SELF-SUPERVISED POLICY ADAPTATION DURING DEPLOYMENT

Nicklas Hansen[1,2], **Rishabh Jangir**[1,3], **Yu Sun**[4], **Guillem Alenyà**[3], **Pieter Abbeel**[4], **Alexei A Efros**[4], **Lerrel Pinto**[5], **Xiaolong Wang**[1]
[1]UC San Diego [2]Technical University of Denmark
[3]IRI, CSIC-UPC [4]UC Berkeley [5]NYU

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. Our method improves generalization in 31 out of 36 environments across various tasks and outperforms domain randomization on a majority of environments.[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 & 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 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; Kalashnikov et al., 2018; Julian et al., 2020). However, while it is relatively easy to craft a dense reward function during training (Gu et al., 2017; Pinto & Gupta, 2016), during deployment it is often impractical and may require substantial engineering efforts.

[1]Webpage and implementation: https://nicklashansen.github.io/PAD/

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 (see Section 2 for details). 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.

We perform experiments both in simulation and with a real robot. In simulation, we evaluate on two sets of environments: DeepMind Control suite (Tassa et al., 2018) and the CRLMaze ViZDoom (Lomonaco et al., 2019; Wydmuch et al., 2018) navigation task. We evaluate generalization by testing in new environments with visual changes unknown during training. Our method improves generalization in 19 out of 22 test environments across various tasks in DeepMind Control suite, and in all considered test environments on CRLMaze. Besides simulations, we also perform Sim2Real transfer on both reaching and pushing tasks with a Kinova Gen3 robot. After training in simulation, we successfully transfer and adapt policies to 6 different environments, including continuously changing disco lights, on a real robot operating solely from an uncalibrated camera. In both simulation and real experiments, our approach outperforms domain randomization in most environments.

2 Related Work

Self-supervised learning is a powerful way to learn visual representations from unlabeled data (Vincent et al., 2008; Doersch et al., 2015; Wang & Gupta, 2015; Zhang et al., 2016; Pathak et al., 2016; Noroozi & Favaro, 2016; Zhang et al., 2017; Gidaris et al., 2018). Researchers have proposed to use auxiliary data prediction tasks, such as undoing rotation (Gidaris et al., 2018), solving a jigsaw puzzle (Noroozi & Favaro, 2016), tracking (Wang et al., 2019), etc. to provide supervision in lieu of labels. In RL, the idea of learning visual representations and action at the same time has been investigated (Lange & Riedmiller, 2010; Jaderberg et al., 2016; Pathak et al., 2017; Ha & Schmidhuber, 2018; Yarats et al., 2019; Srinivas et al., 2020; Laskin et al., 2020; Yan et al., 2020). For example, Srinivas et al. (2020) use self-supervised contrastive learning techniques (Chen et al., 2020; Hénaff et al., 2019; Wu et al., 2018; He et al., 2020) to improve sample efficiency in RL by jointly training the self-supervised objective and RL objective. However, this has not been shown to generalize to unseen environments. Other works have applied self-supervision for better generalization across environments (Pathak et al., 2017; Ebert et al., 2018; Sekar et al., 2020). For example, Pathak et al. (2017) use a self-supervised prediction task to provide dense rewards for exploration in novel environments. While results on environment exploration from scratch are encouraging, how to transfer a trained policy (with extrinsic reward) to a novel environment remains unclear. Hence, these methods are not directly applicable to the proposed problem in our paper.

Generalization across different distributions is a central challenge in machine learning. In domain adaptation, target domain data is assumed to be accessible (Geirhos et al., 2018; Tzeng et al., 2017; Ganin et al., 2016; Gong et al., 2012; Long et al., 2016; Sun et al., 2019; Julian et al., 2020). For example, Tzeng et al. (2017) use adversarial learning to align the feature representations in both the source and target domain during training. Similarly, the setting of domain generalization (Ghifary et al., 2015; Li et al., 2018; Matsuura & Harada, 2019) assumes that all domains are sampled from the same meta distribution, but the same challenge remains and now becomes generalization across meta-distributions. Our work focuses instead on the setting of generalizing to truly *unseen* changes in the environment which cannot be anticipated at training time.

There have been several recent benchmarks in our setting for image recognition (Hendrycks & Dietterich, 2018; Recht et al., 2018; 2019; Shankar et al., 2019). For example, in Hendrycks & Dietterich (2018), a classifier trained on regular images is tested on corrupted images, with corruption types unknown during training; the method of Hendrycks et al. (2019) is proposed to improve robustness on this benchmark. Following similar spirit, in the context of RL, domain randomization (Tobin et al., 2017; Pinto et al., 2017a; Peng et al., 2018; Ramos et al., 2019; Yang et al., 2019; James et al., 2019) helps a policy trained in simulation to generalize to real robots. For example, Tobin et al. (2017); Sadeghi & Levine (2016) propose to render the simulation environment with random textures and train the policy on top. The learned policy is shown to generalize to real

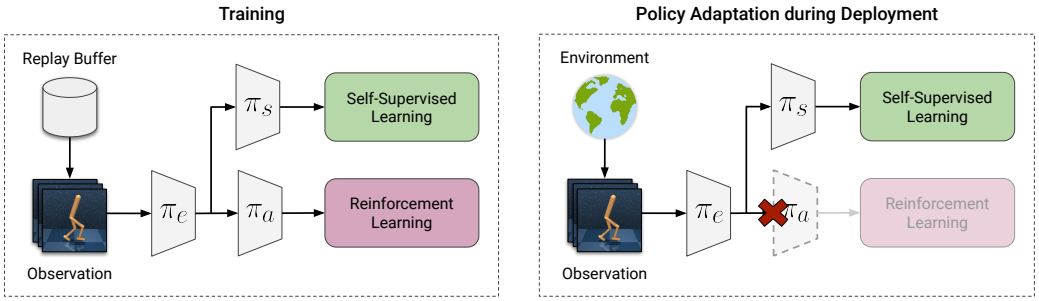


Figure 1. **Left**: Training before deployment. Observations are sampled from a replay buffer for off-policy methods and are collected during roll-outs for on-policy methods. We optimize the RL and self-supervised objectives jointly. **Right**: Policy adaptation during deployment. Observations are collected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

robot manipulation tasks. Instead of deploying a fixed policy, we train and adapt the policy to the new environment with observational data that is naturally revealed during deployment.

Test-time adaptation for deep learning is starting to be used in computer vision (Shocher et al., 2017; 2018; Bau et al., 2019; Mullapudi et al., 2019; Sun et al., 2020; Wortsman et al., 2018). For example, Shocher et al. (2018) shows that image super-resolution can be learned at test time (from scratch) simply by trying to upsample a downsampled version of the input image. Bau et al. (2019) show that adapting the prior of a generative adversarial network to the statistics of the test image improves photo manipulation tasks. Our work is closely related to the test-time training method of Sun et al. (2020), which performs joint optimization of image recognition and self-supervised learning with rotation prediction (Gidaris et al., 2018), then uses the self-supervised objective to adapt the representation of individual images during testing. Instead of image recognition, we perform test-time adaptation for RL with visual inputs in an online fashion. As the agent interacts with an environment, we keep obtaining new observational data in a stream for training the visual representations.

3 Method

In this section, we describe our proposed Policy Adaptation during Deployment (PAD) approach. It can be implemented on top of any policy network and standard RL algorithm (both on-policy and off-policy) that can be described by minimizing some RL objective $J(\theta)$ w.r.t. the collection of parameters θ using stochastic gradient descent.

3.1 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. Intuitively, one can think of π_e as a feature extractor, and π_a as a controller based on these features. 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 control, and rotation prediction for navigation.

3.2 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

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

For continuous actions, ℓ is the mean squared error between the ground truth and the model output. For discrete actions, the output is a soft-max distribution over the action space, and ℓ is the cross-

entropy loss. Empirically, we find this self-supervised task to be most effective with continuous actions, possibly because inverse dynamics prediction in a small space of discrete actions is not as challenging. Note that we predict the inverse dynamics instead of the forward dynamics, because when operating in feature space, the latter can produce trivial solutions such as the constant zero feature for every state[2]. If we instead performed prediction with forward dynamics in pixel space, the task would be extremely challenging given the large uncertainty in pixel prediction.

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 & Zisserman, 2017).

3.3 Training and Testing

Before deployment of the policy, because we have signals from both the reward and self-supervised auxiliary task, we can train with both in the fashion of multi-task learning. This corresponds to the following optimization problem during training $\min_{\theta_a,\theta_s,\theta_e} J(\theta_a,\theta_e) + \alpha L(\theta_s,\theta_e)$, where $\alpha > 0$ is a trade-off hyperparameter. 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. Empirically, we find only negligible difference with keeping θ_s fixed at test-time, so we update both since the gradients have to be computed regardless; we ablate this decision in appendix C. 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.

4 Experiments

In this work, we investigate how well an agent trained in one environment (denoted the *training environment*) generalizes to *unseen* and diverse test environments. During evaluation, agents have no access to reward signals and are expected to generalize without trials nor prior knowledge about the test environments. In simulation, we evaluate our method (PAD) and baselines extensively on continuous control tasks from DeepMind Control (DMControl) 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. We further show that PAD transfers from simulation to a real robot and successfully adapts to environmental differences during deployment in two robotic manipulation tasks. Samples from DMControl and CRLMaze environments are shown in Figure 2, and samples from the robot experiments are shown in Figure 4. Implementation is available at https://nicklashansen.github.io/PAD/.

Network details. For DMControl and the robotic manipulation tasks we implement PAD on top of Soft Actor-Critic (SAC) (Haarnoja et al., 2018), and adopt both network architecture and hyperparameters from Yarats et al. (2019), with minor modifications: the feature extractor π_e has 8 convolutional layers shared between the RL head π_a and self-supervised head π_s, and we split the network into architecturally identical heads following π_e. Each head consists of 3 convolutional layers followed by 4 fully connected layers. For CRLMaze, we use Advantage Actor-Critic (A2C) as base algorithm (Mnih et al., 2016) and apply the same architecture as for the other experiments, but implement π_e with only 6 convolutional layers. Observations are stacks of k colored frames ($k = 3$ on DMControl and CRLMaze; $k = 1$ in robotic manipulation) of size 100×100 and time-consistent random crop is applied as in Srinivas et al. (2020). During deployment, we optimize the self-supervised objective online w.r.t. θ_e, θ_s for one gradient step per time iteration. See appendix F for implementation details.

[2] A forward dynamics model operating in feature space can trivially achieve a loss of 0 by learning to map every state to a constant vector, e.g. $\mathbf{0}$. An inverse dynamics model, however, does not have such trivial solutions.

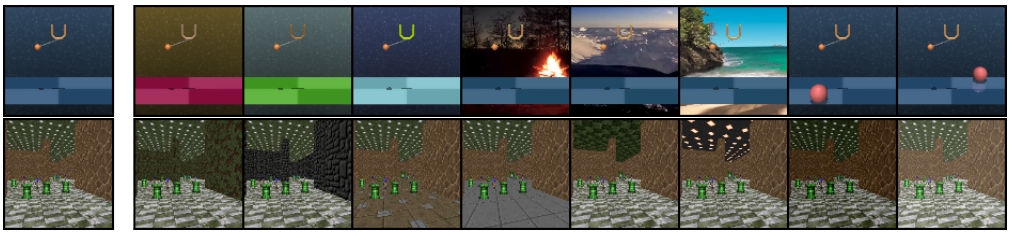


Figure 2. **Left:** Training environments of DMControl (top) and CRLMaze (bottom). **Right:** Test environments of DMControl (top) and CRLMaze (bottom). Changes to DMControl include randomized colors, video backgrounds, and distractors; changes to CRLMaze include textures and lighting.

Table 1. Episodic return in test environments with randomized colors, mean and std. dev. for 10 seeds. Best method on each task is in bold and blue compares SAC+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**

4.1 DeepMind Control

DeepMind Control (DMControl) (Tassa et al., 2018) is a collection of continuous control tasks where agents only observe raw pixels. Generalization benchmarks on DMControl represent diverse real-world tasks for motor control, and contain distracting surroundings not correlated with the reward signals.

Experimental setup. We experiment with 9 tasks from DMControl and measure generalization to four types of test environments: (i) randomized colors; (ii) natural videos as background; (iii) distracting objects placed in the scene; and (iv) the unmodified training environment. For each test environment, we evaluate methods across 10 seeds and 100 random initializations. If a given test environment is not applicable to certain tasks, e.g. if a task has no background for the video background setting, they are excluded. Tasks are selected on the basis of diversity, as well as the success of vision-based RL in prior work (Yarats et al., 2019; Srinivas et al., 2020; Laskin et al., 2020; Kostrikov et al., 2020). 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 dynamics works well for motor control. For completeness, we ablate the choice of self-supervision. Learning curves are provided in appendix B. We compare our method to the following baselines: (i) SAC with no changes (denoted *SAC*); (ii) SAC trained with domain randomization on a fixed set of 100 colors (denoted *+DR*); and (iii) SAC trained jointly with an IDM but without PAD (denoted *+IDM*). Our method using an IDM with PAD is denoted by *+IDM (PAD)*. For domain randomization, colors are sampled from the *same distribution* as in evaluation, but with lower variance, as we find that training directly on the test distribution does not converge.

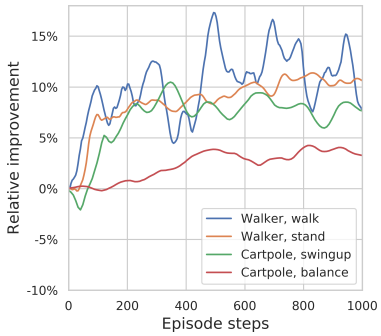


Figure 3. Relative improvement in instantaneous reward over time for PAD on the random color env.

Random perturbation of color. Robustness to subtle changes such as color is essential to real-world deployment of RL policies. We evaluate generalization on a fixed set of 100 colors of foreground, background and the agent itself, and report the results in Table 1 (first 4 columns). We find PAD to improve generalization *in all tasks considered*, outperforming SAC trained with domain

randomization in **6** out of **9** tasks. Surprisingly, despite a substantial overlap between training and test domains of domain randomization, it generalizes no better than vanilla SAC on a majority of tasks.

Long-term stability. We find the relative improvement of PAD to improve over time, as shown in Figure 3. To examine the long-term stability of PAD, we further evaluate on 10x episode lengths and summarize the results in the last two columns in Table 1 (goal-oriented tasks excluded). While we do not 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. In this setting, PAD still improves generalization in **6** out of **7** tasks, and thus naturally extends beyond episodic deployment. For completeness, we also evaluate methods in the environment in which they were trained, and report the results in appendix A. We find that, while PAD improves generalization to novel environments, performance is virtually unchanged on the training environment. We conjecture this is because the self-supervised task is already fully learned and any continued training on the same data distribution thus has little impact.

Non-stationary environments. To investigate whether PAD can adapt in non-stationary environments, we evaluate generalization to diverse video backgrounds (refer to Figure 2). We find PAD to outperform all baselines on **7** out of **8** tasks, as shown in Table 2, 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).

Table 2. Episodic return 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 blue 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**

Scene content. We 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 2. 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.

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: (i) replacing inverse dynamics with the rotation prediction task described in Section 3.2; and (ii) replacing it with the recently proposed CURL (Srinivas et al., 2020) contrastive learning algorithm for RL. As shown in Table 3, 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 A), and conjecture that selecting a self-supervised task that learns features useful to the RL task is crucial to the success of PAD, which we discuss further in Section 4.2.

Table 3. 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 blue 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

Table 4. Episodic return of PAD and baselines in CRLMaze environments. PAD improves generalization *in all considered environments* and outperforms both A2C and domain randomization by a large margin. All methods use A2C. We report mean and std. error of 10 seeds. Best method in each environment is in bold and blue compares rotation prediction with and without PAD.

CRLMaze	Random	A2C	+DR	+IDM	+IDM (PAD)	+Rot	+Rot (PAD)
Walls	−870±30	−380±145	−260±137	−302±150	−428±135	−206±166	**−74±116**
Floor	−868±23	−320±167	−438±59	**−47±198**	−530±106	−294±123	−209±94
Ceiling	−872±30	−171±175	−400±74	166±215	−508±104	128±196	**281±83**
Lights	−900±29	−30±213	−310±106	239±270	−460±114	−84±53	**312±104**

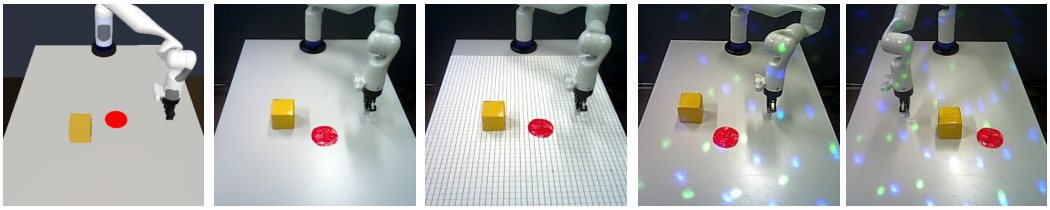
Offline versus online learning. Observations that arrive sequentially are highly correlated, and we thus hypothesize that our method benefits significantly from learning online. To test this hypothesis, we run an *offline* variant of our method in which network updates are forgotten after each step. In this setting, our method can only adapt to single observations and does not benefit from learning over time. Results are shown in Table 3. We find that our method benefits substantially from online learning, but learning offline still improves generalization on select tasks.

4.2 CRLMaze

CRLMaze (Lomonaco et al., 2019) is a time-constrained, discrete-action 3D navigation task for ViZDoom (Wydmuch et al., 2018), in which an agent is to navigate a maze and collect objects. There is a positive reward associated with green columns, and a negative reward for lanterns as well as for living. Readers are referred to the respective papers for details on the task and environment.

Experimental setup. We train agents on a single environment and measure generalization to environments with novel textures for walls, floor, and ceiling, as well as lighting, as shown in Figure 2. We implement PAD on top of A2C (Mnih et al., 2016) and use rotation prediction (see Section 3.2) as self-supervised task. Learning to navigate novel scenes requires a generalized scene understanding, and we find that rotation prediction facilitates that more so than an IDM. We compare to the following baselines: (i) a random agent (denoted *Random*); (ii) A2C with no changes (denoted *A2C*); (iii) A2C trained with domain randomization (denoted *+DR*); (iv) A2C with an IDM as auxiliary task (denoted *+IDM*); and (v) A2C with rotation prediction as auxiliary task (denoted *+Rot*). We denote Rot with PAD as *+Rot (PAD)*. Domain randomization uses 56 combinations of diverse textures, partially overlapping with the test distribution, and we find it necessary to train domain randomization for twice as many episodes in order to converge. We closely follow the evaluation procedure of (Lomonaco et al., 2019) and evaluate methods across 20 starting positions and 10 random seeds.

Results. We report performance on the CRLMaze environments in Table 4. 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 further examine the importance of selecting appropriate auxiliary tasks by a simple ablation: replacing rotation prediction with an IDM for the navigation task. We conjecture that, while an auxiliary task can enforce structure in the learned representations, its features (and consequently gradients) need to be sufficiently correlated with the primary RL task for PAD to be



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

Figure 4. 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).

successful during deployment. While PAD with rotation prediction improves generalization across all test environments considered, IDM does not, which suggests 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.

4.3 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. Both tasks use an XY action space, where the Z position of the actuator is fixed. Agents operate purely from pixel observations with *no access to state information*. During deployment, we make no effort to calibrate camera, lighting, or physical properties such as dimensions, mass, and friction, and policies are expected to generalize with no prior knowledge of the test environment. Samples from the *push* task are shown in Figure 4, and samples from *reach* are shown in appendix E.

Table 5. Success rate of PAD and baselines on a *real* robotic arm. Best method in each environment is in bold and blue compares +IDM with and without PAD.

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%**

Experimental setup. We implement PAD on top of SAC (Haarnoja et al., 2018) and apply the same experimental setup as in Section 4.1 using an Inverse Dynamics Model (IDM) for self-supervision, but without frame-stacking (i.e. $k = 1$). Agents are trained in simulation with dense rewards and randomized initial configurations of arm, goal, and box, and we measure generalization to 3 novel environments in the real-world: (i) default environment with pixel observations that roughly mimic the simulation; (ii) a patterned table cloth that distracts visually and greatly increases friction; and (iii) disco, an environment with non-stationary visual disco light distractions. Notably, all 3 environments also feature subtle differences in dynamics compared to the training environment, such as object dimensions, mass, friction, and uncalibrated actions. In each setting, we evaluate the success rate across 25 test runs spanning across 5 pre-defined goal locations throughout the table. The goal locations vary between the two tasks, and the robot is reset after each run. We perform comparison against direct transfer and domain randomization baselines as in Section 4.1. We further evaluate generalization to changes in dynamics by considering a variant of the simulated environment in which object mass, size, and friction, arm mount position, and end effector velocity is modified. We consider each setting both individually and jointly, and evaluate success rate across 50 unique configurations with the robot reset after each run.

Table 6. Success rate of PAD and baselines for the *push* task on a *simulated* robotic arm in test environments with changes to *dynamics*. Changes include object mass, size, and friction, arm mount position, and end effector velocity. Best method in each environment is in bold and blue compares +IDM with and without PAD.

Simulated robot	SAC	+DR	+IDM	+IDM (PAD)
Push (object)	66%	64%	72%	**82%**
Push (mount)	68%	58%	**86%**	84%
Push (velocity)	70%	68%	70%	**78%**
Push (all)	56%	50%	48%	**76%**

Results. We report transfer results in Table 5. 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)*. While domain randomization proves highly effective in *reach (cloth)*, we observe no significant benefit in the other settings, which suggests that PAD can be more suitable in challenging tasks like *push*. To isolate the effect of dynamics, we further evaluate generalization to a number of simulated changes in dynamics on the *push* task. Results are shown in Table 6. We find PAD to improve generalization to changes in the physical properties of the object and end effector, whereas both *SAC+IDM* and PAD are relatively unaffected by changes to the mount position. Consistent with the real robot results in Section 5, PAD is found to be most effective when changes in dynamics are non-trivial, improving by as much as **28%** in the *push (all)* setting, where all 3 environmental changes are considered jointly. These results suggest that PAD can be a simple, yet effective method for generalization to diverse, unseen environments that vary in both visuals and dynamics.

5 Conclusion

While previous work addresses generalization in RL by learning policies that are invariant to any environment changes that can be anticipated, we formulate an alternative problem setting in vision-based RL: can we instead *adapt* a pretrained-policy to new environments without any reward. We propose Policy Adaptation during Deployment, a self-supervised framework for online adaptation at test-time, and show empirically that our method improves generalization of policies to diverse simulated and real-world environmental changes across a variety of tasks. We find our approach benefits greatly from learning online, and we systematically evaluate how the choice of self-supervised task impacts performance. While the current framework relies on prior knowledge on selecting self-supervised tasks for policy adaptation, we see our work as the initial step in addressing the problem of adapting vision-based policies to unknown environments. We ultimately envision embodied agents in the future to be learning all the time, with the flexibility to learn both with and without rewards, before and during deployment.

Acknowledgements. This work was supported, in part, by grants from DARPA, NSF 1730158 CI-New: Cognitive Hardware and Software Ecosystem Community Infrastructure (CHASE-CI), NSF ACI-1541349 CC*DNI Pacific Research Platform, and gifts from Qualcomm and TuSimple. This work was also funded, in part, by grants from Berkeley DeepDrive, SAP and European Research Council (ERC) from the European Union Horizon 2020 Programme under grant agreement no. 741930 (CLOTHILDE). We would like to thank Fenglu Hong and Joey Hejna for helpful discussions.

References

OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020. 1

David Bau, Hendrik Strobelt, William Peebles, Jonas Wulff, Bolei Zhou, Jun-Yan Zhu, and Antonio Torralba. Semantic photo manipulation with a generative image prior. *ACM Trans. Graph.*, 38(4), 2019. ISSN 0730-0301. 3

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020. 2

Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2051–2060, 2017. 4

Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1422–1430, 2015. 2

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control, 2018. 2

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016. 2

Robert Geirhos, Carlos R. Medina Temme, Jonas Rauber, Heiko H. Schütt, Matthias Bethge, and Felix A. Wichmann. Generalisation in humans and deep neural networks. In *NeurIPS*, 2018. 2

Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pp. 2551–2559. IEEE Computer Society, 2015. ISBN 9781467383912. 2

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations, 2018. 2, 3, 4, 7

Boqing Gong, Yuan Shi, Fei Sha, and Kristen Grauman. Geodesic flow kernel for unsupervised domain adaptation. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2066–2073. IEEE, 2012. 2

Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *2017 IEEE international conference on robotics and automation (ICRA)*, pp. 3389–3396. IEEE, 2017. 1

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *Advances in Neural Information Processing Systems 31*, pp. 2451–2463. Curran Associates, Inc., 2018. 2

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications, 2018. 4, 8, 17

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2020. 2

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv: Learning*, 2018. 2

Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. *ArXiv*, abs/1906.12340, 2019. 2, 4

Olivier J. Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, S. M. Ali Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding, 2019. 2

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks, 2016. 2

Stephen James, Paul Wohlhart, Mrinal Kalakrishnan, D. Kalashnikov, A. Irpan, J. Ibarz, S. Levine, Raia Hadsell, and Konstantinos Bousmalis. Sim-to-real via sim-to-sim: Data-efficient robotic grasping via randomized-to-canonical adaptation networks. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 12619–12629, 2019. 2

R. Julian, B. Swanson, G. Sukhatme, Sergey Levine, Chelsea Finn, and Karol Hausman. Never stop learning: The effectiveness of fine-tuning in robotic reinforcement learning. *arXiv: Learning*, 2020. 1, 2, 15

D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, V. Vanhoucke, and S. Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *ArXiv*, abs/1806.10293, 2018. 1

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. 2020. 5, 17

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 17

Sascha Lange and Martin A. Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2010. 2

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020. 2, 5, 17

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. A simple randomization technique for generalization in deep reinforcement learning. *ArXiv*, abs/1910.05396, 2019. 1

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016. 1

Ya Feng Li, Mingming Gong, Xinmei Tian, Tongliang Liu, and Dacheng Tao. Domain generalization via conditional invariant representations. In *AAAI*, 2018. 2

Vincenzo Lomonaco, Karen Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3d non-stationary environments. *arXiv preprint arXiv:1905.10112*, 2019. 2, 4, 7, 16, 17

Mingsheng Long, Han Zhu, Jianmin Wang, and Michael I Jordan. Unsupervised domain adaptation with residual transfer networks. In *Advances in Neural Information Processing Systems*, pp. 136–144, 2016. 2

Toshihiko Matsuura and Tatsuya Harada. Domain generalization using a mixture of multiple latent domains, 2019. 2

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 1

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016. 4, 7, 17

Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2019. doi: 10.1109/iccv.2019.00367. 3

Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. In *Advances in Neural Information Processing Systems*, pp. 9191–9200, 2018. 1

Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pp. 69–84. Springer, 2016. 2

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning, 2018. 6

Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2536–2544, 2016. 2

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 2

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018. 1, 2

Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *2016 IEEE international conference on robotics and automation (ICRA)*, pp. 3406–3413. IEEE, 2016. 1

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017a. 1, 2

Lerrel Pinto, James Davidson, Rahul Sukthankar, and Abhinav Gupta. Robust adversarial reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2817–2826. JMLR. org, 2017b. 1

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. *arXiv preprint arXiv:1610.01283*, 2016. 1

Fabio Ramos, Rafael Possas, and Dieter Fox. Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators. *Robotics: Science and Systems XV*, Jun 2019. 2

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018. 2

Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do imagenet classifiers generalize to imagenet? *arXiv preprint arXiv:1902.10811*, 2019. 2

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 1

Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 1, 2

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models, 2020. 2

Vaishaal Shankar, Achal Dave, Rebecca Roelofs, Deva Ramanan, Benjamin Recht, and Ludwig Schmidt. A systematic framework for natural perturbations from videos. *arXiv preprint arXiv:1906.02168*, 2019. 2

Assaf Shocher, Nadav Cohen, and Michal Irani. Zero-shot super-resolution using deep internal learning, 2017. 3

Assaf Shocher, Shai Bagon, Phillip Isola, and Michal Irani. Ingan: Capturing and remapping the "dna" of a natural image, 2018. 3

Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv preprint arXiv:2004.04136*, 2020. 2, 4, 5, 6, 7, 17

Yu Sun, Eric Tzeng, Trevor Darrell, and Alexei A Efros. Unsupervised domain adaptation through self-supervision. *arXiv preprint*, 2019. 2

Yu Sun, Xiaolong Wang, Zhuang Liu, John Miller, Alexei A. Efros, and Moritz Hardt. Test-time training with self-supervision for generalization under distribution shifts. *ICML*, 2020. 3

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 2015. 17

Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. DeepMind control suite. Technical report, DeepMind, January 2018. 2, 4, 5, 16, 17

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter 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. 1, 2

Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7167–7176, 2017. 2

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008. 2

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 2

Xiaolong Wang, Allan Jabri, and Alexei A. Efros. Learning correspondence from the cycle-consistency of time. In *CVPR*, 2019. 2

Mitchell Wortsman, Kiana Ehsani, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning, 2018. 3

Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3733–3742, 2018. 2

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018. 2, 7, 16

Wilson Yan, Ashwin Vangipuram, Pieter Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. *arXiv preprint arXiv:2003.05436*, 2020. 1, 2

Jiachen Yang, Brenden Petersen, Hongyuan Zha, and Daniel Faissol. Single episode policy transfer in reinforcement learning, 2019. 1, 2

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images, 2019. 2, 4, 5, 17

Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European conference on computer vision*, pp. 649–666. Springer, 2016. 2

Richard Zhang, Phillip Isola, and Alexei A Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1058–1067, 2017. 2

A 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 DMControl and CRLMaze. DMControl results are reported in Table 7 and results on the CRLMaze environment are shown in Table 8. In this setting, we also compare to an additional baseline on DMControl: 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 7. Episodic return on the training environment for each of the 9 tasks considered in DMControl, mean and std. dev. for 10 seeds. Best method on each task is in bold and blue 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±17	847±71	756±71	**911±24**	895±28
Walker, stand	388±10	959±11	928±36	**966±8**	956±20
Cartpole, swingup	132±41	**850±28**	807±36	849±30	845±34
Cartpole, balance	646±131	978±22	971±30	**982±20**	979±21
Ball in cup, catch	150±96	725±355	469±339	**919±118**	910±129
Finger, spin	3±2	809±138	686±295	**928±45**	927±45
Finger, turn_easy	172±27	**462±146**	243±124	**462±152**	455±160
Cheetah, run	264±75	**387±74**	195±46	384±88	380±91
Reacher, easy	107±11	264±113	92±45	**390±126**	365±114

Table 8. Episodic return 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 blue compares rotation prediction with and without PAD.

CRLMaze	Random	A2C	+DR	+IDM	+IDM (PAD)	+Rot	+Rot (PAD)
Training env.	−868±34	371±198	−355±93	585±246	−416±135	**729±148**	681±99

B LEARNING CURVES ON DEEPMIND CONTROL

All methods are trained until convergence (500,000 frames) on DMControl. 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 5 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.

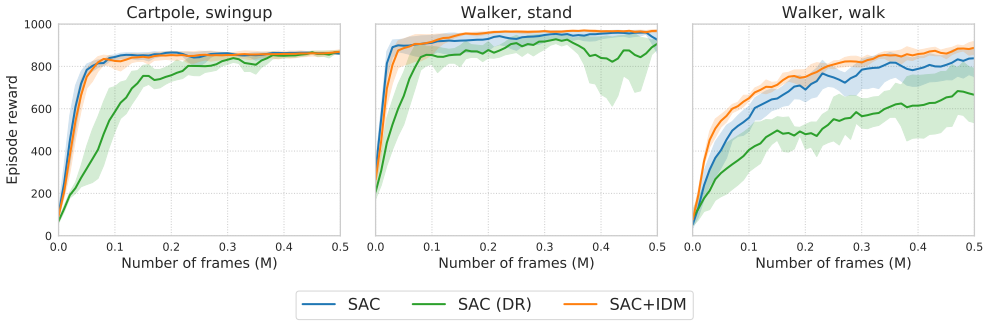


Figure 5. Learning curves for SAC, SAC trained with domain randomization (denoted *SAC (DR)* here), and SAC+IDM on three tasks from the DeepMind Control suite (DMControl). Episodic return is averaged across 10 seeds and the 95% confidence intervals are visualized as shaded regions. SAC and SAC+IDM exhibit 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.

C KEEPING π_s FIXED DURING POLICY ADAPTATION

We now consider a variant of PAD where the self-supervised task head π_s is fixed at test-time such that the self-supervised objective L is optimized only wrt π_e, as discussed in Section 3.3. We measure generalization to test environments with randomized colors and report the results in Table 9 for three tasks from the DeepMind Control suite. We empirically find the difference between updating π_s and keeping it fixed negligible, and we choose to update π_s by default since its gradients are computed by back-propagation regardless.

Table 9. Episodic return in test environments with randomized colors, mean and std. dev. for 10 seeds. All methods use SAC. *IDM (PAD, fixed π_s)* considers a variant of PAD where π_s is fixed at test-time, whereas *IDM (PAD)* denotes the default usage of PAD in which both π_e and π_s are optimized at test-time using the self-supervised objective.

Random colors	IDM	IDM (PAD, fixed π_s)	IDM (PAD)
Walker, walk	$406_{\pm 29}$	$452_{\pm 38}$	$468_{\pm 47}$
Walker, stand	$743_{\pm 37}$	$802_{\pm 41}$	$797_{\pm 46}$
Cartpole, swingup	$585_{\pm 73}$	$623_{\pm 57}$	$630_{\pm 63}$

D COMPARISON TO ADAPTATION WITH REWARDS

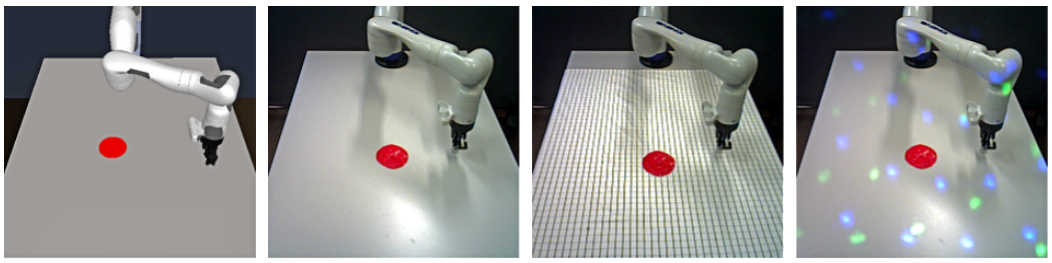
While our method does *not* require data collected prior to deployment and does *not* assume access to a reward signal, we additionally compare our method to a naïve fine-tuning approach using transitions and rewards collected from the target environment prior to deployment. To fine-tune the pre-trained policy using rewards, we collect datasets consisting of 1, 10, and 100 episodes in each target environment using the learned policy while keeping its parameters fixed, and then subsequently fine-tune both π_e and π_a on the collected data, following the same training procedure as during the training phase. This fine-tuning approach is analogous to Julian et al. (2020) but does not use data from the original environment during adaptation. Results are shown in Table 10. We find that naïvely fine-tuning the policy using data collected prior to deployment can improve generalization but requires comparably more data than PAD, as well as access to a reward signal in the target environment. This finding suggests that PAD may be a more suitable method for settings where data from the target environment is scarce and not easily accessible prior to deployment.

E ADDITIONAL ROBOTIC MANIPULATION SAMPLES

Figure 6 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 Figure 4.

Table 10. Episodic return in test environments with randomized colors, mean and std. dev. for 10 seeds. All methods use SAC trained with an inverse dynamics model (IDM) as auxiliary task. Our method is denoted *IDM (PAD)*, and we compare to a naïve fine-tuning approach that assumes access to transitions and rewards collected from 1, 10, and 100 episodes, respectively, from target environments *prior* to deployment.

			Fine-tuning w/ rewards		
Random colors	IDM	IDM (PAD)	1 episode	10 episodes	100 episodes
Walker, walk	$406_{\pm 29}$	$468_{\pm 47}$	$395_{\pm 78}$	$489_{\pm 104}$	$561_{\pm 62}$
Walker, stand	$743_{\pm 37}$	$797_{\pm 46}$	$661_{\pm 65}$	$728_{\pm 44}$	$784_{\pm 31}$
Cartpole, swingup	$585_{\pm 73}$	$630_{\pm 63}$	$538_{\pm 53}$	$605_{\pm 51}$	$650_{\pm 58}$



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

Figure 6. 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) on a real robot, taking observations from an uncalibrated camera.

F Implementation Details

In this section, we elaborate on implementation details for our experiments on DeepMind Control (DMControl) suite (Tassa et al., 2018) and CRLMaze (Lomonaco et al., 2019) for ViZDoom (Wydmuch et al., 2018). Our implementation for the robotic manipulation experiments closely follows that of DMControl. Code is available at https://nicklashansen.github.io/PAD/.

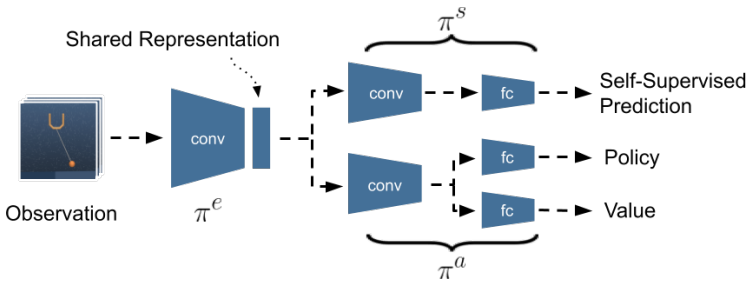


Figure 7. Network architecture for the DMControl, CRLMaze, and robotic manipulation experiments. π^s and π^a uses a shared feature extractor π^e. Observations are stacks of 100×100 colored frames. Implementation of policy and value function depends on the learning algorithm.

Architecture. Our network architecture is illustrated in Figure 7. 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 (DMControl, robotic manipulation) or 6 (CRLMaze) convolutional layers and outputs features of size $32 \times 21 \times 21$ in DMControl 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

Table 11. Hyperparameters used for the DM-Control (Tassa et al., 2018) 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 12. Hyperparameters used for the CRL-Maze (Lomonaco et al., 2019) navigation 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

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 DMControl 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 DMControl 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.

Hyperparameters. When applicable, we adopt our hyperparameters from Yarats et al. (2019) (DM-Control, robotic manipulation) and Lomonaco et al. (2019) (CRLMaze). For the robotic manipulation experiments, our implementation closely follows that of DMControl, only differing by number of frames in an observation. We use a frame stack of $k = 3$ frames for DMControl and CRLMaze, and only $k = 1$ frame for robotic manipulation. For completeness, we detail all hyperparameters used for the DMControl and CRLMaze environments in Table 11 and Table 12.

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 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 by episodic return 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.