Pre-trained Image Encoder for Data-Efficient Reinforcement Learning and Sim-to-Real transfer on Robotic-Manipulation tasks

Anonymous Author(s)
Affiliation
Address
email

Abstract: Sample-efficiency is still a major challenge for reinforcement-learning (RL) algorithms, particularly when learning directly from image inputs. We propose a simple two-step pipeline: First, learn a visual representation of the scene by pre-training an encoder from multiple supervised computer-vision objectives, then train an RL agent which can focus solely on solving the task. We evaluate our method on 3 realistic manipulation tasks with a simulated 6-degrees-of-freedom robot. We show that not only is our method much more sample-efficient than an end-to-end baseline, but it also reaches a higher final success rate, even solving one of the tasks where the baseline fails to make any progress. Additionally, by adding domain randomization techniques into our pipeline, we are able to solve a simpler reaching task consistently in the real world via zero-shot sim-to-real transfer.

Keywords: Reinforcement Learning, Computer Vision, Sim-to-Real

1 Introduction

We present a simple pipeline to tackle vision-based robotic-manipulation tasks in a data-efficient manner. The idea is to pre-train an encoder from multiple computer-vision objectives, such as image segmentation or depth prediction. The pre-trained encoder is then frozen and used by the RL agent to bypass the raw images. We focus on three tasks with sparse rewards: reaching a target, pushing a button, and sliding a block to a target position. We instantiate our approach with Soft Actor-Critic (SAC) [1] and compare it to an end-to-end baseline. Additionally, we present a modified version of the pipeline for sim-to-real zero-shot transfer, and show that our agent is able to solve a simpler reaching task in the real world consistently.

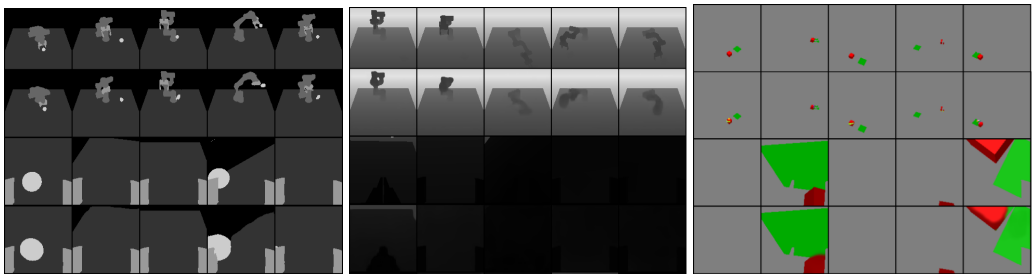
2 Related work

Learning representations for RL. Representation Learning is particularly challenging in RL, since learning good representations requires diverse data, which relies on good exploration, which itself relies on good representations. Multiple works attempt to learn good representations jointly with the RL objectives [2, 3, 4].

Pre-training for RL. Learning a dynamics model is an useful way of learning good representations, that can later be used to feed a model-free RL agent [5, 6]. Several works focus on extracting keypoints from images as a preliminary step [7, 8]. Like in our work, pre-training can also come from computer-vision objectives [9, 10].

Domain randomization for RL. The goal behind domain randomization is to make the agent learn the domain's invariances directly from data. If the data is rich enough, the agent should be able to

Submitted to the 6th Conference on Robot Learning (CoRL 2022). Do not distribute.



(a) Image segmentation (reaching) (b) Depth prediction (pushing) (c) Auto-encoding (sliding)

Figure 1: Qualitative results of the pre-trained encoder on some test images. The first row for each view shows the ground-truth, the second row shows the network outputs.

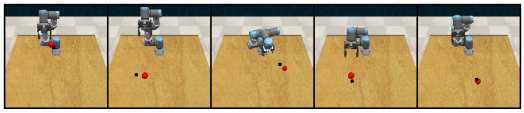


Figure 2: State regression: Qualitative results of the pre-trained encoder for the reaching task on some test images. The black square shows the 2D projection of the 3D output for the ball's coordinates.

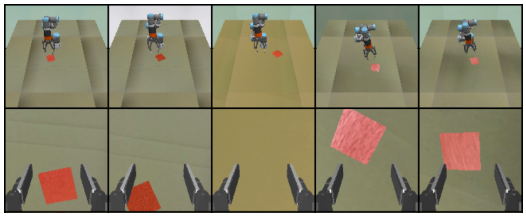


Figure 3: Texture randomization: The texture of the walls, table and target are randomized to bring variety to the dataset.

generalize to new environments. Many works focus on the visual appearance of the scenes [11, 12, 13]. Other elements can also be altered, particularly regarding the dynamics of the environment. A good framework for such works is that of Meta-Learning (e.g. [14]).

3 Method

Our method consists in two steps. First, we train an encoder on a collection of supervised computer-vision objectives. Then, we freeze the weights of the encoder and use it as the backbone of all RL networks to learn a task. Let $s = (s_{\text{image}}, s_{\text{proprioception}})$ the RL state. The goal of the first step is to learn a more compact representation $\tilde{s}_{\text{image}}$, and the goal of the second step is to train an RL agent from the new states $\tilde{s} = (\tilde{s}_{\text{image}}, s_{\text{proprioception}})$. We use SAC as our RL algorithm.

3.1 Training the Image Encoder

All three encoder networks (one per task) were trained with identical hyper-parameters and architecture (ResNet-50). We propose to learn a representation from multiple objectives simultaneously: Image Segmentation, Depth Prediction, Auto-encoding, and State Regression. A simple ResNet decoder is used for the first three objectives (one network per objective), and a simple MLP for regressing the state of the environment (objects' coordinates and robot proprioception).

As shown in Figure 4, our setup consists of two cameras, one mounted on the wrist of the robot and one in front of it. The encoder takes a single image as input, and all three vision decoders take the corresponding 32-dimensional encoded representation as input. However, the MLP decoder for state regression takes in the concatenation of both encoded vectors (one for each camera). This 64-dimensional vector corresponds to $\tilde{s}_{\text{image}}$.

We apply the following modifications to the standard computer-vision objectives in order to prioritize the objects over other visual information:

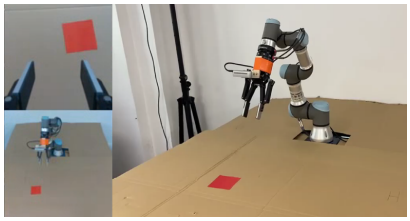


Figure 4: Real UR3 robot. The inputs to the agent are the two camera images as shown in the left.

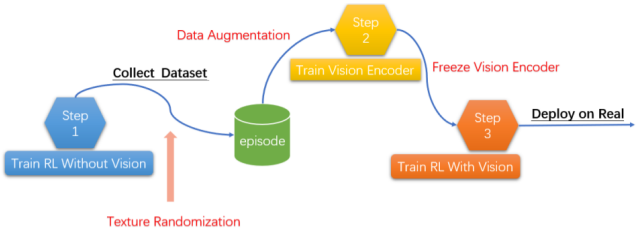


Figure 5: Sim-to-real pipeline. Two additional steps with respect to the fully simulated pipeline: texture randomization and data augmentation.

- Image Segmentation. We consider 5 different tasks for segmentation: Floor and Walls, Table, Robot, Gripper, Objects. We give a larger weight to the object class in the cross-entropy loss (10-to-1 ratio).
- Depth Prediction. We use the ground-truth segmentation image to compute a mask of the objects, in order to increase the loss values of all pixels belonging to it (by a factor of 10). Without this modification, the network might choose to ignore the objects since they represent a tiny portion of the image.
- Auto-encoding. Learning to reconstruct the full image would be redundant with the information extracted by the other two decoders. Instead, we want this decoder to further help the network focus on the objects. To do so, we use the same object-segmentation mask to reconstruct only the objects in the scene, ignoring everything else.

3.2 Zero-Shot Sim-to-Real transfer

Other than learning how to solve the task, sim-to-real transfer presents an additional challenge: learning how to generalize to a new domain. By having a two-step process, we can alleviate the burden of the RL algorithm and rely instead on the more stable supervised-learning training.

As shown in Figure 5, the sim-to-real pipeline is very similar to the one discussed so far. Two additional steps are required. First, we apply texture randomisation to generate the dataset to train the encoder. In our case, as shown in Figure 3, we collected textures from the real target environment (Figure 4) under different lighting and camera conditions. If the transfer domain is unknown, a much more varied dataset would be required for the final agent to be able to generalize.

Additionally, we use standard computer-vision data-augmentation techniques during the training of the encoder, including classic visual augmentations (hue, contrast, blur, saturation...) as well as small image translations to account for any differences in position between the simulated and real-world cameras.

4 Experimental Results

Tasks. We evaluate our method on three simulated tasks for a 6-degrees-of-freedom robot manipulator: reaching a ball, pushing a button, and sliding a block to a target square. The target objects (ball, button, block and square) appear randomly at the beginning of each episode within the reach of the robot: the ball appears anywhere in the 3D space, the button, block and square are bound to the tabletop. The initial position of the target object changes after each episode, but it does not move during an episode. The initial state of the robot is always the same, close to an upright position. An episode ends once the robot has completed the task, or after expiration (20 time-steps for the sliding task, and 100 for the other two). The reward is fully sparse, and is equal to $+100$ if the robot solves the task and 0 otherwise. All tasks belong to the benchmark and learning environment RLBench [15], which is built on top of CoppeliaSim [16]. The backend physics engine is the Bullet physics library [17].

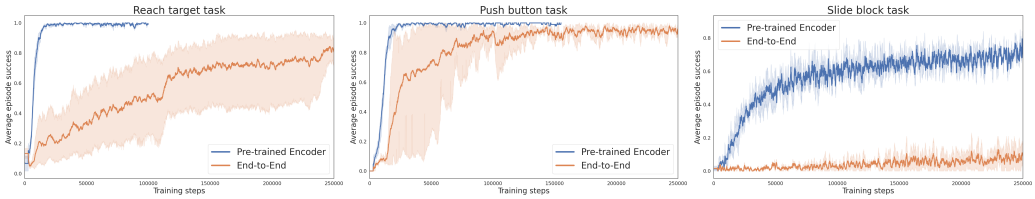


Figure 6: Learning curves for the three tasks in simulation. All the results are smoothed with a rolling window of 100 episodes, and the standard error is computed on three random seeds.

Action space. For the reaching and pushing tasks, the robot receives 6-dimensional joint speed commands. For the sliding task, the robot receives an XYZ end-effector position, while the gripper orientation remains fixed. For all tasks we use an additional dimension to control the gripper state (open or closed).

Observation space and Policy architecture. The encoder receives 64x64x6 images (two cameras) and outputs a 64-dimensional vector. This vector is concatenated to a 20-dimensional vector containing the robot propioception (joint positions, joint speeds, gripper position and orientation, gripper state). This 84-dimensional vector is then fed into an MLP that outputs the action.

4.1 Comparison to End-to-End RL

Figures 6 show that learning with a pre-trained encoder is not only much more sample-efficient, but the final performance is also better.

For the reaching and pushing tasks, the agent with a pre-trained encoder is the only one to reach a final performance of close to a 100%, while also being faster than the end-to-end agent. Suprisingly, for the pushing task, an agent learning from the full state of the environment (i.e. object coordinates) rather than image inputs, is only able to reach a final performance of around 95% accuracy. One possible explanation is that knowing the position of the button doesn't really help with the actual pushing down (the gripper would sometimes get stuck in the sides of the button and the agent is unable to recover). In the case of the more complicated sliding task, the end-to-end baseline is unable to make any significant progress, while the agent with the pre-trained encoder solves the task consistently, reaching around 75% accuracy after 200.000 steps. One possible explanation for the struggles of the end-to-end baseline on this task, is that the wrist camera is not as useful here as in the other two tasks, as the target is almost never in frame, and the block is only in frame when already making contact with it.

4.2 Sim-to-Real qualitative results

As shown in Figure 4, the task we try to tackle in the real world is a simpler version of the reaching task, with a 2D target on the table. We only provide qualitative results for this experiment. As shown in this video, a zero-shot transfer results in a policy that is able to consistently solve the task in the real-world setup, and is even able to adapt mid-episode if the target is displaced.

5 Future work

Future work will include evaluation on more tasks, namely pick-and-place or similar tasks that involve grasping, as well as a quantitative evaluation of the sim-to-real transfer. Ideally, all of the tasks will also be tested in the real robot. Finally, we would also like to compare our method to stronger end-to-end baselines, such as SAC-AE [18] or CURL [2]. Since the training process of the encoder is generic and task-agnostic, one possible direction for future research would be to test it in a multi-task setting, with a single pre-trained encoder for a single multi-task agent.

References

[1] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://openreview.net/forum?id=HJjvxl-Cb.

[2] M. Laskin, A. Srinivas, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119*, 2020. arXiv:2004.04136.

[3] N. Hansen, R. Jangir, Y. Sun, G. Alenyà, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[4] M. Schwarzer, N. Rajkumar, M. Noukhovitch, A. Anand, L. Charlin, R. D. Hjelm, P. Bachman, and A. C. Courville. Pretraining representations for data-efficient reinforcement learning. *Advances in Neural Information Processing Systems*, 34:12686–12699, 2021.

[5] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. In *International Conference on Machine Learning*, pages 2117–2126. PMLR, 2018.

[6] A. X. Lee, A. Nagabandi, P. Abbeel, and S. Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *Advances in Neural Information Processing Systems*, 33:741–752, 2020.

[7] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[8] P. Florence, L. Manuelli, and R. Tedrake. Self-supervised correspondence in visuomotor policy learning. *IEEE Robotics and Automation Letters*, 5(2):492–499, 2019.

[9] B. Chen*, A. Sax*, F. Lewis, S. Savarese, A. Zamir, J. Malik, and L. Pinto. Robust policies via mid-level visual representations: An experimental study in manipulation and navigation. In *4th Annual Conference on Robot Learning, CoRL 2020*, Proceedings of Machine Learning Research. PMLR, 2020. URL https://arxiv.org/abs/2011.06698.

[10] M. Khan, P. Srivatsa, A. Rane, S. Chenniappa, R. Anand, S. Ozair, and P. Maes. Pretrained encoders are all you need. *arXiv preprint arXiv:2106.05139*, 2021.

[11] F. Sadeghi and S. Levine. rl: Real single-image flight without a single real image. 2016. 2.

[12] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.

[13] S. James, P. Wohlhart, M. Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, R. Hadsell, and K. Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12627–12637, 2019.

[14] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[15] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[16] E. Rohmer, S. P. Singh, and M. Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[17] E. Coumans. Bullet physics simulation. In *ACM SIGGRAPH 2015 Courses*, page 1. 2015.

[18] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 10674–10681, 2021.