Self-Supervised Policy Adaptation during Deployment

Nicklas Hansen
Technical University of Denmark

Rishabh Jangir
UC San Diego

Yu Sun
UC Berkeley

Guillem Alenyà
IRI, CSIC-UPC

Pieter Abbeel
UC Berkeley

Alexei A. Efros
UC Berkeley

Lerrel Pinto
NYU

Xiaolong Wang
UC San Diego

Abstract

In most real world scenarios, a policy trained by reinforcement learning in one environment needs to be deployed in another, potentially quite different environment. However, generalization across different environments is known to be hard. A natural solution would be to keep training after deployment in the new environment, but this cannot be done if the new environment offers no reward signal. Our work explores the use of self-supervision to allow the policy to continue training after deployment without using any rewards. While previous methods explicitly anticipate changes in the new environment, we assume no prior knowledge of those changes yet still obtain significant improvements. Empirical evaluations are performed on diverse simulation environments from DeepMind Control suite and ViZDoom, as well as *real* robotic manipulation tasks in continuously changing environments, taking observations from an uncalibrated camera.[1]

1 Introduction

Deep reinforcement learning (RL) has achieved considerable success when combined with convolutional neural networks for deriving actions from image pixels [Mnih et al., 2013, Levine et al., 2016, Nair et al., 2018, Yan et al., 2020, Andrychowicz et al., 2020]. However, one significant challenge for real-world deployment of vision-based RL remains: a policy trained in one environment might not generalize to other new environments not seen during training. Already hard for RL alone, the challenge is exacerbated when a policy faces high-dimensional visual inputs.

A well explored class of solutions is to learn robust policies that are simply invariant to changes in the environment [Rajeswaran et al., 2016, Tobin et al., 2017, Sadeghi and Levine, 2016, Pinto et al., 2017b, Lee et al., 2019]. For example, domain randomization [Tobin et al., 2017, Peng et al., 2018, Pinto et al., 2017a, Yang et al., 2019] applies data augmentation in a simulated environment to train a single robust policy, with the hope that the augmented environment covers enough factors of variation in the test environment. However, this hope may be difficult to realize when the test environment is truly unknown. With too much randomization, training a policy that can simultaneously fit numerous augmented environments requires much larger model and sample complexity. With too little randomization, the actual changes in the test environment might not be covered, and domain randomization may do more harm than good since the randomized factors are now irrelevant. Both phenomena have been observed in our experiments. In all cases, this class of solutions requires human experts to anticipate the changes before the test environment is seen. This cannot scale as more test environments are added with more diverse changes.

Instead of learning a robust policy *invariant* to all possible environmental changes, we argue that it is better for a policy to keep learning during deployment and *adapt* to its actual new environment. A

[1] Project page with code: `https://nicklashansen.github.io/PAD/`

NeurIPS 2020 3rd Robot Learning Workshop: Grounding Machine Learning Development in the Real World.

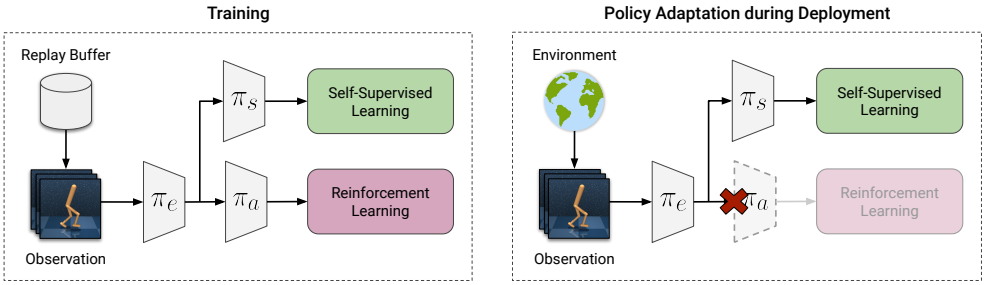


Figure 1. **Left**: Training before deployment. Observations are sampled from a replay buffer for off-policy methods and are collected during roll-outs for on-policy methods. We optimize the RL and self-supervised objectives jointly. **Right**: Policy adaptation during deployment. Observations are collected from the test environment online, and we optimize only the self-supervised objective.

naive way to implement this in RL is to fine-tune the policy in the new environment using rewards as supervision [Rusu et al., 2016]. However, while it is relatively easy to craft a dense reward function during training [Gu et al., 2017, Pinto and Gupta, 2016], during deployment it is often impractical.

In this paper, we tackle an alternative problem setting in vision-based RL: adapting a pre-trained policy to an unknown environment without any reward. We do this by introducing self-supervision to obtain "free" training signal during deployment. Standard self-supervised learning employs auxiliary tasks designed to automatically create training labels using only the input data. Inspired by this, our policy is jointly trained with two objectives: a standard RL objective and, *additionally*, a self-supervised objective applied on an intermediate representation of the policy network. During training, both objectives are active, maximizing expected reward and simultaneously constraining the intermediate representation through self-supervision. During testing / deployment, only the self-supervised objective (on the raw observational data) remains active, forcing the intermediate representation to adapt to the new environment.

2 Method

We now describe our proposed Policy Adaptation during Deployment (PAD) approach, which can be implemented on top of any policy network and standard RL algorithm, both on-policy and off-policy.

Network architecture. We design the network architecture to allow the policy and the self-supervised prediction to share features. For the collection of parameters θ of a given policy network π, we split it sequentially into $\theta = (\theta_e, \theta_a)$, where θ_e collects the parameters of the feature extractor, and θ_a is the head that outputs a distribution over actions. We define networks π_e with parameters θ_e and π_a with parameters θ_a such that $\pi(\mathbf{s}; \theta) = \pi_a(\pi_e(\mathbf{s}))$, where $\mathbf{s}$ represents an image observation. The goal of our method is to update π_e at test-time using gradients from a self-supervised task, such that π_e (and consequently π_θ) can generalize. Let π_s with parameters θ_s be the self-supervised prediction head and its collection of parameters, and the input to π_s be the output of π_e (as illustrated in Figure 1). In this work, the self-supervised task is inverse dynamics prediction for continuous control, and rotation prediction for navigation. We ablate the choice of self-supervised task in appendix E.

Inverse dynamics prediction and rotation prediction. At each time step, we always observe a transition sequence in the form of $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$, during both training and testing. Naturally, self-supervision can be derived from taking parts of the sequence and predicting the rest. An inverse dynamics model takes the states before and after transition, and predicts the action in between. In this work, the inverse dynamics model π_s operates on the feature space extracted by π_e. We can write the inverse dynamics prediction objective formally as

$$L(\theta_s, \theta_e) = \ell\big(\mathbf{a}_t,\ \pi_s\left(\pi_e(\mathbf{s}_t), \pi_e(\mathbf{s}_{t+1})\right)\big)\,. \tag{1}$$

where ℓ is the mean squared error for continuous actions and cross-entropy for discrete actions. As an alternative self-supervised task, we use rotation prediction [Gidaris et al., 2018]. We rotate an image by one of 0, 90, 180 and 270 degrees as input to the network, and cast this as a four-way classification problem to determine which one of these four ways the image has been rotated. This task is shown to be effective for learning representations for object configuration and scene structure, which is beneficial for visual recognition [Hendrycks et al., 2019, Doersch and Zisserman, 2017].

Table 1. Cumulative reward in DMC environments with randomized colors, mean and std. dev. for 10 seeds. Best method on each task is in bold and brown compares +IDM with and without PAD.

					10x episode length	
Random colors	SAC	+DR	+IDM	+IDM (PAD)	+IDM	+IDM (PAD)
Walker, walk	414±74	**594±104**	406±29	468±47	3830±547	**5505±592**
Walker, stand	719±74	715±96	743±37	**797±46**	7832±209	**8566±121**
Cartpole, swingup	592±50	**647±48**	585±73	630±63	6528±539	**7093±592**
Cartpole, balance	857±60	**867±37**	835±40	848±29	**7746±526**	7670±293
Ball in cup, catch	411±183	470±252	471±75	**563±50**	–	–
Finger, spin	626±163	465±314	757±62	**803±72**	7249±642	**7496±655**
Finger, turn_easy	270±43	167±26	283±51	**304±46**	–	–
Cheetah, run	154±41	145±29	121±38	**159±28**	1117±530	**1208±487**
Reacher, easy	163±45	105±37	201±32	**214±44**	1788±441	**2152±506**

Training and testing. Before deployment of the policy, we optimize both the RL task and the self-supervised auxiliary task $\min_{\theta_a,\theta_s,\theta_e} J(\theta_a,\theta_e) + \alpha L(\theta_s,\theta_e)$, where $\alpha > 0$ is a trade-off hyper-parameter. During deployment, we cannot optimize J anymore since the reward is unavailable, but we can still optimize L to update both θ_s and θ_e. As we obtain new images from the stream of visual inputs in the environment, θ keeps being updated until the episode ends. This corresponds to, for each iteration $t = 1...T$:

$$\mathbf{s}_t \sim p(\mathbf{s}_t|\mathbf{a}_{t-1}, \mathbf{s}_{t-1}) \tag{2}$$

$$\theta_s(t) = \theta_s(t-1) - \nabla_{\theta_s} L(\mathbf{s}_t; \theta_s(t-1), \theta_e(t-1)) \tag{3}$$

$$\theta_e(t) = \theta_e(t-1) - \nabla_{\theta_e} L(\mathbf{s}_t; \theta_s(t-1), \theta_e(t-1)) \tag{4}$$

$$\mathbf{a}_t = \pi(\mathbf{s}_t; \theta(t)) \quad \text{with} \quad \theta(t) = (\theta_e(t), \theta_a), \tag{5}$$

where $\theta_s(0) = \theta_s, \theta_e(0) = \theta_e$, $\mathbf{s}_0$ is the initial condition given by the environment, $\mathbf{a}_0 = \pi_\theta(\mathbf{s}_0)$, p is the unknown environment transition, and L is the self-supervised objective as previously introduced.

3 Experiments

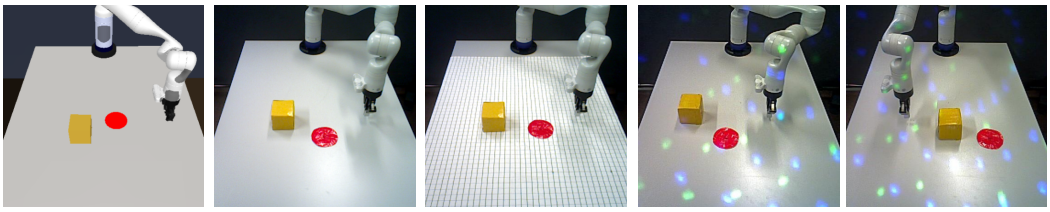
In this work, we investigate how well an agent trained in one environment (denoted the *training environment*) generalizes to *unseen*, visually diverse test environments, with only access to image observations during training and evaluation. During deployment, agents have no access to reward signals and are expected to generalize without trials nor prior knowledge about the test environments. We evaluate our method (PAD) and baselines extensively on a variety of tasks from DeepMind Control (DMC) suite [Tassa et al., 2018], CRLMaze [Lomonaco et al., 2019], as well as robotic manipulation tasks on a real robot, operating solely from an uncalibrated camera.

Network details. For DMC and robotic manipulation tasks we implement PAD on top of Soft Actor-Critic (SAC) [Haarnoja et al., 2018], and adopt both network architecture and hyper-parameters from Yarats et al. [2019] with minor modifications. For CRLMaze, we use Advantage Actor-Critic (A2C) [Mnih et al., 2016] as base algorithm and apply the same architecture as for the other experiments. Observations are colored frames of size 100×100. See appendix H for details.

Baselines. We compare PAD to the following baselines: (i) base algorithm with no changes (*SAC/A2C*); (ii) base algorithm trained with domain randomization (*+DR*); and (iii) joint training with auxiliary task (*+IDM/+Rot* for IDM and rotation prediction, respectively), but without PAD. DR uses less randomization than the test environments as we find it to not converge otherwise.

3.1 DeepMind Control

DeepMind Control (DMC) [Tassa et al., 2018] is a collection of continuous control tasks where agents only observe raw pixels. We experiment with 9 tasks from DMC and measure generalization to environments with randomized colors, as robustness to subtle changes such as color is essential to real-world deployment of RL policies. We implement PAD on top of SAC and use an Inverse Dynamics Model (IDM) for self-supervision, as we find that learning a model of the motors works well for motor control. Results are shown in Table 1. We find PAD to improve generalization *in all tasks considered*, outperforming SAC trained with domain randomization in **6** out of **9** tasks. To examine the long-term stability of PAD, we further evaluate on 10x episode lengths and summarize the results in the last two columns of Table 1 (goal-oriented tasks excluded). While we do not



(a) Simulation. *(b)* Default transfer. *(c)* Table cloth. *(d)* Disco lights.

Figure 2. Samples from the *push* robotic manipulation task. The task is to push the yellow cube to the location of the red disc. Agents are trained in setting (a) and evaluated in settings (b-d).

explicitly prevent the embedding from drifting away from the RL task, we find empirically that PAD does not degrade the performance of the policy, even over long horizons, and when PAD does *not* improve, we find it to hurt minimally. We conjecture this is because we are not learning a new task, but simply continue to optimize the same (self-supervised) objective as during joint training, where both two tasks are compatible.

3.2 CRLMaze

CRLMaze [Lomonaco et al., 2019] is a 3D navigation task for ViZDoom [Wydmuch et al., 2018]. We implement PAD on top of A2C [Mnih et al., 2016] and use rotation prediction as self-supervised task. Results are shown in Table 2. PAD improves generalization *in all considered test environments*, outperforming both A2C and domain randomization by a large margin. Domain randomization performs consistently across all environments but is less successful overall. We ablate the choice of self-supervision in appendix E and find that rotation prediction is more suitable for tasks that require scene understanding, whereas IDM is useful for tasks that require motor control. We leave it to future work to automate the process of selecting appropriate auxiliary tasks.

Table 2. **Top:** Cumulative reward of PAD and baselines in CRLMaze environments, mean and std. error of 10 seeds. **Bottom:** Success rate of PAD and baselines deployed on a real robotic arm. Best method on each task is in bold.

CRLMaze	A2C	+DR	+Rot	+Rot (PAD)
Walls	$-380_{\pm 145}$	$-260_{\pm 137}$	$-206_{\pm 166}$	$\mathbf{-74_{\pm 116}}$
Floor	$-320_{\pm 167}$	$-438_{\pm 59}$	$-294_{\pm 123}$	$\mathbf{-209_{\pm 94}}$
Ceiling	$-171_{\pm 175}$	$-400_{\pm 74}$	$128_{\pm 196}$	$\mathbf{281_{\pm 83}}$
Lights	$-30_{\pm 213}$	$-310_{\pm 106}$	$-84_{\pm 53}$	$\mathbf{312_{\pm 104}}$
Real robot	**SAC**	**+DR**	**+IDM**	**+IDM (PAD)**
Reach (default)	100%	100%	100%	100%
Reach (cloth)	48%	**80%**	56%	**80%**
Reach (disco)	72%	76%	88%	**92%**
Push (default)	88%	88%	92%	**100%**
Push (cloth)	60%	64%	64%	**88%**
Push (disco)	60%	68%	72%	**84%**

3.3 Robotic manipulation tasks

We deploy our method and baselines on a real Kinova Gen3 robot, and evaluate on two manipulation tasks: (i) *reach*, a task in which the robot reaches for a goal marked by a red disc; and (ii) *push*, a task in which the robot pushes a cube to the location of the red disc. Agents operate purely from pixel observations with *no access to state information*. During deployment, we make no effort to calibrate camera, lighting, or dimensions, and policies are expected to generalize with no prior knowledge of the test environment. Samples from the *push* task are shown in Figure 2, and samples from *reach* are shown in appendix G. We implement PAD on top of SAC [Haarnoja et al., 2018] and use an IDM for self-supervision. Agents are trained in simulation with dense rewards and randomized initial configurations of arm, goal, and box, and we measure generalization to novel environments in the real world.

We summarize the results in Table 2. While all methods transfer successfully to *reach (default)*, we observe PAD to improve generalization in all settings in which the baselines show sub-optimal performance. We find PAD to be especially powerful for the *push* task that involves dynamics, improving by as much as **24%** in *push (cloth)*, which suggests that PAD can be more suitable in challenging tasks like *push* in unseen and changing environments.

References

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.

C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2051–2060, 2017.

S. Gidaris, P. Singh, and N. Komodakis. Unsupervised representation learning by predicting image rotations, 2018.

S. Gu, E. Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pages 3389–3396. IEEE, 2017.

T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications, 2018.

D. Hendrycks, M. Mazeika, S. Kadavath, and D. Song. Using self-supervised learning can improve model robustness and uncertainty. *ArXiv*, abs/1906.12340, 2019.

I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. 2020.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

K. Lee, K. Lee, J. Shin, and H. Lee. A simple randomization technique for generalization in deep reinforcement learning. *ArXiv*, abs/1910.05396, 2019.

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

V. Lomonaco, K. Desai, E. Culurciello, and D. Maltoni. Continual reinforcement learning in 3d non-stationary environments. *arXiv preprint arXiv:1905.10112*, 2019.

V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pages 9191–9200, 2018.

C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning, 2018.

X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

L. Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 3406–3413. IEEE, 2016.

L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017a.

L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 2817–2826. JMLR. org, 2017b.

A. Rajeswaran, S. Ghotra, B. Ravindran, and S. Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016.

A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020.

C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9, 2015.

Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. de Las Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, T. Lillicrap, and M. Riedmiller. DeepMind control suite. Technical report, DeepMind, Jan. 2018.

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. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep 2017.

M. Wydmuch, M. Kempka, and W. Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018.

W. Yan, A. Vangipuram, P. Abbeel, and L. Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020.

J. Yang, B. Petersen, H. Zha, and D. Faissol. Single episode policy transfer in reinforcement learning, 2019.

D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images, 2019.

A Samples from DeepMind Control

We evaluate our method (PAD) and baselines extensively on continuous control tasks from DeepMind Control (DMC) suite [Tassa et al., 2018] as well as the CRLMaze [Lomonaco et al., 2019] navigation task, and experiment with both stationary (colors, objects, textures, lighting) and non-stationary (videos) environment changes. Samples from both DMC and CRLMaze are shown in Figure 3.

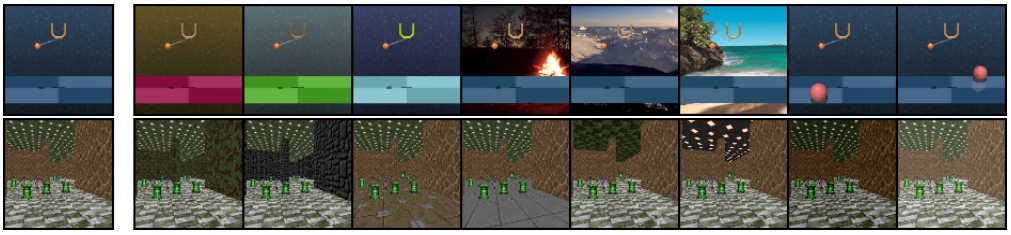


Figure 3. **Left:** Training environments of DMC (top) and CRLMaze (bottom). **Right:** Test environments of DMC (top) and CRLMaze (bottom). Changes to DMC include: randomized colors, video backgrounds, and distracting objects. Changes to CRLMaze include textures and lighting.

B Generalization to non-stationary changes and scene contents

To investigate whether PAD can adapt in non-stationary environments, we evaluate generalization to diverse video backgrounds (refer to Figure 3). We find PAD to outperform all baselines on **7** out of **8** tasks, as shown in Table 3, by as much as **104%** over domain randomization on *Finger, spin*. Domain randomization generalizes comparably worse to videos, which we conjecture is not because the environments are non-stationary, but rather because the image statistics of videos are not covered by its training domain of randomized colors. In fact, domain randomization is outperformed by the vanilla SAC in most tasks with video backgrounds, which is in line with the findings of Packer et al. [2018]. We further hypothesize that: (i) an agent trained with an IDM is comparably less distracted by scene content since objects uncorrelated to actions yield no predictive power; and (ii) that PAD can adapt to unexpected objects in the scene. We test these hypotheses by measuring robustness to colored shapes at a variety of positions in both the foreground and background of the scene (no physical interaction). Results are summarized in Table 3. PAD outperforms all baselines in **3** out of **5** tasks, with a relative improvement of **208%** over SAC on *Ball in cup, catch*. In the two cartpole tasks in which PAD does not improve, all methods are already relatively unaffected by the distractors.

Table 3. Cumulative reward in test environments with video backgrounds (top) and distracting objects (bottom), mean and std. dev. for 10 seeds. Best method on each task is in bold and brown compares SAC+IDM with and without PAD.

Video backgrounds	SAC	+DR	+IDM	+IDM (PAD)
Walker, walk	616±80	655±55	694±85	**717±79**
Walker, stand	899±53	869±60	902±51	**935±20**
Cartpole, swingup	375±90	485±67	487±90	**521±76**
Cartpole, balance	693±109	**766±92**	691±76	687±58
Ball in cup, catch	393±175	271±189	362±69	**436±55**
Finger, spin	447±102	338±207	605±61	**691±80**
Finger, turn_easy	355±108	223±91	355±110	**362±101**
Cheetah, run	194±30	150±34	164 ±42	**206±34**
Distracting objects	**SAC**	**+DR**	**+IDM**	**+IDM (PAD)**
Cartpole, swingup	**815±60**	809±24	776±58	771±64
Cartpole, balance	**969±20**	938±35	964±26	960±29
Ball in cup, catch	177±111	331±189	482±128	**545±173**
Finger, spin	652±184	564±288	836±62	**867±72**
Finger, turn_easy	302±68	165±12	326±101	**347±48**

C Performance on the training environment

Historically, agents have commonly been trained and evaluated in the same environment when benchmarking RL algorithms exclusively in simulation. Although such an evaluation procedure does not consider generalization, it is still a useful metric for comparison of sample efficiency and stability of algorithms. For completeness, we also evaluate our method and baselines in this setting on both DMC and CRLMaze. DMC results are reported in Table 4 and results on the CRLMaze environment are shown in Table 5. In this setting, we also compare to an additional baseline on DMC: a blind SAC agent that operates purely on its previous actions. The performance of a blind agent indicates to which degree a given task benefits from visual information. We find that, while PAD improves generalization to novel environments, performance is virtually unchanged when evaluated on the same environment as in training. We conjecture that this is because the algorithm already is adapted to the training environment and any continued training on the same data distribution thus has little influence. We further emphasize that, even when evaluated on the training environment, PAD still outperforms baselines on most tasks. For example, we observe a **15%** relative improvement over SAC on the *Finger, spin* task. We hypothesize that this gain in performance is because the self-supervised objective improves learning by constraining the intermediate representation of policies. A blind agent is no better than random on this particular task, which would suggest that agents benefit substantially from visual information in *Finger, spin*. Therefore, learning a good intermediate representation of that information is highly beneficial to the RL objective, which we find PAD to facilitate through its self-supervised learning framework. Likewise, the SAC baseline only achieves a 51% improvement over the blind agent on *Cartpole, balance*, which indicates that extracting visual information from observations is not as crucial on this task. Consequently, both PAD and baselines achieve similar performance on this task.

Table 4. Cumulative reward on the training environment for each of the 9 tasks considered in DMC, mean and std. dev. for 10 seeds. Best method on each task is in bold and brown compares +IDM with and without PAD. It is shown that PAD hurts minimally when the environment is unchanged.

Training env.	Blind	SAC	+DR	+IDM	+IDM (PAD)
Walker, walk	$235_{\pm 17}$	$847_{\pm 71}$	$756_{\pm 71}$	**$911_{\pm 24}$**	$895_{\pm 28}$
Walker, stand	$388_{\pm 10}$	$959_{\pm 11}$	$928_{\pm 36}$	**$966_{\pm 8}$**	$956_{\pm 20}$
Cartpole, swingup	$132_{\pm 41}$	**$850_{\pm 28}$**	$807_{\pm 36}$	$849_{\pm 30}$	$845_{\pm 34}$
Cartpole, balance	$646_{\pm 131}$	$978_{\pm 22}$	$971_{\pm 30}$	**$982_{\pm 20}$**	$979_{\pm 21}$
Ball in cup, catch	$150_{\pm 96}$	$725_{\pm 355}$	$469_{\pm 339}$	**$919_{\pm 118}$**	$910_{\pm 129}$
Finger, spin	$3_{\pm 2}$	$809_{\pm 138}$	$686_{\pm 295}$	**$928_{\pm 45}$**	$927_{\pm 45}$
Finger, turn_easy	$172_{\pm 27}$	**$462_{\pm 146}$**	$243_{\pm 124}$	**$462_{\pm 152}$**	$455_{\pm 160}$
Cheetah, run	$264_{\pm 75}$	**$387_{\pm 74}$**	$195_{\pm 46}$	$384_{\pm 88}$	$380_{\pm 91}$
Reacher, easy	$107_{\pm 11}$	$264_{\pm 113}$	$92_{\pm 45}$	**$390_{\pm 126}$**	$365_{\pm 114}$

Table 5. Cumulative reward of PAD and baselines in the CRLMaze training environment. All methods use A2C. We report mean and std. error of 10 seeds. Best method is in bold and brown compares rotation prediction with and without PAD.

CRLMaze	Random	A2C	+DR	+IDM	+IDM (PAD)	+Rot	+Rot (PAD)
Training env.	$-868_{\pm 34}$	$371_{\pm 198}$	$-355_{\pm 93}$	$585_{\pm 246}$	$-416_{\pm 135}$	**$729_{\pm 148}$**	$681_{\pm 99}$

D Learning curves on DeepMind Control

All methods are trained until convergence (500,000 frames) on the DeepMind Control suite. While we do not consider the sample efficiency of our method and baselines in this study, we report learning curves for SAC, SAC+IDM and SAC trained with domain randomization on three tasks in Figure 4 for completeness. SAC trained with and without an IDM are similar in terms of sample efficiency and final performance, whereas domain randomization consistently displays worse sample efficiency, larger variation between seeds, and converges to sub-optimal performance in two out of the three tasks shown.

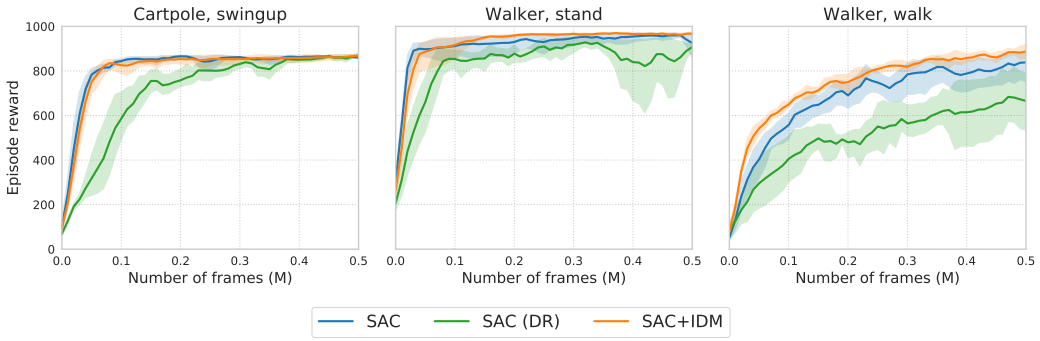


Figure 4. Learning curves for SAC, SAC trained with domain randomization (denoted *SAC (DR)* here), and SAC+IDM on three tasks from the DeepMind Control suite. Episode reward is averaged across 10 seeds and the 95% confidence intervals are visualized as shaded regions. SAC and SAC+IDM exhibit similar sample efficiency and final performance, whereas domain randomization consistently displays worse sample efficiency, larger variation between seeds, and converges to sub-optimal performance in two out of the three tasks shown.

E Choice of self-supervised task

We investigate how much the choice of self-supervised task contributes to the overall success of our method, and consider the following ablations on DMC: (i) replacing inverse dynamics with the rotation prediction task; and (ii) replacing it with the recently proposed CURL [Srinivas et al., 2020] contrastive learning algorithm for RL. As shown in Table 6, PAD improves generalization of CURL in a majority of tasks on the randomized color benchmark, and in 4 out of 9 tasks using rotation prediction. However, inverse dynamics as auxiliary task produces more consistent results and offers better generalization overall. We argue that learning an IDM produces better representations for motor control since it connects observations directly to actions, whereas CURL and rotation prediction operates purely on observations. In general, we find the improvement of PAD to be bigger in tasks that benefit significantly from visual information (see appendix C), and conjecture that selecting a self-supervised task that learns features useful to the RL task is crucial to the success of PAD. We leave it to future work to automate the process of selecting appropriate auxiliary tasks.

F Offline versus online learning

Observations that arrive sequentially are highly correlated, and we thus hypothesize that our method benefits significantly from learning online. To test this hypothesis, we run an *offline* variant of our method in which network updates are forgotten after each step. In this setting, our method can only adapt to single observations and does not benefit from learning over time. Results are shown in Table 6. We find that our method benefits substantially from online learning, but learning offline still improves generalization on select tasks.

G Additional robotic manipulation samples

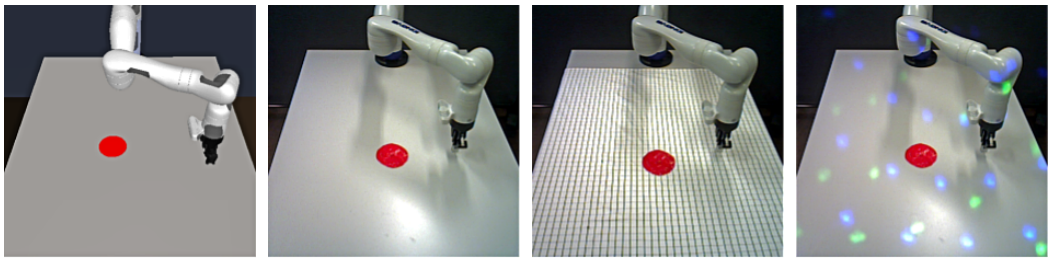
Figure 5 provides samples from the training and test environments for the *reach* robotic manipulation task. Agents are trained in simulation and deployed on a real robot. Samples from the *push* task are shown in the main paper.

H Implementation Details

In this section, we elaborate on implementation details for our experiments on DMC and CRLMaze [Lomonaco et al., 2019]. Our implementation for the robotic manipulation experiments closely follows that of DMC.

Table 6. Ablations on the randomized color domain of DMC. All methods use SAC. CURL represents RL with a contrastive learning task [Srinivas et al., 2020] and Rot represents the rotation prediction [Gidaris et al., 2018]. Offline PAD is here denoted O-PAD for brevity, whereas the default usage of PAD is in an online setting. Best method is in bold and brown compares +IDM w/ and w/o PAD.

Random colors	CURL	CURL (PAD)	Rot	Rot (PAD)	IDM	IDM (O-PAD)	IDM (PAD)
Walker, walk	445±99	**495±70**	335±7	330±30	406±29	441±16	468±47
Walker, stand	662±54	753±49	673±4	653±27	743±37	727±21	**797±46**
Cartpole, swingup	454±110	413±67	493±52	477±38	585±73	578±69	**630±63**
Cartpole, balance	782±13	763±5	710±72	734±81	835±40	796±37	**848±29**
Ball in cup, catch	231±92	332±78	291±54	314±60	471±75	490±16	**563±50**
Finger, spin	691±12	588±22	695±36	689±20	757±62	767±43	**803±72**
Finger, turn_easy	202±32	186±2	283±68	230±53	283±51	**321±10**	304±46
Cheetah, run	202±22	**211±20**	127±3	135±12	121±38	112±35	159±28
Reacher, easy	325±32	**378±62**	99±29	120±7	201±32	241±24	214±44



(a) Simulation. *(b)* Default transfer. *(c)* Table cloth. *(d)* Disco.

Figure 5. Samples from the *reach* robotic manipulation task. The task is to move the robot gripper to the location of the red disc. Agents are trained in setting (a) and evaluated in settings (b-d).

Architecture. Our network architecture is illustrated in Figure 6. Observations are stacked frames ($k = 3$) rendered at 100×100 and cropped to 84×84, i.e. inputs to the network are of dimensions $9 \times 84 \times 84$, where the first dimension indicates the channel numbers and the following ones represent spatial dimensions. The same crop is applied to all frames in a stack. The shared feature extractor π^e consists of 8 (DMC, robotic manipulation) or 6 (CRLMaze) convolutional layers and outputs features of size $32 \times 21 \times 21$ in DMC and robotic manipulation, and size $32 \times 25 \times 25$ in CRLMaze. The output from π^e is used as input to both the self-supervised head π^s and RL head π^a, both of which consist of 3 convolutional layers followed by 3 fully-connected layers. All convolutional layers use 32 filters and all fully connected layers use a hidden size of 1024, as in Yarats et al. [2019].

Learning algorithm. We use Soft Actor-Critic (SAC) [Haarnoja et al., 2018] for DMC and robotic manipulation, and Advantage Actor-Critic (A2C) for CRLMaze. Network outputs depend on the task and learning algorithm. As the action spaces of both DMC and robotic manipulation are continuous, the policy learned by SAC outputs the mean and variance of a Gaussian distribution over actions. CRLMaze has a discrete action space and the policy learned by A2C thus learns a soft-max distribution over actions. For details on the critics learned by SAC and A2C, the reader is referred to Haarnoja et al. [2018] and Mnih et al. [2016], respectively.

Hyper-parameters. When applicable, we adopt our hyper-parameters from Yarats et al. [2019] (DMC, robotic manipulation) and Lomonaco et al. [2019] (CRLMaze). For the robotic manipulation experiments, our implementation closely follows that of DMC, only differing by number of frames in an observation. We use a frame stack of $k = 3$ frames for DMC and CRLMaze, and only $k = 1$ frame for robotic manipulation. For completeness, we detail all hyper-parameters used for the DMC and CRLMaze environments in Table 7 and Table 8.

Data augmentation. Random cropping is a commonly used data augmentation used in computer vision systems [Krizhevsky et al., 2012, Szegedy et al., 2015] but has only recently gained interest as a stochastic regularization technique in the RL literature [Srinivas et al., 2020, Kostrikov et al., 2020, Laskin et al., 2020]. We adopt the random crop proposed in Srinivas et al. [2020]: crop

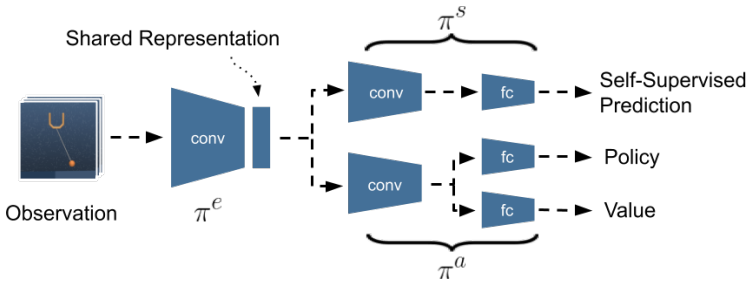


Figure 6. Network architecture for the DMC, CRLMaze, and robotic manipulation experiments. π^s and π^a uses a shared feature extractor π^e. Observations are stacks of 100×100 colored frames. Implementation of policy and value function depends on the learning algorithm.

Table 7. Hyper-parameters for the DMC tasks.

Hyperparameter	Value
Frame rendering	$3 \times 100 \times 100$
Frame after crop	$3 \times 84 \times 84$
Stacked frames	3
Action repeat	2 (finger) 8 (cartpole) 4 (otherwise)
Discount factor γ	0.99
Episode length	1,000
Learning algorithm	Soft Actor-Critic
Self-supervised task	Inverse Dynamics Model
Number of training steps	500,000
Replay buffer size	500,000
Optimizer (π^e, π^a, π^s)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Optimizer (α)	Adam ($\beta_1 = 0.5, \beta_2 = 0.999$)
Learning rate (π^e, π^a, π^s)	3e-4 (cheetah) 1e-3 (otherwise)
Learning rate (α)	1e-4
Batch size	128
Batch size (test-time)	32
π^e, π^s update freq.	2
π^e, π^s update freq. (test-time)	1

Table 8. Hyper-parameters for the CRLMaze task.

Hyperparameter	Value
Frame rendering	$3 \times 100 \times 100$
Frame after crop	$3 \times 84 \times 84$
Stacked frames	3
Action repeat	4
Discount factor γ	0.99
Episode length	1,000
Learning algorithm	Advantage Actor-Critic
Self-supervised task	Rotation Prediction
Number of training episodes	1,000 (dom. rand.) 500 (otherwise)
Number of processes	20
Optimizer	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Learning rate	1e-4
Learning rate (test-time)	1e-5
Batch size	20
Batch size (test-time)	32
π^e, π^s loss coefficient	0.5
π^e, π^s loss coefficient (test-time)	1
π^e, π^s update freq.	1
π^e, π^s update freq. (test-time)	1

rendered observations of size 100×100 to 84×84, applying the same crop to all frames in a stacked observation. This has the added benefits of regularization while still preserving spatio-temporal patterns between frames. When learning an Inverse Dynamics Model, we apply the same crop to all frames of a given observation but apply two different crops to the consecutive observations $(\mathbf{s}_t, \mathbf{s}_{t+1})$ used to predict action $\mathbf{a}_t$.

Policy Adaptation during Deployment. We evaluate our method and baselines with episodic cumulative reward of an agent trained in a single environment and tested in a collection of test environments, each with distinct changes from the training environment. We assume no reward signal at test-time and agents are expected to generalize without pre-training or resetting in the new environment. Therefore, we make updates to the policy using a self-supervised objective, and we train using observations from the environment in an online manner without memory, i.e. we make one update per step using the most-recent observation.

Empirically, we find that: (i) the random crop data augmentation used during training helps regularize learning at test-time; and (ii) our algorithm benefits from learning from a batch of randomly cropped observations rather than single observations, even when all observations in the batch are augmented copies of the most-recent observation. As such, we apply both of these techniques when performing Policy Adaptation during Deployment and use a batch size of 32. When using the policy to take actions, however, inputs to the policy are simply center-cropped.