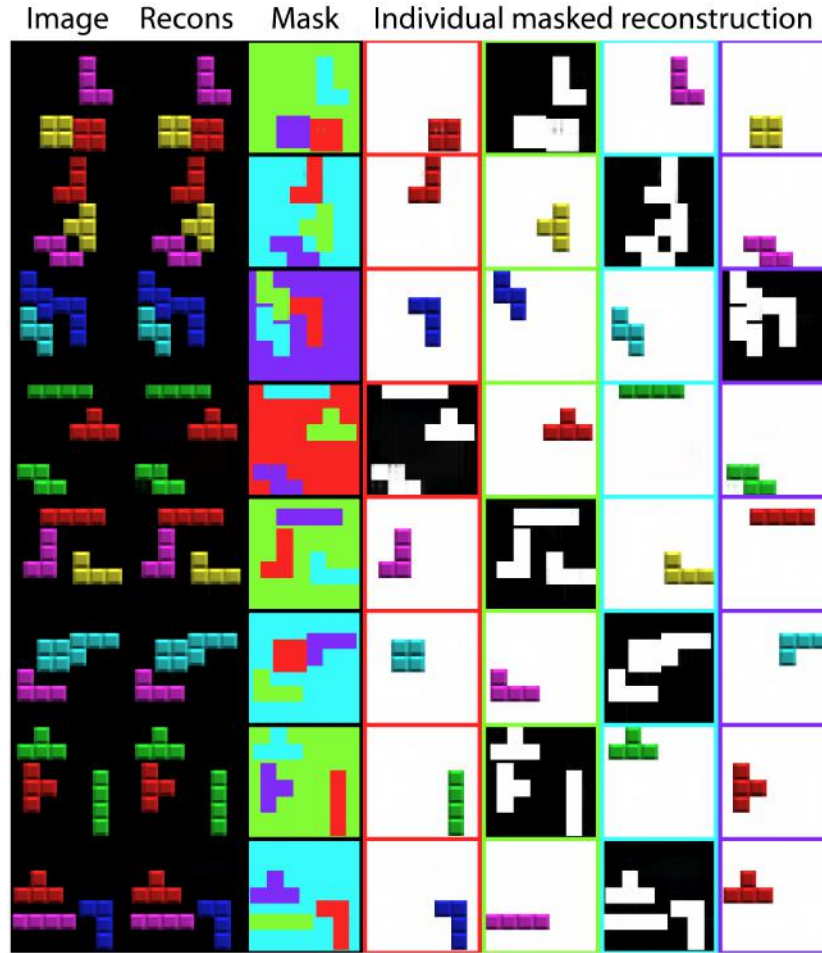*Figure 16.* Additional segmentation and object reconstruction results on CLEVR6. Border colors are matched to the segmentation mask on the left.

D. Additional Plots

Decompositions   Figures 16–18 show additional decomposition samples on our datasets. Figure 19 shows a complete version of Figure 7, showing all individual masked reconstruction slots. Figures 20–22 show a comparison between the object reconstructions and the mask logits used for assigning decoded latents to pixels.

Projections of Object Latents   Figures 24–26 demonstrate how object latents are clustered when projected onto the first two principal components of the latent distribution. Figures 27–29 show how object latents are clustered when projected onto a t-SNE (Maaten & Hinton, 2008) of the latent distribution.

Traversals   Figures 30–32 show additional (randomly chosen) latent traversals for IODINE on CLEVR like on the right side of Figure 6.

Input Ablations   Figures 33–40 give an overview of the impact of each of the inputs to the refinement network on the total loss, mean squared reconstruction error, KL divergence loss term, and the ARI segmentation performance (excluding the background pixels) on the CLEVR and Tetris datasets.

Figure 17. Additional segmentation and object reconstruction results on Multi-dSprites. Border colors are matched to the segmentation mask on the left.

Figure 18. Additional segmentation and object reconstruction results on Tetris. Border colors are matched to the segmentation mask on the left.
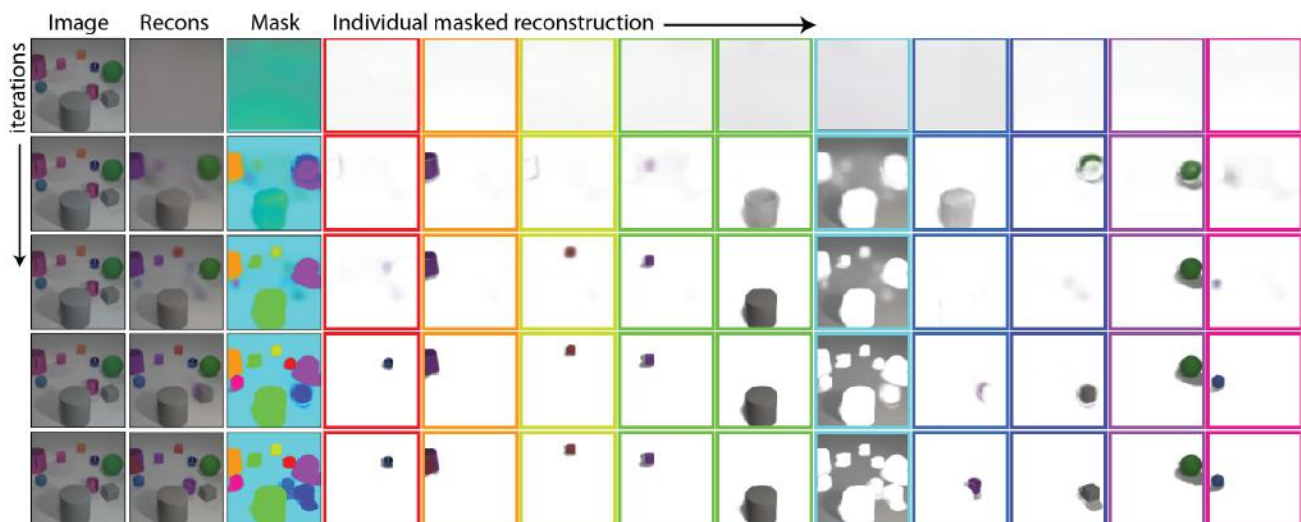
Figure 19. Full version of Figure 7, showcasing all slots.
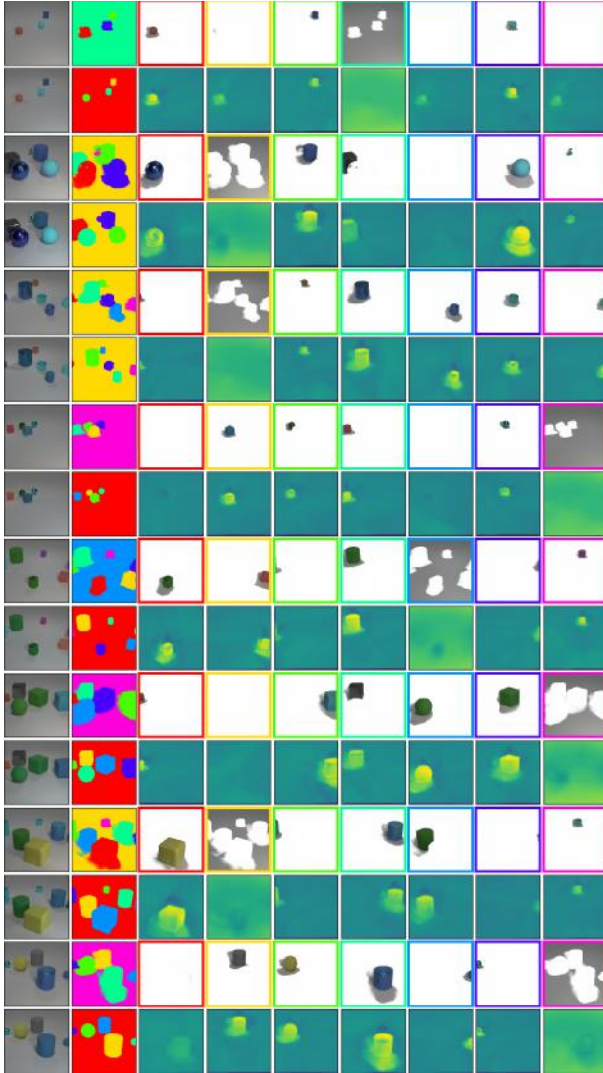
Figure 20. **CLEVR6** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.
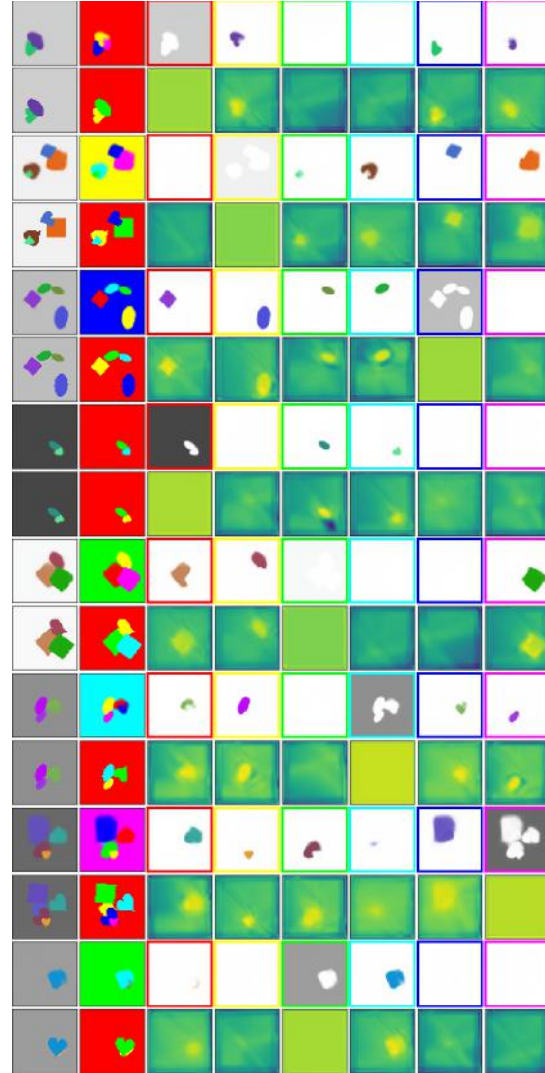
Figure 21. **Multi-dSprites** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.
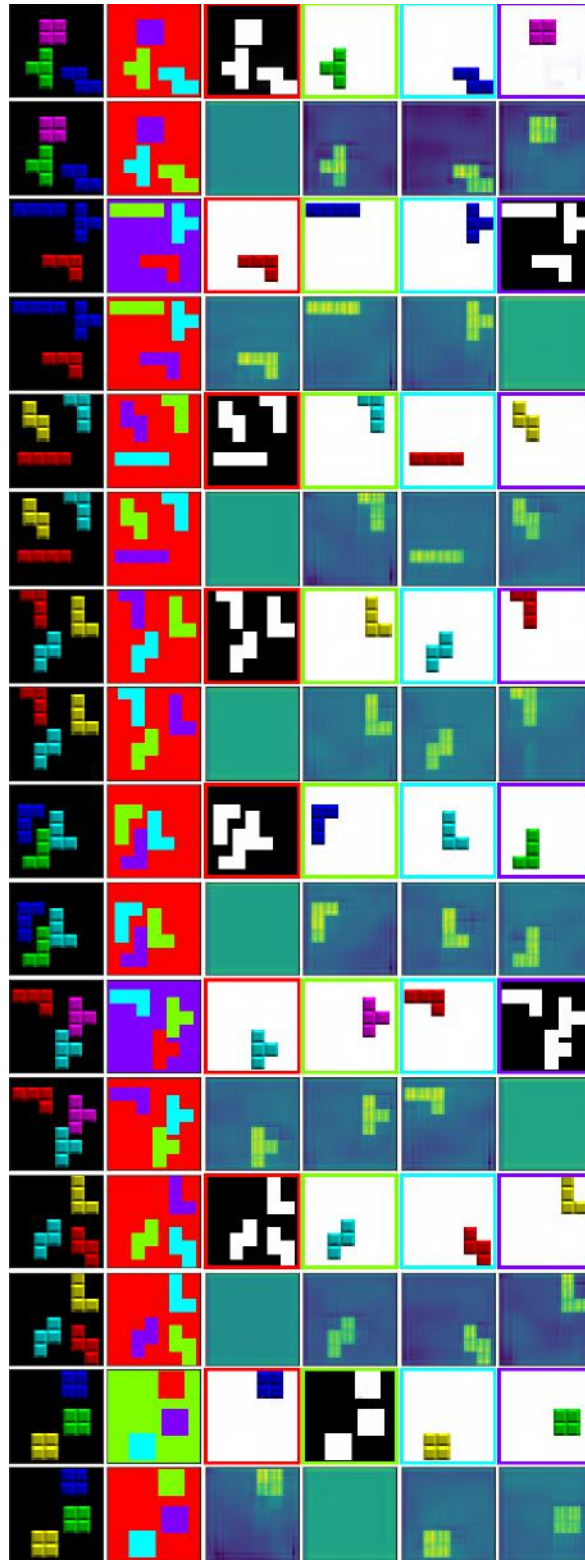
Figure 22. **Tetris** dataset. Odd rows: image and object masks as determined by the model. Even rows: first column is the input image, second one is the ground-truth masks and the following ones are mask logits produced by the model.
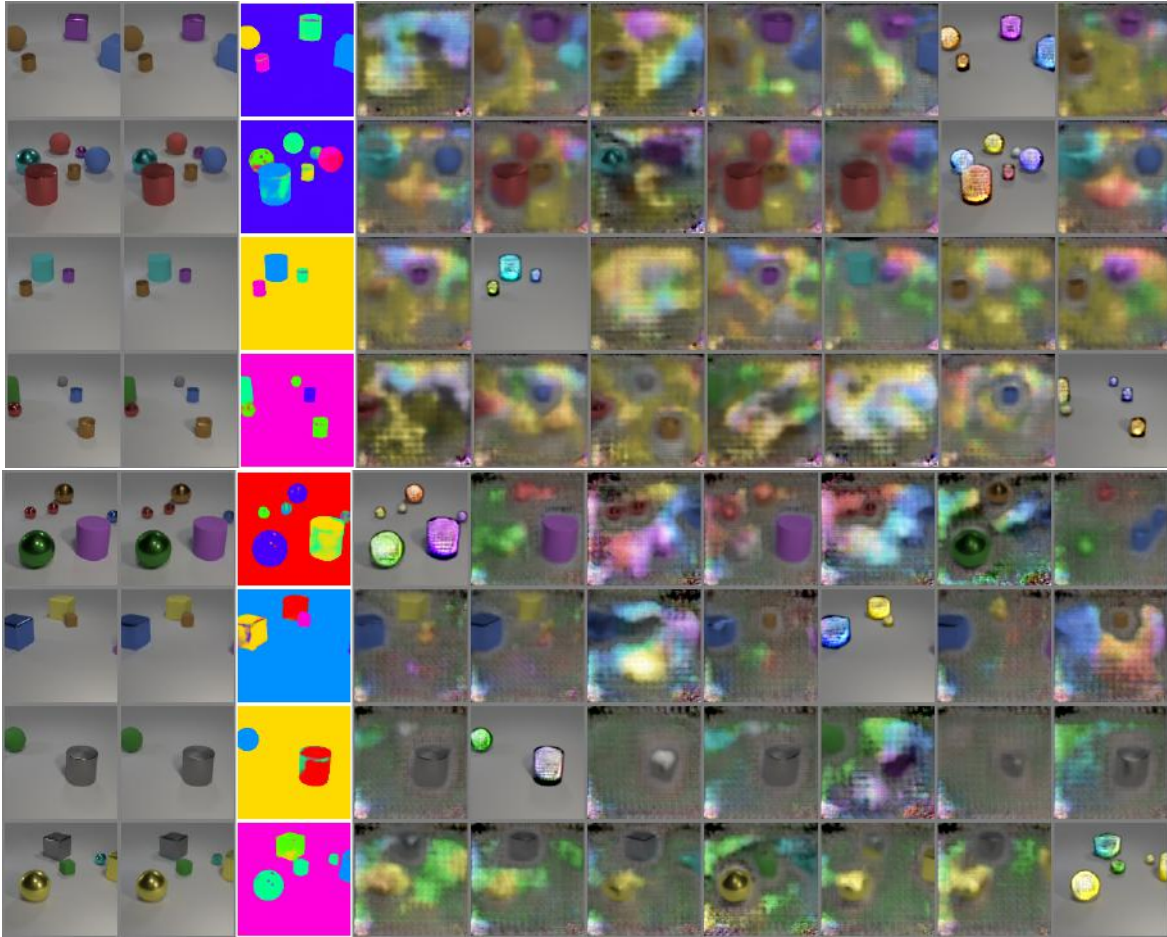
Figure 23. Segmentation and object reconstruction results on CLEVR6 using a deconvolution based decoder instead of the spatial broadcast decoder. Note that IODINE still cleanly segments objects from the background (now ignoring shadows), but specialization of the individual slots is much worse. Both, slots holding multiple objects, and objects replicated across multiple slots are much more frequent now. Slot reconstructions are also much less clean, containing much more noise and residue of other objects. (Note though, that in this figure we didn't mask the reconstructions as we have for Figure 16.)
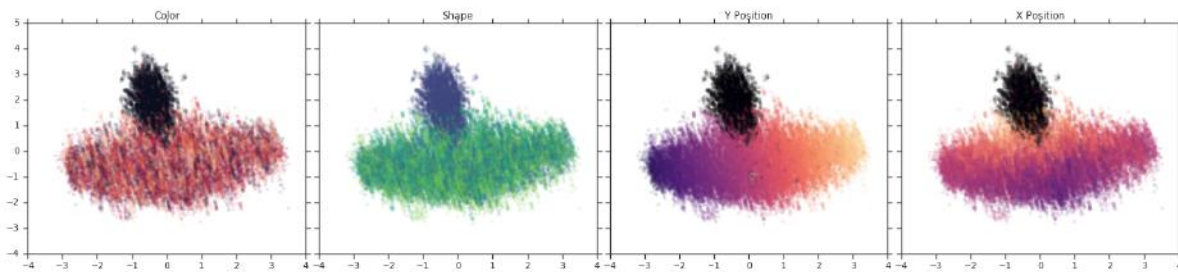
Figure 24. Projection on the first two principal components of the latent distribution for the **CLEVR6** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.
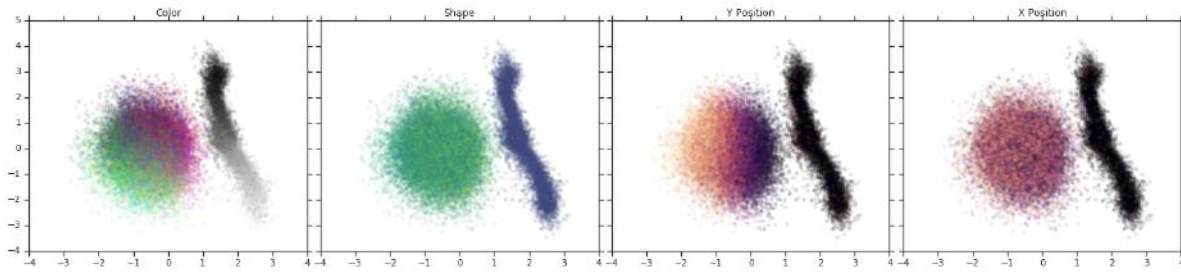
Figure 25. Projection on the first two principal components of the latent distribution for the **Multi-dSprites** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.
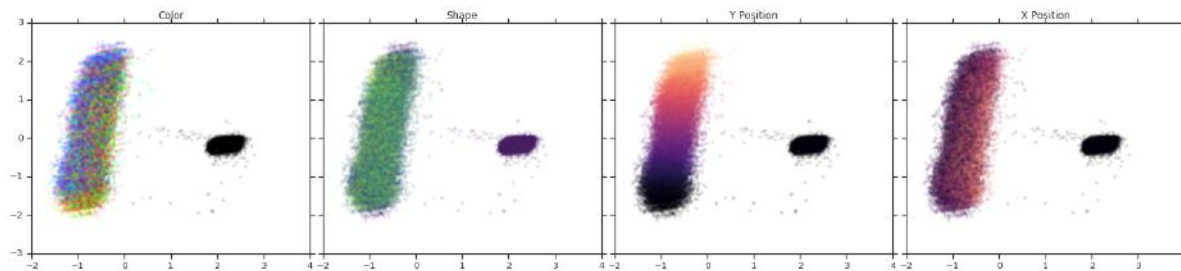
Figure 26. Projection on the first two principal components of the latent distribution for the **Tetris** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor.
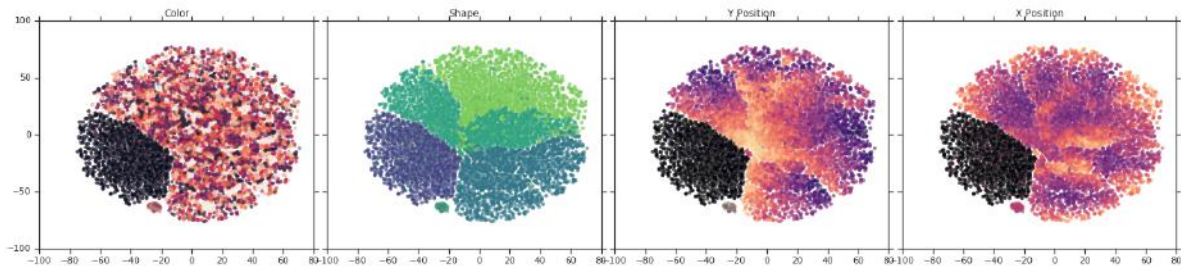
Figure 27. t-SNE of the latent distribution for the **CLEVR6** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor
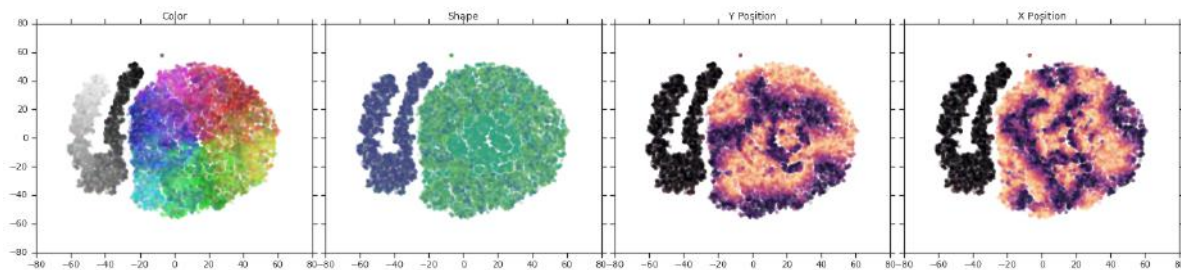
Figure 28. t-SNE of the latent distribution for the **Multi-dSprites** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor
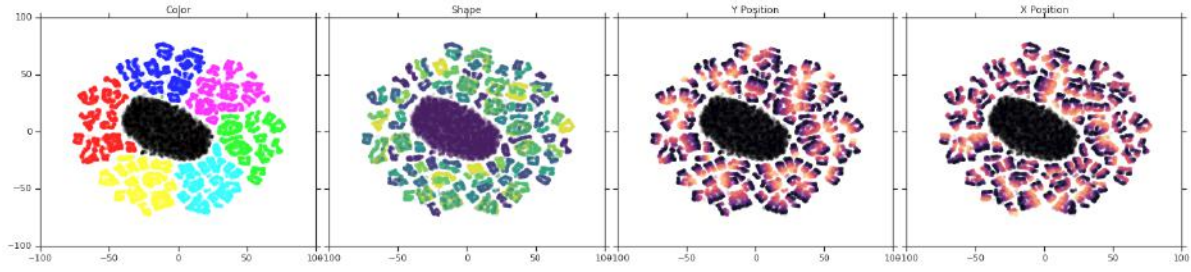
Figure 29. t-SNE of the latent distribution for the **Tetris** dataset. Each dot represents one object latent and is colored according to the corresponding ground truth factor
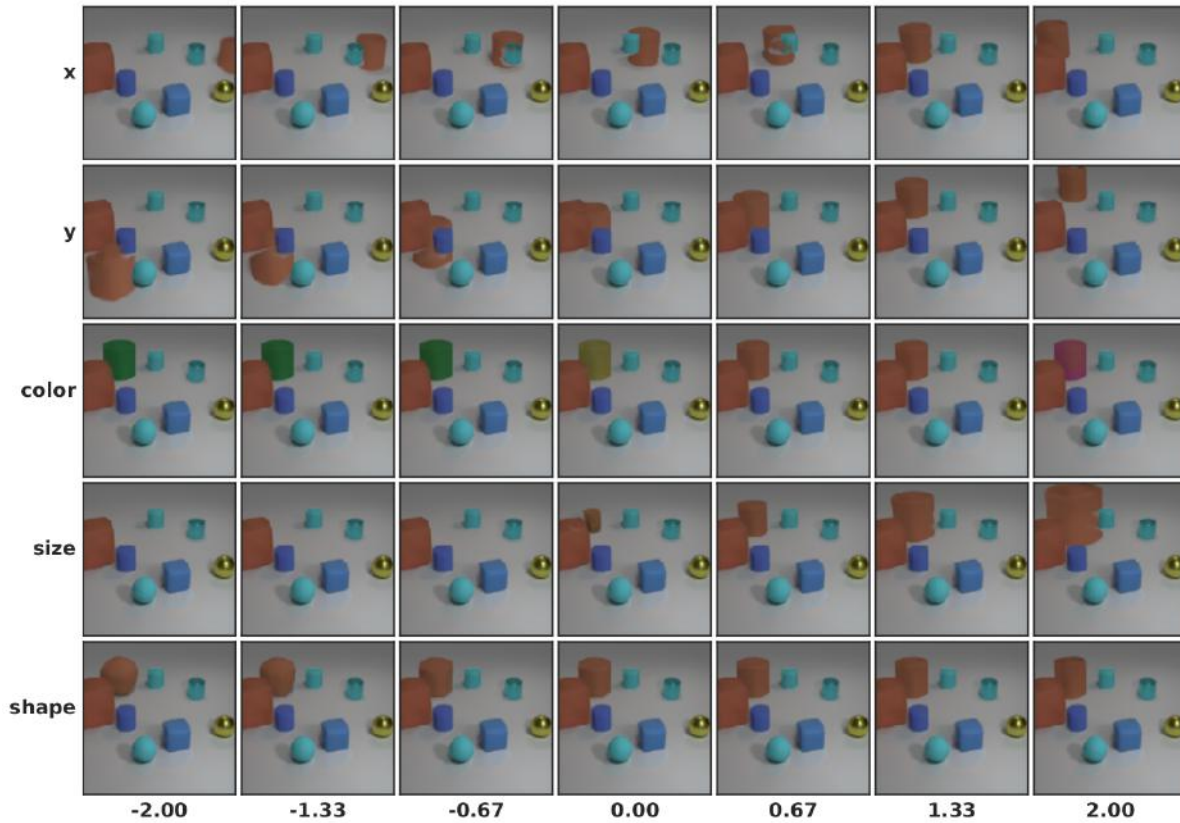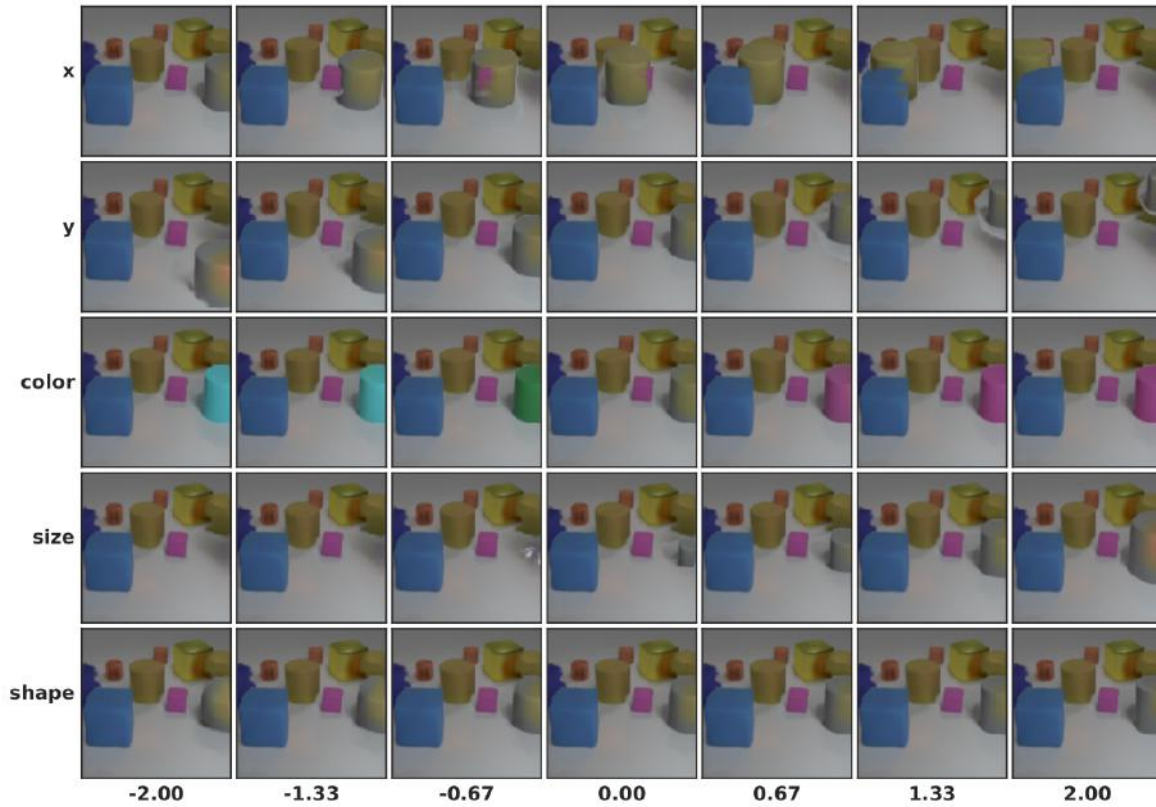

Figure 30. Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the brown cylinder in the back is changing. Occlusion handling shows several flaws, that could be fixed by adjusting another latent (not shown) that encodes the depth ordering.
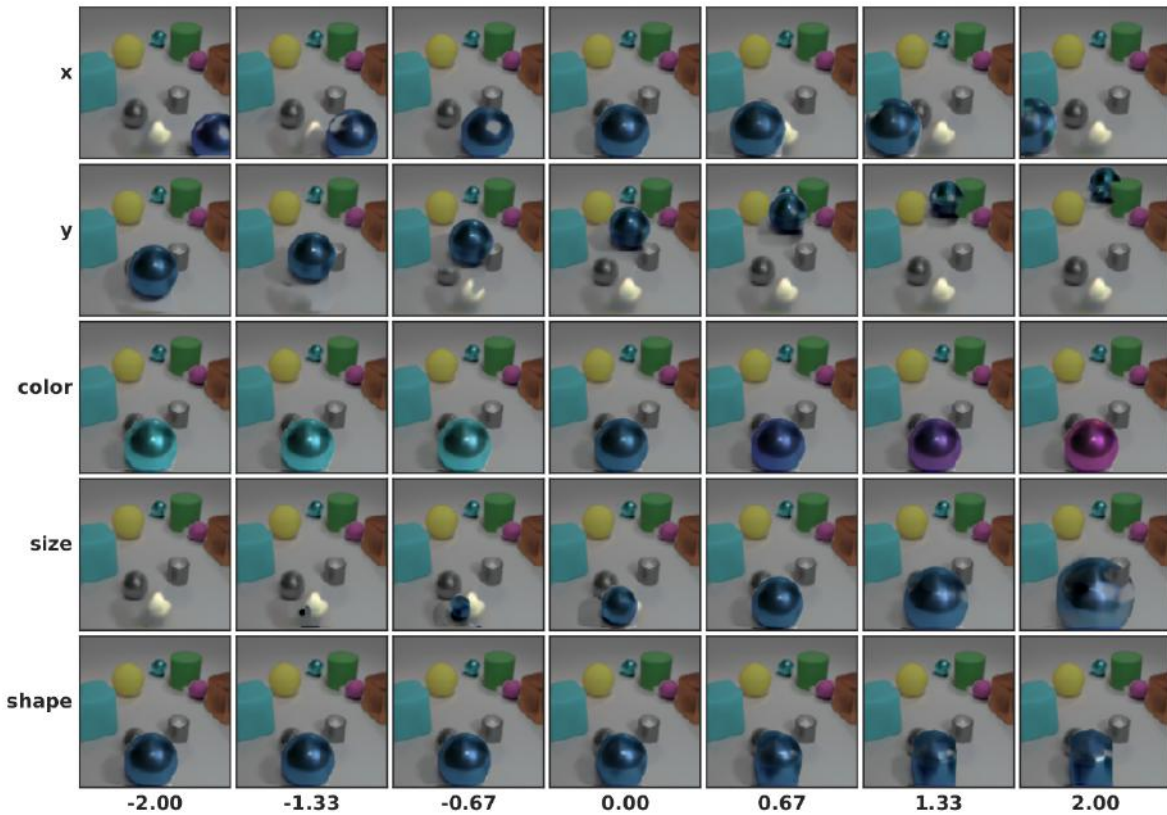
Figure 31. Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the large blue sphere in the front is changing. Note that the background slot contains a bright spot behind the blue sphere that becomes visible when the sphere is moved away.

Figure 32. Latent traversal of IODINE on CLEVR (like right side of Figure 6), for a randomly chosen example and randomly chosen slot. Here the gray cylinder on the right is changing. Occlusion handling shows several flaws, that could be fixed by adjusting another latent (not shown) that encodes the depth ordering.
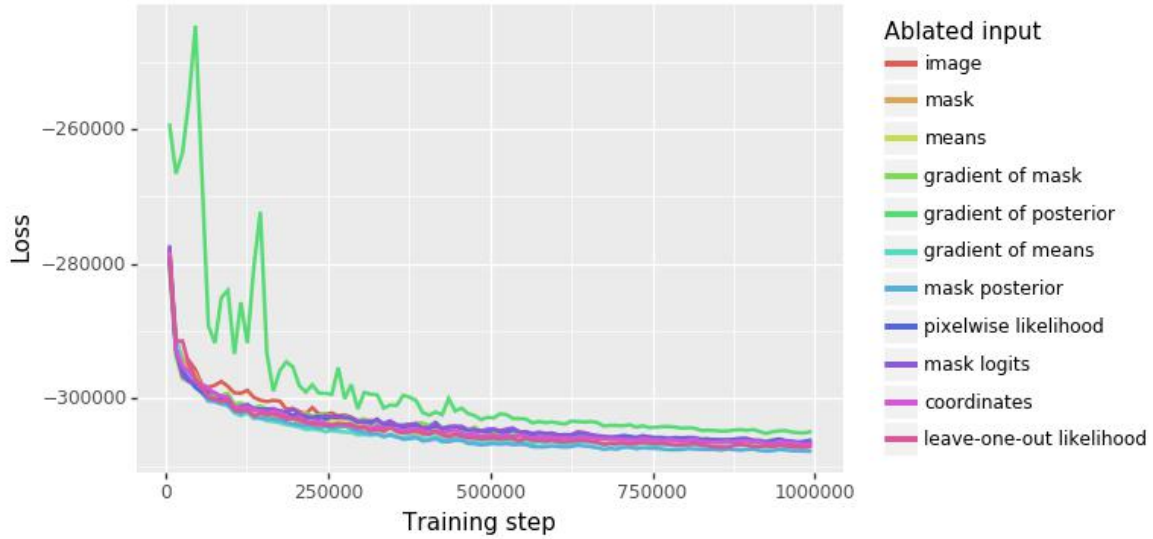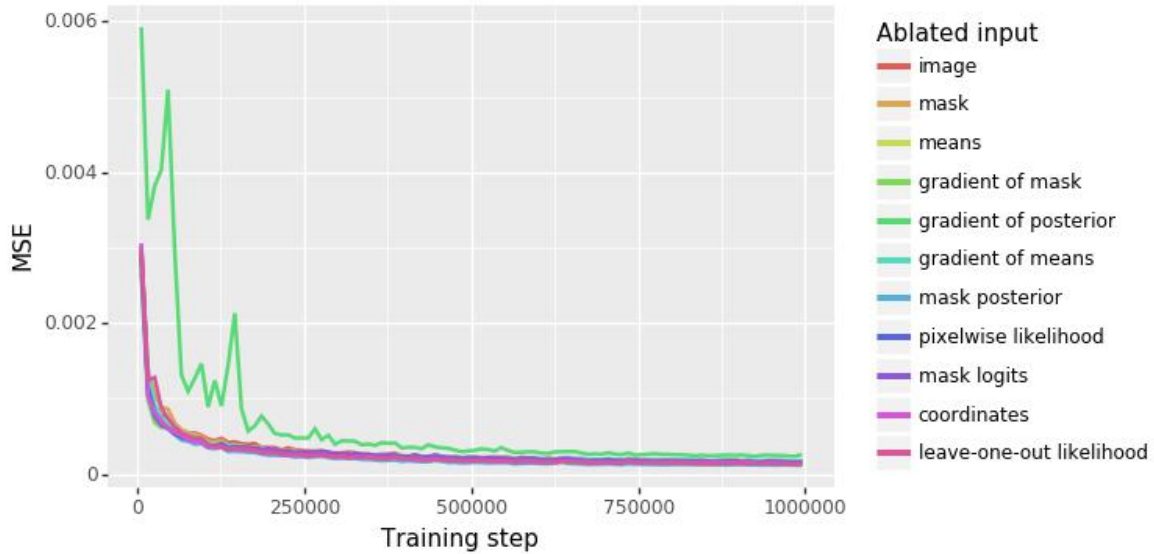

Figure 33. Ablation study for the model's total loss on CLEVR6. Each curve denotes the result of training the model without a particular input.
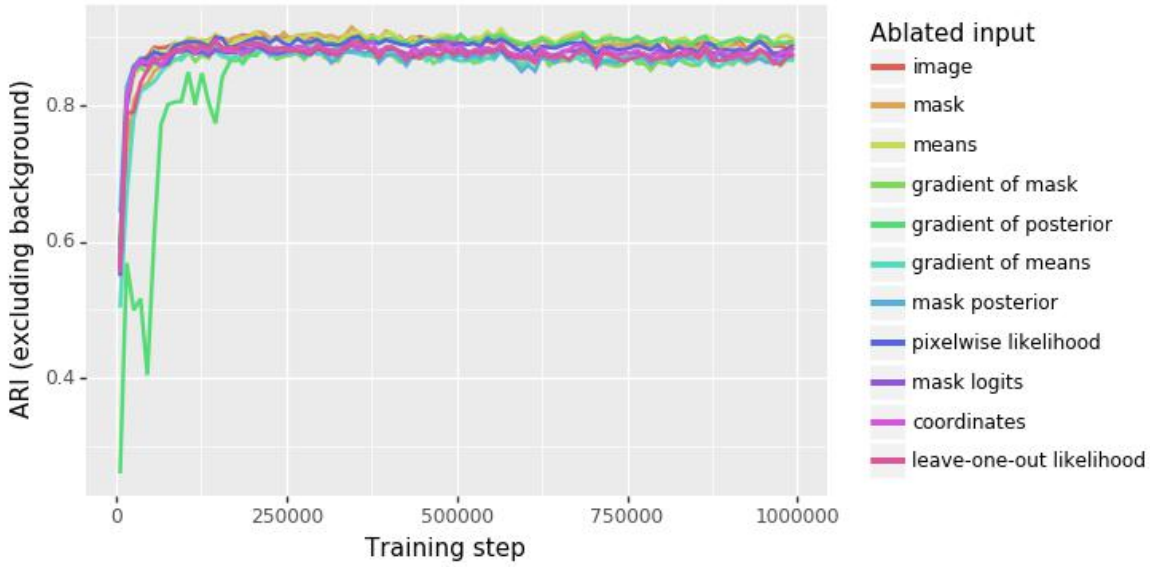
Figure 34. Ablation study for the model's segmentation performance in terms of ARI (excluding the background pixels) on CLEVR6. Each curve denotes the result of training the model without a particular input.

Figure 35. Ablation study for the model's reconstruction loss term on CLEVR6. Each curve denotes the result of training the model without a particular input. The y-axis shows the mean squared error between the target image and the output means (of the final iteration) as a proxy for the full reconstruction loss.
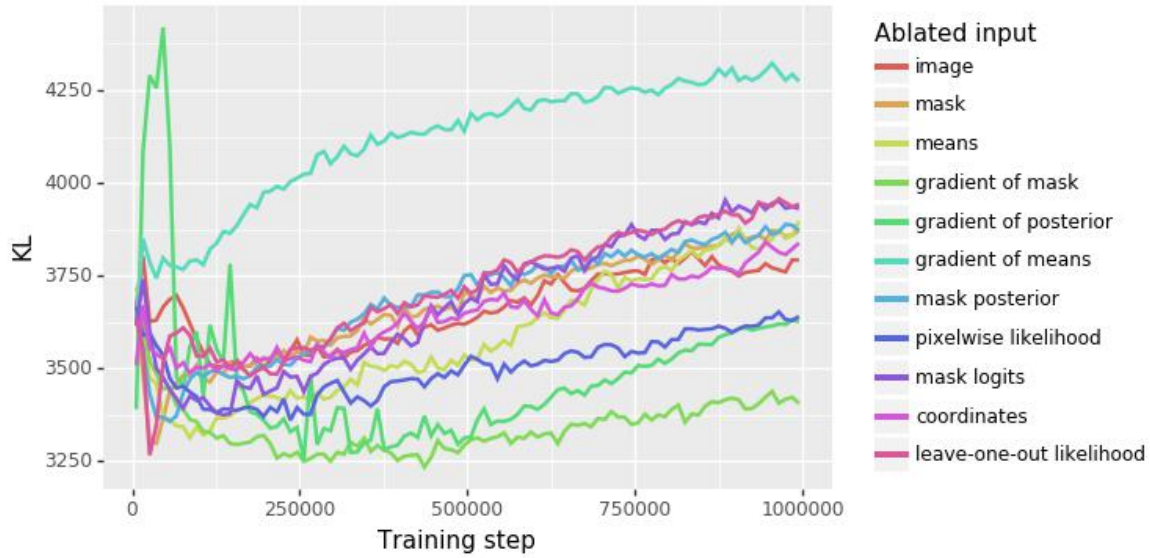
Figure 36. Ablation study for the model's KL divergence loss term on CLEVR6, summed over slots and iterations. Each curve denotes the result of training the model without a particular input.
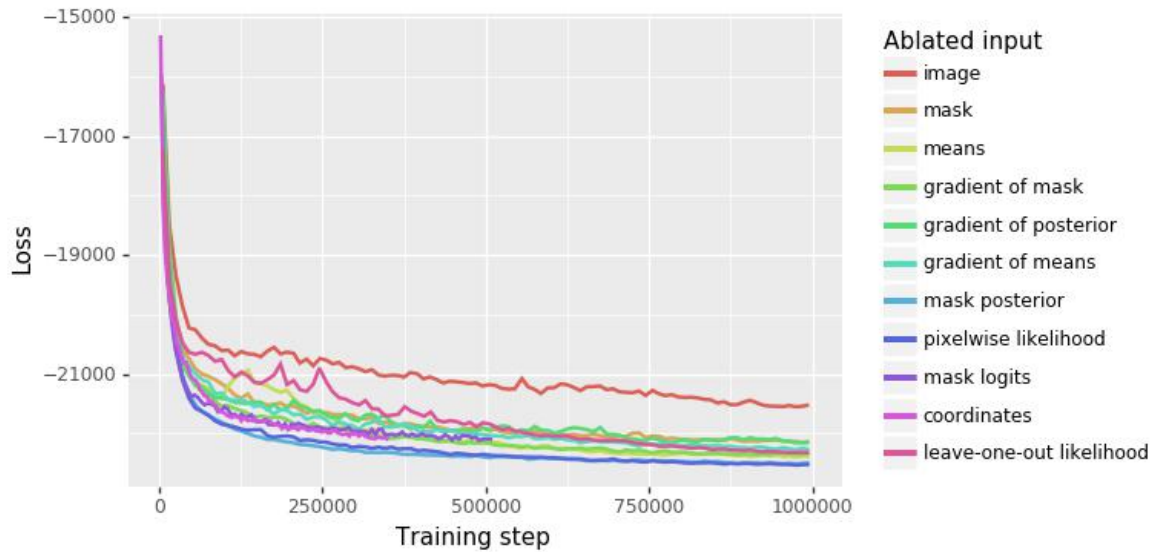

Figure 37. Ablation study for the model's total loss on Tetris. Each curve denotes the result of training the model without a particular input.
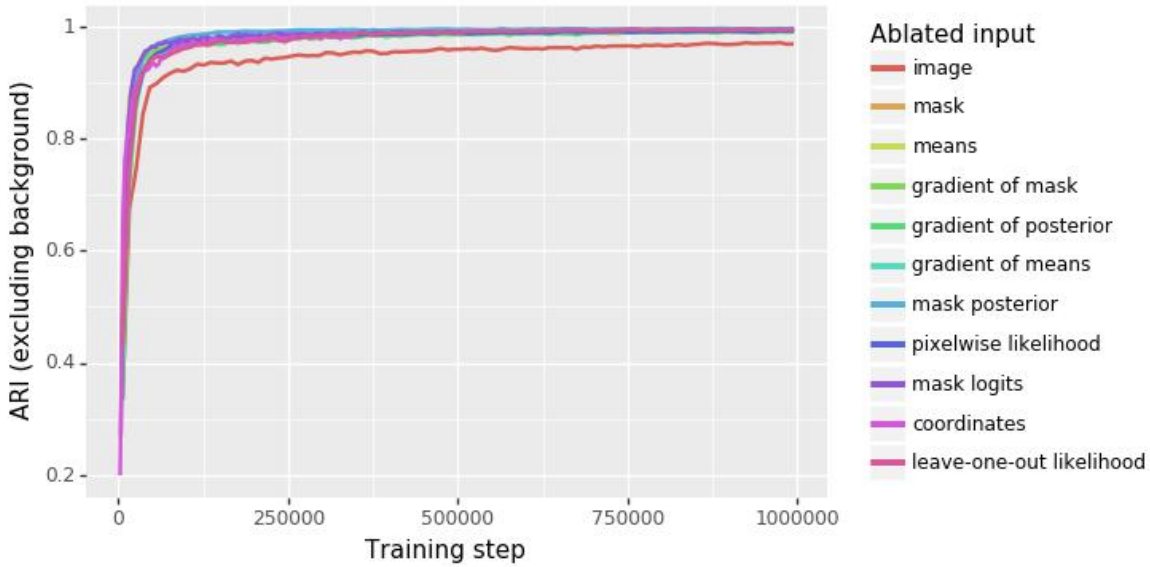
Figure 38. Ablation study for the model's segmentation performance in terms of ARI (excluding the background pixels) on Tetris. Each curve denotes the result of training the model without a particular input.
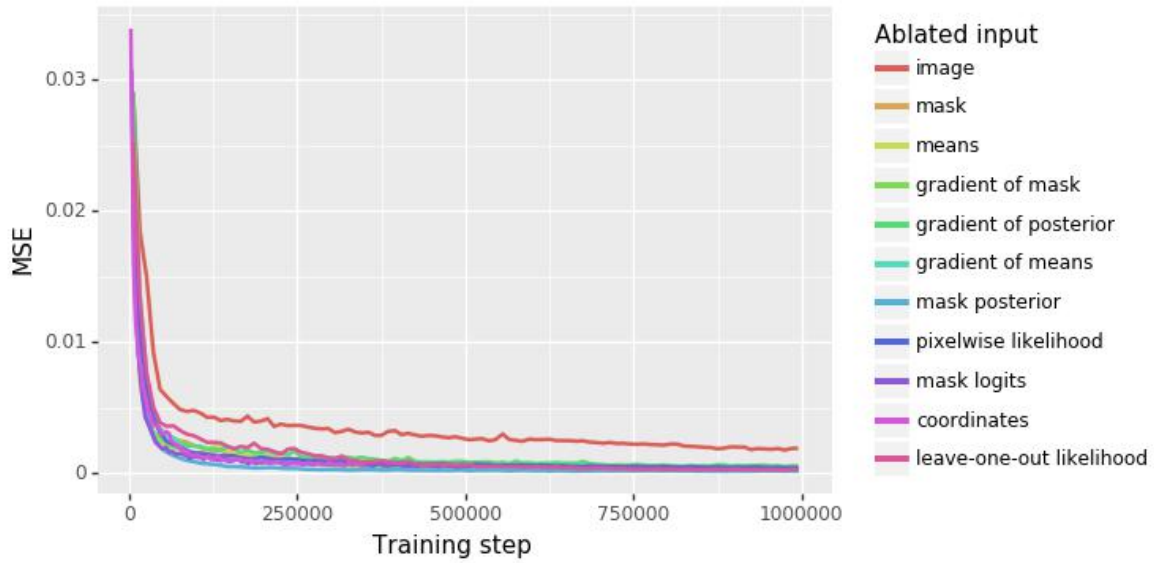

Figure 39. Ablation study for the model's reconstruction loss term on Tetris. Each curve denotes the result of training the model without a particular input. The y-axis shows the mean squared error between the target image and the output means (of the final iteration) as a proxy for the full reconstruction loss.
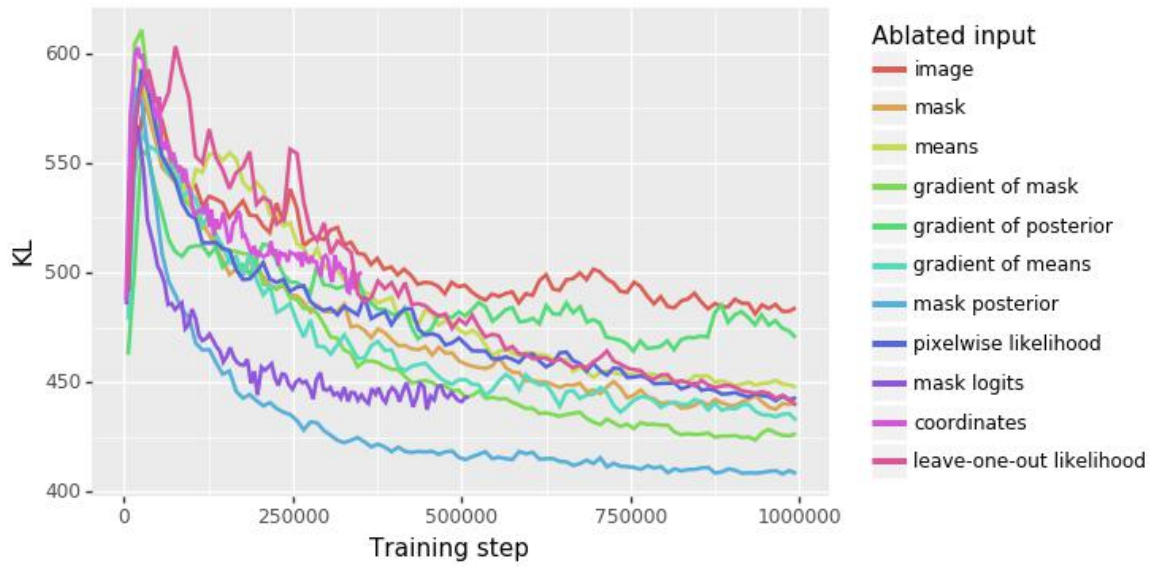
Figure 40. Ablation study for the model's KL divergence loss term on Tetris, summed over slots and iterations. Each curve denotes the result of training the model without a particular input.