Spatially Invariant, Label-free Object Tracking

Eric Crawford
Mila/McGill University
Montreal, QC, Canada
eric.crawford@mail.mcgill.ca

Joelle Pineau
Mila/McGill University
Montreal, QC, Canada
jpineau@cs.mcgill.ca

Abstract

The ability to detect and track objects in the visual world is a crucial skill for any intelligent agent, as it is a necessary precursor to any object-level reasoning process. Moreover, it is important that agents learn to track objects without supervision (i.e. without access to annotated training videos) since this will allow agents to begin operating in new environments with minimal human assistance. The task of learning to discover and track objects in videos, which we call *unsupervised object tracking*, has grown in prominence in recent years; however, most architectures that address it still struggle to deal with large scenes containing many objects. In the current work, we thus propose an architecture that scales well to the large-scene, many-object setting by employing spatially invariant computations (convolutions and spatial attention) and representations (a spatially local object specification scheme). In a series of experiments, we demonstrate a number of attractive features of our architecture; most notably, that it outperforms competing methods at tracking objects in cluttered scenes with many objects, and that it can generalize well to videos that are larger and/or contain more objects than videos seen in training.

In the current work, we address the problem of learning to track objects in cluttered scenes without supervision; at training time, the algorithm has access to videos but no object annotations. Our high-level approach is modeled after Sequential Attend Infer Repeat (SQAIR) [7]. SQAIR formulates a Variational Autoencoder (VAE) [6] for videos, endowed with a highly structured, object-like latent representation. This VAE is composed of a number of modules that are applied each timestep of the input video; notably a discovery module which detects new objects in the current frame, and a propagation module which updates the attributes of objects discovered in previous frames based on information from the current frame. For each input frame, a generative rendering module creates a corresponding output frame from the objects proposed by the discovery and propagation modules. The network is trained by maximizing the VAE evidence lower bound, which encourages the output frames to be accurate reconstructions of the input frames. At the end of training, it is expected that the discovery module will have become a competent object detector, while the propagation module will have learned object dynamics.

However, as we demonstrate empirically, SQAIR struggles at processing spatially large videos that contain many densely packed objects. We hypothesize that this is because SQAIR does not fully exploit the spatial statistics of objects in images. We propose **S**patially **I**nvariant **L**abel-free **O**bject **T**racking (SILOT) (pronounced like "silo"), a differentiable architecture for unsupervised object tracking that is able to scale well to large scenes containing many objects. SILOT achieves this scalability by making extensive use of spatially invariant computations and representations, thereby fully exploiting the structure of objects in images. Through a number of experiments, we demonstrate the concrete advantages that arise from this focus on spatial invariance. In particular, we show that SILOT has a greatly improved capacity for handling large, many-object videos, and that trained SILOT networks can generalize well to videos that are larger and/or contain different numbers of objects than videos encountered in training.

Workshop on Perception as Generative Reasoning, NeurIPS 2019, Vancouver, Canada.

Overview. SILOT is a Variational Auotencoder which models a video as a collection of moving objects. The model is divided into modules. The **discovery** module detects objects from each frame, the **propagation** module updates the attributes of previously discovered objects, the **selection** module selects a small set of objects to keep from the union of the discovered and propagated objects, and the generative **rendering** module renders selected objects into an output frame. Discovery, propagation and selection constitute the VAE inference network $h_\phi(z|x)$, while the rendering makes up the VAE generative model $g_\theta(x|z)$.

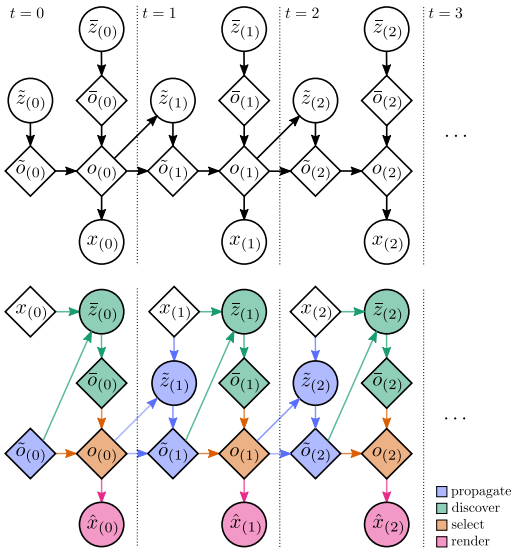

Figure 1: Top: Generative model assumed by SILOT. Diamonds/circles are deterministic/stochastic functions of their inputs. Bottom: Structure of the SILOT neural network. Modules are indicated by color.

Assume we are given an input video x of length T, with $x_{(t)}$ denoting an individual frame for $t \in \{0, \dots, T-1\}$ (we always surround temporal indices with brackets). For each timestep, we consider a number of variable sets: discovered latents $\bar{z}_{(t)}$, discovered objects $\bar{o}_{(t)}$, propagated latents $\tilde{z}_{(t)}$, propagated objects $\tilde{o}_{(t)}$, and selected objects $o_{(t)}$. The total set of latent variables for the VAE is: $z = \bigcup_{t=0}^{T-1} \bar{z}_{(t)} \cup \tilde{z}_{(t)}$. For both discovery and propagation, the object sets are deterministic functions of the corresponding latent variable sets (and possibly other variables). The generative model assumed by SILOT and high-level structure of the SILOT neural network (both modeled after SQAIR) are shown in Figure 1. Within a timestep t, the flow of computation runs as follows:

1. $x_{(t)}$ and objects from the previous frame $o_{(t-1)}$ are passed into the propagation module, which predicts and applies a set of updates $\tilde{z}_{(t)}$, yielding propagated objects $\tilde{o}_{(t)}$.
2. $x_{(t)}$ and $\tilde{o}_{(t)}$ are passed into the discovery module, which discovers objects in the frame that are not yet accounted for, first yielding $\bar{z}_{(t)}$ and then $\bar{o}_{(t)}$ via a deterministic transformation.
3. The selection module selects a subset of objects to retain from $\tilde{o}_{(t)} \cup \bar{o}_{(t)}$, yielding $o_{(t)}$.
4. $o_{(t)}$ is passed into the rendering module which yields an output frame $\hat{x}_{(t)}$.

Object Representation. An object is represented by a set of variables, each called an *attribute*:

$$o^{\text{where}} \in \mathbb{R}^4 \qquad o^{\text{what}} \in \mathbb{R}^A \qquad o^{\text{depth}} \in [0,1] \qquad o^{\text{pres}} \in [0,1]$$

o^{where} specifies the object's size and location. o^{what} acts as a catch-all, storing information about the object that is not captured by other attributes (e.g. appearance, velocity). o^{depth} specifies the relative depth of the object; in the output image, objects with higher values for this attribute appear on top of objects with lower values. o^{pres} specifies the extent to which the object exists; objects with $o^{\text{pres}} = 0$ do not appear in the output image.

Discovery. Object discovery in SILOT is implemented as a convolutional neural network in order to achieve spatial invariance, and is heavily inspired by SPAIR, an unsupervised convolutional object detector [2], as well as single-shot supervised object detectors such as YOLO [12] and SSD [9]. An initial convolutional network d_ϕ^{bu} extracts "bottom-up" information from the current input frame $x_{(t)}$, mapping to a feature volume: $v_{(t)}^{\text{bu}} = d_\phi^{\text{bu}}(x_{(t)})$.

The structure of network d_ϕ^{bu} can be taken to induce a spatial grid over the frame, as follows. Let c_h/c_w be the translation (in pixels) vertically/horizontally between receptive fields of adjacent spatial locations in $v_{(t)}^{\text{bu}}$. Then for an input frame with dimensions $(H_{\text{inp}}, W_{\text{inp}}, 3)$, we divide the frame up into an (H, W) grid of cells, each cell being c_h pixels high by c_w pixels wide, where $H = \lceil H_{\text{inp}}/c_h \rceil$, $W = \lceil W_{\text{inp}}/c_w \rceil$. The output volume $v_{(t)}^{\text{bu}}$ has spatial shape (H, W), and we associate each of its spatial locations with the corresponding grid cell. Importantly, the input frame is padded on all sides to ensure that the receptive field for each spatial location in v_t^{bu} is centered on its grid cell.

The discovery module will ultimately yield a separate object for each grid cell. It is useful to think of the discovery module as consisting of a 2D array of identical object detectors, each operating on a different local region of the image. We call each of these local detectors a *discovery unit*; the structure of a unit is shown in Figure 2. Variables for all discovery units are grouped together into convolutional volumes with spatial dimensions (H, W), and computations are implemented by size-preserving convolutions on these volumes, essentially computing all units in parallel.

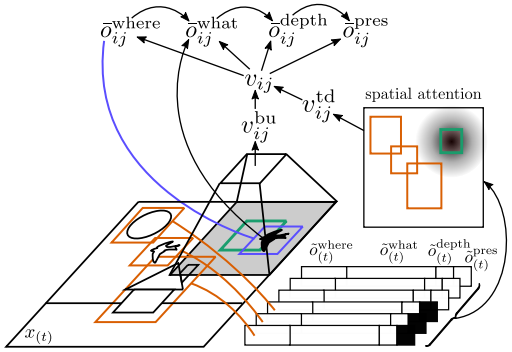

Figure 2: Schematic depicting the structure of a discovery unit with indices ij at time t, discovering the black bird which has just come into view from the right. Local bottom-up information from the current frame is processed by a convolutional filter (trapezoid), which has a receptive field (grey base of the trapezoid) centered on the discovery unit's grid cell (green rectangle). Next, top-down information about nearby objects propagated from the previous frame (orange boxes) is summarized using spatial attention with a Gaussian kernel centered at the grid cell. Bottom-up and top-down information is then fused and used to autoregressively predict object attributes (here we have omitted the latent discovery variables $\bar{z}_{ij}$). $\bar{o}_{ij}^{\text{where}}$ is specified with respect to the grid cell, and the prediction for $\bar{o}_{ij}^{\text{what}}$ conditions on the output of a glimpse parameterized by $\bar{o}_{ij}^{\text{where}}$ (blue rectangle).

In order to avoid rediscovering objects that are already accounted for, discovery needs to be aware of "top-down" information about objects propagated from the previous frame. Note that each discovery unit need only worry about propagated objects that are near its grid cell. We thus employ a spatial attention step which, for each grid cell, weights the propagated objects according to a Gaussian kernel centered at the cell (similar to [1]). The result is a feature volume $v_{(t)}^{\text{td}}$ with spatial shape (H, W) where each spatial location contains information about nearby objects: $v_{(t)}^{\text{td}} = \text{SpatialAttention}_{\phi}^{\text{disc}}(\tilde{o}_{(t)}, \sigma)$ where σ is the standard deviation of the Gaussian kernel.

Finally, the network predicts parameters for distributions over the latent discovery variables $\bar{z}_{(t)}$, samples from these distributions, and maps the sampled latents to the more interpretable $\bar{o}_{(t)}$. This is done on an attribute-by-attribute basis (in order [where, what, depth, pres]), and is autoregressive (predictions for later attributes condition on samples for earlier ones). A *glimpse* is extracted from $x_{(t)}$ at location $\bar{o}_{(t)}^{\text{where}}$ in order to include location-specific information.

Propagation. The propagation module at time t takes in the current frame $x_{(t)}$ and the objects from the previous timestep $o_{(t-1)}$, and propagates the objects forward in time, using information from the current frame to update the object attributes. This yields propagated objects $\tilde{o}_{(t)}$. We begin by computing a feature vector for each object in $o_{(t-1)}$. In order to handle interactions between objects, we can have the feature vector for an object depend on other objects as well. Here we make the assumption that object interactions are spatially local (which is enough to handle collisions, for example). Thus we compute features using a spatial attention step similar to the one used in the Discovery module, allowing the features for an object to depend on attributes of nearby objects: $u_{(t)}^{\text{td}} = \text{SpatialAttention}_{\phi}^{\text{prop}}(o_{(t-1)}, \sigma)$. From here we autoregressively predict new values for the object attributes; this is again similar to attribute prediction in the discovery module, except that rather than directly predicting attribute values, we predict attribute *updates* and subsequently apply them. For brevity, we leave the full details of this process to the supplementary material.

Propagation in SILOT is similar to propagation in SQAIR. However, there is one significant difference. In SQAIR, objects within a timestep are updated sequentially; this allows objects within a timestep to condition on one another, facilitating coordination between objects and supporting behavior such as explaining away. However, this sequential processing can be computationally demanding when there are large numbers of objects. In contrast, SILOT updates all objects within a timestep in parallel; a degree of coordination between (nearby) objects is achieved via the spatial attention step.

Selection. After running propagation and discovery for a given timestep, we are left with $K + HW$ objects, where K is the number of objects in the previous frame. The role of the selection module is to reduce this back to K objects; otherwise the number of propagated objects would grow with the number of timesteps, and training would become computationally intractable. We employ a simple top-K selection strategy, picking the K objects with largest values for the *pres* attribute. While this

hard selection is not differentiable, we found it not to cause problems as long as K is large enough (we chose K to be roughly 25% larger than the maximum number of objects in a single frame).

Rendering. The rendering module is the sole constituent of the VAE generator, taking in the current set of objects $o_{(t)}$ and yielding a frame $\hat{x}_{(t)}$. The rendering process is highly structured, and this structure gives meaning to the object attributes; for example, the object is placed at location o^{where} via spatial transformers, and o^{depth} parameterizes a differentiable approximation of relative depth. Details are left to the supplementary material.

Training. The network is trained by maximizing the evidence lower bound [6]:

$$E_{x\sim h(x), z\sim h_\phi(z|x)}\left[\log\left(\frac{g_\theta(x|z)g(z)}{h_\phi(z|x)}\right)\right]$$

with respect to ϕ and θ. Here $h(x)$ is the distribution over videos defined by the dataset, and $g(z)$ is the prior distribution over z. The optimization is performed by gradient ascent. For the majority of latents, we assume independent Normal prior distributions. However, for the *pres* variables (which can be thought of as BinConcretes [10]) we employ a prior which encourages the network to use few objects in reconstructing a scene (similar to that used in SPAIR [2]), improving the quality of the discovered objects. To increase stability in training, we first train on only the first 2 frames of each video, and gradually grow to include the full videos (a form of curriculum learning). We also employ a technique that we call *discovery dropout* wherein we turn off the entire discovery module with probability 0.5 for $t \neq 0$, which encourages the network to favor using propagation over discovery.

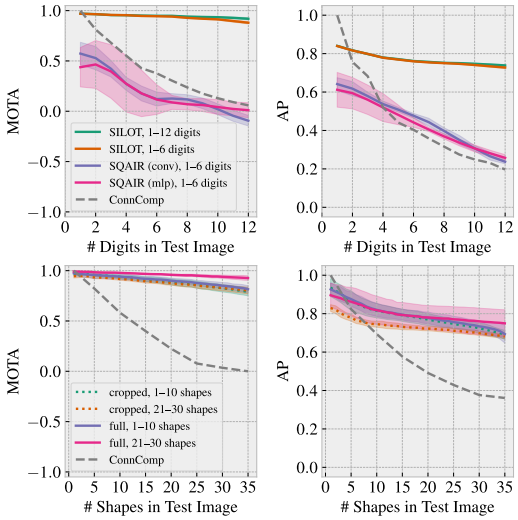

Figure 3: Probing object tracking performance as number of objects per video varies in the Scattered MNIST (top) and Scattered Shapes (bottom) tasks. All points are averages over 6 random seeds for MNIST, 4 for Shapes. Filled regions are standard deviations.

Experiments. We tested SILOT on several challenging object discovery and tracking tasks, emphasizing videos containing large numbers of objects. We used 2 metrics to assess model performance: MOTA and AP, standard measures of object tracking and detection, respectively[11, 3] (higher is better for both). We compare against a baseline algorithm ConnComp, whose performance can be interpreted as a measure of the difficulty of the dataset; it will be successful only to the extent that objects can be tracked by color alone. Results are shown in Figure 3, and videos visualizing SILOT's performance can be found online at `https://sites.google.com/view/silot`. Code is available at `https://github.com/e2crawfo/silot`. A similar model was concurrently developed in [5].

The first task consists of videos with a random number of moving MNIST digits. We used two training conditions: train on videos containing 1–6 digits vs 1–12 digits. All networks were tested on videos containing up to 12 digits; thus, networks trained in the 1–6 condition were required to generalize beyond their training experience. Here we also evaluated SQAIR trained with two different backbone networks. Results for SQAIR trained on 1–12 digits are not shown, as SQAIR's simpler discovery module was unable to process the dense scenes; the resulting highly varied performance would make the plots unreadable. Notice that SILOT trained on 1–6 digits takes only a minor performance hit compared to SILOT trained on 1–12 digits.

	MOTA	AP
Space Invaders	.89	.73
Asteroids	.67	.67

Table 1: SILOT performance on Atari.

The second task consists of videos containing randomly moving colored shapes. Here we perform a similar manipulation as for MNIST: train on 1–10 shapes vs 21–30 shapes, test on up to 35 shapes. We also performed an additional manipulation: training on random 60×60 crops of the full video, vs training on full 96×96 videos. At test time full videos are always used. This manipulation tests SILOT's ability to generalize to larger videos than encountered in training. Notice that SILOT trained on random 60×60 crops of videos containing only 1–10 shapes achieves reasonable performance when applied to full videos containing up to 35 shapes.

Finally, in order to push the scalability of SILOT, we tested it on videos from SpaceInvaders and Asteroids Atari games (with spatial dimensions at least 190×160), results shown in Table 1.

References

[1] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[2] Eric Crawford and Joelle Pineau. Spatially invariant, unsupervised object detection with convolutional neural networks. In *Thirty-Third AAAI Conference on Artificial Intelligence*, 2019.

[3] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[4] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. In *Advances in neural information processing systems*, pages 2017–2025, 2015.

[5] Jindong Jiang, Sepehr Janghorbani, Gerard de Melo, and Sungjin Ahn. Scalable object-oriented sequential generative models. *arXiv preprint arXiv:1910.02384*, 2019.

[6] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[7] Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In *Advances in Neural Information Processing Systems*, pages 8606–8616, 2018.

[8] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[9] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[10] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[11] Anton Milan, Laura Leal-Taixé, Ian Reid, Stefan Roth, and Konrad Schindler. Mot16: A benchmark for multi-object tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[12] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. *arXiv preprint*, 2017.

[13] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

A Module Details

Discovery

Details on the grid structure of the Discovery module were presented in the main text. Here we provide on additional figure, Figure 4, giving some visual intution about grid cells and padding in the discovery module.

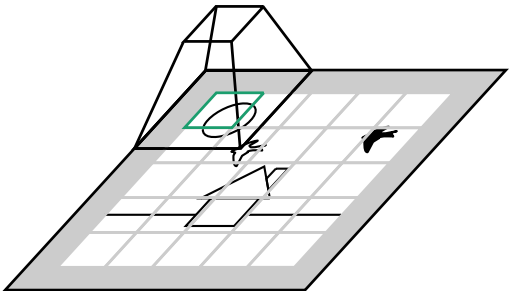

Figure 4: Schematic depicting grid cells and padding in the discovery module. The gray grid is the grid of cells described in the main text. The top of the trapezoid is a spatial location in the output layer of d_ϕ^{bu}. This output location is associated with the grid cell highlighted in green. The bottom of the trapezoid is the receptive field of that spatial location; the frame is padded (solid gray border area) before being passed into d_ϕ^{bu} to ensure that all receptive fields are centered on their corresponding grid cells.

Here we also provide details on the workings of the discovery module after $v_{(t)}^{\text{bu}}$ and $v_{(t)}^{\text{td}}$ have been predicted, which were largely glossed over in the main text. A convolutional network fuses bottom-up and top-down information (here we begin omitting temporal indices):

$$v = d_\phi^{\text{fuse}}(v^{\text{bu}}, v^{\text{td}})$$

We follow the convention that a convolutional network that takes multiple inputs first depth-wise concatenates those inputs into a single volume, which is then fed into the network. The network then predicts parameters for distributions over the latent discovery variables $\bar{z}_{(t)}$, samples from the predicted distributions, and then maps the sampled latent to the more interpretable $\bar{o}_{(t)}$. This is done on an attribute-by-attribute basis (in order [where, what, depth, pres]), and is autoregressive, so that predictions for later attributes are conditioned on samples for earlier attributes.

Predicting $\bar{o}^{\text{where}}$. We first use a network d_ϕ^{where} to predict parameters for a distribution over $\bar{z}^{\text{where}}$, and then sample:

$$\bar{\mu}^{\text{where}}, \bar{\sigma}^{\text{where}} = d_\phi^{\text{where}}(v)$$
$$\bar{z}^{\text{where}} \sim N(\bar{\mu}^{\text{where}}, \bar{\sigma}^{\text{where}})$$

Next we deterministically map to $\bar{o}^{\text{where}}$. We decompose $\bar{z}^{\text{where}}$ as $\bar{z}^{\text{where}} = (\bar{z}^y, \bar{z}^x, \bar{z}^h, \bar{z}^w)$. Here it will be useful to narrow our focus to a single discovery unit with indices ij for $i \in \{0, \ldots, H-1\}$, $j \in \{0, \ldots, W-1\}$. $\bar{z}_{ij}^y$ and $\bar{z}_{ij}^x$ parameterize the position of the object according to:

$$b_{ij}^y = b^{\text{min}} + \text{sigmoid}(\bar{z}_{ij}^y)\left(b^{\text{max}} - b^{\text{min}}\right)$$
$$\bar{o}_{ij}^y = (i + b_{ij}^y)c_h$$
$$b_{ij}^x = b^{\text{min}} + \text{sigmoid}(\bar{z}_{ij}^x)\left(b^{\text{max}} - b^{\text{min}}\right)$$
$$\bar{o}_{ij}^x = (j + b_{ij}^x)c_w$$

where b^{min} and b^{max} are fixed real numbers which, in effect, impose bounds on the distance between the object and the grid cell. $\bar{z}_{ij}^h$ and $\bar{z}_{ij}^w$ parameterize the size of the object as:

$$\bar{o}_{ij}^h = \text{sigmoid}(\bar{z}_{ij}^h)a_h \qquad\qquad \bar{o}_{ij}^w = \text{sigmoid}(\bar{z}_{ij}^w)a_w$$

for fixed real numbers a_h and a_w. (a_h, a_w) can be interpreted as the dimensions of an *anchor box* as used in supervised object detection [13]. Specifying object size with respect to a_h and a_w, as opposed to the the size of the input frame, ensures that $\bar{o}_{ij}^h$ and $\bar{o}_{ij}^w$ are meaningful regardless of the spatial dimensions of the input frame.

Predicting $\bar{o}^{\text{what}}$, $\bar{o}^{\text{depth}}$ and $\bar{o}^{\text{pres}}$. In order to obtain highly location-specific information from the image, an array of glimpses g (one per discovery unit) is extracted from the image using spatial transformers [4] parameterized by $\bar{o}_{(t)}^{\text{where}}$. These are then mapped to a feature volume v^{obj} by a network d_ϕ^{obj}:

$$g = \tau(x, \bar{o}^{\text{where}})$$
$$v^{\text{obj}} = d_\phi^{\text{obj}}(g)$$

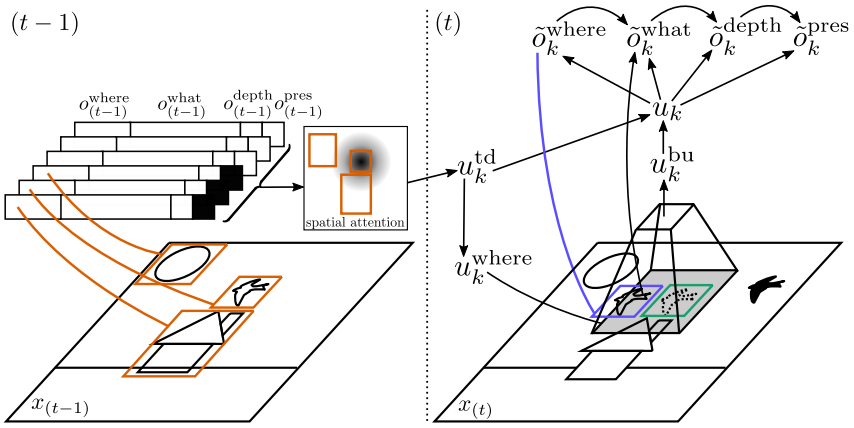

Figure 5: Schematic depicting the propagation module updating an object with index k, tracking the location of the white bird. A feature vector for the object, which also takes into account nearby objects, is first created using spatial attention. Next, an initial glimpse (grey region) is specified with respect to the object's location from the previous time step (green box). This glimpse is then processed by a neural network (trapezoid) and used to predict and apply an update to the *where* attribute, resulting in $\tilde{o}^{\text{where}}_k$. The blue box corresponds to the location of the image referred to by $\tilde{o}^{\text{where}}_k$. Another glimpse is extracted at location $\tilde{o}^{\text{where}}_k$, and updates to the remaining attributes are predicted and applied autoregressively. Here we have omitted the latent propagation variables $\tilde{z}_{(t)}$.

We then autoregressively predict the remaining attributes:

$$
\begin{aligned}
\bar{\mu}^{\text{what}}, \bar{\sigma}^{\text{what}} &= d^{\text{what}}_\phi(v, v^{\text{obj}}, \bar{o}^{\text{where}}) \\
\bar{z}^{\text{what}} &\sim N(\bar{\mu}^{\text{what}}, \bar{\sigma}^{\text{what}}) \\
\bar{o}^{\text{what}} &= \bar{z}^{\text{what}} \\
\bar{\mu}^{\text{depth}}, \bar{\sigma}^{\text{depth}} &= d^{\text{depth}}_\phi(v, v^{\text{obj}}, \bar{o}^{\text{where}}, \bar{o}^{\text{what}}) \\
\bar{z}^{\text{depth}} &\sim N(\bar{\mu}^{\text{depth}}, \bar{\sigma}^{\text{depth}}) \\
\bar{o}^{\text{depth}} &= \text{sigmoid}(\bar{z}^{\text{depth}}) \\
\bar{\mu}^{\text{pres}} &= d^{\text{pres}}_\phi(v, v^{\text{obj}}, \bar{o}^{\text{where}}, \bar{o}^{\text{what}}, \bar{o}^{\text{depth}}) \\
\bar{z}^{\text{pres}} &\sim \text{Logistic}(\bar{\mu}^{\text{pres}}) \\
\bar{o}^{\text{pres}} &= \text{sigmoid}(\bar{z}^{\text{pres}})
\end{aligned}
$$

Propagation

Propagation at time t takes in the current frame $x_{(t)}$ and the K objects from the previous timestep $o_{(t-1)}$, and propagates the objects forward in time, using information from the current frame to update the object attributes. For propagation we assume an additional object attribute $o^{\text{hidden}}_{(t)}$, which stores the hidden state of a recurrent neural network p^{rnn}_ϕ and is fully deterministic. After each propagation step, the hidden state for each object is updated independently by running the recurrent network network, taking the new object attributes as input. We can think of this as providing the propagation module with a deterministic path from an object's past to the present. Newly discovered objects are given a default initial hidden state. The value of the default hidden state is a trainable parameter.

In this section, all variables are matrices with K as their leading dimension, where K is the number of propagated/selected objects. All networks are applied to each row independently (i.e. "object-wise") and in parallel. This is similar to the convention used in the discovery module, but with one less leading dimension. The structure of propagation for a single object is shown in Figure 5.

As discussed in the main text, the first step is to compute features about the previous objects $o_{(t-1)}$ using a spatial attention step:

$$
u^{\text{td}}_{(t)} = \text{SpatialAttention}^{\text{prop}}_\phi(o_{(t-1)}, \sigma)
$$

Next we need to condition attribute updates on the current frame. Rather than condition on the entire frame, we instead extract a glimpse in a region around the object's location from the previous timestep:

$$\begin{aligned} u_{(t)}^{\text{where}} &= o_{(t-1)}^{\text{where}} + \tanh(p_\phi^{\text{glimpse}}(u_{(t)})) \\ g_{(t)}^{\text{prop}} &= \tau(x_{(t)}, u_{(t)}^{\text{where}}) \\ u_{(t)}^{\text{bu}} &= p_\phi^{\text{bu}}(g_{(t)}^{\text{prop}}) \end{aligned}$$

We predict and apply an update to the *where* attribute:

$$\begin{aligned} \tilde{\mu}_{(t)}^{\text{where}}, \tilde{\sigma}_{(t)}^{\text{where}} &= p_\phi^{\text{where}}(u_{(t)}, u_{(t)}^{\text{obj}}) \\ \tilde{z}_{(t)}^{\text{where}} &\sim N(\tilde{\mu}_{(t)}^{\text{where}}, \tilde{\mu}_{(t)}^{\text{where}}) \\ \tilde{o}_{(t)}^{\text{where}} &= o_{(t-1)}^{\text{where}} + \tanh(\tilde{z}_{(t)}^{\text{where}}) \end{aligned}$$

Next we extract and process another glimpse at $\tilde{o}_{(t)}^{\text{where}}$:

$$\begin{aligned} g_{(t)}^{\text{prop}} &= \tau(x_{(t)}, \tilde{o}_{(t)}^{\text{where}}) \\ u_{(t)}^{\text{obj}} &= p_\phi^{\text{obj}}(g_{(t)}^{\text{prop}}) \end{aligned}$$

We then autoregressively predict changes to the remaining attributes and apply them:

$$\begin{aligned} \tilde{\mu}^{\text{what}}, \tilde{\sigma}^{\text{what}} &= p_\phi^{\text{what}}(u, u^{\text{obj}}, \tilde{o}^{\text{where}}) \\ \tilde{z}^{\text{what}} &\sim N(\tilde{\mu}^{\text{what}}, \tilde{\sigma}^{\text{what}}) \\ \tilde{o}^{\text{what}} &= o_{(t-1)}^{\text{what}} + \tanh(\tilde{z}^{\text{what}}) \\ \tilde{\mu}^{\text{depth}}, \tilde{\sigma}^{\text{depth}} &= p_\phi^{\text{depth}}(u, u^{\text{obj}}, \tilde{o}^{\text{where}}, \tilde{o}^{\text{what}}) \\ \tilde{z}^{\text{depth}} &\sim N(\tilde{\mu}^{\text{depth}}, \tilde{\sigma}^{\text{depth}}) \\ \tilde{o}^{\text{depth}} &= o_{(t-1)}^{\text{depth}} + \tanh(\tilde{z}^{\text{depth}}) \\ \tilde{\mu}^{\text{pres}} &= p_\phi^{\text{pres}}(u, u^{\text{obj}}, \tilde{o}^{\text{where}}, \tilde{o}^{\text{what}}, \tilde{o}^{\text{depth}}) \\ \tilde{z}^{\text{pres}} &\sim \text{Logistic}(\tilde{\mu}^{\text{pres}}) \\ \tilde{o}^{\text{pres}} &= o_{(t-1)}^{\text{pres}} \cdot \text{sigmoid}(\tilde{z}^{\text{pres}}) \end{aligned}$$

Notice that propagation cannot increase the value of the *pres* attribute, due to the form of the update (multiplication by a sigmoid). This ensures that objects are only ever discovered by the Discovery module, which is better equipped for it.

Finally, the hidden state is updated by the RNN:

$$\tilde{o}^{\text{hidden}} = p_\phi^{\text{rnn}}(o_{(t-1)}^{\text{hidden}}, \tilde{o}^{\text{where}}, \tilde{o}^{\text{what}}, \tilde{o}^{\text{depth}}, \tilde{o}^{\text{pres}})$$

Rendering

The rendering module is the sole constituent of the VAE decoder. It takes in the current set of objects $o_{(t)}$ and renders them into a frame. We start by focusing on a single object with index k. First, an appearance map and a partial transparancy map are are predicted:

$$\beta_k, \xi_k = \text{sigmoid}(r_\theta^{\text{obj}}(o_k^{\text{what}}))$$

These maps have dimensions $(H_{\text{obj}}, W_{\text{obj}}, 3)$ and $(H_{\text{obj}}, W_{\text{obj}}, 1)$, respectively, for integers H_{obj}, W_{obj}.

ξ_k is multiplied by o_k^{pres} to ensure that objects are only rendered to the image to the extent that they are present, yielding a final transparency map:

$$\alpha_k = \xi_k \cdot o_k^{\text{pres}}$$

Next we combine α_k with o_k^{depth} to get an *importance* map:

$$\gamma_k = \alpha_k \cdot o_k^{\text{depth}}$$

For each object, an inverse spatial transformer parameterized by o_k^{where} is then used to create image-sized versions of these three maps, with the input maps placed in the correct location:

$$\alpha'_k, \beta'_k, \gamma'_k = \tau^{-1}([\alpha_k, \beta_k, \gamma_k], o_k^{\text{where}})$$

For each location in the output of one of these image-sized maps, the value is obtained from a corresponding location in the input space, dictated by the location parameters o_k^{where}. However, some of these locations will lie "outside" of the input map, and for these we use a default value. In particular, we use a default of 0 for α and β, and $-\infty$ for γ. These computations can be performed for all objects in parallel.

To obtain the output frame, the image-sized appearance maps are combined by weighted summation. For each pixel we take the softmax (over objects) of the importance values, and weight each object by the resulting value. We also weight by α' to implement transparency. Thus we have:

$$\hat{x}_{(t)} = \frac{\sum_{k=0}^{K-1} \beta'_k \alpha'_k e^{\gamma'_k}}{\sum_{\ell=0}^{K-1} e^{\gamma'_\ell}}$$

When a given pixel is within the bounding boxes of two or more objects, the softmax implements a differentiable approximation of relative depth, and objects with larger values for o^{depth} (and thus larger values for γ, all else being equal) are rendered on top of objects with lower values. The default of $-\infty$ used when spatially transforming γ ensures that objects do not contribute to the softmax for pixels that are not within their bounding box.

For large frames this scheme can be expensive in terms of both memory and computation. Thus in practice we use an equivalent (but more complex) implementation that avoids explicitly constructing image-sized maps for each object.

B Spatial Attention

In this section we provide details on the spatial attention steps used in the discovery and propagation modules. Both are similar to Neural Physics Engine [1], though more so for the Propagation version.

Spatial Attention for Discovery

Recall that in the discovery module, spatial attention is used to obtain, for each discovery unit, a feature vector summarizing nearby propagated objects. The main motivation is to allow the discovery module to avoid rediscovering objects that are already accounted for:

$$v_{(t)}^{\text{td}} = \text{SpatialAttention}_\phi^{\text{disc}}(\tilde{o}_{(t)}, \sigma)$$

We first narrow our focus to a single discovery unit with indices ij. For every propagated object $\tilde{o}_k$ (dropping temporal indices here), we first use an MLP d_ϕ^{spatial} to compute a feature vector that is specific to ij. As input to this MLP, we supply the object $\tilde{o}_k$, except that we replace the position attributes $\tilde{o}_k^y$, $\tilde{o}_k^x$ with *relative* position attributes $\tilde{o}_{ij,k}^{y'}$ and $\tilde{o}_{ij,k}^{x'}$. Here we are assuming that what is important is the position of the propagated objects relative to the grid cell, rather than their absolute positions. Noting that the center of grid cell ij has location $((i+0.5)\cdot c_h, (j+0.5)\cdot c_w)$ we have

$$\tilde{o}_{ij,k}^{y'} = \tilde{o}_k^y - (i+0.5)\cdot c_h \qquad\qquad \tilde{o}_{ij,k}^{x'} = \tilde{o}_k^x - (j+0.5)\cdot c_w$$

The spatial attention module then just sums these feature vectors over propagated objects k, weighted by a Gaussian kernel:

$$v_{ij}^{\text{td}} = \sum_{k=0}^{K-1} G(\tilde{o}_{ij,k}^{y'}, \tilde{o}_{ij,k}^{x'}, \sigma) \cdot d_\phi^{\text{spatial}}(\tilde{o}_k^{\setminus yx}, \tilde{o}_{ij,k}^{y'}, \tilde{o}_{ij,k}^{x'})$$

where G is the density of a 2 dimensional Gaussian, and $\tilde{o}_k^{\setminus yx}$ contains all attributes of $\tilde{o}_k$ except y and x. Note that this can be computed for all ij and k in parallel.

Spatial Attention for Propagation

In the propagation module, spatial attention is used to compute a feature vector for each object from the previous step, which is subsequently used to predict updates to the attributes of the object.

$$u_{(t)}^{\text{td}} = \text{SpatialAttention}_{\phi}^{\text{prop}}(o_{(t-1)}, \sigma)$$

This vector is supposed to take into account attributes of the object itself, as well as attributes of nearby objects. This should allow the updates to take into account the effect of nearby objects on the target object.

We first narrow our focus to a target object with index ℓ. We use an MLP p_{ϕ}^{td} to compute an initial feature vector:

$$u_{\ell}^{\text{td}'} = p_{\phi}^{\text{td}}(o_{\ell})$$

Then for every object k we compute the position of object k relative to object ℓ:

$$o_{\ell,k}^{y'} = o_k^y - o_{\ell}^y \qquad\qquad o_{\ell,k}^{x'} = o_k^x - o_{\ell}^x$$

Next we use an MLP $p_{\phi}^{\text{spatial}}$ to get feature vectors for o_k in the context of target object o_{ℓ}. This is similar to discovery, except the MLP also takes $u_{\ell}^{\text{td}'}$ as an argument. The results are summed and weighted by a Gaussian kernel, and then $u_{\ell}^{\text{td}'}$ is added in:

$$u_{\ell}^{\text{td}} = u_{\ell}^{\text{td}'} + \sum_{k=0}^{K-1} G(o_{\ell,k}^{y'}, o_{\ell,k}^{x'}, \sigma) \cdot p_{\phi}^{\text{spatial}}(o_k^{\setminus yx}, o_{\ell,k}^{y'}, o_{\ell,k}^{x'}, u_{\ell}^{\text{td}'})$$

We can think of the second term as computing the additive effect of nearby objects on the target object. Again this can be computed for all ℓ and k in parallel.

C Baseline Algorithm: ConnComp

We compare against a simple baseline algorithm called ConnComp (Connected Components), which works as follows. For each frame a graph is created wherein the pixels are nodes, and two pixels are connected by an edge if and only if they are adjacent and have the same color. We then extract connected components from this graph, and call each connected component an object. This yields a set of objects for each frame. In order to track objects over time (i.e. to assign persistent identifiers to the objects), we employ the Hungarian algorithm to find matches between detected objects in each pair of successive frames [8]. As matching cost we use the distance between object centroids, and require that matching objects have the same color (objects pairs with mismatched colors are assigned a cost of ∞). The performance of ConnComp can be interpreted as a measure of the difficulty of the dataset; it will be successful only to the extent that objects can be tracked by color alone.

D Experiment Visualizations

Visualization of object tracking in trained SILOT networks are shown for the Scattered MNIST Figure 6 and the Scattered Shapes task in 7.

Figure 6: Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered MNIST task containing 8 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have passed completely through other objects (e.g. 5 with the green box, 3 with the grey box).

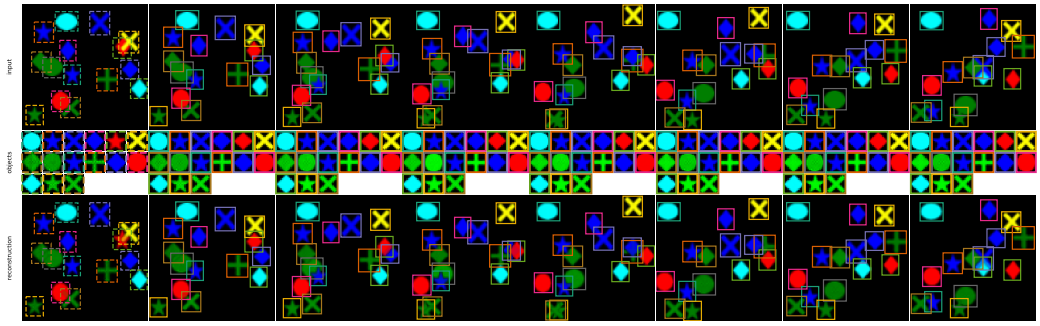

Figure 7: Visualizing a forward pass of a trained SILOT network applied to a video from the Scattered Shapes task containing 15 objects. Top / Bottom: Ground truth / reconstructed frames with bounding boxes for detected objects overlaid. Middle: Predicted appearances for detected objects. Box color represents object identity according to the network. Boxes for objects that SILOT has discovered in a given frame are dashed, while boxes for objects propagated from the previous frame are solid. Notice that the network is able to track objects even after they have been heavily occluded by other objects (e.g. green cross with the orange box that starts near the center).