Discovering Object-Centric Generalized Value Functions From Pixels

Somjit Nath [1 2] **Gopeshh Raaj Subbaraj** [2 3] **Khimya Khetarpal** [† 2 4] **Samira Ebrahimi Kahou** [1 2 5]

Abstract

Deep Reinforcement Learning has shown significant progress in extracting useful representations from high-dimensional inputs albeit using hand-crafted auxiliary tasks and pseudo rewards. Automatically learning such representations in an object-centric manner geared towards control and fast adaptation remains an open research problem. In this paper, we introduce a method that tries to discover meaningful features from objects, translating them to temporally coherent 'question' functions and leveraging the subsequent learned general value functions for control. We compare our approach with state-of-the-art techniques alongside other ablations and show competitive performance in both stationary and non-stationary settings. Finally, we also investigate the discovered general value functions and through qualitative analysis show that the learned representations are not only interpretable but also, centered around objects that are invariant to changes across tasks facilitating fast adaptation.

1. Introduction

Learning control from high-dimensional input such as images is a complex problem relevant to many real world applications. While researchers have made huge strides in Deep Reinforcement Learning (RL), decision making from images remains a challenge due to the difficulty of discovering meaningful features that are invariant across tasks. Levine et al. (2016); Kalashnikov et al. (2018) demonstrate how agents can learn a policy from pixels to be used in real-life applications. One of the standard practices in RL is to learn a control policy from pixels in an end-to-end fashion. The end-to-end learning paradigm presents certain downsides due to the black-box nature of Deep Neural Networks (DNN). A significant challenge faced during learning from pixels is the inability to learn a good control policy from high dimensional visual inputs and limited information in a single scalar reward signal. To address this issue, recent works have shown that the agent performance can be improved by using auxiliary tasks alongside the main RL objective. The corresponding auxiliary losses can be trained using different objectives from supervised (Jaderberg et al., 2016; Schwarzer et al., 2020; Guo et al., 2020), self-supervised (Oord et al., 2018; Laskin et al., 2020), or RL domains (Veeriah et al., 2019).

Designing meaningful auxiliary tasks poses many challenges; most previous works considered them in order to learn visual features entirely without rewards. For example, Schwarzer et al. (2021) learns visual features entirely without considering rewards. In contrast, our focus here is to *discover* auxiliary tasks that are *reward-driven* and help learn useful representations which in turn can facilitate learning better policies. It is generally easier to define predictive knowledge of the environment through value functions, thus learning auxiliary tasks that are driven by such learned reward functions can help to learn many, different, potentially invariant, properties of the environment in order to be robust to any non-stationarity. Thus, these reward-driven auxiliary objectives could specifically serve an important purpose to learn and retain representations when learning in presence of non-stationarity. For instance, in Atari, fundamental objects and corresponding characteristics remain consistent when an agent transitions to a new unseen level. Discovering these objects (features) and learning associated properties (cumulants) could drive generalization and adaptation in realistic continual learning or multi-task settings.

This idea finds its origins in the Horde architecture (Sutton et al., 2011) where auxiliary objective constitutes learning value functions for a multitude of pseudo-rewards termed as General Value Functions (GVFs). GVFs provide a mechanism to learn value functions by choosing any scalar reward signal as a cumulant. These value functions can be used to extract useful predictions based on interactions with the environment thereby learning a rich representation. These estimates of environmental knowledge could potentially act as useful representations for an RL agent in different settings. In this work, we tackle the problem of learning General Value Functions in the pixel space, in an end-to-

[1]École de technologie supérieure [2]Mila-Quebec AI Institute [3]Université de Montréal [4]McGill University [5]CIFAR AI Chair †now at DeepMind. Correspondence to: Somjit Nath <somjit.nath.1@ens.etsmtl.ca>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

end manner by learning *useful* representations. We design a framework for pixel-based agents to take advantage of GVF predictions by treating them as features to ensure a compact yet rich representation. Our main contributions are as follows:

- We propose OC-GVFs: an end-to-end approach to automatically discover object-centric General Value Functions from pixels. Our method can subsequently be used as features for learning the downstream control policy. (Sec 3)
- Instead of learning a huge number of auxiliary tasks as GVFs and only discovering few relevant ones, we aim for our method to focus on discovering the key attributes of the environment. To do this, we consider the **limited GVF regime**, i.e. a small number of GVFs to ensure we get an information-rich representation of the environment.
- We empirically demonstrate that OC-GVFs can outperform the current state-of-the-art algorithms for GVF discovery in both stationary and non-stationary environments. (Sec. 4.4)
- We show that the proposed method can quickly adapt to new unseen situations with increasing complexity. (Sec. 4.5)

2. Background & Related Work

2.1. General Value Functions

GVFs are value functions that are suited to represent predictive knowledge of an agent's environment, such as "how far a particular object is in this gridworld" which represent knowledge about the environment. In Sutton et al. (2011), in addition to the main task, the Horde architecture considers many sub-agents called demons which learn different predictive components of the environment.

GVFs are essentially same as the value functions defined in a Markov Decision Process, except the rewards are not obtained from the task specifically. The input to the GVFs would thus be a policy, a discount factor, and scalar rewards known as cumulants. These cumulants can be described as *questions* to the GVFs. GVF *answers* then are formalized as value functions, henceforth referred to as GVFs. In summary, GVFs are defined as the **expected discounted return over a certain trajectory, where the returns are defined as the discounted sum of the cumulants of interest.** Analogous to value functions in RL, GVFs can be learned by any value-based RL algorithm e.g. Temporal Difference (TD) Learning. In this paper, we use the mean squared TD error, (MSTDE) (Sutton & Barto, 2018a).

2.2. Auxiliary tasks in RL

The idea of auxiliary tasks in RL was introduced to learn from signals other than just scalar reward signals especially when the rewards are sparse, delayed, or noisy. Auxiliary tasks are learned in parallel to the main RL loss (Shelhamer et al., 2016; Sutton & Barto, 2018b).

Auxiliary tasks are an umbrella term and could refer to any task that can aid an RL Agent by predicting observations from the environment. Some of the common auxiliary task setups include reward prediction (Jaderberg et al., 2017). In environments with sparse reward structures, auxiliary tasks provide instantaneous targets for shaping the representation in the absence of reward. Few other works focus on state prediction in the latent space and use the loss from this prediction to drive certain aspects like exploration in RL (Pathak et al., 2017). Paster et al. (2021) focuses on modeling the inverse dynamics of the environment i.e. prediction of actions from state and next-state representations. Veeriah et al. (2019) presented a principled meta-gradient algorithm for the discovery of GVF-based questions to use as auxiliary tasks in the context of RL.

2.3. Cumulant Discovery

To learn GVFs, we need a cumulant for defining the TD target. *Discovery* refers to automatically learning cumulants, that aid in the primary task, referred as *useful cumulants*. Recall that (Sec. 2.2),Veeriah et al. (2019) develop a framework to discover GVF *questions* with a *question network* and learn the cumulants (with the main network). This is in contrast to other auxiliary task methods in RL which use hand-crafted cumulants for learning in the environment (Jaderberg et al., 2016). They propose a meta-learning approach to discover important questions that are useful for learning the main task in the environment and then estimate answers through GVFs for these discovered questions. The idea here is to use the same RL loss driven by the reward from the environment for the discovery of auxiliary tasks. The core intuition behind cumulant discovery is that through the availability of a question network with meta-learnable parameters, we enable the agent to discover useful questions directly from experience. We believe using the value functions learned from these discovered cumulants as part of the input representation for the main RL agent is crucial to our method.

More recently, Kearney et al. (2022) augment the agent's observations with GVFs for control in RL. Kearney et al. also integrate the discovery of GVFs and their use in a single end-to-end framework using a meta gradient descent approach. In a nutshell, they shape the GVF predictions based on the control agent's learning process and use those predictions directly as features for learning a better control policy. In contrast, we only focus on cumulant learning, but we can easily extend our framework to learning γ and policy

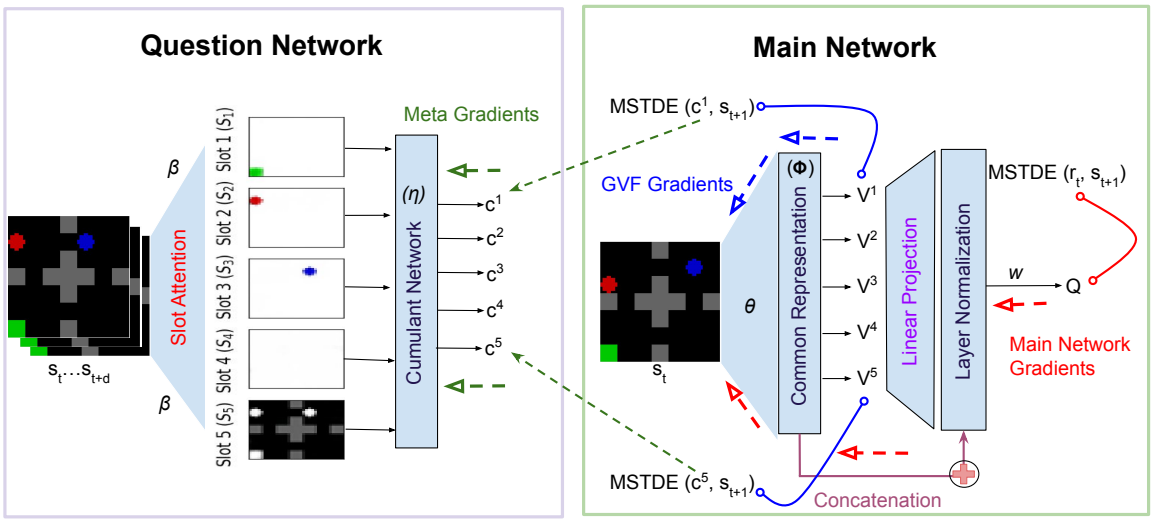


Figure 1. **The Object-Centric GVF Learning Framework**: We have two parts a *Question Network* and a *Main Network*. The Question Network takes in a batch of inputs ($s_t \dots s_{t+d}$) and tries to predict GVF questions (cumulants) from each slot. Each of the cumulants corresponds to the slot outputs (c^k). The Main Network is an RL algorithm with GVF heads each of which is trained by the cumulants from the Question Network. For training GVFs, we use the Mean Squared TD Error (MSTDE) which depends on the cumulants discovered by the Question Network (γ is the same as defined for the main task). The outputs of the GVFs are projected into the latent space where it is concatenated with the representation before normalization and action value prediction with respect to the main task.

correction parameters (Kearney et al., 2022).

2.4. Object-Centric Representations

There have been several recent works that tackle challenging tasks such as object manipulation in presence of multiple objects (Watters et al., 2019; van Steenkiste et al., 2019; Veerapaneni et al., 2020). Much of this work is in a single-task RL setting for a particular reward signal. In contrast, certain works have also focused on learning object-centric representations from images in multi-task RL settings (Pong et al., 2019; Nair et al., 2018; Ghosh et al., 2019; Warde-Farley et al., 2018). These methods rely on some assumptions that the observations can be encoded into a single vector which makes it harder to learn in environments with multiple objects. Zadaianchuk et al. proposed learning object-centric representations which are used for reward shaping. They claim that this approach leads to solving tasks independently and then combining these skills during evaluation.

We chose the slot attention mechanism (Locatello et al., 2020) to learn object centric representations in our approach. These slot representations are used to learn cumulants (through the cumulant network, See Sec.2.3) which in turn are used as part of the input representations to learn a policy. Moreover, there is a one-to-one mapping between the slot representations and the cumulants. This enables discovering useful questions from all the captured slots and ensures that no part of the original image is left unattended.

In this paper, we aim to ask the question: *What are these discovered features learned by these networks and can they be used as input features?* Kearney et al. experimented with a similar architecture for non-image based domains where they concatenated the states with learned GVFs. In our work, we extend this methodology to more realistic image-based domains which are prevalent in RL. A big part of question discovery in these papers is driven by the main RL loss, however, RL loss on its own cannot capture object-centric representation efficiently. Since RL environments heavily rely on object semantics, we add another object discovery loss to the discovery network to bind the discovered cumulants to certain objects discovered by this architecture as explained in Sec. 5.1.

3. Discovering Object-Centric GVFs

Our proposed architecture consists of two separate networks: the *question network* and the *main network*. This two-network meta-gradient approach was introduced for GVF discovery by Veeriah et al. (2019). The primary difference in our work is that we use an Embedded Self prediction (ESP) (Lin et al., 2021) type model to embed GVFs as useful features in our training pipeline. In the original ESP paper, the core intuition was that, if human understandable features are given to the model, the corresponding GVFs would capture meaningful properties of the policy. In a similar way, we directly adapt these trained GVFs as features of the agent's main value function.

We introduce two key modifications to this architecture for

Algorithm 1 Object-Centric GVFs (OC-GVFs)

Input parameters
Num of Slots N, Num of Training Steps(slot module) M
Observations O, Num of GVFs K, Num of episodes E
Initialize parameters of networks β, θ, η
Initialize learning rate of networks α_1, α_2, α_3
function Slot Module Training Phase (N, O, β, M, α_1)
  for $i \leftarrow 1$ to M **do**
    $S_i \leftarrow$ slot_model(O_i)
    $\hat{O}_i \leftarrow$ reconstruct(S_i)
    $\beta_{t+1} \leftarrow \beta_t - \alpha_1 \nabla_\beta \mathcal{L}\left(O_i, \hat{O}_i\right)$
  end for
end function
function GVFs Training Phase(K, E, θ, η)
  for $n \leftarrow 1$ to E **do**
    $t \leftarrow 0$
    $done \leftarrow$ False
    $\theta_{n,0} \leftarrow \theta_n$
    while not $done$ **do**
      $t \leftarrow t + 1$
      $\{C_n^1, \ldots, C_n^k\} \leftarrow f_1(\eta; \{O_t, \ldots, O_{t+d}\})$
      $\{\phi_t\} \leftarrow f_2(\theta; O_t)$
      $\{V_1, \ldots, V_k\} \leftarrow f_3(\theta; \phi_t; \{C_n^1, \ldots, C_n^k\})$
      $\{\psi_1, \ldots, \psi_k\} \leftarrow$ proj$(V_1, \ldots, V_k)$
      $\chi_t \leftarrow$ concat$(\{\psi_1, \ldots, \psi_k\}; \phi_t)$
      $Q_t \leftarrow f_4(w, \chi_t)$
      $\theta_{n,t} \leftarrow \theta_{n,t-1} - \alpha_2 \nabla_{\theta_{n,t-1}} \mathcal{L}^{\mathcal{RL}}(\theta_{n,t-1})$
    end while
    $\eta_{n+1} \leftarrow \eta_n - \alpha_3 \nabla_\eta \sum_{j=1}^{t} \mathcal{L}^{\mathcal{RL}}(\theta_{n,j})$
    $\theta_{n+1} \leftarrow \theta_{n,t}$
  end for
end function

adaptation to all possible input spaces and better stability, namely 1) a key design decision in our approach is to concatenate GVFs with the states in the latent space. This is achieved via linear projection from the outputs of each of the GVFs into the latent space after which they are concatenated with the common representation from the main task. This modification removes the necessity for the states to be vectors which was the case for concatenation with directly the state inputs (Kearney et al., 2022), and 2) addition of layer normalization (Ba et al., 2016) after the concatenation. This helps especially in stability when the slots and hence the cumulants are not learned yet during the early phase of training. Next, we describe the individual components in detail.

3.1. Question Network

The question network (Fig. 1 Left) takes in a batch of state observations ($s_t \ldots s_{t+d}$) which are unrolled from the replay buffer as inputs. These inputs are fed to a slot attention mechanism (parameterized by β) that outputs slots, S_i corresponding to discovered objects from the images. Each of these slots can be considered to have some features of the objects in the images. We learn slot representations through forward propagation of the question network. These slot representations are then mapped to each GVF cumulant using an MLP (parameterized by the meta-parameters η). The slots are trained by reconstruction loss as in Locatello et al. (2020) and the cumulants are trained with the main RL loss similar to Veeriah et al. (2019). This forces the cumulants to capture task-specific properties of each object discovered by each slot. As discussed in the previous section, a GVF question is specified by a cumulant function, a discount function, and a policy. In our method, the question network only explicitly parameterizes the cumulants. Though this is a departure from the architecture used in Veeriah et al. (2019) which parameterized both the cumulants and their corresponding γ, our method similar to Kearney et al. (2022) parameterizes only the cumulants. This is because we wanted to capture the properties of objects which would be relevant to long-horizon settings, but this is definitely something that can be incorporated very easily into this architecture. For our experiments, we use the same γ of the main agent for learning the GVFs.

3.2. Main Network

The main network (on the right side in Fig 1), deals with the training of the GVF answers (parameterized by θ) i.e computing the appropriate value function for each of the cumulants generated by the question network and also learning the main action-value function of the main task (parameterized by w). Note that the number of GVFs is a hyper-parameter in this setup. Once these losses are computed, we do the first backpropagation (blue line) update on the parameter theta as in regular gradient descent. An important detail here is that we do another backpropagation (red line) update on theta while updating the main agent based on the MSTDE with respect to the main task. The meta-parameters are updated based on the cumulative loss incurred over the unrolled data from the buffer. The number of steps of unrolled data is again a hyper-parameter that is tuned as part of the experiments.

Action and State Value GVFs: Before proceeding further, it is imperative to discuss the utility of both using action value GVFs and state value GVFs. Action value GVFs generally contain more information regarding each action than an expectation over actions in the case of state values. However, for our architecture, we preferred to use state value GVFs as they are less prone to divergence from off-policy Q-learning updates which can often be the case in Veeriah et al. (2019). However, since the authors of that paper only use them for learning the representation it does not affect them as much. More details including some empirical

performance plots are in Appendix A.3.3.

To summarize, our method uses feature representations that are output from the slot attention module rather than using human-designed features. We believe this is a key component as we would like the GVFs to automatically discover useful characteristics in an environment. Since many RL environments are image-based and certain aspects of the environment do not change (like objects present in the environment), we believe a mechanism like slot attention adds the most value here. The slot attention module takes in representations from convolution layers and produces abstract representations called slots which bind well to objects in the visual inputs. We refer the reader to the original slot attention paper for more details (Locatello et al., 2020). However, since we have access to an experience replay buffer which is generally present in most RL algorithms, we do not require any pre-training for the slot attention. The data to train slot attention can be directly obtained from the buffer and thus we obtain an end-to-end training pipeline.

4. Experiments

Next, we discuss the empirical performance of our algorithm across different domains and settings. We address the following questions: **Q1.** How does our approach compare to simple baselines in stationary environments? Can this approach significantly outperform other baselines in non-stationary tasks? (Sec. 4.4) **Q2.** Can learned object-centric representations adapt quickly to unseen tasks? (Sec. 4.5) **Q3.** How much do object-centric representations help in cumulant learning when compared to other architectures with similar feature discovery? (Sec. 4.6). To address these questions, we first describe the domains and settings. In Sec. 5.1, we explain our choice of using slot attention for object discovery. In Sec. 5.3, 5.4 we discuss the importance of layer normalization and leveraging the learned GVFs as features respectively.

4.1. Domains

Visualization of the domains are in Appendix A.1.

Collect-objects Environment: is a customized version of the four-room gridworld environment similar to the one used in Veeriah et al.. The agent is rewarded for collecting objects of different colors in the right order. The agent moves deterministically in one of four possible directions. For each episode, the starting position of the agent is chosen at the bottom left. The agent receives a reward of +5 for picking up the red objects and a reward of +10 for picking up the blue object after the red one. We also explore a non-stationary version of this domain where the locations of the objects spawn randomly inside their respective rooms after every episode. If the agent picks up the green object before the red one, the agent does not receive any reward.

MiniGrid-Dynamic Obstacles: For the experiments on non-stationarity, we used the MiniGrid Dynamic Obstacles (Chevalier-Boisvert et al., 2018). In this domain, the agent is placed in a grid where it has to avoid colliding with obstacles and reach the goal. The starting position of the agent and the obstacle positions are all chosen at random.

CoinRun & StarPilot: are a part of procedurally generated environments called ProcGen (Cobbe et al., 2019). In CoinRun, the agent is tasked to capture the coin while avoiding obstacles. In StarPilot, the agent must destroy enemies while avoiding enemy fire and obstacles. We studied the performance of individual levels in CoinRun. In our task adaptation experiments, we study performance at a new unseen level after every episode in a sequential manner.

4.2. Settings

- **Learning in the absence of non-stationarity.** The agent has to collect two objects in the gridworld. The positions of the two objects are fixed across episodes and unchanged. The agent needs to collect the two objects in the same order to receive full rewards.
- **Learning in the presence of non-stationarity.** For Collect Objects, the objects spawn randomly. This creates a non-stationarity in the reward function across these tasks. The task boundaries are not known to the agent as well and are randomly changed after a fixed number of episodes.
- **Quick Adaptation.** To evaluate the agent's ability to quickly adapt to novel situations, we introduce the agent to new unseen levels in the CoinRun and StarPilot domains over time without any prior task information.

4.3. Baselines

We consider the following baselines: 1) **DDQN**, which serves as the main RL algorithm, 2) **Random-GVFs** which uses randomly initialized Question Network, 3) **Hand-Crafted GVFs** that uses human-defined cumulants based on the task,[1] and 4) **Dis-Aux-GVFs** (Veeriah et al., 2019), which is the only[2] prior work that integrates the discovery of cumulants in the pixel space. A key motivation for our method is to ensure that the approach for discovery does not require a huge amount of data, in lieu of which we compare to other baselines in the **limited GVF regime**. We limit the number of GVFs to be 5 for all the environments, the same as the number of slots. We include all the relevant hyperparameters and implementation details in Appendix A.4 and

[1] For Collect Objects, one of the cumulants essentially specifies the location of the red goals.

[2] To the best of our knowledge.

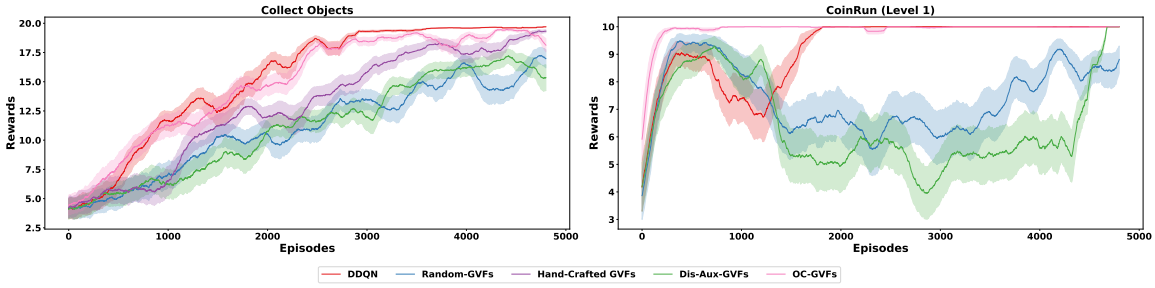


Figure 2. **Learning in the absence of non-stationarity** shows that our method (OC-GVFs) is more sample efficient than using Random-GVFs and Dis-Aux-GVFs. All baselines are expected to show similar performance due to the simple nature of both the Collect Objects and CoinRun stationary domain here. In a simple task, DDQN is marginally better than other methods. However, in CoinRun which is more challenging OC-GVFs is significantly better than all other approaches. Shaded regions correspond to the standard error across 10 independent runs.

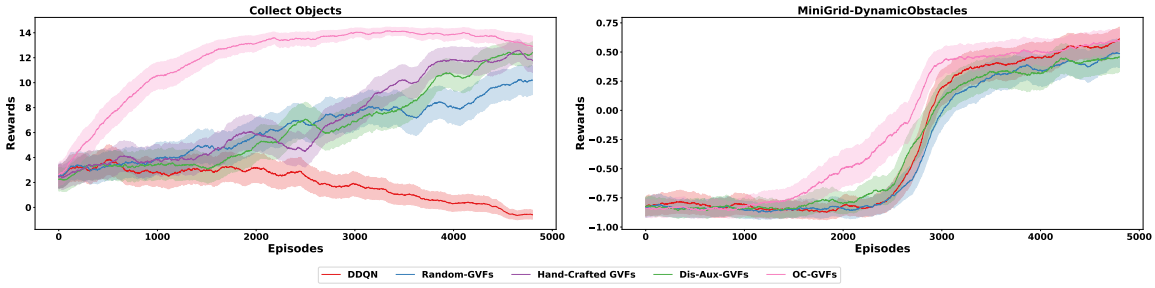


Figure 3. **Learning in the presence of non-stationarity** shows that our approach is sample efficient and quick to adapt when the goal locations are dynamically changed after every episode in the CollectObjects environment. We also observe a similar trend in the Minigrid Dynamic Obstacles environment where the obstacle locations changes across episodes. Baselines such as DDQN cannot adapt to the non-stationarity induced due to changing goals as quickly as OC-GVFs across both domains. All results reported over 10 seeds with shaded region showing the standard error.

open-source the code[3]. Additionally, details of the baselines can be found in Appendix A.2 along with methods that involve GVFs as features.

4.4. Our approach performs competitively in both stationary and non-stationary settings. (Q1.)

In this section, we show the performance of our algorithm (Sec. 3) comparing to the baselines described in Sec. 4.3. We here demonstrate learning in the presence of stationary and non-stationary settings.

Stationary Domains: We note that for simple environments like Collect Objects and CoinRun (first level), DDQN performs very well because the task comprises of fixed locations of objects. As seen in Fig. 2 (a), for Collect Objects all the baselines perform similarly, with OC-GVF slightly better than all the other baselines with GVFs. In Fig. 2 (b), we notice the best performance for our algorithm with DDQN also converging within 2000 episodes. Since this task does not have a lot of *auxiliary* tasks to discover, Dis-Aux-GVFs are not as fruitful in accelerating performance in the main task.

Non-Stationary Domains: These experiments highlight the benefits of using object-centric representations as features. This is evident in Fig. 3 (a) where our algorithm (OC-GVFs) is able to converge much faster than all the other baselines. In the Dynamic Obstacles Environment (Fig. 3 (b)), the difference is not as noticeable as the task is relatively easier, however, our method still outperforms other baselines.

4.5. OC-GVFs is amenable to fast adaptation in the presence of increasing complexity in tasks. (Q2.)

GVFs help in learning predictive knowledge about the environment. As such learned GVFs can often help in adapting to a new task because the agent can utilize previously learned information and quickly adapt to new scenarios. Most end-to-end GVF learning schemes involve discovering cumulants with the main RL loss (Veeriah et al., 2019). While this is useful for the main task, the utility of such GVFs is lost during nonstationarity. On the contrary, as we decouple object discovery (via reconstruction

[3] https://github.com/Somjit77/oc_gvfs

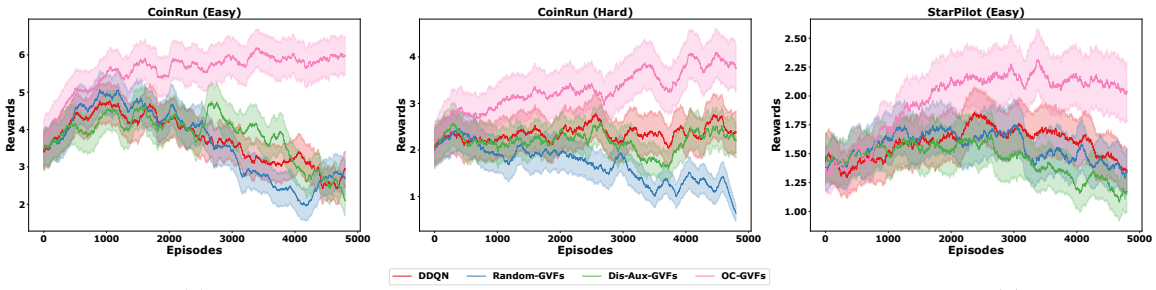


Figure 4. **Adaptation to new tasks:** Our approach OC-GVFs can tackle changing levels better than the baselines. In these experiments, we sample a new task with different difficulty levels after every episode from the first 50 levels. This sampling is carried out from either an easy-to-learn distribution or a hard distribution which is slightly more challenging. The baselines cannot adapt as quickly and perform similarly to DDQN, suggesting no improvement in performance with GVFs. We compare with the baselines mentioned above and report results over 10 seeds.

loss) and cumulant learning (via main RL loss) in our approach, Object-Centric GVFs can adapt and generalize to new situations much faster. During adaptation to a new task, OC-GVFs need to re-learn the GVFs corresponding to the new slots, however, most of the pre-existing slots can be re-utilized when the task complexity increases and the difficulty changes.

Task adaptation refers to adapting to unseen tasks, namely with level changes once algorithms have more or less converged then performance is reported on a new level. These transfer learning experiments are in Appendix A.3.1. We also test the methods for more challenging adaptation by presenting the agent with new unseen levels after *every* episode. This is much more complex for all the algorithms, as seen from Fig. 4, where the baselines do not perform as well including DDQN which was performing well for single levels and transfers across levels. Although OC-GVFs see performance drops due to adaptation to unseen levels every episode, it accumulates a higher average reward across all the sampled levels compared to the baselines.

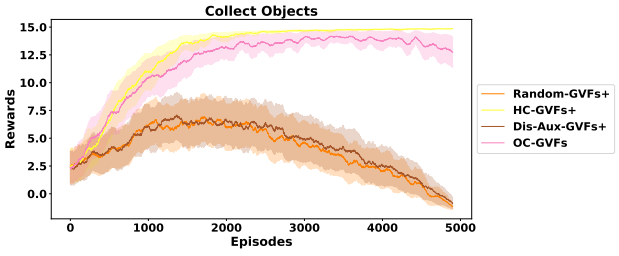


Figure 5. **Object-centric representations** shows the comparison of our approach with other baselines while using GVFs as features(+). This figure singles out the utility of object-centric representations from slot attention mechanism for discovery.

4.6. Feature Discovery without object driven cumulant learning (Q3.)

In this section, we aim to highlight the utility of discovering object-centric cumulants to be used as features. Discovering task-relevant cumulants has proven to be beneficial for learning representations. However, for using GVFs directly as features, it is essential that the cumulants discovered are tethered to some properties of the environment. We compare all the baseline algorithms to investigate whether these learned GVFs can be useful as features (described in Appendix. A.2). Although GVFs discovered for representation learning in Veeriah et al. (2019) were designed for auxiliary tasks, the discovered GVFs from this approach do not work well when used as features.Fig. 5 highlights the disparity between features learned with OC-GVFs and other baselines. Hand Crafted GVFs can learn features best because it can exploit human information and results in best performance.

5. Discussion

5.1. Qualitative Analysis: Understanding slots and how it translates to object properties

One of the crucial components of our approach is the discovery of objects with slot attention. Slot attention is generally trained with the reconstruction loss whereby each slot captures different objects present in the images. For most RL tasks, particularly from pixels, identifying objects can be a big overhead which we are delegating directly to a much more robust model. Since this is a vital component of our framework, it is imperative for slot attention to capture objects on which our cumulant learning is based. In environments where it is difficult to determine objects via slot attention, these methods will not work as well. This is one major limitation of the current framework, however, we have found slot attention to generalize well to different domains with proper architecture changes.

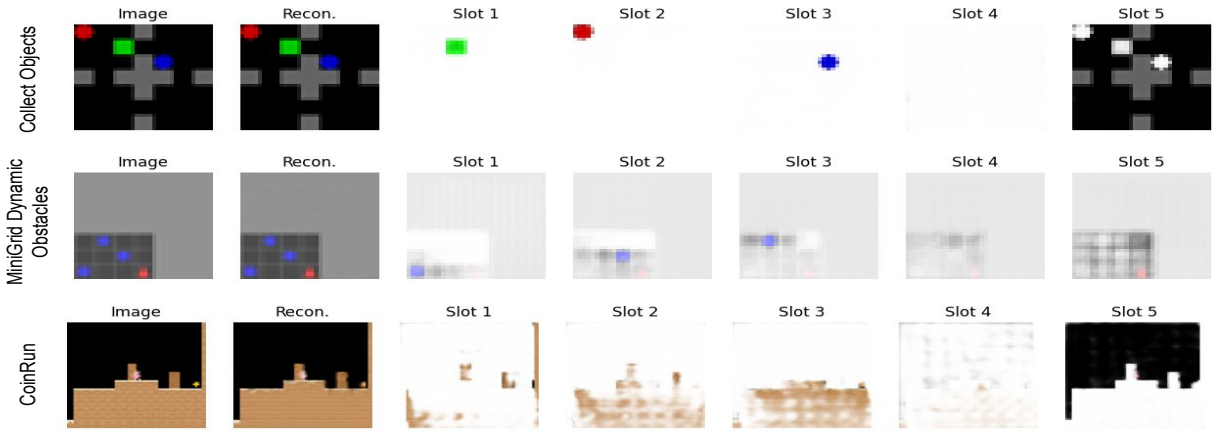


Figure 6. **Slot Attention outputs** shows slots captured by the slot attention mechanism on states sampled from domains. The figure also shows learned state reconstructions for the sampled states across these environments.

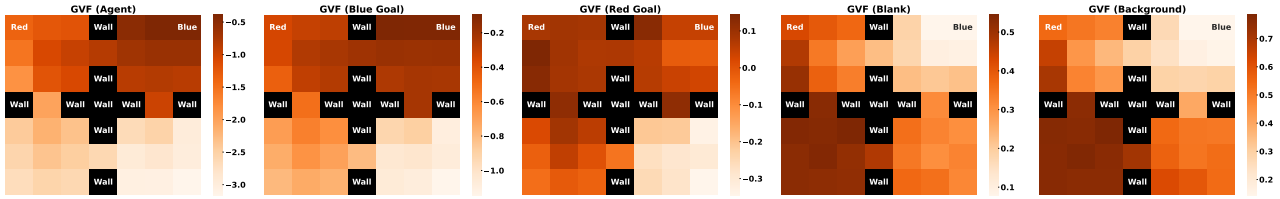


Figure 7. **Visualization of Learned GVFs:** We compare the GVFs learned by OC-GVFs in the CollectObjects environment trained with changing goal positions at every episode. Each of the GVFs is classified based on the slots that are fed as cumulants for training the respective GVFs. For example, the GVF that is learned by the cumulant defined by the slot that captures the Red Goal is GVF (Red Goal). The heatmaps are plotted based on the locations of the agent in the grid with the Red and Blue goal being constant throughout the evaluation. GVF(Blue Goal) and GVF (Red Goal) has higher values near the blue and red goal respectively which highlights it has learned some properties of those objects.

From Fig. 6, we observe how slot attention can segregate the main objects of the state by assigning each slot to those objects. Once features of these objects are learned, they can be used to learn cumulants pertaining to these objects. Slot attention can generalize easily to different locations of such objects which is what enables cumulant learning much faster in comparison to other baseline methods. Another important characteristic of slot attention is that we can be really flexible with the number of slots. In Fig. 6, for both the environments, there is an empty slot that does not contribute to capturing an object, but the relevant objects are still captured in the other slots.

5.2. Visualization of Learned GVFs

Understanding what each GVF actually learns is a important and adds interpretability to our feature design which was missing from previous works like Veeriah et al. (2019). However, adding visualizing GVFs learned from images is tricky because they map images to a scalar value which is difficult to represent in a plot. However, for the Collect Objects domain, the location of the agent can find be a good proxy for the entire observation image provided the other locations remain constant. This is what we use to design our visualization in Figure 7. Each square represents an image with the location of the agent and Red and Blue goal locations are mentioned by text. Now, we plot the GVFs learned in the CollectObjects domain with non-stationary rewards. Each of the GVFs is defined by its own cumulants, which in turn is dependent on the slots that the slot attention model captures. In Figure 7, we add the objects that the slots capture for easy understanding. From the Figure, it is quite evident that for each corresponding object, the GVFs have higher values near the vicinity of that object. This insinuates that the GVFs have learned some form of a distance metric to these objects which can be thought of as compact and rich features when concatenated with the main feature representation. This makes the GVF learning much more interpretable and helps explain the impressive performance of OC-GVFs.

5.3. Importance of Layer Normalization

One of the crucial elements of the proposed architecture is adding a layer normalization layer to the feature space. This is particularly important as in the early phase of learning the slots discovered can be poor leading to absurd cumulant discovery which would make the learned value functions erratic and unstable. This problem would not be as evident for algorithms that only use GVFs for learning representations as they do not directly affect the main value function. As affirmed by Fig. 8, adding layer normalization can significantly improve the stability of our framework leading to faster learning.

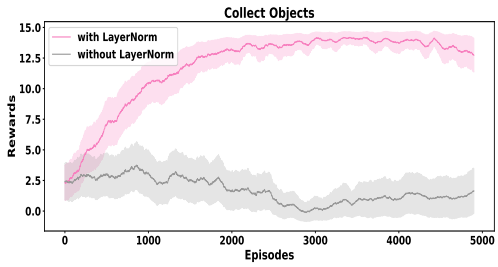


Figure 8. **Layer Normalization** shows how introducing layer normalization in the architecture creates a way for stable and faster learning. This is a critical factor that enables learning GVFs as features.

5.4. Utility of having GVFs as Features

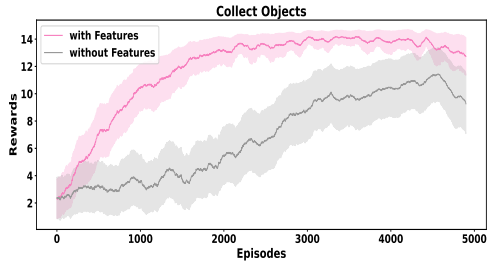


Figure 9. **Utility of GVFs as Features:** We compare the performance of an RL agent using GVFs for learning representations versus using GVFs as features. Even though both the GVFs capture object-centric representations, using the learned GVFs as features makes the most difference.

Another important aspect of the proposed architecture is utilizing GVFs as a part of the feature space. This can be really helpful especially when the learned GVFs contain predictive knowledge about the environment which can be directly utilized by the agent. When the cumulants are well-defined then this approach really shines. Fig. 9 demonstrates the performance of OC-GVFs with and without features. For OC-GVFs without features, we have slot attention capture object properties to be used for representation learning only.

6. Conclusion and Future Work

In this work, we showed the effectiveness of object-centric representation in discovery of GVFs that are used for the control in reinforcement learning. Moreover, we also demonstrated how these learned GVFs capture important components in visual representations and help in quick adaptation to different factors of non-stationarity across tasks.

Limitations: In the current setting, our method relies on the slot attention mechanism to capture distinct objects in the environment. This implies that in the scenarios in which the slot attention mechanism is not able to bind to distinct objects in the pixel space (due to the size of objects, constant movement, etc.), our method OC-GVFs would not perform well without architectural or input representation tweaks. Slot attention is heavily dependent on separating objects by colors so in cases, where the colors of the objects are similar they would always be bound to the same slot which can be problematic depending on the task.

A promising direction for future work would be interesting to explore whether GVFs can perform zero-shot transfer only with the help of previously learned cumulants albeit in the presence of the environments with similar objects. In addition, instead of task-specific cumulants, cumulants can also be trained with task-agnostic losses, which might generalize even better across tasks.

Acknowledgements

The authors would like to thank the ICML reviewers for valuable feedback on an earlier draft of the paper. In addition, we would like to express our sincere gratitude to Google, CIFAR (the Canadian Institute for Advanced Research), NSERC (the Natural Sciences and Engineering Research Council of Canada) and Canada Excellence Research Chairs (CERC) program for their invaluable support and funding. We are also immensely grateful to Compute Canada for providing the computational resources necessary to carry out the experiments. In addition, we would like to extend a special thank you to Alex Kearney for insightful discussions and feedback during the initial research phase.

References

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016. URL https://arxiv.org/abs/1607.06450.

Chevalier-Boisvert, M., Willems, L., and Pal, S. Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid, 2018.

Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. Leveraging procedural generation to benchmark reinforcement learning, 2019. URL https://arxiv.org/abs/1912.01588.

Ghosh, D., Gupta, A., Fu, J., Reddy, A., Devin, C., Eysenbach, B., and Levine, S. Learning to reach goals without reinforcement learning. 2019.

Guo, D., Pires, B. A., Piot, B., Grill, J.-b., Altché, F., Munos, R., and Azar, M. G. Bootstrap latent-predictive representations for multitask reinforcement learning. *arXiv preprint arXiv:2004.14646*, 2020.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks, 2016.

Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W. M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., Fernando, C., and Kavukcuoglu, K. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL http://arxiv.org/abs/1711.09846.

Kalashnikov, D., Irpan, A., Pastor, P., Ibarz, J., Herzog, A., Jang, E., Quillen, D., Holly, E., Kalakrishnan, M., Vanhoucke, V., et al. Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning*, pp. 651–673. PMLR, 2018.

Kearney, A., Koop, A., Günther, J., and Pilarski, P. M. What should i know? using meta-gradient descent for predictive feature discovery in a single stream of experience, 2022. URL https://arxiv.org/abs/2206.06485.

Laskin, M., Srinivas, A., and Abbeel, P. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pp. 5639–5650. PMLR, 2020.

Levine, S., Finn, C., Darrell, T., and Abbeel, P. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Lin, Z., Lam, K.-H., and Fern, A. Contrastive explanations for reinforcement learning via embedded self predictions. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=Ud3DSz72nYR.

Locatello, F., Weissenborn, D., Unterthiner, T., Mahendran, A., Heigold, G., Uszkoreit, J., Dosovitskiy, A., and Kipf, T. Object-centric learning with slot attention, 2020. URL https://arxiv.org/abs/2006.15055.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Nair, A. V., Pong, V., Dalal, M., Bahl, S., Lin, S., and Levine, S. Visual reinforcement learning with imagined goals. *Advances in neural information processing systems*, 31, 2018.

Oord, A. v. d., Li, Y., and Vinyals, O. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Paster, K., McIlraith, S. A., and Ba, J. Planning from pixels using inverse dynamics models. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=V6BjBgku7Ro.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pp. 2778–2787. PMLR, 2017.

Pong, V. H., Dalal, M., Lin, S., Nair, A., Bahl, S., and Levine, S. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. Data-efficient reinforcement learning with self-predictive representations. *arXiv preprint arXiv:2007.05929*, 2020.

Schwarzer, M., Anand, A., Goel, R., Hjelm, R. D., Courville, A., and Bachman, P. Data-efficient reinforcement learning with self-predictive representations. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=uCQfPZwRaUu.

Shelhamer, E., Mahmoudieh, P., Argus, M., and Darrell, T. Loss is its own reward: Self-supervision for reinforcement learning. *arXiv preprint arXiv:1612.07307*, 2016.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018a. ISBN 0262039249.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018b.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., White, A., and Precup, D. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent*

Systems - Volume 2, AAMAS '11, pp. 761–768, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 0982657161.

van Hasselt, H., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning, 2015. URL https://arxiv.org/abs/1509.06461.

van Steenkiste, S., Greff, K., and Schmidhuber, J. A perspective on objects and systematic generalization in model-based rl. *arXiv preprint arXiv:1906.01035*, 2019.

Veerapaneni, R., Co-Reyes, J. D., Chang, M., Janner, M., Finn, C., Wu, J., Tenenbaum, J., and Levine, S. Entity abstraction in visual model-based reinforcement learning. In *Conference on Robot Learning*, pp. 1439–1456. PMLR, 2020.

Veeriah, V., Hessel, M., Xu, Z., Lewis, R., Rajendran, J., Oh, J., van Hasselt, H., Silver, D., and Singh, S. Discovery of useful questions as auxiliary tasks, 2019.

Warde-Farley, D., Van de Wiele, T., Kulkarni, T., Ionescu, C., Hansen, S., and Mnih, V. Unsupervised control through non-parametric discriminative rewards. *arXiv preprint arXiv:1811.11359*, 2018.

Watters, N., Matthey, L., Bosnjak, M., Burgess, C. P., and Lerchner, A. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

Zadaianchuk, A., Seitzer, M., and Martius, G. Self-supervised visual reinforcement learning with object-centric representations. *arXiv preprint arXiv:2011.14381*, 2020.

A. Appendix

A.1. Domains

Fig. 10 are sample input states of all the domains used in the experiments.

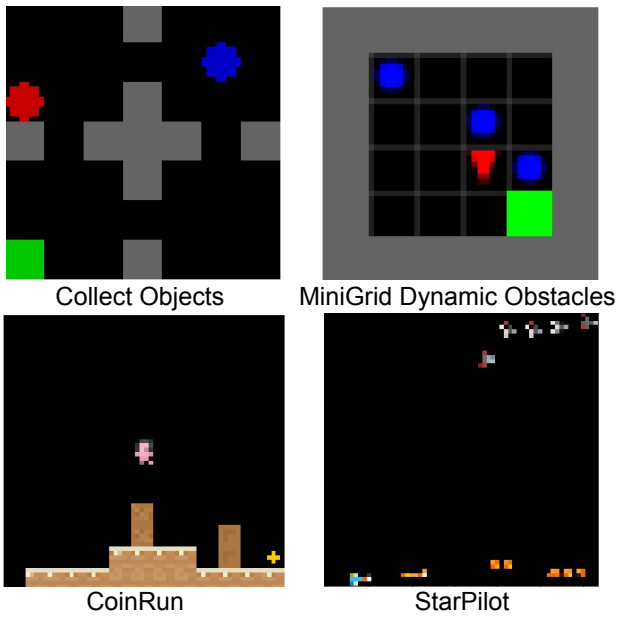


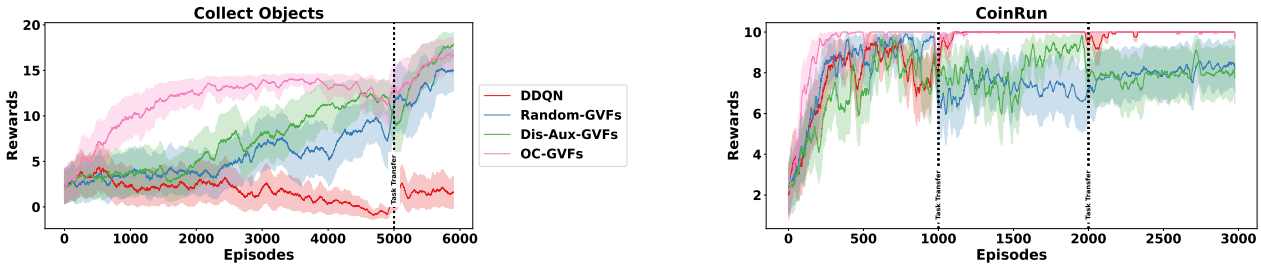
Figure 10. **Domains.** This figure illustrates the domains used in the Experiments. All of the environments require identifying objects in the input space from pixels.

A.2. Baseline Descriptions

Since we claim to learn object-centric representations in the pixel space, we use a 3 convolution layer architecture for the representation layers as in the original DQN implementation (Mnih et al., 2015). This is the part of the main network that was earlier explained in Sec. 3.2 and the main RL network is a DQN agent learning just with the TD loss. In our setup, the question network is a simple MLP network with one fully-connected layer and it outputs the cumulants to be used in the main network as explained in the main paper. All the baselines used in the paper are explained in detail as follows.

1. **DDQN**: For all our experiments, we used Double DQN (van Hasselt et al., 2015) as our base RL algorithm. This is the extension of the DQN (Mnih et al., 2015) algorithm with double Q learning to prevent maximization bias.

2. **Random-GVFs**: We also compared performance against using random cumulants for the learned GVFs. For these experiments, we sampled cumulants from



(a) (b)

Figure 11. **Transfer Learning:** Our approach OC-GVFs can tackle the scenario of changing tasks better than the baselines with no appreciable drop in performance. In Collect Objects environment we change tasks by changing goal locations after certain episodes. Whereas, in Coinrun the different tasks refer to procedurally generated levels varying in task difficulty. Again we compare with the baselines mentioned above and report results over 10 seeds.

a uniform distribution between $[-1, 1]$. The learned GVFs only help in learning the representations, and they are not used as main features.

3. **HandCrafted-GVFs**: This is similar to Random GVFs except we use human knowledge to preselect good cumulants in advance. For the Collect Objects environments, the five cumulants chosen were +1 for reaching the red goal and each of the corridors.

4. **Discovery of Useful Questions as Auxiliary Tasks (Dis-Aux-GVFs)**: This algorithm is from Veeriah et al. (2019) which discovers cumulants from the main RL loss via meta-gradient descent. This is the basic form of discovery that uses the main RL loss to learn cumulants given inputs.

5. **GVFs as Features**: In all of these methods, the GVFs are not only used to influence the representation but are used as features of the main RL network. All these methods include concatenation in the latent space by a linear projection of the GVFs followed by layer normalization. In this category, we have four more algorithms:

   - **Random GVFs as Features (Random-GVFs+)**
   - **Hand Crafted GVFs as Features (HC-GVFs+)**
   - **Using Discovered auxiliary tasks as features (Dis-Aux-GVFs+)**: This algorithm would be most similar to Kearney et al. (2022) where the discovery architecture is kept similar to Veeriah et al. (2019), except we add compatibility with all forms of input space with the help of projected concatenation in the latent space and added stability with layer normalization.
   - **Using discovered Object-Centric GVFs as Features (OC-GVFs)**: The proposed algorithm falls under the umbrella of using GVFs as features.

Note: Algorithms 2-4 all learn action value functions as GVFs similar to how it was designed in Veeriah et al. (2019). All algorithms falling under the umbrella of using GVFs as features (Alg 5) use state value functions as GVFs. Unstable or divergent action values can cause catastrophic failure particularly when they are used as features.

A.3. Additional Experiments

A.3.1. Transfer Learning Experiments

In Fig. 11, we demonstrate settings with increase complexity in tasks; 1) CollectObjects where the task changes once and and 2) CoinRun where the level of difficulty changes twice. In CollectObjects, the new task corresponds to adding one randomly positioned red goal, while CoinRun includes procedural generation of Level $0- > 2- > 3$. We see in Fig. 11 that our approach OC-GVFs not only outperforms the other methods in the initial levels, but also is quick in few-shot adaptation when faced with new harder levels in comparison to the other baselines. DDQN performs poorly in the initial level, but once it is caught up, it remains consistent with the increasing complexity of levels, but suffers from a higher variance as compared to OC-GVFs. Other methods including (Veeriah et al., 2019) struggle in all levels of CoinRun. In these settings, the baselines perform well because we believe this is a much simpler setting as even after transfer we allow the network to train for 1000 episodes which is enough to learn a good representation.

A.3.2. Feature Discovery without object driven cumulant learning

Fig. 12 shows the performance on Collect Objects with static and dynamic object locations and MiniGrid Dynamic Obstacles. In Collect Objects, all the algorithms do relatively well because it is an easy domain, however, we can see the utility of using object-centric representations as features in MiniGrid where the difference is more pronounced.

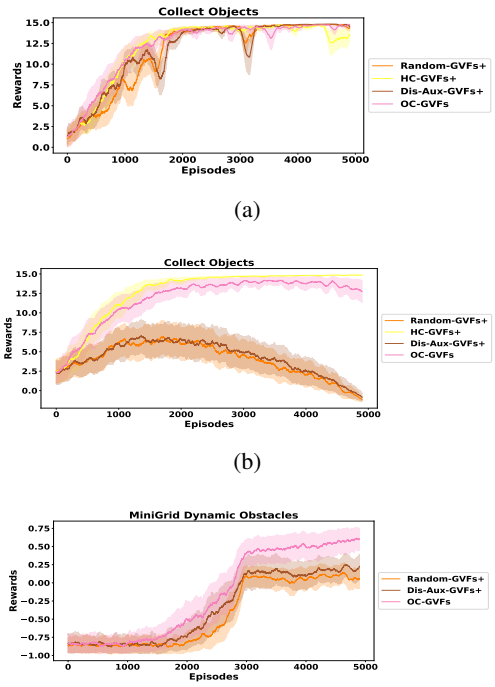


Figure 12. Comparison of the baseline architectures on Collect Objects with stationary(top) & non-stationary rewards (middle) using GVFs as features (+). Each of the baselines algorithms has **projected concatenation** and **layer normalization** added. In spite of these additions, the baselines do not perform well. Only our algorithm OC-GVFs is able to perform close to HandCrafted-GVFs with predefined knowledge of the environment. This further highlights the importance of object-centric discovery.

A.3.3. Action versus State Values

Fig. 13 demonstrates the performance of the baselines for both state and action value functions in the Collect Objects environment with random object placements. As mentioned earlier in Sec. 3.2, GVFs can be learned with both state and action values. Action values are more prone to divergence because of the off-policy nature of action value functions and as a result using them as features results in a slightly worse performance for our algorithm. However, using state values did not seem to work at all for the other baselines when used only for training the common representation. We believe this is due to the less information provided by state value functions, which provide an expectation over all possible actions for each state.

A.4. Implementation Details

For the experiments, we optimized the performance of the main DDQN agent in terms of hyper-parameters and kept them constant for each of the baselines. The other parameters for discovery were kept the same as Veeriah et al. (2019). Most of the slot attention hyper-parameters were kept the same as Locatello et al. (2020). Some modifications were made in the encoder and decoders depending on the resolution of the input images. We used a much smaller resolution compared to Locatello et al. (2020), as finding separating them distinctly was not the objective for our method. This end-to-end pipeline will be undoubtedly more expensive because of the additional overhead of training slot attention. However, in our experiments, we resize the image to a sufficiently small (32x32) resolution which reduces training time and at the same time discovers slots that are 'good enough' for learning cumulants. For CoinRun we used (64x64) images with lesser training steps. The final set of hyper-parameters are listed in Table 1. All our experiments were run on a single V100 GPU.

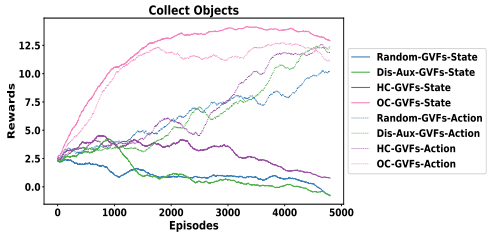


Figure 13. **State & Action Values:** Comparison of baselines algorithms using state v/s action value GVFs. OC-GVFs are robust to both, however, with state values, the baseline algorithms are not as good due to the lack of information captured by them for learning a good representation.

Table 1. Hyper-Parameters of all experiments

Environment	Algorithm Parameters	Encoders and Decoders	Slot Attention Parameters
CollectObjects	"train_episodes": 5000, "batch_size": 32, "target_period": 100, "replay_capacity": 100000, "hidden_arch": [64,32], "epsilon_begin": 1.0, "epsilon_end": 0.01, "epsilon_steps": 0.8, "discount_factor": 0.99, "learning_rate": 0.0001, "eval_episodes": 100, "evaluate_every": 50, "num_gvfs": 5, "unroll_steps": 10	**Main CNN:** Conv2D:filters=16, kernel=3 MaxPool2D:strides=2 Conv2D:filters=32, kernel=3 MaxPool2D:strides=2 Conv2D:filters=64, kernel=3 **Slot Attention Encoder:** Conv2D filters=32, kernel=3 filters=32, kernel=3 filters=64, kernel=3 **Slot Attention Decoder:** Conv2DTranspose filters=64, kernel=3, stride=2 filters=32, kernel=3, stride=2 filters=32, kernel=3, stride=1 filters=4, kernel=3, stride=1	"sa_batch_size": 16, "sa_resolution": 32, "sa_num_slots": 5, "sa_num_iterations": 3, "sa_learning_rate": 0.0004, "sa_num_train_steps": 200000, "sa_warmup_steps": 10000, "sa_decay_rate": 0.5, "sa_decay_steps": 100000
CoinRun	"train_episodes": 5000, "batch_size": 32, "target_period": 100, "replay_capacity": 10000, "hidden_arch": [64,32], "epsilon_begin": 1.0, "epsilon_end": 0.01, "epsilon_steps": 0.8, "discount_factor": 0.99, "learning_rate": 0.0001, "eval_episodes": 100, "evaluate_every": 50, "num_gvfs": 5, "unroll_steps": 10	**Main CNN:** Conv2D:filters=16, kernel=3 MaxPool2D:strides=2 Conv2D:filters=32, kernel=3 MaxPool2D:strides=2 Conv2D filters=64, kernel=3 **Slot Attention Encoder:** Conv2D filters=32, kernel=5 filters=32, kernel=5 filters=64, kernel=5 **Slot Attention Decoder:** Conv2DTranspose filters=64, kernel=5, stride=2 filters=32, kernel=5, stride=2 filters=32, kernel=5, stride=2 filters=32, kernel=3, stride=1 filters=4, kernel=3, stride=1	"sa_batch_size": 16, "sa_resolution": 64, "sa_num_slots": 5, "sa_num_iterations": 3, "sa_learning_rate": 0.0004, "sa_num_train_steps": 100000, "sa_warmup_steps": 10000, "sa_decay_rate": 0.5, "sa_decay_steps": 100000
MiniGrid Dynamic Obstacles	"train_episodes": 5000, "batch_size": 32, "target_period": 100, "replay_capacity": 100000, "hidden_arch": [64, 32], "epsilon_begin": 1.0, "epsilon_end": 0.001, "epsilon_steps": 0.6, "discount_factor": 0.99, "learning_rate": 0.0001, "eval_episodes": 100, "evaluate_every": 50, "num_gvfs": 5, "unroll_steps": 10	**Main CNN:** Conv2D:filters=16, kernel=3 MaxPool2D:strides=2 Conv2D:filters=32, kernel=3 MaxPool2D:strides=2 Conv2D:filters=64, kernel=3 **Slot Attention Encoder:** Conv2D filters=32, kernel=3 filters=32, kernel=3 filters=64, kernel=3 **Slot Attention Decoder:** Conv2DTranspose filters=64, kernel=3, stride=2 filters=32, kernel=3, stride=2 filters=32, kernel=3, stride=1 filters=4, kernel=3, stride=1	"sa_batch_size": 16, "sa_resolution": 32, "sa_num_slots": 5, "sa_num_iterations": 3, "sa_learning_rate": 0.0004, "sa_num_train_steps": 400000, "sa_warmup_steps": 10000, "sa_decay_rate": 0.5, "sa_decay_steps": 100000