Supplementary Materials: Unsupervised Visual Attention and Invariance for Reinforcement Learning

Xudong Wang* Long Lian* Stella X. Yu
UC Berkeley / ICSI
{xdwang,longlian,stellayu}@berkeley.edu

1. Additional Environment Descriptions

1.1. DeepMind Control

We wrote a short description for each environment in DeepMind Control suite [3] in Table 1 to further introduce the environment.

Environment	Descriptions
Walker	A planar walker which encourages an upright torso and minimal torso height in the "stand" task. In "walk" task forward velocity is also encouraged.
Cartpole	A pole tied to a cart at its base, with forces applied to the base. "swingup" task requires the pole to swing up from pointing down while "balance" task requires the pole to balance to be upright.
Ball in cup	A ball attached to a cup, with forces applied to the cup to swing the ball up into the cup in the "catch" task.
Finger	A finger is asked to rotate a rectangular body on a hinge. The top of the body needs to overlap with the object in "turn_easy" task and the body needs to rotate continuously in the "spin" task.
Cheetah	An animal with two feet which is asked to run in the "run" task.
Reacher	A planar reacher with two links connected with a hinge in a plane with a random target location. In the "easy" task, the reacher is asked to reach the object location. The "hard" task is unused in our evaluation since it was not adapted by [1].

Table 1: Descriptions for each environment in DeepMind Control suite.

We also provide samples for the evaluation environments designed by [1] in Fig. 1.

1.2. DrawerWorld

We propose the DrawerWorld, a benchmark with observations in pixels, based on MetaWorld [4] to enable the agent to work in an environment close to real-life scenarios. There are two tasks in DrawerWorld, which are DrawerOpen and DrawerClose. These tasks ask a Sawyer robot to open and close a drawer, respectively.

The multi-component reward function R is a combination of a reaching reward R_{reach} and a push reward R_{push} as follows:

$$\begin{aligned} R &= R_{\text{reach}} + R_{\text{push}} \\ &= -||h-p||_2 + \mathbb{I}_{||h-p||_2<\epsilon} \cdot c_1 \cdot \exp\{||p-g||_2^2/c_2\} \end{aligned} \tag{1}$$

where ϵ is a small distance threshold and is set as 0.08 by default, $p \in \mathbb{R}^3$ be the object position, $h \in \mathbb{R}^3$ be the position of the robot's gripper, and $g \in \mathbb{R}^3$ be goal position. $c_1 = 1000$ and $c_2 = 0.01$ for all tasks in DrawerWorld benchmark.

The goal of distraction-robust RL is to learn a task-conditioned policy $\pi(a|s,z)$, where z indicates an encoding of the task ID, and in this case, different task IDs have different drawer positions. This policy should maximize the average expected return from the task distribution $p(\mathcal{T})$, given by $\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}[\mathbb{E}_{\pi}[\sum_{t=0}^{T}\gamma^t R_t(s_t,a_t)]]$. The success metric, which is evaluate the agent in evaluation time, is described by $\mathbb{I}_{||p-g||_2<\epsilon}$, where ϵ is set to 8cm. The difference between training and evaluation time is the texture and color of the table cloth. Image samples of textures that we use are provided in the main text.

2. De-noise with Past Averages

Since our adapter model works on each frame separately without any assumption on temporal continuity of consecutive frames, our adapter works exactly the same on videos as on fixed backgrounds and is not affected by drastic changes in the background such as flashes of light. However, in some environments where the assumption of temporal continuity holds, i.e. with a relatively slow-moving background, we may make use of this assumption to better de-noise the observations before passing the them into the adapter.

We exploit the assumption here by keeping a mean of past observations $\mathbf{o}_{\text{mean}} = \frac{1}{t}\sum_{i=1}^{t}\mathbf{o}_i$ and subtract the mean from observation $\mathbf{o}_t$ and compute the observations after de-noise with the formulation: This de-noise step happens before the observation is sent into CNN (adaptor), formulated as:

$$\mathbf{o}_{\text{de-noised}} = \text{filter}(\mathbf{o}_t - \alpha\mathbf{o}_{\text{mean}}, \epsilon) + \alpha\mathbf{o}_{\text{mean\_color}} \tag{2}$$

*Equal contribution.

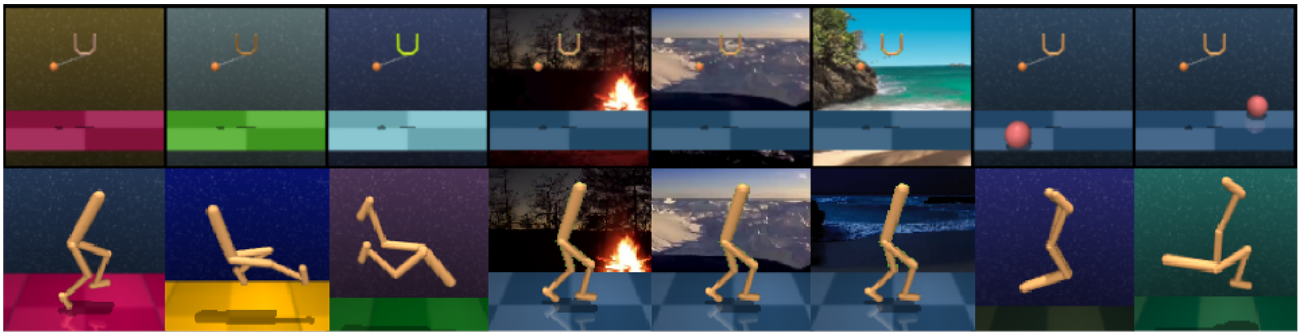

Figure 1: Samples in evaluation environments in DeepMind Control. The samples in the first row are from [1].

where $\mathbf{o}_{\text{mean\_color}}$ is the mean of $\mathbf{o}_{\text{mean}}$ in spatial dimensions, filter is a function that sets the part with value less than ϵ to 0 to remove some noise, and $\alpha \in [0, 1]$ is the strength in noise removal.

This is completely optional, and since it makes use of an additional assumption, they are only used in the cartpole and ball in cup in experiments with background videos and experiments with distracting objects. In addition, we observe that with Places dataset as augmentation, the model is robust enough without this trick, so we disable it for all models in the training of which Places dataset is used.

3. Memory Usage and Speed Comparisons

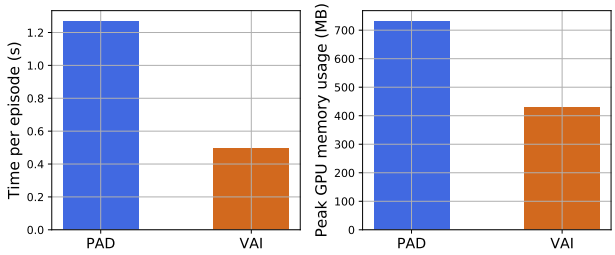

Figure 2: Comparisons on the mean time per episode and GPU memory occupancy at evaluation time for *DrawerClose* task in DrawerWorld between current state-of-the-art method PAD [1] and the proposed method VAI. VAI is more than 2 times faster than PAD during testing time and requires ∼40% less GPU memory usage. Both methods are evaluated with exactly the same backbone network. We take the mean of 10 runs for the latency comparison. Memory usage is obtained with `torch.cuda.max_memory_allocated`.

From Fig. 2, it seems that VAI is about 3 times as fast as PAD in terms of the evaluation time in each episode and requires substantially less GPU memory than PAD. This is largely due to the fact that PAD trains the encoder network at evaluation time with back-propagation, which not only requires the intermediate results to be saved in GPU memory but also requires backward computation to update the model parameters, which consumes both time and memory space. Although VAI has an extra adapter module, the computation and memory it takes are much less than the ones required by backward computation and storing intermediate results. According to the requirements of computational resources in terms of speed and memory, our method is more suitable for robots powered by battery and edge inference devices than PAD from this point of view.

RL Observations	Cumulative Reward
Joint Positions, Velocity, Torso Height from the Environment	969±2
Joint Positions from the Environment	935±3
Keypoints Extracted with KeyNet	709±3
VAI on Training Environment	889±3

Table 2: Cumulative rewards on *Walker, walk* task with 1) joint positions, velocity, and torso height from the environment as observations; 2) joint positions from the environment as observations; 3) keypoints extracted by KeyNet from images; 4) The proposed method VAI. The first two use the ground truth information, which is not accessible during real-world deployment, and serve as upper bounds. For experiment 2, 3, and 4, we use stack of 3 frames as input for the RL agent to infer the velocity since velocity information is missing. Since walker is a planar environment (the walker will not lean towards to away from the screen), the extracted keypoints should roughly correspond to positions from the significant parts of the walker body. The gap between experiments indicate that a limited number of keypoints from KeyNet on its own is not a sufficiently informative or accurate source for observations for an RL agent, which is in accord with our visualization in the main text about the keypoints' temporal inconsistency.

4. Further Experiments with Raw Keypoints

To investigate the question of whether raw keypoints extracted from image observations by KeyNet are able to contribute to effective learning of useful behaviors, we set up experiments based on the *Walker, walk* task in DeepMind Control and list the outcomes in Table 2.

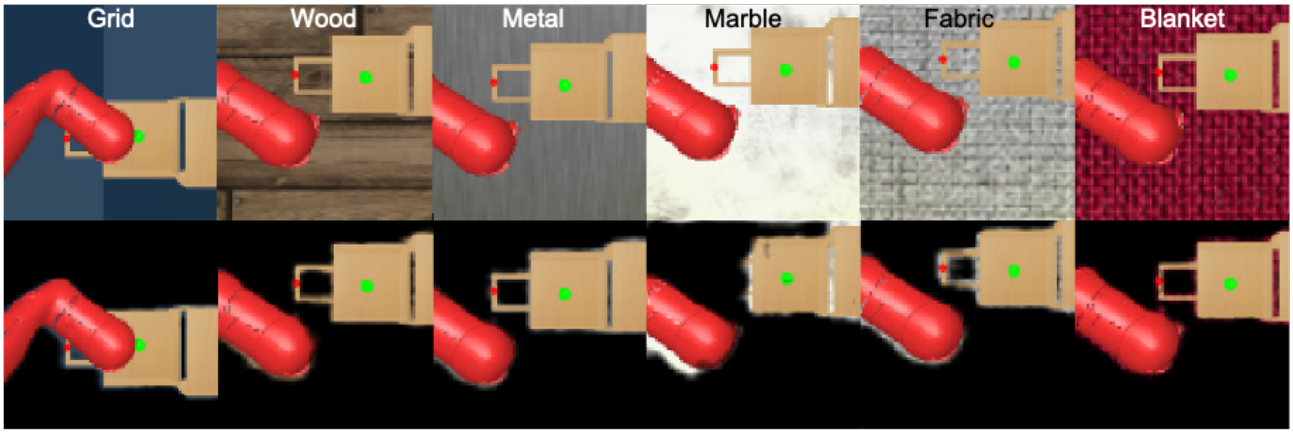

Figure 3: Samples from DrawerWorld, *DrawerClose* task with their corresponding observation processed by the adapter module. The Grid task is the training task for the adapter. All the observations use the same adapter for a fair comparison.

We first train an agent with the state-based observation provided by the environment, a 24-dimensional tensor, which includes positions of the joint, velocity, and walker's torso height. We do not stack frames for this experiment. This experiment indicates an upper bound that our agent is able to achieve in this environment. However, to make a fair comparison with other experiments, where velocity and torso height information is not directly provided, we also remove these parts from our observation, leaving only the positions of the joint as observation in the second experiment. Thus, in the second experiment, the agent directly reads a 14-dimensional tensor per frame from the environment based on the position of each joint. The third experiment is conducted with the RL agent reading a tensor that contains the (x, y) coordinates of 24 keypoints. The keypoints are from a KeyNet which reads an image input. The KeyNet is pre-trained with transporter. In the last experiment, we run VAI, with the adapter module trained from the same KeyNet that is used in the experiment above, on the training environment, although it is able to adapt to other environments as well and thus is more general. To infer velocity information, we stack the observations for three frames for experiments 2, 3, and 4. We run both experiments for 500k steps. We compare their efficiency by evaluating the agent in training environment 10 times with 10 seeds.

According to the performance of these RL agents in the training environment, keypoints on its own do not capture all the information needed by the RL agent accurately. This will be even worse if the agent is evaluated in a different environment is has never seen before, since KeyNet itself does not come with the ability to adapt, although the keypoints it generates are not supposed to carry domain-specific or distraction information. Using keypoints information along with image features as well as history observations may help, as illustrated in [2] and described in the main text, but it will add greatly to the complexity of the RL framework. What's more, agents may need information other than what keypoints provide. For example, keypoints do not carry the shape, size, and color information, which may be of paramount importance in certain tasks. Furthermore, since KeyNet allocates an output dimension for each keypoint, the number of parameters as well as computation time scales linearly with the number of keypoints, which prohibits adding a large number of keypoints to compensate the effect of temporal inconsistency or to capture complicated observations. In contrast, since KeyNet is not used in getting adapted observations in our method, the speed and number of parameters of our RL agent, including the adapter, at evaluation time are not affected by the number of keypoints used to generate ground-truth, which allows our method to scale to complicated environments with many moving parts without losing efficiency.

5. Visualizations of the Observation Adapter

How to make sure that RL will adapt to a certain setting that is different from training setting is still an open problem. Our method opts to work on observation-space. In contrast, PAD works on an intermediate encoder feature space. Our method is much easier to visualize and debug since humans are able to directly understand the quality of adapted observations while it is really difficult to understand what happens in the feature space.

To give examples on how to assess whether an adapter works on a certain environment easily and to illustrate our performance in a visual way in evaluation environments, we gathered 6 pairs of raw samples and samples processed by the adapter in *DrawerClose* task from the same adapter in Fig. 3. As can be seen from the examples, the adapter model differentiates most of the evaluation environments well, with the exception of the marble environment, which the adapter confuses parts of the foreground and background such as the

handle and the patches around the actuator, probably due to the fact that the reflected light on the actuator has a similar white color to the color of background. This indicates why our model performances worse in marble environment, as illustrated in the experiment section in the main text, and, in real-life applications, means that the adapter needs to be re-trained or fine-tuned with observations from similar environments, or if this is not applicable, with augmentation specially-designed to handle this case. We leave the question of handling adapter fine-tuning and re-training to later research.

This visualization has a large impact on the real-world applications of our method: with only a few observations from an intended deployment environment, one could easily visualize and assess whether our method will adapt to such environment. This does not require any ability to run the policy in the dynamics, nor does it require reward functions or consecutive observations which may be difficult to obtain from deployment environments in real-world applications. We strongly believe that this simple assessment provides a direction for future research in explainable, adaptable, and generalizable reinforcement learning and will present great benefit to potential applications of reinforcement learning.

References

[1] Nicklas Hansen, Yu Sun, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[2] Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. In *Advances in neural information processing systems*, pages 10724–10734, 2019.

[3] Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020.

[4] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2019.