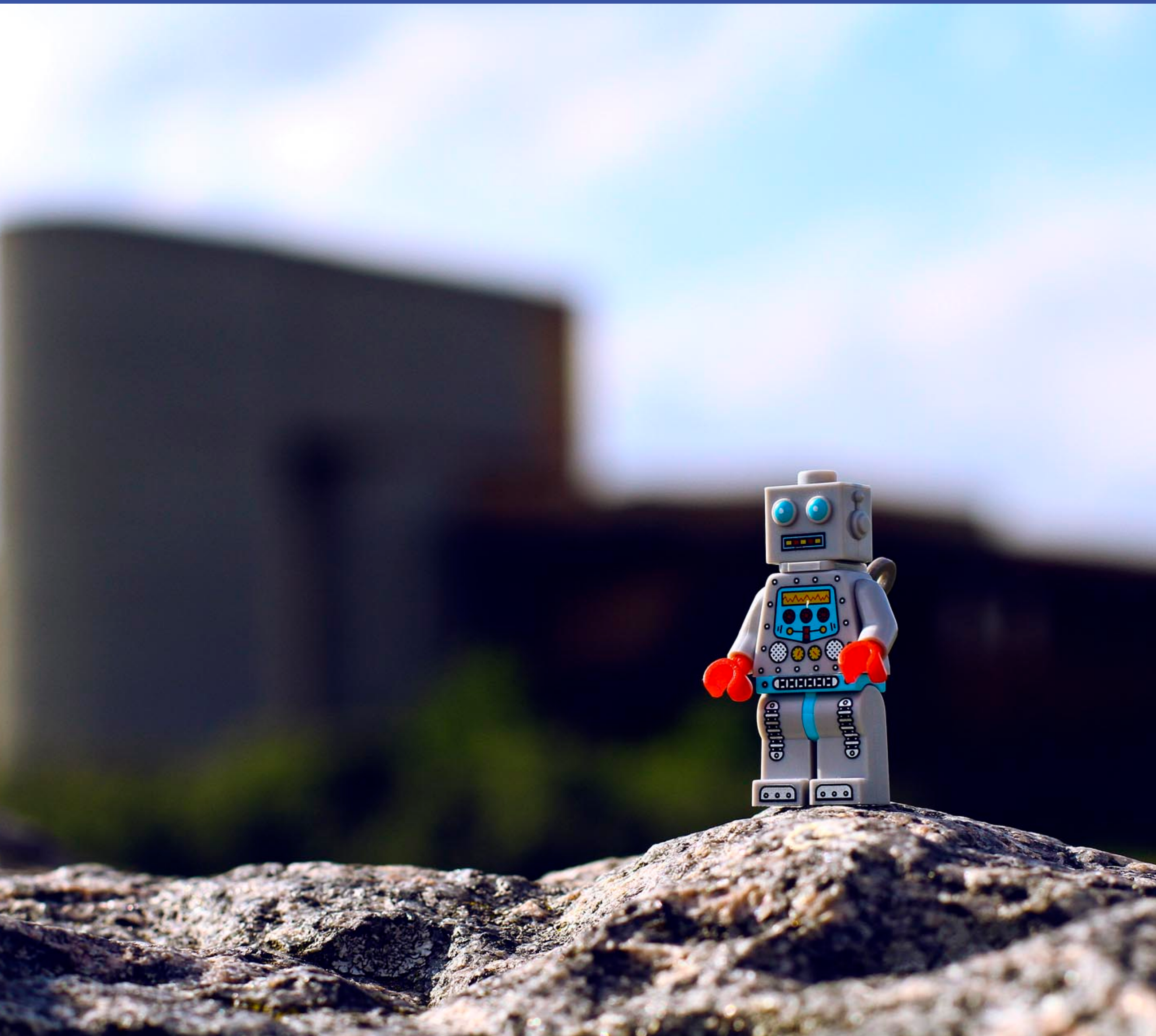
DTU Compute
Department of Applied Mathematics and Computer Science

Generalization in Visual Reinforcement Learning

Master Thesis



Generalization in
Visual Reinforcement Learning

Master Thesis
January, 2021

By
Nicklas Hansen

Supervisors: Ole Winther, Professor, PhD, *Technical University of Denmark*
Xiaolong Wang, Assistant Professor, PhD, *UC San Diego*

Cover photo: Rokas Medonis

Published by: DTU, Department of Applied Mathematics and Computer Science, Richard Petersens Plads, Building 324, 2800 Kgs. Lyngby Denmark
www.compute.dtu.dk

Approval

This thesis has been prepared over six months at the Section for Cognitive Systems, Department of Applied Mathematics and Computer Science, at the Technical University of Denmark, DTU, in partial fulfilment for the degree Master of Science in Engineering, MSc Eng.

Nicklas Hansen (s153077)

....
Signature

....
Date

Abstract

In most real world scenarios, a policy trained by Reinforcement Learning (RL) in one environment needs to be deployed in another, potentially quite different environment. However, generalization across different environments is known to be hard, and the problem is only exacerbated by high-dimensional inputs such as images. A natural solution would be to keep training after deployment, but this cannot be done if the new environment offers no reward signal, as is often the case in real world deployments. Previous work addresses the problem of generalization by learning policies that are invariant to factors of variation that are expected to be present during deployment; this is commonly achieved through domain randomization or data augmentation. While these techniques are easily applicable in supervised learning, in RL they are not. As more factors of variation are introduced during training, policy optimization becomes increasingly more difficult, leading to poor sample efficiency and unstable training, which may prevent the policy from learning altogether. In this thesis, we explore alternative methods for generalization in RL, and propose two distinct frameworks: Policy Adaptation during Deployment (PAD) and Soft Data Augmentation (SODA). PAD is a novel policy adaptation method that uses self-supervision to continue learning even after deployment, without any rewards or human supervision. While previous methods explicitly anticipate changes in the environment, PAD assumes no knowledge of those changes, and adapts only using observations collected during deployment. SODA addresses the scalability issues of data augmentation techniques by decoupling augmentation from policy learning. Specifically, SODA imposes a soft constraint on the encoder that aims to maximize the mutual information between latent representations of augmented and non-augmented data, while the RL optimization process uses strictly non-augmented data. We perform an extensive empirical evaluation of our methods on a diverse set of novel benchmarks for generalization, including deployments on a real robot, and we show that both methods compare favorably to state-of-the-art methods for visual RL. Implementations of our methods and benchmarks are available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark, and video results are available at https://nicklashansen.github.io/PAD and https://nicklashansen.github.io/SODA.

Acknowledgements

I would like to thank my primary supervisor, Prof. Xiaolong Wang, for his brilliant support and feedback throughout the preparation of this thesis and the work that surrounds it. I additionally thank Prof. Ole Winther for his great supervision and academic support throughout my studies and during thesis work, and my collaborators Rishabh Jangir, Yu Sun, Prof. Guillem Alenya, Prof. Pieter Abbeel, Prof. Alexei (Aloysha) A. Efros, and Prof. Lerrel Pinto for their tremendous support, input, and lengthy discussions during our work on the project entitled *Self-Supervised Policy Adaptation during Deployment*. In this context, I would like to especially thank Rishabh Jangir and Prof. Guillem Alenya for making the real robot experiments possible during a global pandemic. I also extend my thanks to Fenglu Hong, Joey Hejna, Ruihan Yang, Mohit Jain, Yueh-Hua (Kris) Wu, Peter E. Christensen, and Frederik Mathiesen for helpful discussions surrounding our work; Denis Yarats, Ilya Kostrikov, Michael (Misha) Laskin, Aravind Srinivas, Kimin Lee, Vincenzo Lomonaco, and the `dm_control` team for their open-source contributions that this thesis builds upon, as well as Alexander R. Johansen and Ali Mohebbi for their moral support.

Part of the work described in this thesis was conducted as an individual study in Prof. Alexei A. Efros' group at the Berkeley Artificial Intelligence Lab (BAIR) during a stay at UC Berkeley. Work on ideation, implementation, and initial results for the PAD method presented in Chapter 5 were carried out at UC Berkeley, whereas our continued efforts in further developing the method and project has been conducted as part of this thesis, along with the proposed DMControl-GB and robotic manipulation benchmarks introduced in Chapter 4, as well as the SODA method presented in Chapter 6, which were developed during a stay in Prof. Xiaolong Wang's lab at the ECE Department of UC San Diego.

This work has been 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, a research grant from Qualcomm, and a gift from TuSimple. This work was also funded, in part, by grants from Berkeley Deep-Drive, SAP, and the European Research Council (ERC) from the European Union Horizon 2020 Programme under grant agreement no. 741930.

Contents

1 Introduction

Deep Reinforcement Learning (RL) combines classical RL methods with the expressiveness of strong function approximators such as neural networks, which have enjoyed rapid development over the past decade, and has gained great interest from especially the artificial intelligence community over its numerous success stories. Notable applications of deep RL include game playing (Mnih et al., 2013; Silver et al., 2016; Pathak et al., 2017; Berner et al., 2019; Cobbe et al., 2019b), robotic manipulation (Pinto & Gupta, 2016; Levine et al., 2016; Nair et al., 2018; Peng et al., 2018; Quillen et al., 2018; Zhu et al., 2019), autonomous vehicles (Gandhi et al., 2017; Siam et al., 2017; Sallab et al., 2017; Zhu et al., 2017; Pan et al., 2018; Kiran et al., 2020), and natural language processing (Grissom et al., 2014; Li et al., 2016; Johnson et al., 2017; Paulus et al., 2018), but the true scope and potential of RL reaches far beyond these applications.

The success of these works can to a large extend be attributed to joint learning of perception and decision-making, both parameterized by deep neural networks and trained by gradient descent in end-to-end optimization frameworks. By leveraging recent advances in machine learning, policies can presently be optimized to perform complex decision-making directly from image observations, without access to any state information, which we in this work shall denote *visual reinforcement learning*, or simply visual RL. Learning from raw sensory inputs such as images and depth sensors is desirable due to their flexibility and accessibility, but also pose difficult perception and generalization problems due to their high dimensionality. Since these problems are not unique to visual RL, it can be helpful to consider literature in related fields.

Although neural networks are wildly popular and successful in supervised learning, they are commonly understood to be prone to *distributional shift* between the finite dataset used during training and the often far more diverse collection of samples encountered in the wild, which is a major obstacle for deployment of reliable machine learning systems in the real world. In the field of computer vision, neural networks have been shown to be prone to catastrophical misclassification on adversarially selected samples, both synthetic (Goodfellow et al., 2015; Sharif et al., 2016; Brown et al., 2017) and naturally occurring (Kurakin et al., 2017; Hendrycks et al., 2019c), that – for the human eye – are indistinguishable to other samples. Despite a decade of architectural improvements and sophisticated optimization methods, neural networks with millions of parameters trained on large-scale image datasets such as ImageNet (Russakovsky et al., 2015) *still* are prone to such seemingly trivial generalization errors.

While one may argue that these adversarially selected samples are isolated events, other works paint a more dire picture. Recht et al. (2019) collects new test sets for two commonly used image recognition benchmarks, CIFAR10 (Krizhevsky, 2009) and ImageNet, closely following the procedure of the original data collections, and evaluate a variety of state-of-the-art convolutional neural networks trained on the original datasets. Perhaps surprisingly, the authors discover a consistent 3%-15% drop in accuracy on the CIFAR10 dataset across all considered models, and a shocking 11%-14% accuracy drop on ImageNet. An explanation for this could be that almost a decade of algorithmic development commonly benchmarked on these two datasets might have led to indirect overfitting to the test sets. However, Recht et al. (2019) finds that the models that perform better on the original test set also generalize better to the new test sets, which effectively invalidates this hypothesis. What, then, could the explanation be? The authors conjecture that the reason for this substantial performance drop may lie in minutiae of the data cleaning process, which is supported by related work on model robustness to perturbations (Hendrycks & Dietterich, 2019; Shankar et al., 2019; Zhang, 2019). These works find that a classifier trained on ImageNet often fails to generalize to even subtle perturbations of the ImageNet test set, such as JPEG-compression and the passing of time in natural videos. Even worse, a model trained to be invariant to e.g. Gaussian noise through data augmentation performs no better than the baseline on test images with speckle noise.

Although these findings are indeed discouraging, stochastic data augmentation and other types of randomization are still highly valuable tools for regularization of neural networks across a multitude of fields. In visual RL, researchers have shown that randomization of visuals and dynamics in a simulated training environment can transfer visio-motor policies trained by deep RL to the real world (Tobin et al., 2017; Sadeghi & Levine, 2017; Peng et al., 2018; Ramos et al., 2019). By randomizing elements such as texture, color, light, camera, and physical properties, the training distribution covers sufficient factors of variation for the policy to generalize to a replica of the simulation constructed in the real world. This technique is commonly referred to as *domain randomization*. While it can be an effective way to build models robust to visual diversity, it still assumes that the target environment is covered by the training distribution, and it therefore simply elevates the problem to a generalization problem across distributions of environments.

Unlike in supervised learning, the policy that we optimize is also commonly used to collect training data. Therefore, the harder the learning problem, the harder it is to generate meaningful training data. Since domain randomization relies on injecting additional factors of variation into the training environment, increasing the size of the training distribution also negatively impacts the sample efficiency of learning algorithms. Consequently, if the training distribution becomes too wide, policy optimization becomes unstable and as a result the policy may never improve. On the other hand, it has also been shown that, just like in the field of computer vision, policies trained by domain randomization generalize no better than their fixed-environment baselines to test environments outside of the randomized training distribution (Packer et al., 2018). For those reasons, practitioners cannot indiscriminately add factors of variation, and can only afford to randomize elements that are deemed likely to vary during policy deployment. To do this effectively, practitioners rely on their own intuition, experimentation, and privileged information about test environments, which is hardly a scalable solution to broadly intelligent behavior in machines.

Another solution to the problem of generalization, albeit naïve, is to simply continue training the policy in the new environment that it is deployed in (Julian et al., 2020). However, this cannot be done if the new environment offers no reward signal. Even if a reward signal can be engineered for each environment that the policy is deployed in, it may require prohibitively many trials to achieve satisfactory behavior, and the approach cannot scale to the endless amount of variability that may be encountered during deployment.

In this work, we explore new methods for generalization in visual RL that are *not* detrimental to the sample efficiency and stability of the policy optimization, do *not* rely on privileged information about the test environments, and do *not* require explicit human supervision for adaptation to their deployment environments. To satisfy these properties, we draw inspiration from the literature of self-supervised learning, a sub-field of representation learning concerning methods that learn expressive representations without explicit supervision or semantic categories. Instead, self-supervised methods derive meaningful training signals from the data itself, and is thus capable of learning powerful representations from vast amounts of unlabelled data. Joint optimization of auxiliary self-supervised objectives together with the primary RL objective(s) can therefore be a useful tool for improving the representational expressiveness in policies parameterized by neural networks. It can both act as a regularization method since the joint optimization imposes a soft constraint on the learned latent space, and simultaneously also be an effective way to derive a complementary training signal in instances where the RL training signal is unreliable or perhaps entirely absent. Concretely, this work discusses two distinct applications of self-supervised learning for improving the generalization in visual RL, both of which prove highly effective in part because image observations contain rich and structured information that can be exploited for self-supervised learning.

The first method, *Policy Adaptation during Deployment* (PAD), explores the use of auxiliary self-supervised learning methods for adaptation of policies to unseen environments. During deployment, a reward signal may be unavailable if the agent is deployed e.g. in the real world, and adaptation using the RL objective is therefore not possible. We propose to instead adapt policies by optimizing a self-supervised objective during deployment, using only raw sensory observations collected during interaction with the new environment. PAD proves to be a simple and intuitive, yet powerful method for adaptation to diverse environmental changes, including both visuals and dynamics. By adapting to environments in an online manner rather than randomizing during training, PAD does not affect the sample efficiency of the learning algorithm, and requires no privileged knowledge about the environment in which it is deployed, which is of importance in e.g. sim2real problems where the environment variations can be difficult to specify formally.

The second method, *Soft Data Augmentation* (SODA), addresses two of the major obstacles in training robust policies by domain randomization and data augmentation methods: sample efficiency and stability of the RL optimization process. While previous work attempts to learn policies directly from augmented observations (either by domain randomization or by data augmentation), SODA uses strictly *non-augmented* observations for policy learning, and instead performs auxiliary representation learning using *augmented* observations. Through its auxiliary task, SODA learns to generalize visually by maximizing the mutual information between latent representations of augmented and non-augmented observations. The proposed auxiliary task shares a learned encoder with the policy, and SODA therefore imposes a soft constraint on the learned representation. As the policy is learned only from non-augmented observations, the difficulty of RL optimization is greatly reduced, while SODA still benefits substantially from data augmentation through its representation learning, hence we refer to it as a *soft* data augmentation.

The proposed methods, as well as a collection of strong baselines from recent literature, are extensively evaluated empirically in simulation on novel benchmarks for generalization in visual RL, and are shown to transfer successfully to robotic manipulation tasks on a real robotic arm under both stationary and non-stationary conditions. To systematically evaluate the proposed methods, we propose the novel *DMControl Generalization Benchmark* (DMControl-GB), a benchmark for visual generalization in continuous control by RL, based on the *DeepMind Control* suite (DMControl) Tassa et al. (2018). DMControl is a collection of challenging and diverse continuous control tasks with both dense and sparse reward signals, and the controls are operated either from ground-truth states or from image observations. We benchmark algorithms on a subset of tasks from DMControl, where both baselines and the individual tasks are selected on the basis of recent advances in RL from image observations Yarats et al. (2019); Laskin et al. (2020b;a); Kostrikov et al. (2020); Stooke et al. (2020); Sekar et al. (2020); Lee et al. (2020a). Whereas prior work only considers sample efficiency and end-performance of RL methods, the proposed DMControl-GB benchmark evaluates generalization to a variety of visually diverse test environments. The test environments are designed to mimic distribution shifts that can be expected in real-world deployment of policies, and serve to provide a common benchmark for visual RL research. To encourage further research in this direction, both PAD, SODA, DMControl-GB, and baselines are open-sourced.

This thesis aims to provide a modern perspective on the challenges of generalization in RL from visual observations, and offers novel approaches to the problem that serve as initial steps towards new directions of research in visual RL. The main body of the thesis is divided into 7 chapters and covers three stand-alone products of our work. Chapter 2 presents the reader to relevant background, Chapter 3 discusses related work, Chapter 4 introduces the proposed benchmarks and procedures for systematic evaluation of generalization ability, Chapter 5 proposes the PAD method for self-supervised policy adaptation, Chapter 6 proposes SODA as a self-supervised alternative to domain randomization, Chapter 7 offers a holistic discussion of the proposed methods as well as our perspective on the future of visual RL, and Chapter 8 concludes the thesis.

2 Background

The main contributions of this thesis are at the intersection of reinforcement learning, computer vision, and robotics. This section aims to provide the unfamiliar reader with the necessary background to better appreciate our work. Section 2.1 introduces core RL concepts and algorithms, and Section 2.2 presents the reader to self-supervised learning methods related to our work. Content presented in this chapter is to be considered self-contained, but assumes a background in deep learning, probability, and optimization, which are all fundamental to our work.

2.1 Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning that is generally concerned with the learning of sequential decision-making from a sequence of observations. In RL, the goal is to learn a *policy* – a mapping from states to immediately executable actions – that maximizes a potentially unknown but observable numerical reward signal. Unlike supervised learning, the learner is not told which actions to take in which state, and instead have to iteratively interact with its environment to learn which behavior is more favorable in terms of the cumulative reward that it receives. The learner that executes a policy within an environment is called an *agent*, and the state of the environment as observed by the agent is formally referred to as an *observation*. The reward signal can be thought of as a numerical measure for how well an agent is performing, and the objective in RL problems is to achieve some *goal* by maximizing the cumulative reward.

A goal can be a concrete state of a dynamical system as in classical control theory, it can be a numerical threshold (e.g. obtaining a specific score in a video game) or it can be a more abstract and temporally spaced-out behavior, such as driving a car without crashing. The purpose of the reward signal is to *reward* an agent for performing a desired behavior and/or *punish* an agent for performing undesired behavior, similar to operant conditioning as popularized by the work of Skinner (1938) on animal learning from a psychological perspective. Some of the key challenges in RL include partial observability of states, high-dimensional and potentially infinite state, observation and action spaces, and delayed rewards which make credit assignment difficult. In this thesis, we address the problem of high-dimensional observation spaces. This section aims to provide the reader with the necessary background in RL to appreciate our work.

2.1.1 Markov Decision Processes

A Markov Decision Process (MDP) is a classical formulation of sequential decision-making problems, and can be considered a mathematically idealized form of reinforcement learning for which precise theoretical statements can be made. In an MDP, the agent continuously interacts with an environment specified by the MDP over a sequence of discrete time steps that may or may not be of finite length. Each interaction can be considered a *transition* between two states of the MDP. In the finite time horizon MDPs that we consider in this work, a sequence of interactions is called an *episode*, and the sequence of state-action pairs observed at each time step by following some policy is called a *trajectory*. In the following, we formally define an MDP in the context of RL.

Let $\mathcal{S}$ and $\mathcal{A}$ denote the state and action spaces, respectively, let $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ denote the transition probability distribution, i.e. the probability distribution over possible next states from being in a state **s** and taking action **a** at a given time step $t \in T$, and let $R(\mathbf{s}_t, \mathbf{a}_t)$ be a reward function. Starting in an initial state $\mathbf{s}_0$ sampled from some fixed distribution $p(\mathbf{s}_0)$, at each time step t an agent takes an action $\mathbf{a}_t \in \mathcal{A}$ which takes the agent from a state $\mathbf{s}_t$ to a new state $\mathbf{s}_{t+1} \sim p(\cdot|\mathbf{s}_t, \mathbf{a}_t)$. One of the key assumptions made in an MDP is that the conditional probability $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ depends only on the current state and action, i.e. the state of the system is fully expressed by $\mathbf{s}_t$; systems fulfilling this property are said to be *Markovian*. Whenever an agent is in a state and follows a transition by taking an action, it simultaneously receives a reward $r_t = R(\mathbf{s}_t, \mathbf{a}_t), \;\; r_t \in \mathbb{R}$. In the MDP framework that we consider, there are generally made very few assumptions about the reward function $R(\cdot)$; it can be dense or sparse, deterministic or stochastic, and is not even expected to be differentiable nor even known (Sutton & Barto, 2018). In practice, the exact specification of the reward function can however play a key role to the success of RL even though it may not necessarily be something that practitioners have any control over; we will return to this discussion later.

We generally distinguish between MDPs with discrete and continuous state-action spaces $\mathcal{S} \times \mathcal{A}$. In the case of a discrete space, we refer to the process as a *finite* MDP since the state-action space contains a finite number of states and actions, e.g. the number of board configurations and moves, respectively, of a game of chess. In the more general case of continuous state-action spaces, the number of states and/or actions are infinite, which makes modeling challenging. Another common distinction is whether an agent observes the full state of the system or only has access to imperfect or partial information about the state – this class of MDPs are commonly referred to as Partially Observable MDPs (POMDPs) and were originally introduced by Åström (1965). In a POMDP, rather than observing a state **s** upon a transition, the agent instead observes an observation $\mathbf{o}_t \in \mathcal{O}$, where $\mathcal{O}$ is the *observation space*. While the system is still assumed to be fully determined by an MDP, the agent only receives observations that contain incomplete information about the underlying state, and therefore needs to learn a policy that acts in the presence of partial information (Kaelbling et al., 1998). In the context of visual RL where POMDPs are most prevalent, the state $\mathbf{s}_t$ at time t may for example contain all the physical information such as the current joint positions, velocities, accelerations, as well as any physical property that may describe the world and fully distinguish two states of the environment from each other, e.g. the random access memory of a computer program running a simulation. Conversely, the corresponding observation $\mathbf{o}_t$ may then be the raw output of a sensory signal, e.g. a camera, depth sensor, or screen rendering. Many real world applications of RL in e.g. robotics – including the applications that we will discuss in this thesis – can be described as both continuous *and* partially observable MDPs with high-dimensional observations, and finding an optimal policy can therefore be very challenging.

2.1.2 The Reinforcement Learning Objective

We will now turn to a formal definition of the RL objective, followed by the derivation of an intuitive algorithm that can be used to solve real problems. As previously described, the objective in RL is to learn a policy that maximizes the cumulative reward (also known as *return*) an agent receives by following a trajectory generated by a policy in an environment described by an MDP. Since both reward function and transition dynamics may be stochastic, we generally try to learn policies that simply maximize the return *in expectation* over some finite number of steps T, which can be formalized as

$$\sum_{t=1}^{T} \mathbb{E}_{\tau \sim p(\tau)} \left[r_t \right] \tag{2.1}$$

for some trajectory τ obtained by following a policy π; this formalization is known as the *reinforcement learning objective*. There are many ways to optimize the RL objective, but in the following we consider a practical algorithm that emerges by direct differentiation of (2.1). *Policy gradient* methods are a class of methods that optimize a policy π_θ parameterized by a neural network with parameters θ by differentiation of the RL objective:

$$
\begin{aligned}
\nabla_\theta \mathcal{L}_{PG}(\theta) &= \nabla_\theta \sum_{t=1}^{T} \mathbb{E}_{\tau\sim p(\tau)}\left[r_t\right] && (2.2)\\
&= \nabla_\theta \mathbb{E}_{\tau\sim p(\tau)}\left[\sum_{t=1}^{T} r_t\right] && (2.3)\\
&= \int \nabla_\theta p(\tau)\left(\sum_{t=1}^{T} r_t\right) d\tau && (2.4)\\
&= \int p(\tau)\nabla_\theta \log p(\tau)\left(\sum_{t=1}^{T} r_t\right) d\tau && (2.5)\\
&= \mathbb{E}_{\tau\sim p(\tau)}\left[\nabla_\theta \log p(\tau)\sum_{t=1}^{T} r_t\right] && (2.6)\\
&= \mathbb{E}_{\tau\sim p(\tau)}\left[\nabla_\theta \left(\sum_{t=1}^{T} \log \pi_\theta(\mathbf{s}_t,\mathbf{a}_t)\right)\sum_{t=1}^{T} r_t\right] && (2.7)\\
&= \mathbb{E}_{\tau\sim p(\tau)}\left[\nabla_\theta \sum_{t=1}^{T} r_t \,\log \pi_\theta(\mathbf{s}_t,\mathbf{a}_t)\right]. && (2.8)
\end{aligned}
$$

The resulting expectation in (2.8) is known as the (direct) *Policy Gradient* (PG), and is central to the REINFORCE algorithm originally proposed by Williams (1992); Sutton et al. (1999). REINFORCE optimizes the RL objective directly by gradient ascent using Monte Carlo estimates of the expected gradient w.r.t. parameters θ, i.e. it solves the problem

$$\theta^* = \arg\max_\theta \mathcal{L}_{PG}(\theta) \qquad (2.9)$$

by sampling trajectories from the policy π_θ. This comes from the realization that, although $\pi_\theta(\cdot)$ is a distribution over actions for a current state $\mathbf{s}_t$, it can be extended to trajectories by letting $p(\tau) = \pi_\theta(\tau) = p(\mathbf{s}_0)\prod_t \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)\, p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$. Note that, while only π_θ depends on θ, both reward function, policy, and transitions may be stochastic, so there can be great uncertainty in trajectories. Further, note that because (2.8) is an expectation over the entire trajectory τ, this stochastic gradient estimator is prone to high variance, and much of the PG literature is therefore concerned with variance reduction techniques such as baseline subtraction, advantage estimation, and reward normalization (Greensmith et al., 2002; Kakade & Langford, 2002; Schulman et al., 2016; Mnih et al., 2016; Chung et al., 2020).

Because of the direct policy optimization, PG methods are classified as *on-policy* algorithms, i.e. they optimize using transitions collected using the *current* policy π_θ. To obtain meaningful gradient estimates, on-policy methods need to average over a large number of trajectories. Since they cannot readily reuse trajectories collected by previous versions of the policy, PG methods exhibit poor sample efficiency in practice. While this may not be a problem for applications for which trajectories are easily generated, the sample inefficiency of (current) PG methods make them unsuitable for applications that rely on real world interaction or costly simulations, such as simulators for e.g. computational fluid dynamics or high-fidelity renderings.

2.1.3 Approximate Dynamic Programming

In this work, we will primarily focus our attention on *off-policy* algorithms that (in principle) can learn from trajectories collected by *any* policy. In practice, this allows for sample reuse and dataset aggregation over the course of training of an agent (and potentially even across agents), and such a dataset is in the RL literature known as a *replay buffer* and denoted $\mathcal{B}$. By learning not only from trajectories collected by the current policy, but also trajectories collected by past versions of the policy, sample efficiency can be greatly improved. Since *past* trajectories cannot be used for reliable gradient estimation of the *current* policy in the on-policy regime from (2.8), we now turn to a class of algorithms that instead rely on (approximate) *dynamic programming*.

Central to approximate dynamic programming in RL is the concept of a state-action value function $Q(\mathbf{s}_t, \mathbf{a}_t)$, which we shall denote the Q-function. A Q-function is a learned mapping $Q : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ that estimates the future cumulative reward attainable by taking action $\mathbf{a}_t$ in state $\mathbf{s}_t$, and is typically parameterized by a neural network. While a Q-function can be implemented as a tabular method in finite MDPs with relatively small state-action spaces, it is infeasible for continuous state-action spaces (and large finite spaces) since a state might never be visited twice and the memory requirements for storing such a table may be prohibitively high. We therefore rely on *approximate* methods implemented as neural networks, which in principle can scale to arbitrarily large problems. Before addressing the approximate methods, however, we first introduce central concepts in the tabular case. By learning the expected Q-value of transitions in a *finite* MDP, the Q-function can be used to identify an optimal policy π^* by always taking the maximum value (in expectation) action in a given state $\mathbf{s}_t$:

$$\mathbf{a}_t^*(\mathbf{s}_t) = \arg\max_{\mathbf{a}_t} Q^*(\mathbf{s}_t, \mathbf{a}_t)\,, \tag{2.10}$$

assuming an optimal Q-function $Q^*(\cdot)$. To learn the Q-function, we iteratively apply *value iteration* updates

$$Q_{n+1}(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q_n(\mathbf{s}_t, \mathbf{a}_t) + \alpha \cdot \delta\,, \tag{2.11}$$

where n is the iteration count, α is a learning rate (step size), and δ is known as the *temporal difference* (TD) error. The TD error is a measure for the error between our current Q-estimate for $(\mathbf{s}_t, \mathbf{a}_t)$ and a target Q-value based on the observed reward r_t, and (2.11) can be directly applied as update rule for an iterative algorithm. Since the Q-function is a measure for the maximum attainable reward (in expectation) from the current state to the end of an episode, we can define the TD(0)-error (one-step) as a relationship known as the *Bellman equation*:

$$\delta = r_t + \gamma \cdot \max_{\mathbf{a}_t} Q(\mathbf{s}_{t+1}, \mathbf{a}_t) - Q(\mathbf{s}_t, \mathbf{a}_t)\,, \tag{2.12}$$

where $\gamma \in (0, 1)$ is a *discount factor* that serves to weight near-future rewards more than rewards obtained far into the future. This class of algorithms is known as TD-learning algorithms, more specifically Q*-learning*, and related ideas will play a central role in this thesis. One of the great advantages of Q-learning is that, by bootstrapping the future return using the Q-function itself, we only need to estimate the single-step error, whereas policy gradient methods require entire trajectories that naturally have high variance. As we shall see, this bootstrapping procedure is not without issues when used along with strong function approximators for Q, but in the tabular case (i.e. discrete state and action spaces, no function approximation), the Q-function is provably convergent to Q^* as $n \rightarrow \infty$ (Watkins & Dayan, 1992). We provide the tabular Q-learning method as proposed by

Algorithm 1 Tabular Q-learning

```
   $\alpha,\ \gamma$: hyper-parameters
1: for each iteration do
2:     for each environment step do
3:         $\mathbf{a}_t = \arg\max_{\mathbf{a}_t} Q(\mathbf{s}_t, \mathbf{a}_t)$                              ▷ Select action using policy
4:         $(\mathbf{s}_{t+1}, r_t) \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$                       ▷ Take environment step
5:         $\delta = r_t + \gamma \cdot \max_{\mathbf{a}_t} Q(\mathbf{s}_{t+1}, \mathbf{a}_t) - Q_\theta(\mathbf{s}_t, \mathbf{a}_t)$     ▷ Compute TD-error
6:         $Q(\mathbf{s}_t, \mathbf{a}_t) \leftarrow Q(\mathbf{s}_t, \mathbf{a}_t) + \alpha \cdot \delta$                  ▷ Update $Q$-function
```

Watkins & Dayan (1992) in Algorithm 1, and delay presentation of a concrete algorithm using strong function approximators to Section 2.1.4.

We will now turn to the arguably more general case where $\mathcal{S}$ and $\mathcal{A}$ are continuous spaces. Since continuous spaces yield infinitely many states, the Q-function cannot be represented as a discrete map unless the state-action space is quantized. Instead, we turn to strong function approximators such as neural networks to learn an approximate Q-function mapping continuous states and actions onto expected returns. We shall assume that the reader is familiar with neural networks as well as fundamental deep learning theory and practices, and instead shift our attention to its implications on Q-learning. Although strong function approximators such as neural networks naturally extend to continuous maps, the Q-learning algorithm itself is not provably convergent in this case, and may lead to severe instability and divergence if not treated carefully. In the following, we will discuss some of the algorithmic challenges in continuous-space Q-learning, and delay discussion of other sources of instability to Chapter 3, Chapter 5, and Chapter 6.

With strong function approximation of the target Q-value used as bootstrapping in the (approximate) dynamic programming of Q-learning, we rely on the neural network's ability to extrapolate to states that may be previously unseen and possibly generated by an unknown behavior policy. Since this extrapolated estimate is used directly in the computation of the TD-error, inaccurate estimates can cause the learning process to diverge (Maei et al., 2009; Kumar et al., 2019). To make matters worse, recall that we in Q-learning rely on these Q-value estimates for policy inference; incorrect estimates therefore lead to sub-optimal policies, and we might never be able to recover from such updates since we typically rely on the inferred policy for data generation. This problem is often referred to as *the deadly triad* of reinforcement learning: function approximation, bootstrapping, and off-policy learning (Sutton & Barto, 2018). Although this paints a somewhat dire picture on the scalability of Q-learning, it turns out that algorithms can be carefully developed and tuned to mitigate these effects, which is covered in more detail in Section 2.1.4.

Another challenge faced in Q-learning from continuous state-action spaces is the $\arg\max_{\mathbf{a}_t}$ and $\max_{\mathbf{a}_t}$ from (2.10) and (2.12), respectively, that are intractable operators for continuous functions. To circumvent this issue, practitioners often opt to also approximate these operators. Since the application of function approximation is distinct in the two cases, we treat them separately in the following.

Policy Learning A continuous policy that maximizes the Q-function (in expectation) can be learned simultaneously with the Q-function itself in a sequence of iterative updates. In the simplest case, we can learn a deterministic policy $\pi(\mathbf{s}_t)$, that outputs an action $\mathbf{a}_t$ approximately maximizing the Q-function. If we assume both π and Q to be differentiable, parameterized functions, then the parameters of π can be optimized by gradient ascent (treating parameters of Q as constants) using the simple objective

$$\max_{\theta} \mathop{\mathbb{E}}_{\mathbf{s}_t \sim \mathcal{B}} \left[Q_{\phi} \left(\mathbf{s}_t, \pi_{\theta}(\mathbf{s}_t) \right) \right] , \tag{2.13}$$

where θ, ϕ are parameters of π and Q, respectively.

Target Q-value As in the previous case, we approximate the max operation using a strong function approximator. We are interested in estimating the Q-value of the maximum value action for a given state $\mathbf{s}_t$, which is in fact a readily available estimate in the form of $Q\left(\mathbf{s}_t, \pi(\mathbf{s}_t)\right)$ from (2.13). While this is a reasonable estimate, it is impractical to apply as a target in the TD-error: because π and Q are iteratively updated, (2.11) becomes a non-stationary, rapidly changing objective susceptible to high variance. A simple mitigation is to instead define a *target* policy (and a corresponding target Q-function) with a separate set of parameters $\bar{\theta}$ (and $\bar{\psi}$), and use those parameters for the target Q-value estimation. While this solves the problem of non-stationarity in the target, we ultimately need to select appropriate parameters for $\bar{\theta}, \bar{\psi}$ for the target to be of any value. A middle-ground between constant target networks and the iteratively updated target from (2.11) is to update targets using the update rule

$$\bar{\theta} \leftarrow \theta \tag{2.14}$$

$$\bar{\psi} \leftarrow \psi \, , \tag{2.15}$$

but only update targets every N-th iteration, e.g. $N = 10$. While (2.14) renders the target of (2.11) stationary in $N-1$ out of every N steps, it turns out that the sudden changes in $\bar{\theta}, \bar{\psi}$ can still lead to instability (Lillicrap et al., 2016). A better way to update target parameters is by using a slow-moving Exponential Moving Average (EMA) of parameters of the *online* (updated by gradient ascent) policy (and Q-function):

$$\bar{\theta}_{n+1} \leftarrow (1-\tau)\bar{\theta}_n + \tau\theta_n \, , \tag{2.16}$$

where τ is a *momentum* coefficient controlling the rate of parameter change. This solution ensures that the target is sufficiently slow-moving to reduce variance in updates, while the target parameters are still updated by an equal amount at every iteration. As we shall see in later discussions, EMAs can help stabilize a variety of target networks in different applications, similar to how they in the optimization literature have emerged as powerful techniques for variance reduction in the step direction of stochastic gradient descent methods (Qian, 1999; Robbins, 2007; Sutskever et al., 2013; Ruder, 2016; Kingma & Ba, 2015).

Together, policy learning and target Q-functions enable deep Q-learning methods to solve high-dimensional continuous control tasks directly from raw pixel inputs. This is first shown by Lillicrap et al. (2016) where it was originally proposed as a *Deep Deterministic Policy Gradient* (DDPG) method for continuous action spaces, inspired by earlier work from Silver et al. (2014). The discrete variant of DDPG, *Deep Q-Network* (DQN) is the neural network equivalent to the Q-learning algorithm as previously described (Mnih et al., 2015). Central to both algorithms is the use of a replay buffer for dataset aggregation, which has become the standard method for data sampling (as opposed to e.g. Monte Carlo sampling) for off-policy RL since it relaxes the assumptions made for data collection. However, since their action selection is deterministic, both methods explicitly add randomization to the optimization process to encourage exploration of the state space during learning. As DQN operates in a discrete action space, a natural way to inject stochasticity is to enforce a random action selection with some probability ϵ, and act greedily in accordance to the Q-function with probability $1-\epsilon$; this exploration scheme is commonly known as ϵ-greedy exploration. In the continuous case, one may also be tempted to simply sample an action uniformly from the action space, but this can lead to instability and inefficient exploration since error scales quadratically in time. A better solution is to instead apply additive noise sampled from some noise source, e.g. a unit Gaussian distribution (or an Ornstein–Uhlenbeck stochastic process as proposed by Lillicrap et al. (2016)), scaled by some (constant) factor, which still ensures action variability but only explores a neighborhood around the current policy in state-action space; this is known as the *reparameterization trick*.

The joint learning of policy and value function has become a prevalent class of deep reinforcement learning algorithms often referred to as *Actor-Critic* (AC) methods. Its name is derived from the idea that the policy learns to *act*, and the purpose of the value function is to *criticize* actions taken by a behavior policy. In the following section, a brief overview of a widely adopted AC method is given, which serves as base algorithm for the majority of work presented in this thesis.

2.1.4 Soft Actor-Critic

Despite the celebrated success of AC methods like DDPG, these algorithms still suffer from poor sample complexity, largely fail to generalize, and are extremely brittle to selection of hyper-parameters (Duan et al., 2016), which makes them of limited practical use. In this section, we introduce *Soft Actor-Critic* (SAC), a recent AC algorithm that aims to address these concerns through a number of algorithmic improvements (Haarnoja et al., 2018a). We will give a brief overview of the SAC method in comparison to DDPG, and broadly discuss the implications of the algorithmic improvements. This will serve as a theoretical foundation for our experiments in Chapter 5 and Chapter 6, although the main contributions of this thesis are independent of the choice of RL algorithm.

Entropy Maximization One of the key components of SAC is its unification of maximum entropy RL (Bagnell & Ziebart, 2010) and off-policy RL. In maximum entropy RL, the conventional RL objective from (2.8) is modified to include an auxiliary entropy term

$$\mathbb{E}_{\tau\sim p(\tau)}\left[\sum_{t=1}^{T} r_t \, \log \pi_\theta(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right)\right] \,, \tag{2.17}$$

where $\alpha \geq 0$ is the *temperature*, a scalar constant that weights the entropy term relative to the reward term, and $\mathcal{H}$ is (in the continuous case) the differential entropy defined as

$$\mathcal{H}\left(\pi_\theta\left(\cdot \mid \mathbf{s}_t\right)\right) = \mathbb{E}_{\mathbf{a}_t}\left[-\log \pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right] \,. \tag{2.18}$$

By jointly maximizing both rewards and the entropy of the policy π, the temperature α effectively determines the relative importance of maximizing returns and maintaining a stochastic policy, such that for $\alpha = 0$ we recover the conventional RL objective and for $\alpha \to \infty$ we maintain an approximately uniform action distribution across all states. While entropy maximization clearly aids in exploration (Schulman et al., 2017a;b), it also results in policies that are more robust to e.g. sensor or state estimation errors as minor deviations from the perceived optimal trajectory are less likely to be out-of-distribution. To make the maximum entropy framework compatible with off-policy RL, Haarnoja et al. (2018b) propose an alternative policy objective

$$J_\pi(\theta) = \mathbb{E}_{\mathbf{s}_t\sim\mathcal{B}}\left[\mathbb{E}_{\mathbf{a}_t\sim\pi_\theta(\mathbf{s}_t)}\left[\alpha \log\left(\pi_\theta\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right) - Q_\psi\left(\mathbf{s}_t, \mathbf{a}_t\right)\right]\right] \,, \tag{2.19}$$

which is the one-step Bellman backup operator (Sutton & Barto, 2018) equivalent of (2.17). Since the SAC algorithm itself is not central to our work, we refer the reader to Haarnoja et al. (2018a) and Haarnoja et al. (2018b) for the derivation of this objective. The gradient of the expectation over $\mathbf{s}_t$ is estimated by sampling transitions from the replay buffer $\mathcal{B}$, and there are several (reasonable) choices of gradient estimators for the expectation over $\mathbf{a}_t$. The simplest choice would be the direct policy gradient estimator that we derived in (2.8), but it is known to have high variance even in the one-step case. However, since the target (our Q-function) of (2.19) is differentiable (unlike the reward function in (2.8)), we can instead apply the reparameterization trick as in DDPG.

Automatic Entropy Tuning Although entropy maximization can greatly improve learning, it introduces the temperature as an extra hyper-parameter. Because the scaling of rewards are not consistent across different tasks (and not even over the course of training in a single task due to non-stationarity), inappropriate selection of α can be detrimental to the success of SAC. Haarnoja et al. (2018b) proposes a convenient method for automatic tuning of the temperature by reformulating the entropy maximization as a constrained optimization problem. Concretely, the authors propose to make α a learnable parameter, optimized by minimizing the difference between the estimated policy entropy $\mathcal{H}$ and a target entropy $\bar{\mathcal{H}}$:

$$J_{\alpha}(\theta) = \mathbb{E}_{\mathbf{s}_t}\left[\alpha\left(\mathcal{H}\left(\pi\left(\cdot \mid \mathbf{s}_t\right)\right) - \bar{\mathcal{H}}\right)\right]. \tag{2.20}$$

Because $\mathcal{H}$ is constrained to be close to $\bar{\mathcal{H}}$ only *in expectation* across visited states, this gives some flexibility for the algorithm to assign more entropy to regions in state space where the policy is uncertain of the optimal action, while allowing the policy to be approximately deterministic in regions where there is a clear distinction between the outcome of actions. While the automatic tuning introduces a new hyper-parameter $\bar{\mathcal{H}}$ in the process of eliminating the reliance on a task-dependent temperature selection, the target entropy is empirically found to be far less of a brittle hyper-parameter than the constant α.

Soft Q-function It is desirable to learn a stochastic policy π since it is more expressive than its deterministic counterpart, and naturally extends off-policy AC algorithms to be compatible with maximum entropy RL. Rather than minimizing the TD-error as formulated in (2.12), Haarnoja et al. (2018a;b) instead propose to minimize a *soft* formulation of the Bellman residual:

$$J_Q(\psi) = \mathbb{E}_{(\mathbf{s}_t,\mathbf{a}_t)\sim\mathcal{B}}\left[\frac{1}{2}\left(Q_{\psi}\left(\mathbf{s}_t,\mathbf{a}_t\right) - \left(r_t + \gamma\mathbb{E}_{\mathbf{s}_{t+1}\sim p(\mathbf{s}_{t+1}\mid\mathbf{s}_t,\mathbf{a}_t)}\left[V_{\bar{\psi}}\left(\mathbf{s}_{t+1}\right)\right]\right)\right)^2\right] \tag{2.21}$$

where

$$V_{\bar{\psi}}\left(\mathbf{s}_t\right) = \mathbb{E}_{\mathbf{a}_t\sim\pi_{\theta}(\mathbf{s}_t)}\left[Q_{\bar{\psi}}\left(\mathbf{s}_t,\mathbf{a}_t\right) - \alpha\log\pi_{\theta}\left(\mathbf{a}_t \mid \mathbf{s}_t\right)\right] \tag{2.22}$$

is the *state*-value function, an estimator for the expected cumulative reward from following a policy π from state $\mathbf{s}_t$, and $\bar{\psi}$ are parameters of a target Q-function (an EMA of ψ) as previously discussed. By insertion of (2.22) into (2.21) and taking the gradient of $J_Q(\psi)$, it is easy to see that the objective can be directly optimized by stochastic gradient descent w.r.t. Q-function parameters ψ:

$$\begin{aligned}\nabla_{\psi}J_Q(\psi) = &\nabla_{\psi}\left[Q_{\psi}\left(\mathbf{s}_t,\mathbf{a}_t\right)\left(Q_{\psi}\left(\mathbf{s}_t,\mathbf{a}_t\right)\right.\right. && (2.23)\\ &\left.\left.- \left[r_t + \gamma\left(Q_{\bar{\psi}}\left(\mathbf{s}_{t+1},\mathbf{a}_{t+1}\right) - \alpha\log\left(\pi_{\theta}\left(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1}\right)\right)\right)\right]\right)\right]. && (2.24)\end{aligned}$$

The reader is referred to Haarnoja et al. (2018a;b) for a derivation of the objective.

Double Q-learning To reduce overestimation of learned Q-values, SAC adopts a double Q-learning formulation from Hasselt (2010); Hasselt et al. (2016). Instead of learning a single Q-function, Hasselt et al. (2016) propose to learn two independent Q-functions, Q_1, Q_2, as well as their corresponding target Q-functions, and simply use the minimum value prediction in policy learning, which has been shown empirically to be a great way to mitigate the consequences of overestimation.

We conclude the overview of SAC with a summary of the algorithm provided as pseudo-code in Algorithm 2. Structurally, the SAC algorithm (and most other off-policy algorithms) resembles the tabular Q-learning algorithm presented in Algorithm 1, but with the addition of a replay buffer and a policy parameterized by neural networks. For each iteration of the algorithm, the policy is first used to collect new transitions from the environment which are added to the replay buffer, and each of the individual objectives are then optimized using transitions sampled (uniformly) from the replay buffer.

Algorithm 2 Soft Actor-Critic (SAC)

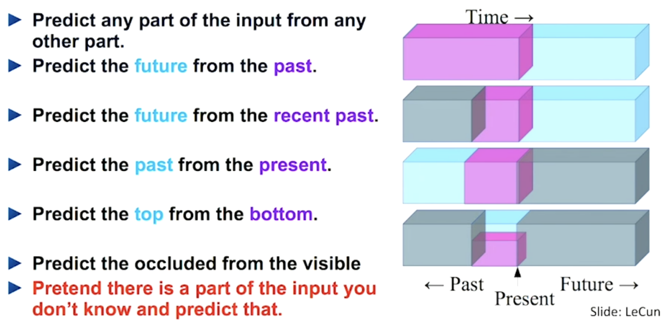
```
    $\theta, \psi$: randomly initialized network parameters
    $\mathcal{B}$: empty replay buffer
 1: for each iteration do
 2:     for each environment step do
 3:         $\mathbf{a}_t \sim \pi_\theta(\mathbf{s}_t)$                                  ▷ Sample action from policy
 4:         $(\mathbf{s}_{t+1}, r_t) \sim p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$           ▷ Take environment step
 5:         $\mathcal{B} \leftarrow \mathcal{B} \cup \{\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1}\}$       ▷ Store transition in replay buffer
 6:     for each update do
 7:         $\nu \sim \mathcal{B}$                                         ▷ Sample transitions from replay buffer
 8:         Optimize $J_{Q_1}(\nu; \psi) + J_{Q_2}(\nu; \psi)$                 ▷ Update $Q$-functions
 9:         Optimize $J_\pi(\nu; \theta)$                                 ▷ Update policy
10:         Optimize $J_\alpha(\nu; \theta)$                                ▷ Update entropy temperature
11:         $Q_1^{\bar{\psi}} \leftarrow (1-\tau) Q_1^{\bar{\psi}} + \tau Q_1^{\psi}$                 ▷ Update target $Q_1$-function
12:         $Q_2^{\bar{\psi}} \leftarrow (1-\tau) Q_2^{\bar{\psi}} + \tau Q_2^{\psi}$                 ▷ Update target $Q_2$-function
```

For further description of RL algorithms and their practical applications, the reader is referred to the relevant literature. We will now conclude the chapter with Section 2.2, which provides a brief and non-exhaustive overview of representation learning methods using self-supervised objectives.

2.2 Self-Supervised Representation Learning

Self-supervision is the art of deriving meaningful supervisory training signal from raw data, without the need for semantic labels (as is commonplace in supervised learning) or rewards (as in RL). High-dimensional data sources and modalities such as RGB images are rich in information and have inherent structure even without labels, which we can exploit to generate a multitude of supervisory signals directly from the data itself. Figure 2.1 illustrates a non-exhaustive list of self-supervisory signals that often can be derived from raw data sources, irrespective of the application area.

As an illustrative example, consider an RGB image: by masking a region in the image and learning to predict pixel values in the masked region (Pathak et al., 2016), the model implicitly learns to leverage structure in the image. Another supervisory signal can be derived by masking one of the color channels and predicting color intensities from the



- Predict any part of the input from any other part.
- Predict the future from the past.
- Predict the future from the recent past.
- Predict the past from the present.
- Predict the top from the bottom.
- Predict the occluded from the visible
- Pretend there is a part of the input you don't know and predict that.

Figure 2.1: **Self-supervised learning.** Illustration of the self-supervised learning paradigm. Training signal is derived directly from input data. Source: LeCun (2018).

remaining two color channels (Zhang et al., 2017), and a third example could be to predict whether two RGB images were sampled from the same distribution or not. While there are countless opportunities to derive training signal from raw data, in reality not all training signals are created equal; the representational expressiveness of a model trained by self-supervision is largely governed by the design of the self-supervised task, and it is therefore crucial for practitioners to design self-supervised tasks that capture information from the data that will be useful for downstream tasks in their desired application area.

In this work, we will focus on two avenues of research in self-supervised representation learning: (1) dynamics modeling, which is central to reinforcement learning applications such as robotics; and (2) contrastive representation learning, a collection of methods that seek to map similar samples (by some metric derived from the context) to similar points in a potentially high-dimensional embedding space. We treat each of the two topics separately, and delay discussion on their individual relevance in the context of visual reinforcement learning to Chapter 3, Chapter 5, and Chapter 6.

2.2.1 Dynamics Modeling

In many real life problems, planning, control, and decision-making relies on the construction of a model that captures information about the state and dynamics of the system, e.g. Linear-Qaudratic-Gaussian Control (Söderström, 2002) and Model Predictive Control (Alba, 1999). In Markovian systems that can be modeled as MDPs, a dynamics model can be interpreted as a predictive model that, given some information about a state or transition, can predict presently unavailable or uncertain information by levering prior knowledge about the structure of the MDP. By learning a model of the MDP, the model can be utilized in planning and control to simulate agent-environment interactions, which allows for e.g. sample-efficient learning and long-horizon control.

Because of the Markov property, the dynamics of a system are fully expressed by their transition probabilities $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$, and it is therefore reasonable to learn an approximation of the transition probabilities; such a model is called a *forward dynamics model*. Although the true transition probabilities are usually unknown, we can estimate $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ by sampling transitions from the environment, i.e. through agent-environment interaction. In states where the agent has no control over the environment, the transition probability can in principle be conditioned solely on the current state. On the other end of the spectrum are systems in which the transition can be conditioned only on the current action; such problems are known as (multi-armed) *bandit problems*, a class of problems concerning single-state systems. In the general case, future states of a system depend on both the current state and action, and learning a model of $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ can therefore

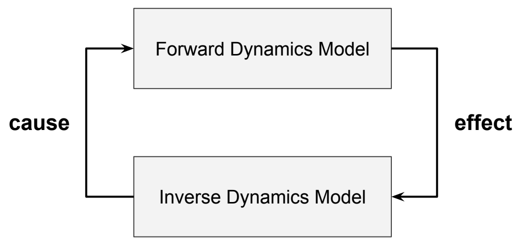


Figure 2.2: **Dynamics modeling.** Illustration of the forward and inverse dynamics models. Conceptually, the *forward* dynamics model the *effect* of an action (resulting observation), and the *inverse* dynamics model the *cause* of a transition (the action taken).

be very valuable for planning. Since the ground-truth $\mathbf{s}_{t+1}, \mathbf{a}_t$ and $\mathbf{s}_{t+1}$ are available at every transition, learning a forward dynamics model can be considered a self-supervised learning problem, where $p(\mathbf{s}_{t+1})$ is predicted from $(\mathbf{s}_{t+1}, \mathbf{a}_t)$.

Another modeling approach is to learn an *inverse dynamics model*: predicting $\mathbf{a}_t$ from $(\mathbf{s}_t, \mathbf{s}_{t+1})$, e.g. predicting forces based on the kinematics and properties of a body. In robotics, this can be useful for estimating how much force should be applied to move a joint from one position to another. We illustrate both the forward dynamics and the inverse dynamics in Figure 2.2. Like the forward dynamics model, learning the inverse dynamics can also be framed as a self-supervised learning problem; in both cases, however, learning the dynamics of a system with potentially high-dimensional states such as images can be challenging.

The forward dynamics model in particular poses a difficult prediction problem, because the dimensionality of an image observation can be in the order of 1×10^6. Furthermore, it should be noted that we are generally not interested in assigning equal importance to all pixels in an image observation, since it often has a poor signal-to-noise ratio and only a small part of the image is relevant to the task. For such high-dimensional problems, it can therefore be useful to instead learn the dynamics in a lower-dimensional latent space that we will denote $\mathcal{K}$. By splitting a forward dynamics model

$$f(\cdot) = f(g(\cdot), \cdot), \quad f : \mathcal{K} \times \mathcal{A} \to \mathcal{K}, \quad g : \mathcal{S} \to \mathcal{K}, \quad \mathcal{K} \ll \mathcal{S} \tag{2.25}$$

implemented by e.g. a neural network, and simply performing dynamics modeling on the output of g while treating g as an observation *encoder* (a learned mapping from a high-dimensional space $\mathcal{S}$ into a lower-dimensional space $\mathcal{K}$), the dimensionality of the problem is significantly reduced (Hafner et al., 2019). However, if f and g are jointly optimized to reduce the expected prediction error of f in latent space, the forward dynamics model yields trivial solutions. As an example, consider the mean squared error objective

$$\min_{\theta} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \sim \mathcal{B}} \left[\| f_\theta(g_\theta(\mathbf{s}_t), \mathbf{a}_t) - g_\theta(\mathbf{s}_{t+1}) \|_2^2 \right] , \tag{2.26}$$

where θ denotes the collection of learnable parameters in a neural network. While the error indeed decreases as the predictive power of f increases, there exists an infinite number of trivial solutions of the form $f(\mathbf{c}, \cdot) = \mathbf{c},\ g(\cdot) = \mathbf{c}$ where $\mathbf{c}$ is a constant feature vector, e.g. $\mathbf{0}$, which yield an error of 0 for all inputs and f can then simply be the identity mapping. When learning dynamics in latent space, it is therefore often desirable to learn an inverse dynamics model rather than the forward dynamics (or a combination of the two), as it has no such trivial solutions. An additional benefit of learning an inverse dynamics model is that the dimensionality of an action space is typically low-dimensional, and the inverse dynamics can therefore learn to predict directly in the action space.

In either case, a dynamics model generally learns representations that are good predictors for the dynamics of the system, while information that yield no predictive power is discarded. This can be problematic if the learned representation is to be deployed outside its original environment and/or is used in downstream tasks that would benefit from the discarded information. Depending on the problem, it can therefore sometimes be useful to consider representation learning tasks that are generally useful, i.e. make few assumptions about the downstream task. In the following section, we discuss one such class of methods.

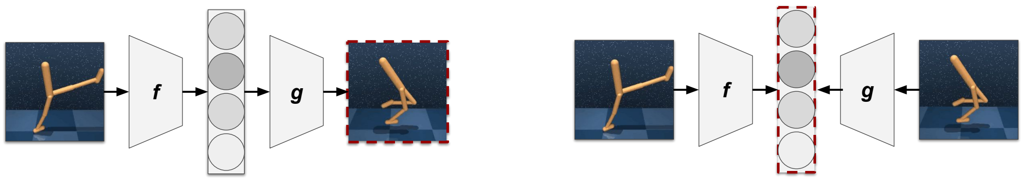
2.2.2 Contrastive Representation Learning

Learning generally useful representations without explicit supervision is a long-standing problem. Many prior works (Vincent et al., 2008; Doersch et al., 2015; Wang & Gupta, 2015; Jayaraman & Grauman, 2015; Pathak et al., 2016; Noroozi & Favaro, 2016; Zhang et al., 2017; Gidaris et al., 2018) in representation learning rely on self-supervised tasks that are well-aligned with the intended downstream tasks. The learned representations therefore lack generality and empirically do not transfer effectively to new tasks. In the context of visual representation learning, generative models (Ballard, 1987; Kingma & Welling, 2014; Bengio et al., 2014; Goodfellow et al., 2014b) have enjoyed great success in lossy data compression and novel image generation, but they are computationally expensive and typically penalizes error in each pixel independently (as in the forward dynamics model in observation space). Although generative models have their own merit, it is yet unclear whether pixel-level image generation is really necessary for expressive representation learning.

Contrastive learning methods (Mikolov et al., 2013; Sermanet et al., 2017; Oord et al., 2018; Wu et al., 2018; Bachman et al., 2019; Tian et al., 2020a; He et al., 2020; Hénaff et al., 2020; Khosla et al., 2020; Chen et al., 2020b; Grill et al., 2020) have recently emerged as a promising alternative class of representation learning methods that *contrast* samples and measure their similarity by some metric formulated as a *score* function. Intuitively, contrastive methods can be thought of as learning to map samples onto an embedding space based on inter-sample similarity, i.e. mapping similar (as measured by the score function) samples to similar points in embedding space, while simultaneously maximizing the latent distance between samples that are deemed dissimilar. Figure 2.3 illustrates the conceptual differences between contrastive learning and predictive learning (e.g. image reconstruction and other self-supervised prediction tasks). Because the embedding space generally is much smaller than the source domain (e.g. image space), contrastive methods learn lower-dimensional representations that encode information relevant for *instance discrimination* rather than for e.g. image reconstruction as in predictive learning, which is arguably more aligned with how humans are thought to learn.

There are currently a multitude of different frameworks for contrastive learning, and it is yet unclear which approach(es) will prevail. In the following, we will give an overview of ideas in contrastive learning that are similar in spirit to the methods that we discuss in Chapter 3, Chapter 5, and Chapter 6, but emphasize that it is not to be considered an exhaustive overview of contrastive methods as a paradigm.

There are a number of open questions regarding the design of contrastive learning methods. What should be compared? How should the comparison be made? How is similarity measured? While there are no conclusive answers to these questions, we aim to discuss a number of recent methods that have proven successful in the computer vision literature. In a recent work, Tian et al. (2020a), the authors learn to map different *views* of the same instance to similar points in latent space, framed as a prediction problem. Given two views



(a) Predictive learning. (b) Contrastive learning.

Figure 2.3: **Predictive learning versus contrastive learning.** Conceptual difference between predictive representation learning and contrastive representation learning. *(a)* Predictive learning learns a latent representation that allows for prediction in the output space. *(b)* Contrastive learning learns a latent representation by contrasting extracted features of congruent and incongruent views. In contrastive learning, f and g may or may not be the same function. The red dashed outline illustrates at which component the loss is applied. Best viewed in color.

v_1^x, v_2^x of a sample x, a discriminating score function $f(\cdot)$ is trained to correctly select v_2^x (the *positive* view) from a set $S = \{v_2^x, v_2^{y_1}, v_2^{y_2}, \cdots, v_2^{y_n}\}$ given an *anchor* view v_1^x using a contrastive objective

$$\mathcal{L}_{\text{contrast}}(v_1^x, v_2^x) = -\mathbb{E}_S \left[\log \frac{f(v_1^x, v_2^x)}{f(v_1^x, v_2^x) + \sum_{j=1}^{n} f\left(v_1^x, v_2^{y_j}\right)} \right], \tag{2.27}$$

which can be interpreted as the log-loss of a multi-way softmax classification with v_2^x as the label for v_1^x. Here, the positive pair x is sampled from the joint distribution $p(v_1, v_2)$, whereas the n negative views in S come from samples $y_1, \ldots, y_n$ sampled from the product of marginals $p(v_1)p(v_2)$. The objective in (2.27) is often referred to as an *InfoNCE* loss as originally proposed by Oord et al. (2018). It can trivially be symmetrized by also considering the opposite scenario where v_1^x is the positive view for the anchor view v_2^x, and can even be generalized to a symmetric loss between M views of a sample x:

$$\mathcal{L}_{\text{multi-contrast}}(v_1^x, \ldots, v_M^x) = \sum_{1 \le i < j \le M} \mathcal{L}_{\text{contrast}}(v_i^x, v_j^x), \quad i \neq j\,. \tag{2.28}$$

The score function $f(\cdot)$ can be implemented in a number of ways. Tian et al. (2020a) and Chen et al. (2020b) propose to implement $f(\cdot)$ as a cosine similarity between latent representations of the two views encoded by some parameterized function $g(\cdot)$, optionally scaled by a constant β:

$$f\left(v_1, v_2\right) = \exp\left(\frac{g\left(v_1\right) \cdot g\left(v_2\right)}{\|g\left(v_1\right)\| \cdot \|g\left(v_2\right)\|} \cdot \frac{1}{\beta} \right). \tag{2.29}$$

Other choices of score function include the dot product $g(v_1)^T g(v_2)$ (Sutton, 1990; He et al., 2020), a bilinear model $g(v_1)^T W g(v_2)$ with parameters W (Oord et al., 2018; Hénaff et al., 2020; Laskin et al., 2020b), or euclidean distance (Wang & Gupta, 2015; Grill et al., 2020). In Appendix B, we provide pseudo-code for Laskin et al. (2020b), a contrastive method that uses the objective from (2.27) and implements $f(\cdot)$ as a bilinear model.

Although Tian et al. (2020a) consider views as different data modalities of a sample (e.g. RGB image, depth map, image segmentation), views can in principle be arbitrarily abstract concepts that relate two parts of a sample to each other, such as color channels or random perturbations of the sample image (Chen et al., 2020b), but views can also

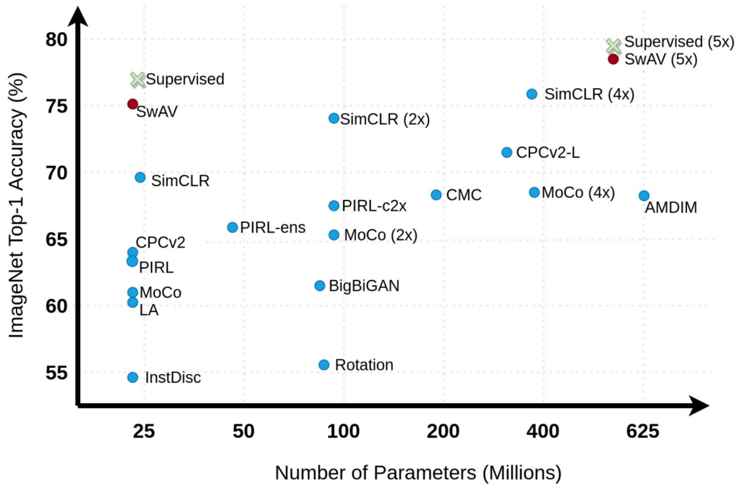


Figure 2.4: **Linear evaluation benchmark.** ImageNet top-1 accuracy of contrastive methods under a linear evaluation. Networks are pre-trained on ImageNet without labels, and a linear classifier is then trained on features extracted from the networks with their weights frozen. Supervised learning on ImageNet is marked with a cross for comparison. *InstDisc* corresponds to Wu et al. (2018), *Rotation* corresponds to Gidaris et al. (2018), *MoCo* corresponds to He et al. (2020), *CMC* corresponds to Tian et al. (2020a), *SimCLR* corresponds to Chen et al. (2020b), and *SwAV* corresponds to Caron et al. (2020).

be two *different* samples that share some property, e.g. contents, meta-data, or even a category-level semantic label as in supervised learning (Khosla et al., 2020). In any instance, the set S is constructed by a single positive view v_2^x as well as a number of *negative* views $v_2^{y_1}, v_2^{y_2}, \cdots, v_2^{y_n}$ that are incongruent to the anchor view v_1^x, e.g. unrelated images sampled uniformly from a dataset.

Design choices such as the selection of views, negative samples, and score function is crucial to the success of contrastive learning methods, and a large body of recent literature studies how to design an effective and generalized framework for contrastive learning. A standard benchmark for representation learning is linear evaluation of methods on ImageNet (Russakovsky et al., 2015): networks are pre-trained on ImageNet without labels, and a linear classifier is then trained on features extracted from the networks with their weights frozen. Figure 2.4 reports ImageNet top-1 accuracy under the linear evaluation.

One of the major obstacles in the adoption of contrastive learning has historically been the enormous amount of compute and negative samples required to produce useful representations. For example, Tian et al. (2020a) use 4096 negative samples for each gradient step and Chen et al. (2020b) use as much as 16382 negative samples. In standard computer vision pipelines for representation learning on ImageNet, this roughly translates to 7,5 GB of memory usage solely for the set S (*after* pre-processing), which is prohibitively large for the majority of the machine learning community. Therefore, researchers are actively developing new contrastive methods that yield effective representations without the need for specialized infrastructure. Wu et al. (2018) propose to sample *features* of negative samples from a memory bank, He et al. (2020) use a momentum encoder as in Section 2.1.4 and maintains a queue of recently encoded negative samples, and Grill et al. (2020) propose an alternative objective that abolishes negative samples altogether. Further discussion on design choices and their implications on visual reinforcement learning is delayed to Chapter 3 and Chapter 6.

3 Related Work

The work presented in this thesis relates to a number of different topics from the deep learning, computer vision, and reinforcement learning literature. This chapter aims to expand upon the discussions in Chapter 2, and summarize recent work that most closely relates to our contributions. The chapter has been divided into 4 sections, providing insights into recent work on representations, self-supervision, and generalization, as well as their applications in RL, which helps the reader put our work into the context of a larger, ongoing research effort on building intelligent systems that function in the real world.

3.1 Learning Visual Invariances

Reinforcement learning from visual observations is an increasingly popular avenue of research, because it holds the promise of enabling agents to more easily interact with unstructured environments where agents with proprioceptive inputs fail. Training RL agents directly in the real world can however be costly. Therefore, practitioners leverage simulated environments during training with the hope of agents being able to generalize their knowledge to the real world. Building simulations that accurately replicate the real world is however challenging, and outside ideal lab environments the visual and dynamical diversity that agents encounter is much greater than what can feasibly be built in a simulation.

A convenient method for improving generalization of agents is to randomize elements of the simulation that can be expected to vary during deployment; this approach is broadly referred to as domain randomization (Tobin et al., 2017; Sadeghi & Levine, 2017; Peng et al., 2018; Tremblay et al., 2018; Ramos et al., 2019). Randomized elements may include visuals such as color, lighting, and texture, or dynamics such as object morphology, physical properties, and environmental layout. Generally, determining which elements to randomize – and to which degree – remains an engineering challenge, since RL optimization becomes increasingly difficult as more factors of variation are added to the distribution of training environments. Unlike in supervised learning, RL algorithms optimize on data generated by the algorithm itself, and increasing the difficulty of policy optimization can therefore prevent the policy from ever improving. Even worse, studies find that agents trained by domain randomization perform no better than their simple baselines when evaluated on environments outside of the training domain (Packer et al., 2018), consistent with findings in computer vision (Hendrycks & Dietterich, 2019; Shankar et al., 2019).

A recent line of work (Lee et al., 2019; Laskin et al., 2020a; Kostrikov et al., 2020; Raileanu et al., 2020b) show that simple data augmentations can improve sample efficiency and generalization of RL agents substantially, without assuming access to the simulation itself. However, (Laskin et al., 2020a) find that, while RL agents can benefit from data augmentation, the choice of augmentation is task-dependent and non-trivial. Further, their experiments show that a sub-optimal choice of data augmentation can be severely detrimental to the end-performance of trained agents, and with bad selections of data augmentation, agents may even fail to learn at all. Selecting appropriate data augmentations for learning invariances in RL therefore easily becomes a tedious engineering challenge, and ultimately suffers from the same problems as domain randomization. In this thesis, we investigate alternative methods for generalization in visual RL that require no significant engineering, and scale to test environments that differ substantially from the training environment, with no prior knowledge about the nature of the changes.

3.2 Self-Supervised Learning

Learning visual representations from unlabelled data has been of increasingly bigger interest in recent years, both for pre-training of computer vision models as good initializations for future downstream tasks (Noroozi & Favaro, 2016; Pathak et al., 2016; Zhang et al., 2017; Gidaris et al., 2018; Zamir et al., 2018; Hendrycks et al., 2019a; Tian et al., 2020a; He et al., 2020; Khosla et al., 2020; Chen et al., 2020b; Grill et al., 2020), and as auxiliary tasks with the purpose of increasing representational expressiveness and robustness to aid the main task in its objective (Hendrycks et al., 2019b; Sun et al., 2020; Chen et al., 2020a; Zamir et al., 2020).

For example, Gidaris et al. (2018) show that simply rotating images by 0, 90, 180, or 270 degrees at random and formulating a self-supervised task as predicting how an image was rotated is a surprisingly effective method for pre-training image classifiers. Hendrycks et al. (2019b) further show that jointly learning a self-supervised auxiliary task together with a classifier on ImageNet improves classification performance on out-of-distribution samples, and Carmon et al. (2019); Kim et al. (2020) show that self-supervision also improves robustness to adversarially selected samples. The common conjecture in works on auxiliary self-supervision is that auxiliary tasks help regularize the shared representation of deep neural networks. Zamir et al. (2018) show that additional modalities and self-supervised pre-training tasks improve transfer learning capabilities of neural networks in computer vision, which would suggest that it may indeed be true.

Section 2.3 introduces the concept of *contrastive* representation learning as an alternative to predictive self-supervised tasks. A significant drawback of contrastive learning methods have been their immense compute and data requirements. Recent work (Grill et al., 2020) propose a framework for contrastive learning *without* negative samples, which alleviates one of the most pressing concerns in contrastive learning: the large batch sizes. Additionally, they show that their self-supervised pre-training can learn representations for image classification that are on par with models trained with ImageNet labels in a supervised fashion, demonstrating that semantic labels are not necessary for meaningful representations. The method proposed by Grill et al. (2020), denoted `BYOL`, uses stochastic data augmentations $t, t' \sim \mathcal{T}$ to produce two different views $v = t(x),\ v' = t'(x)$ from an input image x. An online network encodes and projects v into a lower-dimensional feature vector z, and an offline network defined as an EMA of the online network similarly encodes and projects v' into a vector z'. `BYOL` then trains a prediction network to predict z' from z, using the (normalized) mean squared error between the predicted feature vector and z' as training objective. The authors show that `BYOL` can learn surprisingly expressive representations through careful selection of data augmentations, and additionally find that their method is comparably more stable than previous contrastive learning methods when training with small batch sizes, which makes `BYOL` a viable self-supervised learning task in application domains that historically have used much smaller batch sizes, such as RL.

In this thesis, we propose a framework for generalization in RL that optimizes for invariance through strong data augmentation and an auxiliary self-supervised objective similar to that of `BYOL`, which we introduce in Chapter 6. However, rather than learning to align representations of two augmented views of the same instance, we learn to map the features of *augmented* views onto their *non-augmented* counterparts in latent space. We also propose to perform reinforcement learning together with other self-supervised tasks, including rotation prediction as proposed by Gidaris et al. (2018), and inverse dynamics models as introduced in Chapter 2.2.1. In the following section, we discuss the role of self-supervision in unsupervised adaptation.

3.3 Test-Time Adaptation

Deep domain adaptation is the practice of adapting a deep network pre-trained on one data distribution (the *source domain*) to perform well on a different distribution (the *target domain*). Some approaches to domain adaptation rely on labeled data in the target domain and propose methods for supervised fine-tuning (Hoffman et al., 2015; Hu et al., 2015; Peng et al., 2016; 2019b) without catastrophic forgetting (Goodfellow et al., 2014a), while others only assume access to unlabelled data in the target domain (Ganin & Lempitsky, 2015; Long et al., 2016; Sun et al., 2019). For example, Sun et al. (2019) propose to jointly learn a set of auxiliary self-supervised tasks using data from both the source and target domain, which serves to bring the learned representation closer to the target domain. Common to these works on domain adaptation is the assumption that one has access to a substantial amount of samples from the target domain, which makes them of limited use in the general case of adaptation to unseen distributions.

Recent work in computer vision explores the possibility of mitigating distributional shift by adaptation of deep learning models *at test-time*, with access only to a limited number of test samples (e.g. one) and no previous data from the target domain (Shocher et al., 2017; 2018; Bau et al., 2019; Mullapudi et al., 2019; Wortsman et al., 2019; Sun et al., 2020; Zhou & Levine, 2020; Zhang et al., 2020a; Alet et al., 2020). For example, Shocher et al. (2018) show that it is possible to train a network – from scratch – to perform image super-resolution using only a single test image, simply by learning to upsample artificially downsampled versions of the image. Bau et al. (2019) show that adapting the prior of a Generative Adversarial Network to the image statistics of individual test images improves photo manipulation tasks, and Zhang et al. (2020a) propose a meta-learning task that exploits statistics of a batch of test samples to adapt to individual samples in the batch.

The adaptive mechanism in this thesis work more closely resembles that of Sun et al. (2020), as well as concurrent works Zhou & Levine (2020); Alet et al. (2020); Raileanu et al. (2020a). As evident from the literature on self-supervised domain adaptation, auxiliary tasks are helpful for adaptation to new domains. In Sun et al. (2020), the authors propose to optimize an auxiliary self-supervised task jointly together with the main supervised objective during training, and opt for rotation prediction (Gidaris et al., 2018) in their experiments. They show that performing self-supervised updates on randomly perturbed versions of individual test images before main task prediction improves the performance of the main task under various distribution shifts. Similarly, Zhou & Levine (2020) propose self-supervised test-time adaptation as a method for improved uncertainty estimates, and Alet et al. (2020) reformulates the test-time adaptation framework as a meta-learning problem. The primary benefit of adaptation through self-supervision is its flexibility; while other works rely on stationarity in the test environment (Yang et al., 2020a) or access to the test environment prior to deployment (Hanna & Stone, 2017; Desai et al., 2020; Karnan et al., 2020), self-supervision allows an agent to continuously adapt to an environment through interaction.

With empirical evidence that self-supervision can be used for adaptation to distribution shifts encountered only at test-time, it is natural to ask whether self-supervision can be used for adaptation in reinforcement learning, where an agent continuously collects new data from the test environment through interaction. In this thesis, we explore a novel approach to generalization that continuously adapts using observations received at test-time. Through interaction with an environment, we can extract information about discrepancies in both visuals and dynamics, and we propose to optimize a self-supervised objective at test-time to better align an agent's representation with its new environment.

3.4 Visual Representations in Reinforcement Learning

Reinforcement learning is *hard*. In many cases, reward signals are sparse, and the rewards may even be delayed in time which makes credit assignment challenging. It is not uncommon for an agent to only receive a single bit of feedback for an entire trajectory's worth of experience, which leaves the agent with very little training signal. To aid the process of policy learning, it can be useful to employ auxiliary tasks to increase the density of training signal and to produce immediate feedback for the agent. For instance, Jaderberg et al. (2017) introduce a number of novel self-supervised tasks for visual RL, e.g. predicting whether the agent will be rewarded, penalized, or neither within a short time horizon, and they find each additional training signal to improve sample efficiency and end-performance of the agent on a diverse set of tasks when optimized jointly together with a sparse reward signal. Shelhamer et al. (2017) extends this work by providing an extensive study on auxiliary tasks for RL in Atari games. They find that it is highly task-dependent whether an auxiliary task is helpful or not to a particular RL task, but that auxiliary tasks generally do not hurt the end-performance, similar to the findings of Hendrycks et al. (2019b); Sun et al. (2020) in computer vision.

A vast amount of work in RL focuses on the learning of environment dynamics through forward and inverse dynamics models. Model-based RL methods (Deisenroth & Rasmussen, 2011; Chua et al., 2018; Janner et al., 2019; Hafner et al., 2019; Yan et al., 2020; Yu et al., 2020b) learn a model of the environment and use the learned model for trajectory planning, whereas model-free algorithms may simply learn a dynamics model as an auxiliary objective to provide additional training signal (Nguyen-Tuong & Peters, 2010; Stadie et al., 2015; Baranes & Oudeyer, 2013; Agrawal et al., 2016; Pathak et al., 2017; Ha & Schmidhuber, 2018; Sekar et al., 2020); in this work, we shift our attention to model-free RL algorithms.

World models are a promising direction of research. Originally proposed by Ha & Schmidhuber (2018), this class of algorithms learn a low-complexity model of the environment using a generative self-supervised objective, and afterwards learn a compact policy using the world model representation as input. Furthermore, the authors show that, given a powerful enough world model, it is possible to train a policy solely on generated data, and that the learned policies successfully transfer back to the original environment that was used to train the world model.

Another interesting direction of research is curiosity-based exploration using dynamics models (Pathak et al., 2017), which has been found to learn complex navigation skills without any reward signal. The authors train both a forward dynamics model and an inverse dynamics model jointly, and simultaneously train a policy for environment interaction using the loss of the dynamics models as reward signal. Concretely, the received reward is high when the dynamics losses are high, and vice versa. Although this may seem counter-intuitive at first, it can be motivated as follows: since the dynamics loss will be high for out-of-distribution data, the loss is a good indication of whether the policy is exploring new parts of the environment or not. While both of these works are interesting applications of dynamics models, they are not directly applicable to the problem of generalization. We propose a novel application of dynamics models to improve the generalization of RL agents by optimizing a self-supervised objective at test-time.

Aside from learning the environment dynamics, visual RL also requires the ability to perceive and act based on those perceptions. With high-dimensional inputs such as images, perception is a non-trivial task to learn, and the difficulty is only exacerbated by the simultaneous policy learning. It is therefore natural to ask whether one can draw inspiration from computer vision literature to improve the perception of agents. Yen-Chen et al. (2020) shows that visual pre-training on large datasets such as ImageNet can provide good visual priors for certain robotics applications, and they find that robots can learn to manipulate objects with just a small amount of interaction when pre-trained visually, which is more in line with how humans learn new skills. A recent line of research investigates the use of self-supervised tasks from computer vision literature for joint visual learning together with RL optimization (Yarats et al., 2019; Laskin et al., 2020b; Lee et al., 2020a; Stooke et al., 2020; Zhan et al., 2020). For example, Yarats et al. (2019) show that learning an auxiliary autoencoder improves both sample efficiency and end-performance of policies for continuous control from images. Laskin et al. (2020b) builds upon this work and find that a simple contrastive learning objective similar to Chen et al. (2020b) further improves policy learning, and nearly closes the gap in sample efficiency between learning from proprioceptive states and image observations; an encouraging result for the future of visual reinforcement learning.

3.5 Reinforcement Learning in the Real World

A common problem setting for generalization and adaptation in RL is *sim2real*; training a policy in simulation with the intent to deploy it in the real world. Learning in simulation is attractive in part because training in the real world can be both costly and outright *unsafe*, and in part because simulations offer a lot more flexibility during experimentation. However, building an accurate replica of the real world in a simulation is challenging, and even minutiae discrepancies between the simulation and the real world can be devastating to the transfer of agents. In this section, we explore recent work on sim2real, which can be considered a special case of the general transfer problem where the target environment is either partially or fully known, and can possibly be queried during training.

A number of prominent works have shown that it is possible for agents to learn deep visuomotor policies directly in the real world (Levine et al., 2016; Pinto et al., 2016; Pinto & Gupta, 2016; Finn & Levine, 2017; Gu et al., 2017; Levine et al., 2018; Ebert et al., 2018; Gupta et al., 2018; Kalashnikov et al., 2018; Zhan et al., 2020). The common approach in these works is to collect large real-world robot interaction datasets and then use that data for off-policy Q-learning to produce a policy. For example, Pinto & Gupta (2016) learns robotic grasping of various objects from 700 robot hours worth of grasping attempts (corresponding to more than 500,000 attempts), where a sparse reward signal is derived from a tactile sensor, and Kalashnikov et al. (2018) similarly collects a dataset consisting of approximately 800 robot hours. In all of these works, the policy is learned and deployed in the same environment, though some studies consider generalization across grasping objects (Pinto & Gupta, 2016; Kalashnikov et al., 2018; Zhan et al., 2020).

The research community has recently started to address generalization across environments. Julian et al. (2020) show that visuomotor policies for robotic grasping can be trained on pre-collected data from Kalashnikov et al. (2018) and then fine-tuned in a select number of robot environments that differ both in visuals, objects, and robot morphology. The authors find that a naïve fine-tuning strategy works surprisingly well when trained on data collected from 700 trials in the new environment. While 700 trials from each individual environment that we wish to deploy to is still a lot, it shows that policies can indeed adapt to both the visuals and dynamics of their new environments, even in the real world.

Concurrent to our work, Zhan et al. (2020) show that, by application of techniques similar to those proposed in this thesis, diverse robotic manipulation skills can be learned from just 10 human demonstrations and less than one hour of robot interaction. Specifically, the authors propose to pre-train the encoder of a Soft Actor-Critic (Haarnoja et al., 2018b) agent using a contrastive loss similar to that of Laskin et al. (2020b) and a dataset collected from human demonstrations. After pre-training, both policy and encoder is jointly trained through robot interaction and data augmentation. In our work, we employ similar techniques but require no demonstrations nor robot interaction in the real world. Instead, we train policies entirely in simulation and only subsequently transfer to a real robot. While learning tasks from scratch in the real world with little data is a remarkable feat, we argue that future robots will also need to leverage prior experiences to learn more complex and compositional behaviors in just a few interactions.

To minimize real-world interaction during training, related works aim to learn a policy in simulation and then iteratively align either the policy or the simulation itself with the real world (James et al., 2019; Karnan et al., 2020; Desai et al., 2020; Zhang et al., 2020b). For instance, James et al. (2019) learns to transform the visuals of randomized simulations to a *canonical view* before inputting it into the policy, and (Karnan et al., 2020; Desai et al., 2020) learns to transform the dynamics of a simulation to its real-world deployment by iteratively collecting data in both domains. Concurrent to our work, Zhang et al. (2020b) learns dynamics correspondence between two domains using *unpaired* data, and uses the learned correspondence to transfer a policy from simulation to a real robot without any fine-tuning. While these are encouraging steps towards scalable real-world robot learning, in this thesis we consider the general case of policy transfer without prior knowledge about the target environment. To address this, we investigate how to adapt learned policies *during deployment*, and the discussed methods are therefore not directly applicable to our problem setting.

3.6 Summary and Open Problems

In recent years, increasingly powerful RL algorithms have rapidly improved the capabilities of agents, as well as the rate at which they can acquire new skills. However, many successes have stemmed from policy learning directly from low-dimensional proprioceptive states, which can be difficult to obtain reliably outside ideal conditions, e.g. when deployed in the real world. In pursuit of learning systems that better reflect how humans perceive and interact with the world, RL from visual observations have attracted interest. When an agent's environment is unstructured and may contain unanticipated elements, visual perception can aid the agent in its decision-making and allow it to dynamically adjust its behavior to better reflect the environment it interacts with.

Learning to perceive, however, has by itself proven challenging, and it is therefore worth considering techniques for visual perception that have brought success in the field of computer vision, such as randomization, data augmentation, and self-supervised representation learning. Although the adoption of randomization and data augmentation techniques have contributed to remarkable improvements in both sample efficiency and generalization ability of agents trained by RL, policy learning on increasingly large training distributions leads to optimization difficulties and may prevent learning entirely. At the same time, it also remains non-trivial to design appropriate randomization for each task, and training policies that generalize therefore becomes a major engineering hurdle. As a result, practitioners are forced to manually balance the generalization versus convergence trade-offs that emerge with current methods. To improve generalization, recent works have proposed to learn mappings of visuals and dynamics between two domains; these mappings are however equally prone to distributional shift, and the problem of generalization across environments therefore becomes a meta-problem of generalization between *distributions* of environments as in domain randomization and data augmentation approaches.

To address these challenges, we need to rethink how agents learn and represent their knowledge. A promising direction of research is the application of self-supervised representation learning together with policy optimization. Depending on the formulation of the self-supervised task(s), previous work has individually shown that auxiliary tasks can improve both sample efficiency, generalization, and end-performance. While the majority of recent work on visual representation learning in RL train and evaluate on the same environment, generalization to unseen, but similarly structured, environments is arguably just as crucial to more widespread deployment of visual RL algorithms, which has been acknowledged by a number of recent works. In this thesis, we investigate novel approaches to generalization that seek to address the problems of scalability of current learning algorithms. To benchmark generalization ability of the developed methods against recent state-of-the-art methods for visual RL, we systematically evaluate our methods and baselines on a number of new benchmarks for generalization in RL from images, which we introduce in Chapter 4.

4 Benchmarks for Generalization in Visual Reinforcement Learning

To accurately measure advances in algorithmic design, standardized benchmarks are a necessity. For the past decade, ImageNet (Russakovsky et al., 2015) has served as a large-scale benchmark for computer vision research, enabling practitioners to compare algorithms and neural network designs, and continuously push the field of computer vision forward in measurable increments. Benchmarks such as GLUE (Wang et al., 2018; 2019) and SQuAD (Rajpurkar et al., 2016) have played a similar role in the field of natural language processing, and likewise in RL and robotics research other such benchmarks have been proposed (Bellemare et al., 2015; Dosovitskiy et al., 2017; Wydmuch et al., 2018; Lomonaco et al., 2019; Juliani et al., 2019; Tassa et al., 2020; Zhu et al., 2020; Cobbe et al., 2020). The Arcade Learning Environment (ALE) (Bellemare et al., 2015) has been of particular significance to the field due to its accessibility, diversity in environments (55 Atari games, optionally visual observations), and right level of difficulty (easy enough for – at that time – state-of-the-art algorithms to achieve non-trivial results, but still leaving a substantial gap between algorithms and humans), which has made it a long-standing challenge to achieve superhuman performance on the benchmark.

More recently, DeepMind Control Suite (DMControl) (Tassa et al., 2018) has been proposed as a benchmark for continuous control, featuring challenging and diverse tasks that reflect real robotic tasks such as locomotion, manipulation, and grasping, offering both proprioceptive states and image renderings as observations for RL. Figure 4.1 shows a number of tasks proposed in DMControl. Like ALE, DMControl has been adopted by the community as a benchmark for algorithmic improvements, and has been used in a number of related works on sample efficient reinforcement learning from visual observations (Hafner et al., 2019; Yarats et al., 2019; Laskin et al., 2020b; Kostrikov et al., 2020; Laskin et al., 2020a; Stooke et al., 2020). DMControl is particularly well-suited for algorithmic benchmarks in visual RL because of its standardized interfacing and task design, while still offering tasks that are technically challenging (e.g. large action spaces, precise control, random initializations, sparse rewards) and require a diverse set of skills. While both ALE and DMControl are excellent benchmarks for measuring algorithmic progress in visuomotor control, as per their original proposals they do not consider generalization across distributions of related environments.

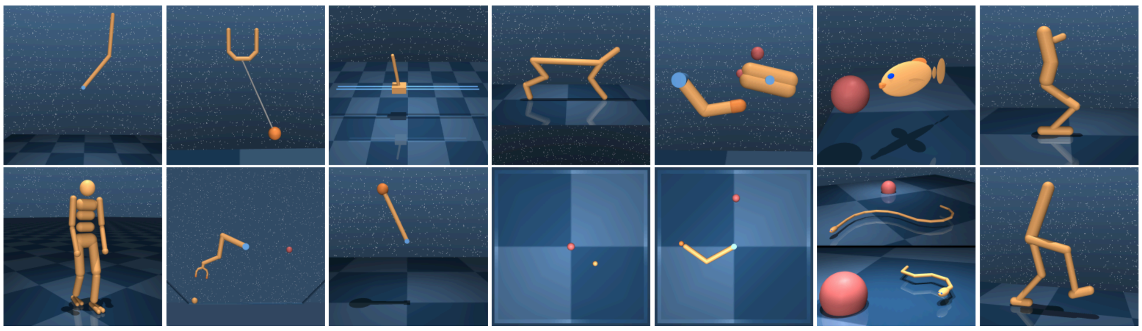


Figure 4.1: **DMControl environments.** Visualization of common tasks from the DMControl environments for benchmark of continuous control. Source: Tassa et al. (2018).

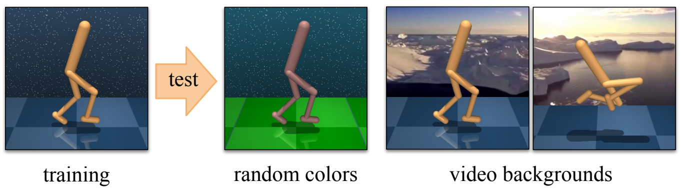


Figure 4.2: **DMControl-GB evaluation.** Agents are trained in a fixed environment and evaluated on a wide range of unseen and visually diverse test environments, including environments with randomized colors and video backgrounds. Sample images correspond to actual observations.

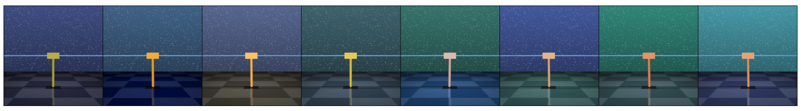
Machado et al. (2018); Dubey et al. (2018) propose variants of the ALE environment for evaluating generalization to held-out game modes, and Packer et al. (2018) evaluates generalization to modified dynamics in continuous control tasks from proprioceptive states similar to those of DMControl. Common to these works is their focus on generalization to changes in environment dynamics. Related work, Song et al. (2020), studies the concept of observational overfitting in a video game played directly from pixels, and find that deep RL algorithms rely heavily on spurious correlations in the rendered environment, demonstrating the need for benchmarks to also consider the challenge of perceptual generalization when training in small, visually homogeneous environments that do not represent real world deployments of agents trained by RL.

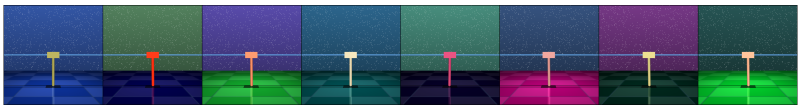
In this work, we propose a new benchmark for generalization in continuous control from images, based on the DMControl environments. It systematically benchmarks visual generalization of RL agents trained in a single, fixed environment, and evaluated on a variety of challenging and visually diverse environments. Section 4.1 introduces the proposed benchmark, Section 4.2 present an additional benchmark for visual RL, and Section 4.3 concludes the chapter by proposing a systematic evaluation of generalization when deploying agents trained in simulation onto real robots perceiving and interacting in unstructured environments.

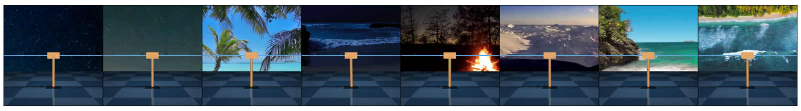
4.1 DMControl Generalization Benchmark

The *DMControl Generalization Benchmark* (DMControl-GB) (Hansen & Wang, 2020) is a benchmark for perceptual generalization in continuous control from visual observations, based on DMControl (Tassa et al., 2018). It features a variety of challenging and diverse tasks reflecting real applications such as robotic locomotion, manipulation, and grasping, and measures generalization to diverse visual changes of incremental difficulty. In DMControl-GB, agents are trained in a single, fixed environment and their generalization is evaluated on a wide variety of test environments. Evaluation is designed to mimic real world deployment of RL agents, and include visual changes such as color randomization and non-stationary video backgrounds.

To develop RL agents that successfully navigate and operate in the real, unstructured world, agents need to perceive and interpret an ever-changing environment that may not have been previously seen during training. For example, imagine a cooking robot that is trained to cook meals for the elderly in their own homes. If the aim is to learn a general policy with satisfactory behavior in any kitchen that is it deployed in, it will inevitably be necessary to extrapolate and generalize learnings from a finite set of training environments to the kitchen that it eventually is expected to act in, since it is infeasible to train a robot on every possible kitchen. Even within a single kitchen the agent can be exposed to


(a) `color_easy`


(b) `color_hard`


(c) `video_easy`

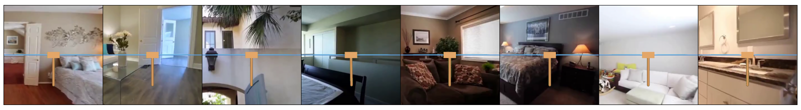

(d) `video_hard`

Figure 4.3: **Samples from DMControl-GB test environments.** Agents are trained in a fixed environment, and the generalization ability of agents is evaluated on the test distributions shown in (a-d). Environments (a-b) randomize the color of background, floor, and the agent itself, while (c-d) replaces the background with natural videos. In (c), only the skybox is replaced, whereas the entirety of the background is replaced in (d).

a multitude of environmental changes, such as those caused by its own interactions, but also external factors such as humans, other robots, and natural changes such as lighting conditions and tear on tools and the robot's joints. While true generalization is a long-term goal of the artificial intelligence community, DMControl-GB aims to provide a test-bed for visual generalization that is achievable in the near future.

We systematically evaluate performance under distributional shift: agents are trained in the environments proposed by Tassa et al. (2018) and evaluated in two distinct categories of environments: (i) *randomized colors*, where the colors of the floor, background, and the agent itself is randomized; and (ii) *video backgrounds*, where the static background of DMControl is replaced by continuously changing backgrounds collected from natural videos. The evaluation procedure is visualized in Figure 4.2, and samples from the test environments are shown in Figure 4.3. Each of the two distribution shifts (colors and videos) are offered in two varieties: *easy* and *hard*. For the randomized color environments, the two varieties differ in color diversity with *hard* being significantly more difficult. In the video background environments, the *easy* variant features natural videos with diverse colors and textures, but limited non-stationarity, and only the skybox is replaced by video. In the *hard* variant, both floor and skybox is replaced by video, and videos are highly non-stationary, i.e. significant changes in scene and camera pose over the course of an episode. Each set of randomized color environments feature 100 unique environments, while the video sets contain a total of 100 videos; *easy* videos are collected in the wild, and *hard* videos are a subset of the RealEstate10K (Zhou et al., 2018) dataset.

As a whole, DMControl-GB poses a significant generalization challenge since agents are only trained on *a single environment* with no visual diversity, and are expected to

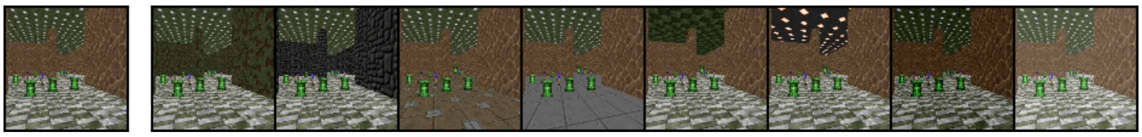


Figure 4.4: **Samples from the modified CRLMaze environments.** Agents are trained in a fixed environment (left) and evaluated in environments with unseen textures of floor, walls, and ceiling, as well as changing light conditions (right). Sample images correspond to actual observations.

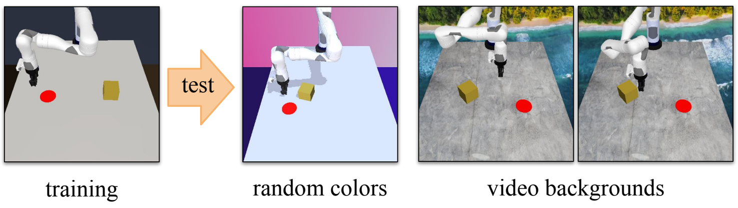


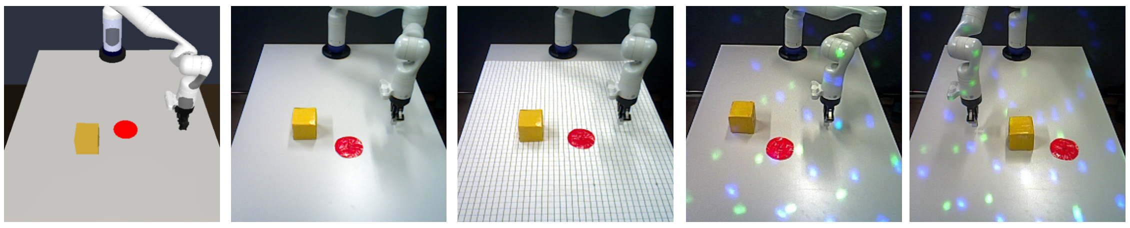
Figure 4.5: **Robotic manipulation evaluation.** Agents are trained in a fixed environment and evaluated on a wide range of unseen and visually diverse test environments, including environments with randomized colors and video backgrounds. Sample images correspond to actual observations.

generalize to a diverse set of environments with visual characteristics that differ significantly from the training environment, unlike e.g. the recently proposed ProcGen benchmark (Cobbe et al., 2019b) that both trains and evaluates on a large set of procedurally generated video games. We therefore believe that DMControl-GB fills an important gap in visual RL, and envision it as a generally useful benchmark for future research on generalization. Documentation and open-sourced implementation of DMControl-GB is available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark, and the DMControl-GB tasks and implementation is discussed in detail in Appendix A.

4.2 Generalization in Navigation

While DMControl and DMControl-GB make great benchmarks for measuring sample-efficiency and generalization in continuous control tasks, their tasks inherently require limited scene understanding. To better evaluate generalization in tasks that require scene understanding, we propose an additional generalization benchmark based on CRLMaze (Lomonaco et al., 2019), a 3D navigation task based on the ViZDoom platform (Wydmuch et al., 2018).

In CRLMaze, agents are to navigate a maze and collect objects; there is a positive reward associated with green columns, and a penalty associated with red lanterns. The task is time-constrained and the agent receives a small penalty at each time step. At each step, the agent can move either forward, backward, left, or right, and the initial position of the agent is sampled from a set of 20 fixed points. As in DMControl, agents are by default trained and evaluated in the same fixed environment. We therefore propose a new set of test environments that differ visually from the training environment, and choose to evaluate generalization to environments with unseen textures of the floor, walls, and ceiling, as well as changing light conditions. The set of variations have been selected on the basis of visual diversity and to replicate some of the distribution shifts that are likely to be relevant for real-world deployment of embodied agents for navigation. Samples from the training and test environment of the modified CRLMaze task are shown in Figure 4.2.



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

Figure 4.6: **Push: robotic manipulation in real world.** Samples from the *push* robotic manipulation task. The task is to position the gripper of the robotic arm at the location of a physical red disc. Agents are trained in setting (a) and evaluated in settings (b-d). Sample images correspond to actual observations. In the real world, the agent receives observations from an uncalibrated camera. Samples from the *reach* task are shown in Figure 4.7.

4.3 From Simulations to Real Robots

Evaluation of the generalization ability of agents under artificial distribution shifts in simulated environments is a highly accessible and inexpensive benchmark for measuring algorithmic improvements. However, we are ultimately interested in developing agents that solve real-world problems with vision-based RL. Therefore, it is natural to consider simulations as a test-bed for the development of algorithms, with the purpose of eventually transferring trained agents to the real world – commonly referred to as *sim2real*. To allow for successful transfers, simulations are generally designed to closely mimic the intended deployment environment, both in terms of visuals and the underlying dynamics.

In the context of robotics, MuJoCo (Todorov et al., 2012) is a popular and highly modular physics engine for training agents in simulation. Using MuJoCo and a simulated Kinova Gen 3 robotic arm, we develop an environment for robotic manipulation tasks from visual observations, and consider the following two tasks: (i) *reach*, a simple control task in which the robotic arm needs to position its gripper at the location of a physical red disc; and (ii) *push*, an object manipulation task in which the robotic arm needs to push a yellow cube to the location of the red disc. In both tasks, the robot is controlled using end-effector movement only, the action space is restricted to the plane, i.e. no vertical movement, and we randomize the initial configuration of robotic arm, goal, and cube at each reset. At each step, the agent receives a penalty proportional to the distance between gripper and goal in the reach task, and between cube and goal in the push task. When the goal position of the task is reached, the agent receives a positive reward at each step until the episode terminates. We use an episode length of 50 for each of the two tasks. As in DMControl, we train in a fixed environment and evaluate generalization to test environments with randomized colors and video backgrounds. To simulate deployment on a real robot, we additionally perform random perturbations of camera, lighting, and texture during evaluation only. The evaluation protocol is visualized in Figure 4.5.

Besides evaluating agents in simulation, we also transfer trained policies to the real world, and deploy on a physical Kinova Gen 3 robotic arm with communication handled through a custom-built ROS interface. Samples from the two robotic manipulation tasks in simulation and the real world are shown in Figure 4.6 and Figure 4.7. We first consider a direct policy transfer, where we deploy in a real environment that approximately resembles the simulation, which we shall denote the *default transfer*. In the sim2real setting,

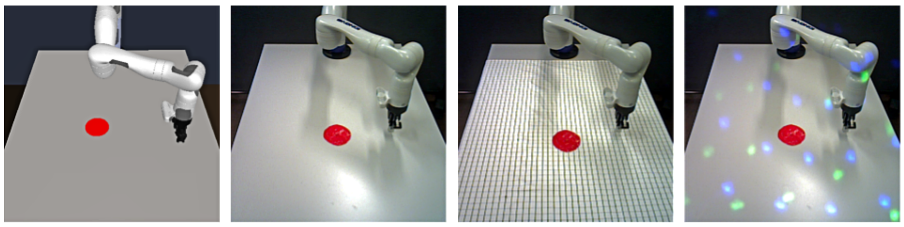

(a) Simulation. (b) Default transfer. (c) Table cloth. (d) Disco lights.

Figure 4.7: **Reach: robotic manipulation in the real world.** Samples from the *reach* 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). Sample images correspond to actual observations. In the real world, the agent receives observations from an uncalibrated camera. Samples from the *push* task are shown in Figure 4.6.

a policy needs to generalize to subtle visual differences between simulation and the real world, such as camera pose, object dimensionality, lighting, and texture. Deployment also presents further challenges in terms of dynamics, since the simulation is often a coarse and inaccurate model of the real environment. In our evaluations, we make no attempt to address disparities in dynamics such as action calibration, object dimensionality and mass, as well as friction and subtle inconsistencies in the movement of the robotic arm, and the policy takes observations directly from an uncalibrated camera.

For a policy to truly generalize to the real setting, it should function in unstructured environments without extensive engineering efforts. Therefore, we also propose to evaluate policy transfer to two other real settings: (i) a *table cloth* that increases friction substantially and has a pattern that distracts visually; and (ii) continuously moving *disco lights* that serve to provide distraction and environmental non-stationarity. In each test environment, we benchmark generalization to 5 distinct initial configurations, and repeat each evaluation 5 times for a total of 75 real-world evaluations per algorithm that we benchmark. Videos of the reach and push tasks, both in simulation and the real world, are available at https://nicklashansen.github.io/PAD and https://nicklashansen.github.io/SODA.

The following two chapters describe our proposed methods and provide extensive experimental results that demonstrate their effectiveness. Chapter 5 describes *Policy Adaptation during Deployment*, and Chapter 6 describes *Soft Data Augmentation*. In Chapter 7, we discuss the implications of our work more broadly, and provide our perspective on the future of visual RL.

5 Policy Adaptation during Deployment

In most practical applications of reinforcement learning, a policy learned in one environment needs to be deployed in another, potentially quite different environment. It may be that a policy trained in simulation needs to be deployed on a real robot, or it may be that an agent trained to perform a task under a finite number of environmental conditions needs to generalize to a set of unseen conditions. In either scenario, generalization across different environments is known to be challenging. A natural solution could be to naïvely train a policy from scratch in each new environment, but this is extremely inefficient considering that the new environment likely contains elements of what the agent has seen previously. A less naïve solution could be to fine-tune a learned policy in each of the encountered environments, but this cannot be done if the new environment offers no reward signal (e.g. the real world), and constructing a new reward signal can be difficult.

In this chapter, we tackle the problem of generalization and policy transfer in visual RL from a novel perspective: adapting a policy pre-trained in one environment to an unknown environment *without* any reward signal nor previous experience in that environment. We do this by leveraging self-supervision as an intrinsic training signal during deployment, and rapidly adapt policies to their new environments through interaction *within a single episode*. We propose our method for policy adaptation during deployment in Section 5.1, and present our experimental results in simulation and on a real robot in Section 5.2.

5.1 Method

In this section, we describe our proposed Policy Adaptation during Deployment (PAD) method for generalization in visual RL (Hansen et al., 2020). PAD is a general framework for self-supervised policy transfer and can be implemented on top of any policy network and standard deep RL algorithm (both on-policy and off-policy) that can be described by minimizing an RL objective $\mathcal{L}_{RL}(\theta)$ using gradient descent and optimizing w.r.t. a collection of network parameters θ that parameterize the policy. We here describe PAD as an algorithm-agnostic framework, and delay discussion of implementation details specific to our experiments to Section 5.2.

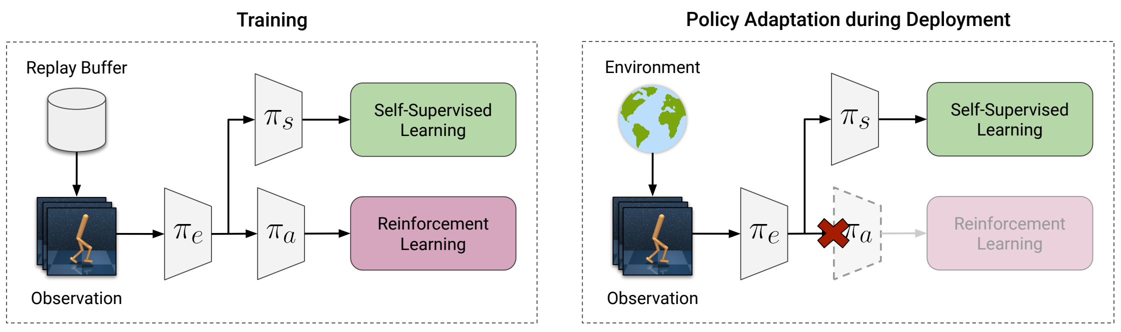


Figure 5.1: **Overview of method.** *Left:* Training before deployment. Observations are sampled from a replay buffer for off-policy methods and are collected during roll-outs for on-policy methods. We optimize the RL and self-supervised objectives jointly. *Right*: Policy adaptation during deployment. Observations are collected from the test environment online, and we optimize only the self-supervised objective.

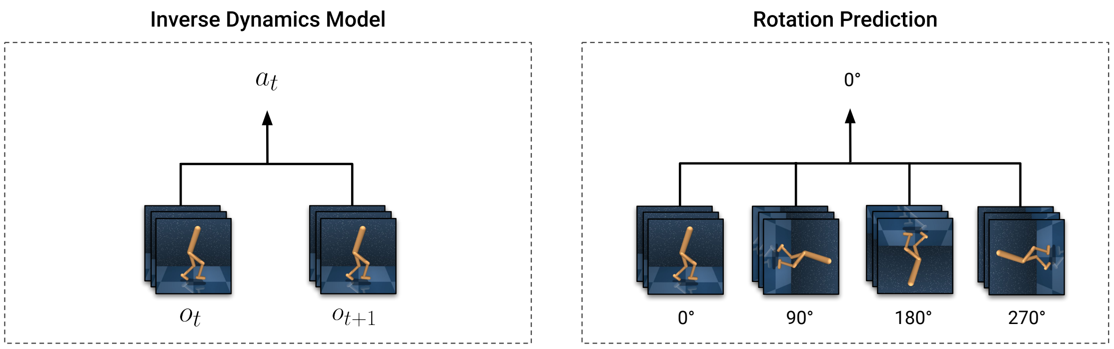


Figure 5.2: **Self-supervised tasks.** *Left:* Inverse dynamics modeling. Given two consecutive observations the action taken in between them is predicted. *Right*: Rotation prediction as proposed by Gidaris et al. (2018) and discussed in Section 2.2. An observation is randomly rotated by one of 0, 90, 180, or 270 degrees and the task is to determine which one of these four ways the observation was rotated.

5.1.1 Network Architecture

A core design choice of PAD is to let a self-supervised prediction network share parameters with the policy network for RL. For a given policy network π parameterized by a collection of parameters θ, we split the network architecture sequentially into two parts $\theta = (\theta_e, \theta_a)$; we denote θ_e as parameters of the *encoder* and denote θ_a as parameters of the policy task head that outputs e.g. a distribution over actions as well as any algorithm-specific values. Given this split of network parameters, we define π_e as the encoder and π_a as the policy task head, such that $\pi(o_t) = \pi_a(\pi_e(o_t))$ for an image observation o_t, and $\pi(o_t) = a_t$ is a distribution over actions for a finite MDP and continuous action values in the continuous case. In the general case, both o_t and a_t may be high-dimensional, but we will in the following drop the bold notation introduced in Chapter 2 to simplify notation. To provide intuition into the network architecture, one can think of π_e as a feature extractor that takes an image observation as input and outputs a feature vector, while π_a is a controller that instead takes a feature vector extracted by the encoder as input and then outputs a probability distribution over actions.

We then let π_s be the self-supervised prediction head with its own collection of parameters θ_s, and we let the output of π_e be the input of π_s, such that π_a and π_s share an encoder π_e. The encoder is generally implemented as a convolutional neural network, but can in principle consist of any parameterized neural network layers. The goal of our method is to update the shared encoder π_e at test-time using gradients from the self-supervised task, such that π_e extracts feature representations that are better aligned with the new environment. Since π_e is shared between the two tasks, we can expect the self-supervised updates to influence the decision-making of π_a through the finetuned representation. Figure 5.1 provides an overview of our method and architecture.

5.1.2 Self-Supervised Tasks

While our proposed framework makes no assumptions about the choice of self-supervised task, in this thesis we primarily consider adaptation using one of two distinct self-supervised tasks: (i) inverse dynamics modeling; and (ii) rotation prediction as proposed by Gidaris et al. (2018). Figure 5.2 illustrates the two tasks. The inverse dynamics are modeled as a simple prediction task in feature space: two consecutive observations o_t and o_{t+1} are encoded by π_e, and the task is then for π_s to predict a_t given a concatenation of features from the two observations. We can formally define the inverse dynamics objective as

$$\mathcal{L}_{ss}^{idm}(\theta_s, \theta_e) = \ell\big(a_t,\ \pi_s\left(\pi_e(o_t), \pi_e(o_{t+1})\right)\big)\,. \tag{5.1}$$

For continuous actions, ℓ is the mean squared error between prediction action and ground-truth action, whereas in the discrete case ℓ is a cross-entropy loss between the predicted action distribution and the one-hot encoded ground-truth action. Rotation prediction is implemented as in Gidaris et al. (2018); an input image is randomly rotated by one of 0, 90, 180, or 270 degrees, and the task is then for π_s to predict the ground-truth rotation from the features extracted by π_e, which we model as a classification problem that can be optimized using a cross-entropy loss. The rotation prediction objective can similarly be defined as

$$\mathcal{L}_{ss}^{rot}(\theta_e, \theta_s) = \ell\big(\omega_t,\ \pi_s\left(\pi_e(\text{rotate}(o_t, \omega_t))\right)\big),\quad \omega_t \sim \mathcal{W}\,, \tag{5.2}$$

where $\mathcal{W}$ is a uniform distribution over possible rotations, ω_t denotes a rotation, and rotate is a function that rotates an input observation o_t by ω_t. We optimize π_s and π_e jointly by propagating errors from the self-supervised task through both parts of the network. Empirically, we find that inverse dynamics modeling works well for motor control tasks where there is a close relationship between observations and actions, while we find that rotation prediction works better for navigation tasks that require scene understanding, which is in line with previous work (Hendrycks et al., 2019b; Doersch & Zisserman, 2017).

5.1.3 Training and Deployment

Our method consists of the following two phases: initial policy learning in a fixed environment (denoted the *training* phase), and self-supervised policy adaptation during deployment in a previously unseen environment (denoted the *deployment* or *test* phase). During training, we learn a policy jointly together with the self-supervised task such that both tasks share an encoder, similar to other works on multi-task learning (Baxter, 1996; Argyriou et al., 2006; Pathak et al., 2016; Jaderberg et al., 2017; Doersch & Zisserman, 2017; Zamir et al., 2018; Hendrycks et al., 2019b; Yarats et al., 2019; Sun et al., 2020; Laskin et al., 2020b; Stooke et al., 2020). Training signals are trivial to obtain for the self-supervised task, and it is assumed that the reward signal is well-defined in the training environment, e.g. a simulation. The training phase then corresponds to the following optimization problem:

$$\min_{\theta_e, \theta_a, \theta_s} \mathcal{L}_{RL}(\theta_a, \theta_e) + \alpha \mathcal{L}_{ss}(\theta_s, \theta_e)\,, \tag{5.3}$$

where $\alpha > 0$ is a hyper-parameter that controls the auxiliary task weighting.

During deployment, we can no longer assume access to a reward signal, since the agent may be deployed e.g. in the real world. We can however still update both θ_e and θ_s by optimizing the self-supervised objective $\mathcal{L}_{ss}$ using observations collected through interaction in the new environment. Empirically, we only find negligible difference with keeping θ_s fixed at test-time, so we update both since gradients have to be computed regardless. As new observations are obtained at each step in the environment, we make one gradient step (using the self-supervised objective) per environment step until the episode terminates. We find it sufficient to make self-supervised updates using only the most recently arriving observation, i.e. we maintain no replay buffer and update purely in an online manner. To reduce variance in updates made during deployment, one can dynamically obtain a batch of training data by copying the current observation and applying a stochastic data augmentation. While PAD has no explicit mechanism for preventing catastrophical forgetting during deployment, we do not find it problematic in our experiments; we discuss this further in Chapter 7. Finally, we summarize the deployment phase of PAD in Algorithm 3, and in the following section we discuss our experimental setup and results extensively.

Algorithm 3 Policy Adaptation during Deployment (PAD) – *Deployment Phase*

```
    θ: network parameters pre-trained in source environment
    α: auxiliary task weight
    $\mathcal{T}$: stochastic data augmentation
1:  for each environment step do
2:      $a_t \sim \pi^t_\theta(o_t),\ \ \theta^t = (\theta^t_e, \theta^t_s)$        ▷ Sample action from current policy
3:      $o_{t+1} \sim p(o_{t+1} \mid o_t, a_t)$                    ▷ Take environment step
4:      $o'_t,\ o'_{t+1} = t(o_t),\ t(o_{t+1}),\ \ t \sim \mathcal{T}$      ▷ Optionally augment observations
5:      $\theta^{t+1}_e = \theta^t_e - \alpha \nabla \mathcal{L}_{ss}(o'_t, a_t, o'_{t+1}; \theta^t_e, \theta^t_s)$   ▷ Update shared encoder
6:      $\theta^{t+1}_s = \theta^t_s - \alpha \nabla \mathcal{L}_{ss}(o'_t, a_t, o'_{t+1}; \theta^t_e, \theta^t_s)$   ▷ Update self-supervised task head
```

5.2 Experiments

In this thesis, we investigate how well an agent trained directly from image observations in one environment generalizes to a variety of unseen, but similar, test environments. During evaluation, agents have no access to a reward signal and are expected to generalize without previous trials or privileged information about the test environments that they are to be deployed in; it can therefore be considered a *zero-shot* generalization problem.

We evaluate PAD and a number of strong baselines extensively on (i) continuous control tasks from DMControl Generalization Benchmark, our proposed benchmark based on DeepMind Control suite (Tassa et al., 2018); (ii) a novel navigation benchmark based on CRLMaze (Lomonaco et al., 2019) and ViZDoom (Wydmuch et al., 2018); and (iii) robotic manipulation tasks both in simulation and on a real robotic arm. We evaluate performance on the environments in which the agents are trained, and systematically benchmark generalization to a number of environmental changes, including both stationary changes (e.g. colors, objects, textures, lighting, dynamics) and non-stationary changes (natural videos backgrounds). An open-sourced implementation of PAD is made available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark and a webpage with video results from our experiments are available at https://nicklashansen.github.io/PAD.

5.2.1 Implementation Details

We opt for a similar experimental setup for each of the three benchmarks, and adopt hyper-parameters from previous work whenever applicable. In this section, we first describe the implementation details that relate to our DMControl-GB experiments, and then proceed to describe the modifications we make to adapt our implementation to the two additional benchmarks.

We implement PAD using a Soft Actor-Critic (SAC) (Haarnoja et al., 2018b) agent as base algorithm, and we adopt both network architecture and hyper-parameters from Yarats et al. (2019) with minor modifications. The network architecture is illustrated in Figure 5.3, and hyper-parameters are detailed in Table 5.1. Our network consists of an encoder π_e with 8 convolutional layers shared between the self-supervised task π_s and the actor-critic π_a of SAC. Each branch consists of 3 non-shared convolutional layers, a linear projection, and a layer normalization, followed by task heads modeled as multilayer perceptrons (MLP) with 3 fully-connected layers. By task head, we refer to the self-supervised prediction head, as well as task-specific prediction heads of SAC, e.g. policy and value functions. Observations are stacks of the k most recent frames ($k = 3$ for DMControl-GB), where each frame is a 100×100 RGB image rendering. Network outputs depend on the task and learning algorithm. As the action space of DMControl-GB is continuous, the

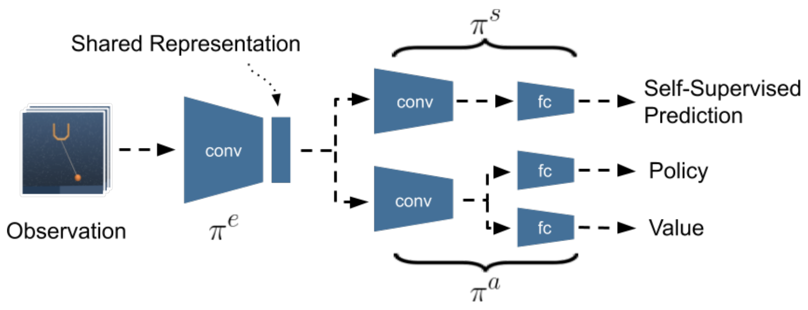


Figure 5.3: **Network architecture.** Architecture for the DMControl-GB, CRLMaze, and robotic manipulation experiments. π^s and π^a use a shared encoder π^e. Observations are stacks of 100×100 colored frames. Implementation of policy and value function depends on the choice of learning algorithm.

policy learned by SAC outputs the mean and variance of a Gaussian distribution over actions. For details on the SAC algorithm, the reader is referred to Section 2.1.4, Haarnoja et al. (2018a), and Haarnoja et al. (2018b).

As in previous work (Laskin et al., 2020b;a), we apply a time-consistent random crop to observations. Random cropping is data augmentation that is commonly 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. We randomly crop observations to dimensions 84×84 (down from 100×100), and apply the same crop to each frame in a stack to preserve spatio-temporal information in the observation, whereas different crops are applied to each observation in a batch. When learning an inverse dynamics model, we apply two different crops to the consecutive observations (o_t, o_{t+1}) used to predict the ground-truth action a_t. During deployment, we create a batch of 32 copies of the most recent observation, and apply random cropping to produce a batch of augmented samples, which we then use to make a single gradient step with the self-supervised objective per step in the environment. When using the policy for inference, we simply apply a 84×84 center-crop to observations before passing it into the network. We provide pseudo-code and further implementation details in Appendix A and Appendix B.

In the robotic manipulation experiments, we employ a similar experimental setup as in DMControl-GB, but opt for observations that instead are a stack of $k = 1$ frames (down from $k = 3$) to improve transfer to environments with different dynamics than the training environment. In the CRLMaze experiments, we use Advantage Actor-Critic (A2C), a synchronous variant of the A3C algorithm proposed by Mnih et al. (2016), as base algorithm, and we apply the same network architecture as for DMControl-GB, but use a smaller encoder π_e that consists of just 6 convolutional layers. In all of our experiments, each convolutional layer uses 32 learnable filters. Like DMControl-GB, our robotic manipulation experiments use a continuous action space. CRLMaze has a discrete action space and the policy learned by A2C thus learns a softmax distribution over actions. By default, we use an inverse dynamics model (which we abbreviate as *IDM* for clarity) as self-supervised task for DMControl-GB and robotic manipulation, and rotation prediction (which we abbreviate as *Rot*) for CRLMaze. Table 5.1 details hyper-parameters used for our DMControl-GB and robotic manipulation experiments, and Table 5.2 details hyper-parameters used for CRLMaze.

We discuss our empirical evaluations in the following. Section 5.2.2 describes our evaluations on DMControl-GB, while Section 5.2.3 and Section 5.2.4 discuss our CRLMaze and robotic manipulation evaluations, respectively.

Table 5.1: **DMControl-GB and robotic manipulation.** Hyper-parameters for PAD and baselines in the DMControl-GB and robotic manipulation experiments.

Hyper-parameter	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 5.2: **CRLMaze.** Hyper-parameters for PAD and baselines in the CRLMaze experiments; uses hyper-parameters from Table 5.1 whenever applicable. We employ rotation prediction as default auxiliary task.

Hyper-parameter	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

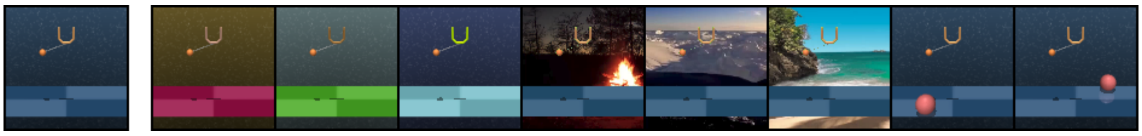


Figure 5.4: **DMControl Environments.** Training and test environments of the PAD experiments on DMControl-GB and the additional visual distractor experiments. *Left:* training environment. *Right:* samples from each of the three considered test distributions.

5.2.2 DMControl-GB

DMControl-GB is our proposed benchmark for generalization in continuous control from images. It is a collection of tasks that represent diverse real-world tasks for motor control, and consist of training and test environments that differ visually.

We experiment with a total of 9 tasks from DMControl-GB and measure generalization to four types of test environments: (i) randomized colors, corresponding to the `color_hard` benchmark; (ii) natural videos, corresponding to the `video_easy` benchmark; (iii) visual distractors placed in the scene, an additional benchmark proposed specifically for the PAD experiments; and (iv) the unmodified training environment to benchmark algorithmic performance when trained and evaluation on the same environment. Samples from each of the test environments are shown in Figure 5.4. In all evaluations, methods are evaluated on 100 random initial configurations and we report results across 10 random seeds because RL experiments are susceptible to high variance. 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 from the benchmark. We select tasks for the benchmark on the basis of task diversity as well as the success of previous work on RL from image inputs in DMControl (Yarats et al., 2019; Laskin et al., 2020b;a; Kostrikov et al., 2020; Stooke et al., 2020).

Table 5.3: **Randomized colors.** Episodic return in test environments from DMControl-GB with randomized colors, mean and standard deviation of 10 random seeds. Best method on each task is in bold font, and best method in a comparison between +IDM with and without PAD is highlighted in blue. PAD improves generalization in 15 out of 16 instances.

					10x episode length	
`color_hard`	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**

PAD is implemented using an IDM as self-supervised task, as we find that learning a model of the environment dynamics works well for continuous control tasks that rely on fine-grained motor control; for completeness, the choice of self-supervision is ablated. In our main results, we compare PAD to a number of strong baselines from recent literature on vision-based RL: (i) SAC with no algorithmic changes, which we will denote *SAC*; (ii) SAC trained with domain randomization on the `color_easy` environment distribution, denoted *SAC+DR* for brevity; (iii) PAD trained jointly with the IDM self-supervised task but without any test-time adaptation, denoted *SAC+IDM*. Our proposed method, PAD, trains an IDM jointly and adapts the representation at test-time; we will denote this method as *SAC+IDM (PAD)* in the following. We apply a random cropping data augmentation to PAD and baselines, since we find methods to not converge on all tasks without it. Hence, the unmodified SAC baseline is in fact equivalent to RAD (Laskin et al., 2020a) that applies random cropping to SAC. We further emphasize that the domain randomization (DR) baseline is trained on the easier `color_easy` distribution as we find the method to not converge on harder training distributions in a majority of environments. Hence, the DR baseline is trained on a distribution similar to that of the `color_hard` test benchmark, but with a lower variance in colors. Refer to Appendix A for more details.

Randomized colors. To reliably deploy agents trained by visual RL in an environment that differs from their training environment, the agents need to be robust to subtle and superficial changes in visuals such as color and lighting conditions. While colors are relatively easy to randomize during training, it can make the RL optimization difficult and even result in an agent failing to learn at all. As such, it is natural to ask whether robustness to shifts in an environment's colors can be achieved without domain randomization. We systematically evaluate the generalization of agents to environments with randomized colors from the `color_hard` distribution of DMControl-GB, and report the results in Table 5.3 (first four columns). We find that PAD consistently improves generalization of the SAC+IDM baseline *in all environments considered*, and exceeds the performance of domain randomization in a majority of environments. It is observed that, despite training the domain randomization baseline on a distribution of colors that are similar to those seen at test-time (but with lower variance), it performs no better than the vanilla SAC baseline in a majority of tasks. These findings are in line with previous work on generalization in visual RL (Packer et al., 2018), where the authors also find domain randomization to be ineffective

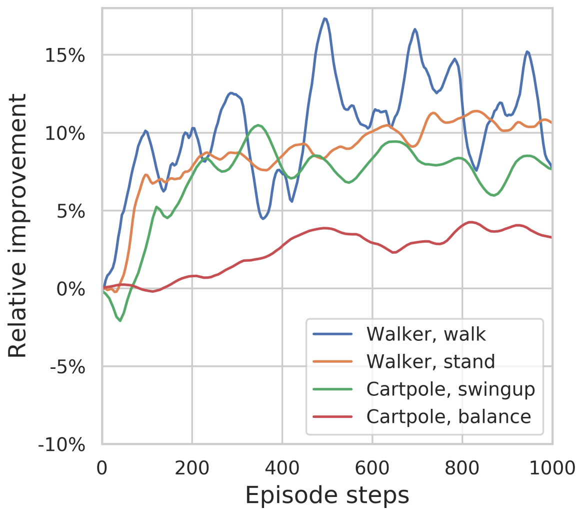


Figure 5.5: **Relative improvement over time.** Relative improvement in instantaneous reward over time for PAD versus the non-adaptive SAC+IDM baseline on the `color_hard` test distribution of DMControl-GB. Average of 5 seeds and 100 initial configurations. Note that baseline performance is already near-optimal on the `cartpole_balance` task.

outside its training distribution. We additionally investigate the rate at which our proposed method adapts to its new environment; Figure 5.5 shows the relative improvement of PAD as compared to the non-adaptive SAC+IDM baseline on four tasks from DMControl-GB on the `color_hard` test distribution. In three of the tasks, most of the improvement from PAD occurs within the first 200 episode steps, which is equivalent to 25-50 frames using appropriate action repeats as defined in the DMControl-GB benchmark and reported in Table 5.1. We see no significant improvement on the `cartpole_balance` task, but we also find that the SAC+IDM baseline is near-optimal before adaptation, so there is little room for improvement.

Long-term stability of PAD. The DMControl and DMControl-GB benchmarks by default consider episodes of 1000 steps (excluding action repeat), but real-world deployment of RL agents can potentially consist of longer duration episodes or even infinite time horizon deployment scenarios. While the relative improvement shown in Figure 5.5 indicates stability of the adaptation process around episode durations that are of similar length as the training episodes, this result may not generalize to long-horizon deployments. Therefore, we further evaluate the long-term stability and behavior of PAD on a set of tasks with 10x episode length (10,000 steps), and report the results in Table 5.3 (last two columns). We have excluded two goal oriented tasks from the evaluation, since their long-term performance is solely determined by whether the agent succeeds or not. We find PAD to consistently improve generalization over the SAC+IDM baseline in all tasks considered in the 10x episode length experiments, which suggests that the adaptation process of PAD remains stable in the long term. Perhaps surprisingly, we even find the long-term performance of PAD to exceed that of the default episode length on the `walker_walk` task, yielding a relative improvement of 44% on average, which we conjecture may be because it is a complex locomotion task and subtle inconsistency in representation and observations thus has greater impact on the long-term return. While PAD has no component that explicitly enforces long-term stability of the adaptation process, we find that the self-supervised task gradients are negligible once the representation has been adapted to any changes there might be in the test environment, and no catastrophical forgetting is therefore observed; we elaborate on this phenomenon in Chapter 7.

Table 5.4: **Video backgrounds.** Episodic return in test environments from DMControl-GB with video backgrounds, mean and standard deviation of 10 random seeds. Best method on each task is in bold font, and best method in a comparison between SAC+IDM with and without PAD is highlighted in blue. PAD improves in 7 out of 8 instances.

`video_easy`	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**

Table 5.5: **Visual distractors.** Episodic return in test environments with visual distractors, mean and standard deviation of 10 random seeds. Best method on each task is in bold font, and best method in a comparison between SAC+IDM with and without PAD is highlighted in blue. PAD improves in 3 out of 5 tasks, where the SAC+IDM baseline already has optimal or near-optimal performance in the remaining two tasks.

Visual distractors	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**

Video backgrounds. Environments with natural videos as background pose a particularly difficult adaptation challenge since the environment visuals become non-stationary. While video frames that arrive sequentially are highly correlated, it requires an agent to continuously adapt its representation to pixel-level changes in the video background. We evaluate the generalization of PAD and baselines on the `video_easy` test distribution of DMControl-GB, which features 10 natural non-repeating video backgrounds of various color patterns and textures that are incongruent with the training environment. Results from the experiments are shown in Table 5.4. We find PAD to improve generalization of the SAC+IDM baseline and outperform all other methods in 7 out of 8 tasks. It is worth emphasizing that PAD improves generalization by up to 20% on `ball_in_cup_catch`, and outperforms domain randomization by as much as 60% on the same task and 104% on the `finger_spin` task. We observe that domain randomization performs comparably worse on video backgrounds than color backgrounds, which we conjecture is *not* because the test environment is non-stationary since there is no adaptation involved, but rather because video backgrounds have image statistics that are considerably different from what was seen during training (randomized colors). Our observation is similar to the conclusion of Packer et al. (2018) that find domain randomization to be ineffective beyond its training distribution, and we do in fact find that domain randomization performs no better than (and is often outperformed by) vanilla SAC on the proposed `video_easy` benchmark.

Table 5.6: **Self-supervised tasks.** Ablation on the choice of self-supervised task for PAD on the `color_hard` test distribution of DMControl-GB. All methods use SAC as base algorithm. We compare IDM to CURL (Laskin et al., 2020b) and rotation prediction (Rot) (Gidaris et al., 2018). Best method on each task is in bold font, and best method in a comparison between SAC+IDM with and without PAD is highlighted in blue.

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

Visual distractors. When deploying agents in the real world, it is not unreasonable to expect the agent to encounter objects that were not present in the training environment. Even though the agent may not need to interact with such alien objects, they can still be visually distracting. Therefore – aside from the DMControl-GB experiments – we also consider an additional test setting for PAD: visually distracting objects placed in the scene. Samples of the visual distractors are shown in Figure 5.4, and results are shown in Table 5.5. We find PAD to improve generalization in 3 out of 5 tasks, and observe that the SAC+IDM baseline already has optimal or near-optimal performance in the remaining two tasks, `cartpole_swingup` and `cartpole_balance`. We additionally find that SAC+IDM both with and without PAD generalizes significantly better to visual distractors compared to vanilla SAC and SAC trained with domain randomization. We conjecture that this is because the shared encoder is jointly optimized together with an IDM, and IDMs only encode information that is predictive of actions taken, whereas SAC and SAC+DR more easily overfit to spurious correlations in the environment.

Self-supervised tasks. We explore how much the choice of self-supervised auxiliary task contributes to the overall success of PAD in each of the 9 continuous control tasks from DMControl-GB that we consider. We compare an IDM with two alternative sources of self-supervision: CURL (Laskin et al., 2020b), a recently proposed framework for contrastive representation learning in RL from images, and rotation prediction (abbreviated as *Rot* in the following) as proposed by Gidaris et al. (2018) and described in Section 5.1.2. Generalization results for each of the three self-supervised tasks on the `color_hard` test distribution are shown in Table 5.6. We find self-supervised adaptation using CURL in our proposed PAD framework to improve generalization in 5 out of 9 tasks, outperforming PAD with an IDM in a total of three tasks. It is however also observed that adaptation using CURL actually *decreases* the generalization ability in some tasks, and the non-adaptive CURL baseline is inferior to SAC+IDM in many instances, which would suggest that the representations produced by CURL do not generalize as well as those of an IDM. PAD generally does not improve generalization when using the Rot self-supervised task, but also does not hurt performance significantly. We therefore conclude that PAD is most successful with an IDM on the DMControl-GB benchmark, which we conjecture is because an IDM explicitly learns the relation between observations and actions – a relation that is of particular importance in the fine-grained motor control tasks of DMControl.

Table 5.7: **Learning offline versus learning online.** Ablation on the online learning formulation of PAD on the `color_hard` test distribution of DMControl-GB, mean and standard deviation of 10 random seeds. All methods use SAC as base algorithm. Best method on each task is in bold font. Offline PAD only adapts to individual observations, whereas online PAD is the proposed variant of PAD that does not forget previous updates.

`color_hard`	IDM	IDM (PAD)	IDM (PAD)
		offline	*online*
Walker, walk	406±29	441±16	**468±47**
Walker, stand	743±37	727±21	**797±46**
Cartpole, swingup	585±73	578±69	**630±63**
Cartpole, balance	835±40	796±37	**848±29**
Ball in cup, catch	471±75	490±16	**563±50**
Finger, spin	757±62	767±43	**803±72**
Finger, turn_easy	283±51	**321±10**	304±46
Cheetah, run	121±38	112±35	**159±28**
Reacher, easy	201±32	**241±24**	214±44

Table 5.8: **Keeping θ_s fixed during deployment.** Ablation on keeping θ_s fixed at test-time on the `color_hard` test distribution of DMControl-GB, mean and standard deviation of 10 random seeds. All methods use SAC. *IDM (PAD, fixed θ_s)* considers a variant of PAD where θ_s is fixed at test-time, whereas *IDM (PAD)* denotes the default usage of PAD in which both θ_e and θ_s are adapted at test-time using the self-supervised objective.

`color_hard`	IDM	IDM (PAD, fixed θ_s)	IDM (PAD)
Walker, walk	406±29	452±38	**468±47**
Walker, stand	743±37	**802±41**	797±46
Cartpole, swingup	585±73	623±57	**630±63**

Learning offline versus learning online. We hypothesize that one of the main reasons for why PAD successfully adapts to changes in the test environments – even when the environments are non-stationary – is because input frames arrive sequentially in a stream of highly correlated observations. Because of this sequential arrival of frames, adapting the representation of the agent to a sequence of consecutive observations inevitably also adapts the representation to observations arriving in the immediate future, at least to some degree. While it is difficult to formally quantify this effect, we can measure the benefit of the *online* learning formulation of PAD empirically. To do this, we consider an additional ablation: a variant of PAD that instead learns *offline*, i.e. adapts to each observation individually and resets network parameters to the pre-trained weights after each environment step. Intuitively, this offline variant of PAD can be interpreted as an algorithm that ”forgets” any information about the test environment it has acquired prior to receiving the most-recent observation. We report the results of this ablation in Table 5.7. We find that PAD benefits substantially from learning from the sequentially arriving observations, outperforming the offline variant of PAD in 7 out of 9 tasks. Adapting to just a single observation at each step does however still improve generalization over the SAC+IDM baseline on a number of tasks.

Table 5.9: **Training environment.** Episodic return in the training environment, mean and standard deviation of 10 random seeds. The domain randomization baseline is also evaluated on the fixed training environment in which other methods were trained. *Blind* is an additional baseline that does not have access to visual observations nor proprioceptive states; it only observes its k previous actions. Best method on each task is in bold font, and best method in a comparison between +IDM with and without PAD is highlighted in blue. 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**$\pm$**24**	895 ± 28
Walker, stand	388 ± 10	959 ± 11	928 ± 36	**966**$\pm$**8**	956 ± 20
Cartpole, swingup	132 ± 41	**850**$\pm$**28**	807 ± 36	849 ± 30	845 ± 34
Cartpole, balance	646 ± 131	978 ± 22	971 ± 30	**982**$\pm$**20**	979 ± 21
Ball in cup, catch	150 ± 96	725 ± 355	469 ± 339	**919**$\pm$**118**	910 ± 129
Finger, spin	3 ± 2	809 ± 138	686 ± 295	**928**$\pm$**45**	927 ± 45
Finger, turn_easy	172 ± 27	**462**$\pm$**146**	243 ± 124	**462**$\pm$**152**	455 ± 160
Cheetah, run	264 ± 75	**387**$\pm$**74**	195 ± 46	384 ± 88	380 ± 91
Reacher, easy	107 ± 11	264 ± 113	92 ± 45	**390**$\pm$**126**	365 ± 114

Keeping θ_s fixed during deployment. Another design choice of PAD is to continuously adapt both θ_e and θ_s to the test environment, even though the self-supervised task head π_s is not used by the policy. We now consider a variant of PAD where θ_s is fixed at test-time such that the gradient of the self-supervised objective $\mathcal{L}_{ss}$ as proposed in Section 5.1.3 is only taken w.r.t. θ_e; this variant is equivalent to Algorithm 3 where line 6 is not executed. Generalization results for three tasks on the `color_hard` test distribution of DMControl-GB are shown in Table 5.8. Empirically, we find a negligible difference between updating θ_s and keeping it fixed, and we therefore choose to update both θ_e and θ_s in our proposed variant of PAD since $\nabla_{\theta_s}\mathcal{L}_{ss}$ needs to be computed by back-propagation regardless.

Performance on the training environment. In a majority of prior work on visual RL, agents are trained and evaluated on the same environment, and generalization is therefore not considered in those works. Although such evaluations do not consider generalization, they are still very valuable for benchmarking performance, stability, and sample efficiency of RL algorithms, and for completeness we therefore also evaluate PAD and baselines in this setting. We consider the fixed training environment from DMControl, i.e. the left-most environment shown in Figure 5.4, and report the results in Table 5.9. In addition to the baselines that we provide in previous ablations, we consider an additional baseline as well: a blind agent that does not have access to visual observations nor proprioceptive states, and only observes its k previous actions. Although it is a strikingly naïve baseline, it provides insight into how much visual perception contributes to the success of RL agents in each of the 9 tasks from DMControl. In tasks where the blind agent performs poorly, we expect visual perception to be of particular importance, and likewise if the blind agent achieves a high episodic return on a task, e.g. `cartpole_balance`, then we do not expect visual perception (and consequently PAD) to be that impactful. We find that, while PAD improves the generalization ability of RL agents in novel test environments, performance is virtually unchanged when deployed in the fixed training environment. We conjecture that this is because the representation is already aligned with the environment, and any additional training using the self-supervised task is thus of little impact, which is in line with our findings in the long-horizon experiments shown in Table 5.3. We further

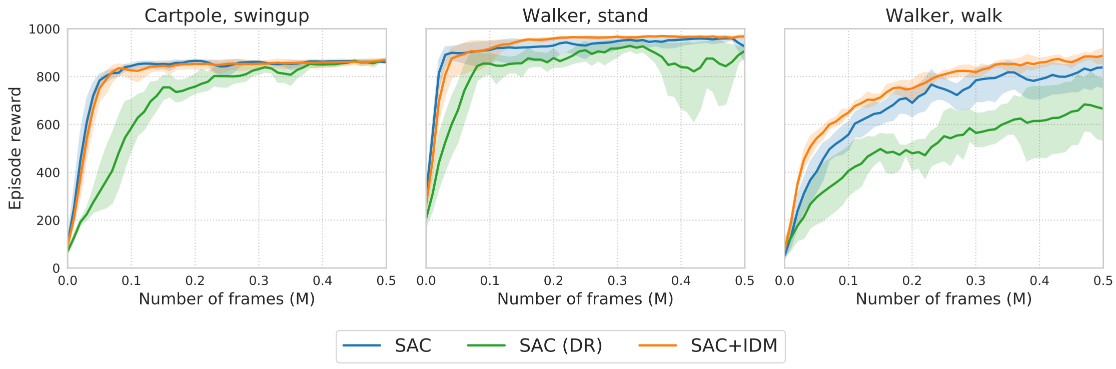


Figure 5.6: **Learning curves.** Learning curves for SAC, SAC trained with domain randomization (denoted *SAC (DR)*), and SAC+IDM on three tasks from DMControl using image observations. Episodic return on the training environment is averaged across 10 random seeds and the 95% confidence intervals are visualized as shaded regions.

report the learning curves for three tasks from DMControl in Figure 5.6. We find SAC and SAC+IDM to exhibit similar sample efficiency and end-performance, whereas SAC trained with domain randomization consistently displays worse sample efficiency, larger variation between random seeds, and converges to sub-optimal performance in some tasks, demonstrating the difficulty of scaling domain randomization.

5.2.3 Modified CRLMaze

CRLMaze (Lomonaco et al., 2019) is a 3D navigation task for ViZDoom (Wydmuch et al., 2018), where an agent is to navigate a maze and collect objects associated with positive and negative rewards (penalties). Readers are referred to Section 4.2 for a description of the CRLMaze environment. We propose a number of modified CRLMaze test environments that differ visually, with changes such as novel textures of the floor, walls, and ceiling, as well as lighting conditions. As in the DMControl-GB benchmark, we train agents in a fixed environment and measure their generalization ability to unseen test environments by episodic return. Concretely, we experiment with the original CRLMaze navigation task and report performance on five distinct types of test environments: (i) unseen wall textures; (ii) unseen floor textures; (iii) unseen ceiling textures; (iv) unseen lighting conditions, where the illumination of the scene is adjusted upwards/downwards; and finally (v) the unmodified training environment as originally proposed by Lomonaco et al. (2019). Samples from each of the test environments are shown in Figure 4.4. In all evaluations, we follow the evaluation procedure of Lomonaco et al. (2019): we evaluate methods on 20 predefined starting positions and measure the mean episodic return across all starting positions, and repeat our experiments with 10 random seeds because the CRLMaze environment is susceptible to high variance.

We implement PAD using rotation prediction (abbreviated as *Rot* in the following) as self-supervised task, as it has been shown to capture scene understanding in previous work (Hendrycks et al., 2019b; Doersch & Zisserman, 2017), which we find to be especially useful for navigation tasks. In line with the DMControl-GB experiments, we compare our method to the following baselines: (i) the A2C base algorithm with no modifications, but using random cropping; (ii) A2C trained with domain randomization (denoted *A2C+DR*; (iii) A2C trained with an IDM as auxiliary task (denoted *A2C+IDM*); (iv) A2C trained with rotation prediction as auxiliary task (denoted *A2C+Rot*); and we additionally compare to (v) a random agent, i.e. a policy with a randomly initialized policy network, which we

Table 5.10: **Modified CRLMaze.** Episodic return of PAD and baselines in the modified CRLMaze environments. All methods use A2C as base algorithm. *Random* uses a randomly initialized policy network. We report mean and standard error of 10 random seeds. Best method on each task is in bold font, and best method in a comparison between +IDM with and without PAD is highlighted in blue. PAD with rotation prediction improves generalization *in all considered test environments* and outperforms both A2C and domain randomization by a large margin, while also hurting minimally in the training environment.

CRLMaze	Random	A2C	+DR	+IDM	+IDM (PAD)	+Rot	+Rot (PAD)
Training	$-868{\pm}34$	$371{\pm}198$	$-355{\pm}93$	$585{\pm}246$	$-416{\pm}135$	**729±148**	$681{\pm}99$
Walls	$-870{\pm}30$	$-380{\pm}145$	$-260{\pm}137$	$-302{\pm}150$	$-428{\pm}135$	$-206{\pm}166$	**−74±116**
Floor	$-868{\pm}23$	$-320{\pm}167$	$-438{\pm}59$	**−47±198**	$-530{\pm}106$	$-294{\pm}123$	$-209{\pm}94$
Ceiling	$-872{\pm}30$	$-171{\pm}175$	$-400{\pm}74$	$166{\pm}215$	$-508{\pm}104$	$128{\pm}196$	**281±83**
Lighting	$-900{\pm}29$	$-30{\pm}213$	$-310{\pm}106$	$239{\pm}270$	$-460{\pm}114$	$-84{\pm}53$	**312±104**

denote *Random*. While the random agent is a very naïve baseline, it serves well as a reference point for readers unfamiliar with the CRLMaze task. The domain randomization baseline is trained on a distribution of 56 combinations of textures with diverse colors and patterns, and its training distribution partially overlaps with the test environments to make transfer easier. We additionally find it necessary to train the A2C+DR baseline for twice as many episodes as all other methods in order to converge to a satisfactory policy.

Results. Generalization results from the modified CRLMaze environments are shown in Table 5.10. While all methods perform much better than random regardless of the test environment, we do find PAD with rotation prediction to improve generalization to all considered test environments, outperforming both A2C and the A2C+DR (domain randomization) baselines by a large margin. We find PAD to be especially powerful for adaptation to superficial changes such as novel lighting conditions, where the mean episodic return is increased from -84 to 312. As in the DMControl-GB experiments, we find PAD to hurt minimally on the training environment when using an appropriately selected self-supervised auxiliary task, and we find sub-optimal choices of self-supervision to be unsuitable for PAD. Our results thus show that the success of PAD is highly dependent on the self-supervised task, but it remains non-trivial to select appropriate self-supervision for new tasks that practitioners may encounter. So while PAD largely removes the need for trial-and-error in the design of domain randomization, practitioners still have to rely on their intuition for which self-supervised task works better for a given RL task. We leave the act of automating the choice of self-supervision for future work, but discuss possible avenues for research on automation in Chapter 7.

5.2.4 Robotic Manipulation

DMControl-GB and the modified CRLMaze benchmarks offer great ways to empirically study the effectiveness of PAD under a diverse set of visual distributional shifts. However, we are ultimately interested in developing algorithms for generalization that provide real benefits to the deployment of agents trained by RL. To benchmark the effectiveness of PAD in real-world deployments, we consider the *sim2real* problem setting and transfer our method and baselines from a simulated environment to a real Kinova Gen3 robotic arm. The sim2real problem setting is of particular interest since the real world not only differs visually, but also has different dynamics than the training simulation. We consider the two tasks, *reach* and *push*, described in Section 4.3, and use an IDM as the self-

Table 5.11: **Real robot deployment.** Success rate of PAD and baselines when deployed on a *real* robotic arm. Best method on each task is in bold font, and best method in a comparison between +IDM with and without PAD is highlighted in blue.

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%**

supervised task. We compare PAD to the same baselines as in the previous experiments: (i) the unmodified SAC base algorithm, but with random cropping; (ii) SAC trained with domain randomization; and (iii) SAC trained jointly with an IDM but without PAD. The domain randomization baseline only randomizes color of table and background, as we find it to otherwise not converge for the push task. During deployment on the real robot, we evaluate each method for 25 trials (episodes) across 5 pre-defined initial configurations, and we reset the robotic arm after each trial. We further evaluate generalization only to changes in dynamics by considering a variant of the simulated environment in which object mass, dimensions, and friction, as well as robotic arm mount position and end-effector velocity is modified. We consider each dynamics change both individually and jointly, and evaluate the success rate across 50 unique initial configurations with the robot reset after each trial.

Real robot deployment. Results from the real robot experiments are shown in Table 5.11. It is observed that all methods achieve a 100% success rate on the default transfer setting of *reach*. We find the random cropping data augmentation to be a big contributor to the transfer success, and conjecture that the translation invariance that random cropping provides is especially helpful because we do not calibrate the camera in our experiments. In the remainder of the settings, PAD improves the success rate of the non-adaptive SAC+IDM baseline and outperforms both vanilla SAC and SAC trained with domain randomization in all but one instance. We find agents to benefit the most in challenging transfer settings such as changes in dynamics and disco lights, which is also the settings in which the non-adaptive baselines are impacted the most.

Simulated robot deployment. To better isolate the impact that dynamics mismatches have on transfer success, we consider an additional experiment in the simulated environment for robotic manipulation; results are shown in Table 5.12. PAD improves generalization to both changes in object dynamics as well as end-effector velocity, and both SAC+IDM and consequently PAD are relatively unaffected by changes in the mount position whereas the SAC and SAC+DR baselines degrade substantially. Consistent with the real robot experiments, PAD is found to be most effective under non-trivial changes in dynamics. In the hardest simulated setting, *Push (all)*, we find PAD to nearly recover the success rate of the individual settings, improving over the SAC+IDM baseline by as much as 28%. Our simulated dynamics experiments would therefore suggest that PAD can be an effective adaptation mechanism for changes in both visuals and dynamics, which are commonly encountered in the sim2real problem setting.

Table 5.12: **Simulated robot deployment.** 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, as well as robotic arm mount position and end-effector velocity. Best method on each task is in bold font, and best method in a comparison between +IDM with and without PAD is highlighted in blue.

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**%

5.3 Summary

In many real applications of RL, a policy trained in one environment (e.g. a simulation) needs to be deployed in another, potentially quite different environment. Generalization across environments is known to be hard, and the problem is only exacerbated by high-dimensional inputs such as images that are increasingly common in RL applications. It is tempting to tackle the problem of generalization as a domain adaptation problem, where we continue training the policy in the new environment until it reaches satisfactory performance, but this cannot be done if the new environment offers no reward signal, and the fine-tuning process itself can be both costly and outright unsafe.

In this chapter, we proposed PAD, a novel approach to generalization in visual RL. While previous work addresses the problem of generalization by learning policies that are invariant to any environment changes that can be anticipated, we formulate an alternative problem setting in visual RL: can we instead *adapt* a pre-trained policy to a new environment *without* any rewards or human supervision. To this end, we explore the use of self-supervision to continue fine-tuning the policy during deployment. Our method makes no assumptions about prior knowledge of environmental changes, and PAD adapts to new environments without any trials or pre-collected data.

We perform an extensive empirical evaluation of our method, and demonstrate that PAD can adapt to a wide variety of test environments using simple self-supervised tasks and observations collected from the environment in an online manner. We find that an inverse dynamics model works well for adaptation in motor control tasks from DMControl-GB and robotic manipulation tasks both in simulation and on a real robot, while rotation prediction works better for the CRLMaze discrete navigation task. In total, our method is shown to improve generalization in 31 out of 36 distinct types of environments, and outperforms domain randomization in a majority of environments. While the current framework relies on prior knowledge on selecting an appropriate self-supervised task for policy adaptation, we see our work as an initial but encouraging step in addressing the problem of adapting policies with visual inputs to unknown environments only through interaction. In the following chapter, Chapter 6, we present *Soft Data Augmentation* SODA), an alternative direction of research for generalization in visual RL.

6 Soft Data Augmentation

Domain randomization, and most recently data augmentation as well, are powerful techniques for learning policies that are invariant to pre-defined factors of variation, such as color, texture, and lighting. The assumption made in these approaches is that the practitioners who are to deploy robust policies can preemptively identify which elements will vary during deployment, and then implement appropriate randomization during training. In practice, it can be difficult to correctly specify all relevant factors of variation, especially when the agent is to be deployed in the real, unstructured world. While it may be tempting to just randomize every element that conceivably could vary during deployment, randomization has been found to greatly decrease sample efficiency, stability, and even end-performance of the learned policy, as previously discussed in Chapter 5, and previous methods based on domain randomization and data augmentation therefore do not scale well to real applications of visual RL where the target environment may be unknown.

Policy Adaptation during Deployment (PAD) is a promising direction for adaptation and generalization in visual RL that does not rely on domain randomization. However, depending on the nature of changes in the test environment, PAD may not adapt perfectly to its new setting, and even then the added computational cost of continuously adapting can be prohibitive for certain real world applications of RL. It is therefore natural to ask whether there are other ways to learn robust policies, which do not rely solely on domain randomization nor solely on policy adaptation. In this chapter, we explore an alternative direction of research that can be considered orthogonal to adaptive methods such as PAD. We identify the primary culprit for instability and difficulty of RL optimization in previous works that rely on domain randomization and data augmentation, and propose a new method that alleviates the problem while still providing strong generalization results to a number of challenging test environments. We propose our alternative approach to generalization in visual RL in Section 6.1, and present our experimental results in Section 6.2.

6.1 Method

Extensive efforts have been made to improve the generalization ability of RL methods via domain randomization and data augmentation. However, as more factors of variation are introduced during training, the optimization process becomes increasingly more difficult, leading to low sample efficiency and unstable training. Instead of learning policies directly from augmented data, we propose Soft Data Augmentation (SODA), a method that decouples augmentation and generalization from the policy learning itself (Hansen & Wang, 2020). SODA aims to learn representations that effectively encode information shared between an observation and its augmented counterpart, while interfering minimally with the RL objective. The main intuition behind SODA is that by maximizing the mutual information between augmented and non-augmented views of the same observation, the encoder will learn to extract meaningful features from observations regardless of their visual variations. With strong and varied data augmentation, SODA learns policies that are invariant to high noise levels and is shown to generalize to a visually diverse set of environments. In this section, we describe our proposed SODA approach to generalization in visual RL. Section 6.1.1 provides an architectural overview of SODA, and Section 6.1.2 then introduces the proposed SODA representation learning task and algorithm.

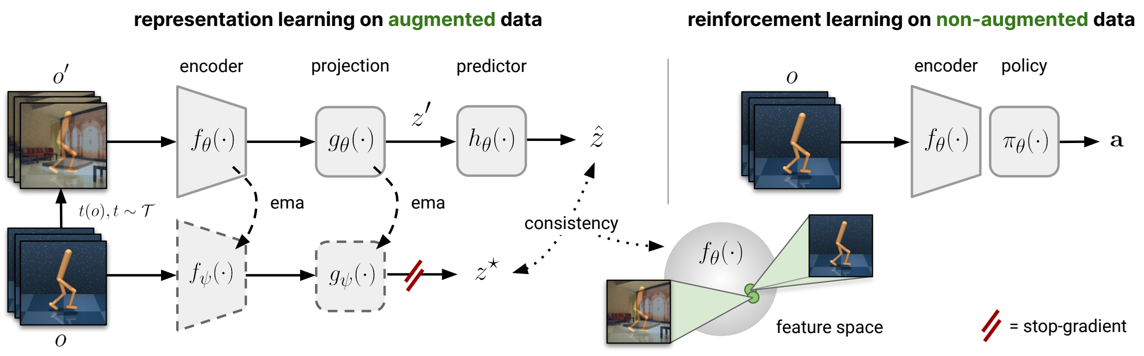


Figure 6.1: **SODA architecture.** *Left:* an observation o is augmented to produce a view o', which is then encoded and projected into $z' = g_\theta(f_\theta(o'))$. Likewise, o is encoded by $f_\psi(\cdot)$ and projected by $g_\psi(\cdot)$ to produce features $z^\star$. The SODA objective is then to predict $z^\star$ from z' by $h_\theta(\cdot)$ formulated as a consistency loss. *Right:* Reinforcement Learning in SODA. The RL task and objective remains unchanged and the policy is learned directly from the non-augmented observations o. *ema* denotes an exponential moving average.

6.1.1 Architectural Overview

Before describing the SODA algorithm in detail, we first outline its architectural design. SODA is implemented as a self-supervised auxiliary task that shares a common encoder $f(\cdot)$ with an RL policy $\pi(\cdot)$, similar to the architecture of PAD as introduced in Section 5.1. We reiterate the formal description of the architecture as follows: for a given policy network parameterized by a collection of parameters θ, we split the network and corresponding parameters sequentially into an encoder $f_\theta(\cdot)$ and a policy $\pi_\theta(\cdot)$ such that $a = \pi_\theta(f_\theta(o))$ outputs a distribution over actions (and any other algorithm-specific values) for a given input observation o. This is illustrated in Figure 6.1 (right). Note that both a and o may be high-dimensional, but we omit the bold notation introduced in Chapter 2 to preserve clarity in our following discussions.

With these definitions in mind, SODA then consists of an additional two architectural components: a projection $g_\theta(\cdot)$, and a prediction head $h_\theta(\cdot)$ that jointly implement the self-supervised task proposed in SODA. Aside from these two components, we define an architecturally identical encoder $f_\psi(\cdot)$ and projection $g_\psi(\cdot)$, where ψ denotes a collection of parameters separate from θ. Then, let f_θ, g_θ be denoted as the *online* encoder and projection, respectively, and similarly let f_ψ, g_ψ be denoted as the *target* encoder (and projection). Similar to the definition of the target Q-functions discussed in Section 2.1.3, we then define the collection of parameters ψ as an exponential moving average (EMA) of θ, and update ψ with every iteration of SODA using a Polyak averaging update rule

$$\psi_{t+1} \leftarrow (1-\tau)\psi_t + \tau\theta_t \tag{6.1}$$

for an iteration step t and a momentum coefficient $\tau \in (0, 1]$, which may or may not be a constant. As such, only parameters θ are optimized by gradient descent. In previous work on Q-learning (Lillicrap et al., 2016; Haarnoja et al., 2018b) and optimization (Kingma & Ba, 2015), EMAs are typically implemented with a constant momentum coefficient, whereas previous work on representation learning (Grill et al., 2020) utilizes a monotonically decreasing momentum coefficient. Empirically, we do not find a monotonically decreasing coefficient to offer any advantages over a constant coefficient in SODA, and we therefore opt for the simpler constant coefficient in this work. Figure 6.1 (left) provides an overview of the architecture for self-supervised learning in SODA, and the proposed representation learning task is detailed in the following section.

6.1.2 Representation Learning

In this section, we first describe the SODA representation learning task, and then seek to provide intuition for the task afterwards. Given an observation $o \in \mathbb{R}^{C \times H \times W}$, we sample a data augmentation $t \sim \mathcal{T}$ and apply it to produce an augmented observation $o' \triangleq t(o)$. As such, o and o' can be considered different *views* of the same observation, where o is the original observation and o' is corrupted by some noise source; this is in contrast to previous work on contrastive learning such as Laskin et al. (2020b); Chen et al. (2020b); Grill et al. (2020); Stooke et al. (2020) that produce two different augmented views. If we assume $f : \mathbb{R}^{C \times H \times W} \rightarrow \mathbb{R}^{C' \times H' \times W'}$ such that $H' \times W'$ is a feature map down-sampled from $H \times W$, then both projections g_θ, g_ψ are learned mappings $g : \mathbb{R}^{C' \times H' \times W'} \rightarrow \mathbb{R}^{K}$ where $K \ll C' \times H' \times W'$. Given a feature vector $z' \triangleq g_\theta(f_\theta(o'))$, the proposed SODA task is then to learn a differential mapping $h_\theta : \mathbb{R}^K \rightarrow \mathbb{R}^K$ that predicts the target projection $z^\star \triangleq g_\psi(f_\psi(o))$ of the non-augmented observation o, as shown in Figure 6.1 (left). We optimize projection g_θ, predictor h_θ, and shared encoder f_θ jointly together with the RL task(s), and employ a simple consistency loss defined as

$$\mathcal{L}_{SODA}\left(\hat{z}, z^\star; \theta\right) = \mathbb{E}_{t \sim \mathcal{T}}\left[\|\hat{z}_\circ - z^\star_\circ\|_2^2\right] \tag{6.2}$$

where $\hat{z} \triangleq h_\theta(z')$, and $\hat{z}_\circ \triangleq \hat{z}/\|\hat{z}\|_2$, $z^\star_\circ \triangleq z^\star/\|z^\star\|_2$ are ℓ_2-normalizations of $\hat{z}$ and $z^\star$, respectively. Because we only use data augmentation to enforce a representational constraint through an *auxiliary* consistency objective, we refer to our method as a *soft* data augmentation. Without the ℓ_2-normalizations, (6.2) would be equivalent to the Mean Squared Error (MSE) between $\hat{z}$ and $z^\star$. While the MSE in principle can be used in place of the proposed objective, we find it too restrictive: whereas an MSE objective explicitly optimizes for similarity, $\mathcal{L}_{SODA}$ instead optimizes for topological equivalence between the prediction space and feature space. Empirically, we find this relaxation of the consistency constraint imposed by MSE to produce more generalized representations.

We motivate the use of both projections g_θ, g_ψ and predictor h_θ following a similar argument. Although we in principle could drop these components and simply optimize for consistency between feature maps outputted by encoders f_θ and f_ψ, we find it to be too strong of a representational constraint, which ultimately decreases expressiveness of the joint representation learned by SODA and the RL task. Concurrent to our work, Lee et al. (2020b) imposes such a representational constraint to improve robustness to sensory errors in a multi-modal setting. The authors conclude that, while their proposed framework improves robustness, it simultaneously decreases expressiveness, i.e. any information that is not shared between views is forcibly discarded, which is found to be detrimental to policy learning and therefore suffers from the same scalability issues as e.g. domain randomization and data augmentation techniques. Our proposed formulation of the SODA task is designed to produce robust policies *without* negatively impacting policy learning.

Since target features $z^\star$ are encoded using a slow-moving encoder f_ψ and projection g_ψ, learning to map z' directly to $z^\star$ may also be too strict of a (soft) constraint, because the non-stationarity of the RL task is known to gradually change encodings over the course of training. While this is also true for purely (self-)supervised learning, it is only exacerbated by jointly performing RL optimization. The introduced predictor h_θ learns a mapping from the online projections to the target projections, which extends the representation learning temporally as θ and ψ are updated at different rates, ultimately easing the representational constraint further. We empirically find the addition of a predictor to improve expressiveness in the learned representations, which is consistent with both prior work Chen et al. (2020b); Grill et al. (2020) and concurrent work (Schwarzer et al., 2020; Dabney et al., 2020) on representation learning.

Algorithm 4 Soft Data Augmentation (SODA)

```
    $\theta, \psi$: randomly initialized network parameters
    $\omega$: RL updates per iteration
    $\tau$: momentum coefficient
 1: for every iteration do
 2:     for update $= 1, 2, ..., \omega$ do
 3:         Sample batch of transitions $\nu \sim \mathcal{B}$
 4:         Optimize $\mathcal{L}_{RL}(\nu)$ w.r.t. $\theta$
 5:     Sample batch of observations $o \sim \mathcal{B}$
 6:     Augment observations $o' = t(o),\ t \sim \mathcal{T}$
 7:     Compute online predictions $\hat{z} = h_\theta(g_\theta(f_\theta(o')))$
 8:     Compute target projections $z^\star = g_\psi(f_\psi(o))$
 9:     Optimize $\mathcal{L}_{SODA}(\hat{z}, z^\star)$ w.r.t. $\theta$
10:     Update $\psi \leftarrow (1-\tau)\psi + \tau\theta$
```

The policy π_θ (including any algorithm-specific task heads) uses f_θ to extract features and is optimized with no modifications to architecture nor inputs, i.e. we optimize an algorithm-specific RL objective $\mathcal{L}_{RL}$ directly on non-augmented observations o and update f_θ, π_θ by gradient descent, as shown in Figure 6.1 (right). This corresponds exactly to policy learning in any common base algorithm, e.g. DDPG (Lillicrap et al., 2016) or SAC (Haarnoja et al., 2018b) as discussed in Section 2.1.3. During training, the proposed SODA algorithm continuously alternates between optimizing $\mathcal{L}_{RL}$ and $\mathcal{L}_{SODA}$, such that the shared representation used for policy learning simultaneously is constrained by the SODA representation learning task. Whereas direct application of data augmentation in policy learning is shown to destabilize the RL optimization process, the *soft* data augmentation employed by SODA only affects the policy indirectly. The complete training procedure is summarized in Algorithm 4; lines 3-4 can be substituted by any standard RL algorithm.

We now seek to further motivate the SODA algorithm from the perspective of mutual information. In its essence, SODA reformulates the problem of generalization as a representational consistency learning problem: the encoder f should learn to map different views of the same underlying state to related feature vectors. This encourages the encoder to learn features that are shared across views, e.g. physical quantities and object interactions, and discard information that yield no predictive power such as background, lighting, and high-frequency noise. Given an observation o and a data augmentation t, we seek to encode o and $o' = t(o)$ into compact feature vectors $z', z^\star \in \mathbb{R}^K$, respectively, by learned mappings f_θ, g_θ and f_ψ, g_ψ such that the mutual information I between o and o' is maximally preserved. The mutual information between $z^\star$ and z' is then given by

$$I(z^\star; z') = \sum_{z'} \sum_{z^\star} p(z^\star, z') \log \frac{p(z^\star | z')}{p(z^\star)} \tag{6.3}$$

and is naturally bounded by the mutual information between o and o', i.e. $I(z^\star; z') \leq I(o; o')$. As $I(z^\star; z')$ is impractical to optimize directly, we approximate I as a simple consistency loss defined in (6.2) by the following intuition. If we assume f_ψ, g_ψ to maximally preserve information in o, then $I(z^\star; z') \propto I(o; o')$, and minimizing $\mathcal{L}_{SODA}$ thus maximizes $I(z^\star; z' | o')$. In other words, by learning a mapping from z' to $z^\star$, we maximally preserve information in o by learning to discard noise added by the data augmentation $t(\cdot)$. With strong and varied data augmentation, f_θ learns to ignore factors of variation that are irrelevant to the RL task, and reduces observational overfitting Song et al. (2020); Packer et al. (2018). While the assumption of maximally preserved information may not hold in practice due to ease of constraints, we find (6.2) to be a good enough approximation for expressive representation learning.

It should furthermore be emphasized that although the SODA objective does not explicitly prevent trivial solutions, e.g. $g_\theta(\cdot) = g_\psi(\cdot) = h_\theta(\cdot) = \mathbf{0}$ as in the latent forward dynamics model discussed in Section 2.2.1, such behavior is implicitly discouraged in SODA through a number of design choices. Firstly, we note that the collection of online parameters θ are updated by stochastic gradient descent in the direction of $\nabla_\theta \mathcal{L}_{SODA}$, whereas the target parameters ψ are *not* updated in this direction, but rather in the direction of the EMA. While this dynamic does not prevent trivial equilibria per se, we conjecture that it makes such equilibria unstable; in fact, the dynamics of the SODA objective and previous work on contrastive learning without negative samples (Grill et al., 2020) can be considered similar to that of the minimax objective of GANs (Goodfellow et al., 2014b). Secondly, we implement both the projector g and predictor h as MLPs with normalization layers, e.g. batch normalization (Ioffe & Szegedy, 2015), which we discuss further in Section 6.2.1. The immediate implication of using normalization layers, however, is that it helps mitigate improper weight initialization which may otherwise lead to these trivial equilibria as shown by concurrent work Richemond et al. (2020), and other concurrent work further hypothesize that batch normalization specifically prevents trivial equilibria through gradient dependence on batch elements, resulting in an implicit contrastive term where other elements in a batch act as negative samples (Fetterman & Albrecht, 2020; Tian et al., 2020c). Lastly, f_θ is jointly optimized between SODA and the RL objective(s), and trivial equilibria in f_θ are therefore in direct conflict with the RL objective since they entail that f_θ would preserve no information in o. While this argument only applies to f_θ, it inevitably contributes to the overall stability of the optimization process, and we empirically find it sufficient to avoid trivial equilibria in f_θ, f_ψ, without the need for normalization layers or careful initialization schemes in f_θ as otherwise proposed by Richemond et al. (2020). In the following section, we discuss our empirical evaluation of SODA.

6.2 Experiments

In this section, we provide empirical evidence for the generalization, sample-efficiency and stability of SODA, and compare to a number of recent state-of-the-art methods for representation learning and generalization in visual RL, including PAD as presented in Chapter 5. We follow a similar experimental setup as in Chapter 5, and investigate how well an agent trained exclusively from image observations in a fixed training environment generalizes to a variety of visually challenging test environments in 5 continuous control tasks from DMControl-GB, and additionally benchmark SODA on two novel sets of challenging test environments for the proposed *push* robotic manipulation task introduced in Chapter 4. Our novel test sets for robotic manipulation include changes such as colors and video backgrounds, as well as random perturbations of camera, lighting, and texture to simulate real-world conditions during testing.

While our experiments primarily aim to demonstrate the effectiveness of SODA, we also provide useful insights into the mechanisms involved in representation learning through soft data augmentation, and our comparison to strong baselines from recent literature seek to foster a broader discussion on generalization in RL which we resume in Chapter 7. An open-sourced implementation of SODA and baselines is made available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark and a webpage with video results from our experiments is available at https://nicklashansen.github.io/SODA.

Table 6.1: **Hyper-parameters.** A complete overview of hyper-parameters for SODA and all baselines in experiments on DMControl-GB and robotic manipulation.

Hyper-parameter	Value
Frame rendering	$3 \times 100 \times 100$
Frame after crop	$3 \times 84 \times 84$
Stacked frames	3 (DMControl-GB) 1 (robotic manipulation)
Number of conv. layers	11
Number of filters in conv.	32
Action repeat	2 (`finger_spin`) 8 (`cartpole_swingup`) 4 (otherwise)
Discount factor γ	0.99
Episode time steps	1,000
Learning algorithm	Soft Actor-Critic
Number of training steps	500,000
Replay buffer size	500,000
Optimizer (RL/aux.)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Optimizer (α)	Adam ($\beta_1 = 0.5, \beta_2 = 0.999$)
Learning rate (RL)	1e-3
Learning rate (α)	1e-4
Learning rate (SODA)	3e-4
Batch size (RL)	128
Batch size (SODA)	256
Actor update freq.	2
Critic update freq.	1
Auxiliary update freq.	1 (CURL) 2 (PAD/SODA) N/A (otherwise)
Momentum coef. τ (SODA)	0.005

6.2.1 Implementation Details

SODA is a general framework that can be implemented on top of any standard RL algorithm using parameterized policies. To make comparison between methods easier, we apply a similar experimental setup as in Chapter 5, and adopt both the network architecture and hyper-parameters from our PAD experiments. In each of the two benchmarks – DMControl-GB and robotic manipulation – SODA is implemented using Soft Actor-Critic (SAC) (Haarnoja et al., 2018b) as base algorithm, and observations are stacks of k colored frames of dimensions 100×100, where $k = 3$ in DMControl-GB and $k = 1$ in robotic manipulation as in Chapter 5. Whereas PAD implements separate convolutional branches for the RL task(s) and the self-supervised task, SODA simplifies the architecture by only implementing the RL branch. We denote all 11 convolutional layers as f_θ, and the policy and any algorithm-specific task heads (e.g. the Q-value functions of SAC) are jointly referred to as π_θ. We implement both $g(\cdot)$ and $h(\cdot)$ as MLPs with batch normalization (Ioffe & Szegedy, 2015) as in previous work on contrastive learning without negative samples (Grill et al., 2020), and use a batch size of 256 for the SODA representation learning task. Projections are of dimension $K = 100$, and the target components f_ψ, g_ψ are updated using an EMA of f_θ, g_θ with a momentum coefficient $\tau = 0.005$. The two objectives $\mathcal{L}_{RL}$ and $\mathcal{L}_{SODA}$ are optimized using Adam (Kingma & Ba, 2015), and in practice we find it sufficient to only make a SODA update after every second RL update, i.e. $\omega = 2$ as defined in Algorithm 4. The reader is referred to Appendix A and Appendix B for more details.

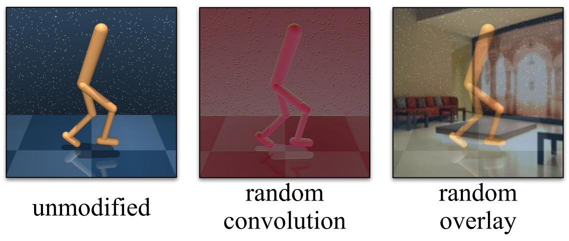


Figure 6.2: **Data augmentation.** We consider the following two data augmentations: *random convolution* (as proposed by Lee et al. (2019); Laskin et al. (2020a)) and *random overlay* (novel). Refer to Figure 6.3 for additional data augmentation samples.

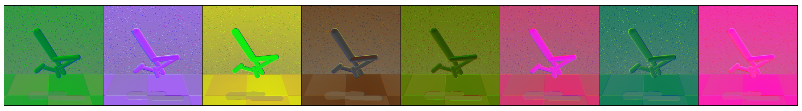
6.2.2 Data Augmentation

The value proposition of SODA is to improve generalization of RL agents without negatively impacting stability and sample efficiency. As evident from previous work (Kostrikov et al., 2020; Laskin et al., 2020a) and the PAD experiments, not all data augmentations are detrimental to policy learning. Augmentations such as random cropping has in fact been shown to *improve* policy learning, but it also provides no obvious benefit in terms of generalization. While SODA makes no assumptions about the data augmentation(s) used, we argue that it is useful to distinguish between *weak* augmentations that may improve sample efficiency but contribute minimally to generalization, and *strong* augmentations that improve generalization at the cost of sample efficiency (Tian et al., 2020b; Xiao et al., 2020). An example of the former is the random cropping, whereas examples of the latter are domain randomization (Tobin et al., 2017; Pinto et al., 2017) and random convolution (Lee et al., 2019; Laskin et al., 2020a). As in previous work (Laskin et al., 2020b;a) and the PAD experiments, we apply temporally consistent random cropping (from 100×100 down to 84×84) by default in SODA and all baselines, and we refer to the cropped images as non-augmented observations.

In this thesis, we seek to stabilize training with *strong* augmentations that are designed to improve generalization. To demonstrate the effectiveness of SODA, we first consider the *random convolution* data augmentation which was initially proposed by Lee et al. (2019) and has been shown to cause instabilities in policy learning on DMControl tasks (Laskin et al., 2020a). The proposed data augmentation applies a randomly initialized convolutional layer to an observation, producing an augmented observation with structured color changes and local texture distortions. We apply padding to preserve original image dimensionality and additionally apply a hyperbolic tangent non-linearity to squash pixel intensities of the random convolution output to match the original $[0, 1)$ range of values. We hypothesize that SODA benefits further from data augmentation with even more diversity and many factors of variation. To test this hypothesis, we propose a novel *random overlay* data augmentation that linearly interpolates between an observation o and an image ε to produce an augmented view

$$t_{overlay}(o) = (1 - \alpha)o + \alpha\varepsilon, \ \varepsilon \sim \mathcal{D} \tag{6.4}$$

where $\alpha \in [0, 1)$ is the interpolation coefficient and $\mathcal{D}$ is an unrelated dataset. In practice, we use $\alpha = 0.5$ and sample images from Places (Zhou et al., 2017), a dataset containing 1.8M diverse scenes, which produces very diverse data augmentations. Both data augmentations are applied in a temporally consistent manner to preserve spatio-temporal information in the observations. Samples from each of the two data augmentations are shown in Figure 6.2, and additional samples are provided in Figure 6.3. We provide pseudo-code for each of the considered data augmentations in Appendix A.


(a) *Random convolution* data augmentation.

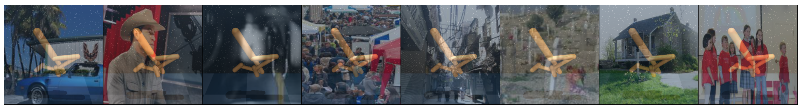

(b) *Random overlay* data augmentation.

Figure 6.3: **Additional data augmentation samples.** In this work, we consider the *random convolution* data augmentation (Lee et al., 2019; Laskin et al., 2020a) shown in (a), as well as a novel *random overlay* data augmentation shown in (b).

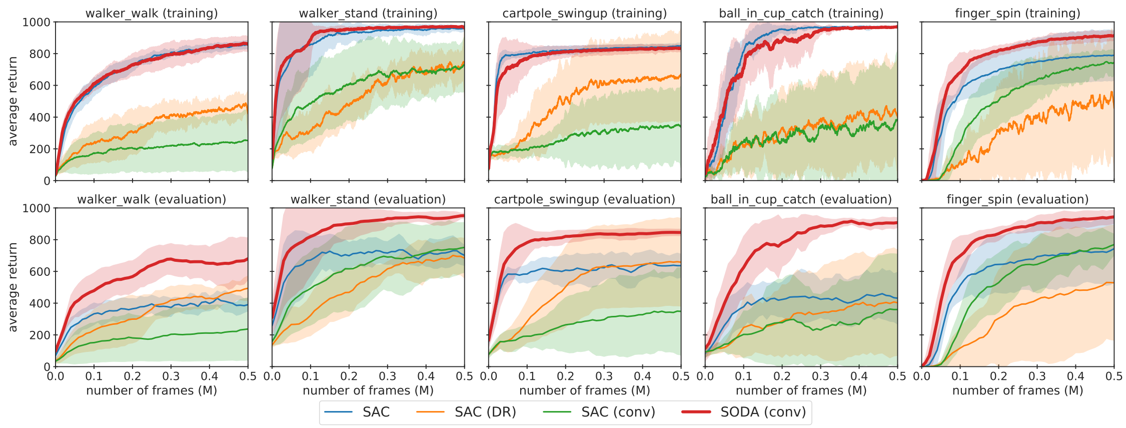


Figure 6.4: **Random convolution.** *Top:* average return on the training environment during training. *Bottom:* periodic evaluation of generalization ability measured by average return on the `color_hard` test distribution. SODA exhibits sample efficiency and convergence similar to SAC but improves generalization significantly. Average of 5 random seeds, shaded area is standard deviation.

6.2.3 DMControl-GB

We train agents in a fixed training environment and continuously evaluate generalization to visually diverse test environments throughout training. We consider five diverse continuous control tasks from the proposed DMControl-GB benchmark for generalization, and compare SODA to the following baselines: (i) a baseline SAC that is functionally equivalent to RAD (Laskin et al., 2020a) since we apply random cropping to all methods; (ii) SAC trained with domain randomization on the *color_hard* test distribution (denoted *SAC (DR)*); (iii) SAC using random convolution as data augmentation (denoted *SAC (conv)*; and (iv) SAC using random overlay as data augmentation (denoted *SAC (overlay)*). Likewise, our method is denoted *SODA (conv)* and *SODA (overlay)*, depending on the choice of data augmentation. Each of the data augmentation baselines perform RL on augmented observations and use no auxiliary tasks, whereas SODA performs RL strictly on non-augmented observations and instead apply data augmentation in its auxiliary representation learning task. Note that the distribution of environments used for the domain randomization baseline considered in this section (`color_hard`) differs from that of the PAD experiments (`color_easy`).

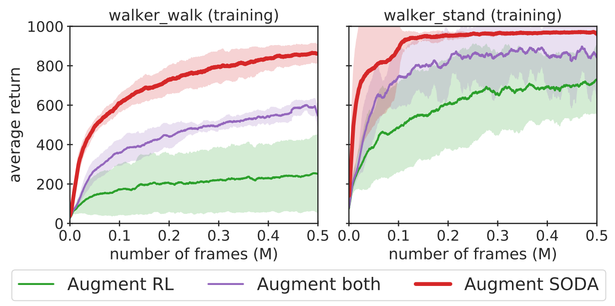


Figure 6.5: **Soft data augmentation.** Average return on the fixed training environment for the `walker_walk` and `walker_stand` tasks. *Augment RL* corresponds to the SAC (conv) baseline, *Augment both* applies random convolution in both SODA representation learning and RL, and *Augment SODA* is the proposed formulation of SODA. Average of 5 random seeds, shaded area is standard deviation.

Stability under strong data augmentation. Domain randomization and data augmentation is known to decrease sample efficiency and can destabilize policy learning. Figure 6.4 shows the training performance (top row) and generalization ability (bottom row) of baselines and SODA using a random convolution data augmentation, and Figure 6.6 similarly shows results for SODA with the novel random overlay data augmentation. While SAC converges to (approximately) optimal performance in the training environment in each of the five tasks, its generalization remains poor, resulting in a substantial drop in performance on the `color_hard` test distribution. Interestingly, we find the generalization ability of SAC to remain approximately constant once the policy learning has converged in the training environment, which is in contrast to the common issue of *overfitting* known from e.g. supervised learning. As previously mentioned, the domain randomization baseline is trained directly on the test distribution, and it therefore achieves similar performance during training and evaluation. However, it evidently exhibits poor sample efficiency and converges to sub-par policies in all tasks considered, and the data augmentation baselines (without SODA) display similar behavior. We find SODA to improve generalization substantially using either of the two data augmentations, recovering optimal behavior in test environments in 4 out of 5 tasks. SODA additionally exhibits stable learning and a sample efficiency similar to that of the SAC baseline, clearly demonstrating the benefits of our *soft* data augmentation proposal as opposed to other approaches to generalization.

Soft data augmentation. To gain more insight into the effect of soft data augmentation, we consider a variant of SODA in which random convolution augmentation is applied in both representation learning and RL. The empirical findings are summarized for two tasks, `walker_walk` and `walker_stand`, in Figure 6.5. While the variant of SODA that applies data augmentation in both the RL objective and the auxiliary SODA task (*Augment both*) improves significantly over only applying data augmentation in RL (*Augment RL*), we find the proposed formulation of SODA (*Augment SODA*) superior in both sample efficiency and end-performance. We find that, although performing auxiliary representation learning improves performance of the data augmentation baseline, avoiding data augmentation altogether in policy learning allows SODA to sustain the same training performance as SAC. Our findings are supported by recent work on data augmentation in RL (Laskin et al., 2020a; Raileanu et al., 2020b) that extensively evaluate data augmentations on tasks from both DMControl (Tassa et al., 2018) and ProcGen (Cobbe et al., 2019b), and the authors find that a majority of data augmentations do in fact destabilize the RL optimization process, resulting in agents that fail to solve the task entirely.

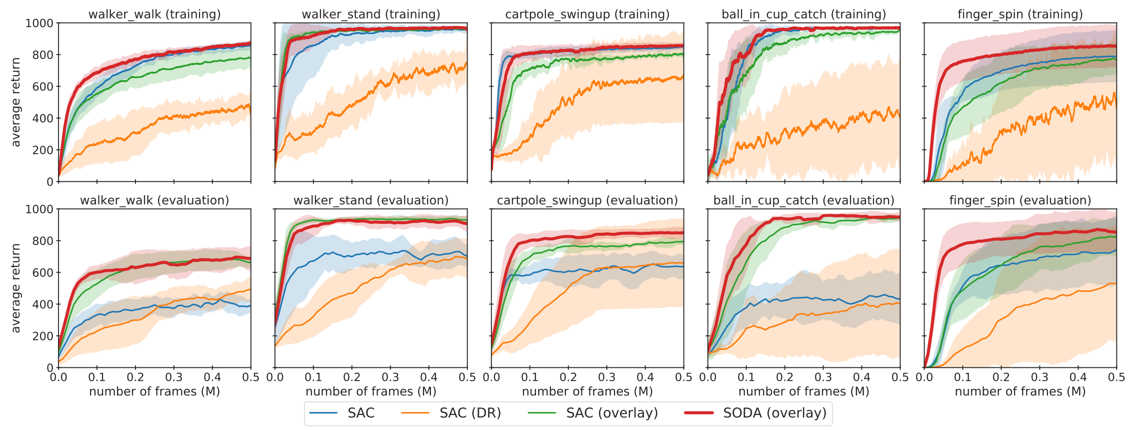


Figure 6.6: **Random overlay.** *Top:* average return on the training environment during training. *Bottom:* periodic evaluation of generalization ability measured by average return on the `color_hard` test distribution. SODA offers better sample-efficiency than the novel *SAC (overlay)* baseline and similar generalization to *SODA (conv)*. Average of 5 random seeds, shaded area is standard deviation.

Choice of data augmentation. SODA is a general framework for generalization in visual RL by data augmentation, and can in principle be implemented with any augmentation. While SODA is designed to stabilize policy learning, an agent's generalization ability is directly impacted by the choice of data augmentation. In this thesis, we consider two data augmentations, random convolution and the novel random overlay, and evaluate their generalization to both environments with randomized colors and environments with video backgrounds. Empirical evaluations on the `video_easy` test distribution is shown in Table 6.2, and evaluations on the `color_hard` test distribution are shown in Figure 6.4 and Figure 6.6. While SODA with random convolution data augmentation generalizes well to the `color_hard` test distribution, it generalizes comparably worse to videos. We conjecture that this is because convolution captures less factors of variation than overlay (e.g. textures and local color changes), which is in line with the findings of Packer et al. (2018); Cobbe et al. (2019a) and our experiments from Chapter 5. SAC with random overlay data augmentation displays stable behavior and near-optimal policy learning despite its strong augmentation, but we still find SODA to improve sample efficiency and end-performance over the random overlay baseline. We additionally argue that, although random overlays do not cause any significant instability in our experiments in DMControl, they may still do so in other environments, and SODA is therefore preferred; we discuss this further in Section 6.2.4. *SODA (overlay)* is found to generalize well to both test distributions even though there is no significant visual similarity between random overlays and the `color_hard` test distribution. These results suggest that SODA indeed does benefit from strong and varied data augmentation, and we leave it to future work to explore the effectiveness of additional (soft) data augmentations for generalization in visual RL.

Comparison to state-of-the-art methods. We now aim to provide a complete overview of the generalization ability of recent methods for visual RL. We evaluate the generalization ability of SODA on the full DMControl-GB benchmark and compare to a number of recent state-of-the-art methods: (i) CURL Laskin et al. (2020b), a contrastive representation learning RL method; (ii) RAD Laskin et al. (2020a), a study on data augmentation for RL that uses random cropping and is functionally equivalent to the SAC baseline in our previous experiments; and (iii) PAD (Hansen et al., 2020), a method for self-supervised

Table 6.2: **Video backgrounds.** Average return of SODA and baselines trained in a fixed environment and evaluated on the `video_easy` test distribution from DMControl-GB. The domain randomization baseline is trained on `color_hard`. Mean and standard deviation of 5 random seeds. * Laskin et al. (2020b). † Laskin et al. (2020a). ‡ Chapter 5.

DMControl-GB (`video_easy`)	CURL *	RAD †	PAD ‡	SAC (DR)	SAC (conv)	SODA (conv)	SAC (overlay)	SODA (overlay)
`walker, walk`	556 ±133	606 ±63	717 ±79	520 ±107	169 ±124	635 ±48	718 ±47	**768 ±38**
`walker, stand`	852 ±75	745 ±146	935 ±20	839 ±58	435 ±100	903 ±56	**960 ±2**	955 ±13
`cartpole, swingup`	404 ±67	373 ±72	521 ±76	524 ±184	176 ±62	474 ±143	718 ±30	**758 ±62**
`ball_in_cup, catch`	316 ±119	481 ±26	436 ±55	232 ±135	249 ±190	539 ±111	713 ±125	**875 ±56**
`finger, spin`	502 ±19	400 ±64	691 ±80	402 ±208	355 ±88	363 ±185	607 ±68	**695 ±97**

policy adaptation that was introduced in Chapter 5. Results are shown in Table 6.3; empirical evaluations on the `video_easy` test distribution are duplicated from Table 6.2. We find SODA to outperform all other methods in 18 out of 20 instances, and often by a large margin. Although PAD improves the generalization ability of RAD in nearly all instances, SODA improves over PAD by as much as 310% and 439% on the extremely challenging `video_hard` test distribution (shown in Figure 4.3 (d)) in `walker_walk` and `finger_spin`, respectively.

Discussion on representation learning. From the results in Table 6.3 we further find that, while both PAD and SODA improve over the RAD baseline using a random cropping, the additional contrastive learning proposed in CURL does not appear to improve generalization. In fact, it appears that CURL *decreases* the generalization ability of agents in certain tasks, e.g. `cartpole_swingup` and `ball_in_cup_catch` on the two color benchmarks. This is particularly noteworthy because all four methods employ the same architecture, hyper-parameters, and random cropping mechanism. Whether the disparity in improvements between CURL and SODA can be attributed solely to the choice of data augmentation (being that CURL only applies random crop, whereas SODA additionally applies a stronger data augmentation), or the design of representation learning task also contributes to the success of SODA remains unclear and would be an interesting direction for future research. Concurrent work Stooke et al. (2020) build upon the CURL method and show empirically that a better design of auxiliary contrastive learning does indeed improve the quality of representations. While the authors do not consider generalization in their experiments, we conjecture that both a good representation learning task and choice of data augmentation will be necessary in future work on generalization in visual RL.

Table 6.3: **Comparison to state-of-the-art methods.** Average return of SODA and state-of-the-art methods on each of the four test distributions from DMControl-GB. Mean and standard deviation of 5 random seeds. SODA outperforms all other methods in 18 out of 20 instances, and often by a large margin. Although PAD improves the generalization ability of RAD in nearly all instances, SODA improves over PAD by as much as 310% and 439% on the extremely challenging `video_hard` test distribution in `walker_walk` and `finger_spin`, respectively. * Laskin et al. (2020b). † Laskin et al. (2020a). ‡ Chapter 5.

	`color_easy`				`color_hard`			
DMControl-GB (colors)	CURL *	RAD †	PAD ‡	SODA (overlay)	CURL *	RAD †	PAD ‡	SODA (overlay)
`walker, walk`	645 ±55	636 ±33	687 ±119	**811 ±41**	445 ±99	400 ±61	468 ±47	**692 ±68**
`walker, stand`	866 ±46	807 ±67	894 ±39	**960 ±4**	662 ±54	644 ±88	797 ±46	**893 ±12**
`cartpole, swingup`	668 ±74	763 ±29	812 ±20	**859 ±15**	454 ±110	590 ±53	630 ±63	**805 ±28**
`ball_in_cup, catch`	565 ±168	727 ±87	775 ±159	**969 ±3**	231 ±92	541 ±29	563 ±50	**949 ±19**
`finger, spin`	781 ±139	789 ±160	**870 ±54**	855 ±93	691 ±12	667 ±154	**803 ±72**	793 ±128

	`video_easy`				`video_hard`			
DMControl-GB (videos)	CURL *	RAD †	PAD ‡	SODA (overlay)	CURL *	RAD †	PAD ‡	SODA (overlay)
`walker, walk`	556 ±133	606 ±63	717 ±79	**768 ±38**	58 ±18	56 ±9	93 ±29	**381 ±72**
`walker, stand`	852 ±75	745 ±146	935 ±20	**955 ±13**	45 ±5	231 ±39	278 ±72	**771 ±83**
`cartpole, swingup`	404 ±67	373 ±72	521 ±76	**758 ±62**	114 ±15	110 ±16	123 ±24	**429 ±64**
`ball_in_cup, catch`	316 ±119	481 ±26	436 ±55	**875 ±56**	115 ±33	97 ±29	66 ±61	**327 ±100**
`finger, spin`	502 ±19	400 ±64	691 ±80	**695 ±97**	27 ±21	34 ±11	56 ±18	**302 ±41**

Table 6.4: **Generalization in robotic manipulation.** Average return of SODA and baselines trained in a fixed environment and evaluated on the proposed test distributions for robotic manipulation. The domain randomization baseline is trained on randomized colors. Mean and standard deviation of 5 random seeds.

`robot, push`	SAC	SAC (DR)	SAC (conv)	SODA (conv)	SAC (overlay)	SODA (overlay)
training env.	14.5 $\pm$5.2	9.3 $\pm$7.5	0.2 $\pm$1.7	14.4 $\pm$4.6	15.8 $\pm$7.3	**21.3** **$\pm$5.0**
random colors	7.2 $\pm$3.4	7.8 $\pm$7.0	-3.1 $\pm$1.4	1.4 $\pm$3.4	13.5 $\pm$4.4	**18.0** **$\pm$3.7**
video bg.	0.9 $\pm$6.0	7.0 $\pm$7.4	-4.9 $\pm$1.9	-4.5 $\pm$1.1	7.6 $\pm$7.1	**13.9** **$\pm$2.6**

6.2.4 Robotic manipulation

While DMControl-GB provides a good platform for benchmarking algorithms, we are ultimately interested in developing algorithms that solve real-world problems with vision-based RL. To better emulate real-world deployment scenarios, we also consider the *push* robotic manipulation task from Section 4.3 on a robotic arm (in simulation). As in DMControl-GB, agents are trained in a fixed environment and evaluated on environments with randomized colors and video backgrounds, and we additionally perform random perturbations of camera, lighting, and texture to simulate real-world conditions during testing. In our empirical evaluation, we will denote these test distributions as *random colors* and *video backgrounds*, and we additionally report performance on the fixed *training* environment. Samples from the robotic manipulation environments that we consider are shown in Figure 4.5.

We follow the evaluation procedure of both our DMControl-GB experiments and the robotic simulation experiments from Chapter 5, but choose to report mean episodic return instead of success rate since experiments are in simulation where a dense reward signal is available. Episodes consist of 50 time steps, and at each time there is a positive reward of 1 for having the cube at the goal location (red disc), and a small penalty (negative reward) proportional to the distance between the cube and goal. Therefore, an average return of 25 means that the cube is successfully relocated to the goal location fore more than half of all time steps. Due to the formulation of the dense reward signal, our evaluation considers not only a binary success criteria, but also rewards the agent for solving the task faster and penalizes less if the agent fails but nearly solves the task.

Agents are evaluated for 100 initial configurations, and we report the mean episodic return across 5 random seeds. As in Section 6.2.3, we compare SODA to the following baselines: (i) a baseline SAC that is functionally equivalent to RAD (Laskin et al., 2020a) since we apply random cropping to all methods; (ii) SAC trained with domain randomization on the *randomized colors* test distribution (denoted *SAC (DR)*); (iii) SAC using random convolution as data augmentation (denoted *SAC (conv)*; and (iv) SAC using random overlay as data augmentation (denoted *SAC (overlay)*). Likewise, our method is denoted *SODA (conv)* and *SODA (overlay)*, depending on the choice of data augmentation. Each of the data augmentation baselines perform RL on augmented observations and use no auxiliary tasks, whereas SODA performs RL strictly on non-augmented observations and instead apply data augmentation in its auxiliary representation learning task.

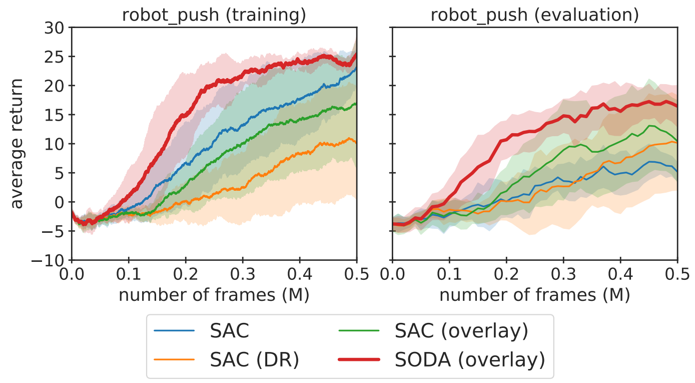


Figure 6.7: **Robotic manipulation.** *Left:* average return on the training environment during training. *Right:* periodic evaluation of generalization measured by average return on the test distribution with randomized colors. SODA outperforms all baselines in both sample efficiency and generalization, and reduces variance. Average of 5 random seeds, shaded area is standard deviation.

Sample efficiency in robotic manipulation. Training and generalization curves for the *push* task are shown in Figure 6.7. It is observed that both the domain randomization baseline and the *SAC (overlay)* baseline achieve a positive mean episodic return in the training environment, i.e. the task is solved in some instances, but they exhibit a sample efficiency that is significantly worse than the vanilla SAC baseline. *SODA (overlay)* is found to have a *better* sample efficiency than the SAC baseline, achieving a mean episodic return of 15 in half as many steps (frames). At the end of training (after 500k steps), SODA achieves a mean episodic return of 21.3 versus 14.5 for the SAC baseline. This would suggest that agents benefit from the additional training signal of auxiliary representation learning in robotic manipulation tasks – even when considering only a single environment – which is supported by the experiments of Yen-Chen et al. (2020); Zhan et al. (2020) that come to a similar conclusion when learning policies from visual observations directly in the real world.

Generalization in robotic manipulation. Table 6.4 shows our empirical evaluation of the generalization of SODA and baselines to the two test distributions, as well as their end-performance in the training environment. As in our DMControl-GB experiments from Section 6.2.3, the generalization ability of the SAC baseline remains poor to both of the two test distributions, while domain randomization and data augmentation is found to improve generalization at the cost of sample efficiency and ultimately end-performance (see Figure 6.7). SODA outperforms all other methods in each of the three environments shown in Table 6.4, and we observe that SODA is comparably more robust to the challenging *randomized colors* and *video backgrounds* test distributions. Whereas the performance of SODA drops by 34.7% when transferred from the training environment to the video backgrounds environment, *SAC (overlay)* drops by 51.9% and the unmodified SAC baseline drops by 93.7%. We find the *SAC (DR)* baseline to be surprisingly robust, only experiencing a drop of 24.7%, but the numerical difference between the mean episodic return of SODA and the domain randomization baseline is significant. Given enough training time and network capacity (i.e. number of trainable parameters), we can expect *SAC (DR)* to eventually outperform the current returns of SODA, but we argue that SODA is desirable since it does not have the issues of sample efficiency and scalability that domain randomization techniques suffer from.

6.3 Summary

Extensive efforts have been made to improve the generalization ability of RL methods via domain randomization and data augmentation, and they remain the most prominent techniques amongst practitioners. However, as more factors of variation are introduced during training with domain randomization, the optimization process becomes increasingly more difficult, leading to poor sample efficiency, unstable learning, and ultimately a decrease in end-performance. But if we on the other hand constrain the randomization to be small, the distribution is unlikely to cover variations in the test environments. Finding the right balance between sample efficiency and generalization therefore requires significant engineering efforts. Similarly, recent studies show that both sample efficiency and generalization can be improved by using the right kinds of data augmentation, but determining which data augmentations to use – and to which degree – is found to be task-dependent and a naïve application of augmentation may lead to unstable training and poor performance, just like domain randomization.

In this chapter, we revisit the use data augmentation, and question whether we can obtain the benefits of augmentation while at the same time interfering minimally with the RL optimization process. We propose Soft Data Augmentation (SODA), a method that stabilizes training by decoupling data augmentation from policy learning. While previous work attempts to learn policies directly from augmented data, SODA uses strictly *non-augmented* data for policy learning, and instead performs auxiliary representation learning using *augmented* data. Our empirical evaluations are performed on the novel DMControl-GB benchmark, as well as a robotic manipulation task. We demonstrate the effectiveness of SODA in stabilizing policy learning with augmented data, and show that SODA improves generalization over recent state-of-the-art methods for visual RL, while maintaining a sample efficiency that is competitive with a non-augmented SAC baseline. We show that SODA can stabilize a random convolution augmentation proposed in previous work, and develop a novel *random overlay* data augmentation that we find to further improve generalization across all benchmarks. We additionally ablate the formulation of SODA, and find that our proposed decoupling of data augmentation from policy learning is crucial to sample efficient RL, but the policy still benefits substantially from auxiliary representation learning even when augmenting data in both SODA and RL optimization.

7 Discussion and Future Perspectives

In this thesis, we have explored two distinct approaches to generalization in visual RL. PAD uses a self-supervised mechanism to continuously adapt to changes in the environment that it is deployed in, and SODA learns strong invariances by improving the stability and sample-efficiency of data augmentation techniques for RL through its soft data augmentation formulation. To systematically evaluate the generalization ability of agents trained by visual RL, we additionally develop the DMControl-GB and robotic manipulation benchmarks, and successfully transfer policies learned in simulation to a real robot.

While our experimental results are encouraging, we acknowledge that our proposed methods are only initial steps towards broadly intelligent agents. 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, and from multiple modalities. In this chapter, we aim to provide a holistic view on the current state of visual RL, the implications of our work, and finally our perspective on the future of visual RL. To this end, we address both concurrent work, opportunities for future research surrounding our methods, as well as recent studies that extend our work.

7.1 Towards Broadly Intelligent Behavior

Before starting to address concrete opportunities for future work, we first discuss our long-term research goal. Today's agents require vast amounts of interactions to learn visuomotor skills, largely fail to generalize beyond their simplistic training environment, and generally do not have any mechanisms for learning once deployed. We ultimately envision a future in which we can teach personalized agents new skills only from a few *natural* interactions in *unstructured* environments. For instance, imagine a motor-impaired person instruct a robot how to fetch and operate items in their household. How would they communicate their knowledge and intentions to the robot, and how would the robot consolidate its new experiences when a reward signal is not given explicitly? A natural way for embodied agents to interact with the real, unstructured world is by visual and auditory perception. Humans are excellent at learning and quickly adapt to new surroundings after a few interactions, and can largely learn by experimentation without any explicit supervision. Yet, real-world adoption of agents trained by reinforcement learning is hindered by their fragility and notoriously poor generalization ability, which is only exacerbated by the high-dimensional nature of visual and auditory perception problems. Machine learning systems today are largely trained to be invariant to certain factors of variation through techniques like domain randomization and data augmentation, and are typically trained and deployed in similar environments. This approach is problematic because (i) it does not scale well; and (ii) it does not address the desire for continual knowledge acquisition – something that comes naturally to humans.

To address these challenges, we need to rethink how agents learn and represent knowledge. As a research community, we will need to explore more flexible learning frameworks in which perception and skills can be learned both independently and jointly, from various sources and modalities, and be continuously adapted over an agent's lifetime. This thesis takes small but important steps towards addressing these challenges. SODA directly addresses the problem of scalability of data augmentation in RL methods (problem

(i) stated above), and PAD addresses the desire for continual knowledge acquisition and adaptation in RL without explicit supervision (problem (ii) stated above).

More generally, it can be useful to formulate a set of properties that we believe will be useful (but not necessarily sufficient) for broadly intelligent agents. We believe that such agents will be equipped with learning frameworks that (1) represent knowledge in a way that allows for adaptive behavior and life-long learning; (2) learn from both direct and indirect supervision; and (3) leverage its understanding of the world to intelligently obtain new experiences. To this end, it is natural to consider how data sources external to the agent can be fused with its own experiences, in part to address generalization concerns (e.g. ”learning to see before learning to act” (Yen-Chen et al., 2020; Yu et al., 2020b; Hansen & Wang, 2020)), but also in the context of continual knowledge acquisition (e.g. fine-tuning and adaptation (Lomonaco et al., 2019; Julian et al., 2020; Hansen et al., 2020)), since real environments are seldom stationary, and in some application areas it can be prohibitively expensive to retrain an agent from scratch.

A recent direction of research, *offline RL*, seek to learn useful policies without any environment interactions prior to deployment, which is achieved by leveraging prior experiences collected from data sources external to the agent itself. In principle, off-policy methods like DQN (Mnih et al., 2015), DDPG (Lillicrap et al., 2016) and SAC (Haarnoja et al., 2018a;b) can be trained entirely offline in this regime through pre-population of their replay buffers, but this approach appears to work poorly in practice (Agarwal et al., 2020). The reasons for that are multitude, but generalization under distributional shift is a core challenge in RL that is only exacerbated by the offline RL problem setting (Levine et al., 2020). SODA improves the observational generalization of RL methods by learning invariances through its self-supervised representation learning framework, and an interesting avenue for future research could be to explore application of similar representation learning techniques for the offline RL regime as a means for reducing distributional shift. In the offline RL problem setting, however, agents not only experience distributional shift in observations, but also in the trajectories and transitions that were taken during generation of the pre-collected experiences. While recent work on offline RL attempts to mitigate this shift by e.g. importance weighted sampling (Kumar et al., 2019; Peng et al., 2019a; Mahmood et al., 2014), it may also be an interesting problem setting for adaptive methods such as PAD that attempt to address the distributional shift only at test-time. To achieve more broadly intelligent behavior in machines, we argue that a combination of techniques that address the problems of generalization and distributional shift both before *and* during deployment will likely be necessary.

7.2 Policy Adaptation

Adapting a policy for a deployment environment that differs from where the policy is trained is not a new idea. However, concurrent work on policy adaptation primarily considers the problem of domain adaptation, where the policy is to be deployed in a single target environment that is known and to some extent accessible during the training phase (Julian et al., 2020; Karnan et al., 2020; Desai et al., 2020; Zhang et al., 2020b). The self-supervised policy adaptation mechanism of PAD adapts to environments in an online manner *during* deployment, without any prior knowledge about the target environment, and is conceptually similar to the ideas proposed in both previous work (Sun et al., 2020) and concurrent work (Zhang et al., 2020a; Raileanu et al., 2020a; Alet et al., 2020).

Our experiments in Chapter 5 show that the proposed framework for adaptation in PAD does not hurt performance of the policy when fine-tuning to an environment that is indistinct to its original training environment, and it generally does not degrade performance even in long-horizon deployment scenarios. However, we also find that the success of PAD depends (to some extent) on the choice of self-supervised task. Since selecting an appropriate self-supervised task for PAD remains non-trivial, successful application of PAD may still require some domain knowledge and experimentation. An interesting opportunity for future research is to explore methods that automate this experimentation and selection of self-supervised tasks for policy adaptation. We hypothesize that auxiliary objectives whose gradients are aligned with those of the RL objective(s) are better proxies for adapting the policy directly using supervision, which is supported by related findings in computer vision literature (Sun et al., 2020). A natural solution could be to initially optimize a set of auxiliary tasks together with the RL objective(s) during training, and then exploit their gradient information to continually prune the set of tasks until only the most gradient-aligned auxiliary task remains. Similarly, a softer task selection could be to simply use gradient information to assign adaptive *weights* to each task, such that the auxiliary objective may consist of more than one task with individual influences (weights) that may or may not sum to 1 (Lin et al., 2019; Shi et al., 2020; Verboven et al., 2020; Yang et al., 2020b). Another solution may be to apply alternative methods from multi-task learning and auxiliary learning to the RL problem setting. Yu et al. (2020a) propose a method for reducing task interference in the multi-task setting by artificially realigning task gradients that are deemed problematic from an optimization point-of-view, and Wortsman et al. (2019) propose to leverage meta-learning techniques to *learn* a self-supervised objective with the desired properties.

A different opportunity for future work in policy adaptation is to revisit the adaptation algorithm of PAD. Empirically, PAD is not found to be detrimental to the performance of the policy at test-time, but we conjecture that a policy adapted to a new environment may need to be re-adapted to its original environment if it is later re-deployed there. A recent study (Bodnar et al., 2020) shows that the adaptation mechanism of PAD indeed does decrease performance in the original environment after adaptation to a different environment, which is a problem setting distinct from the single-episode deployment considered in Chapter 5. The authors empirically study the changes in feature space that occur during adaptation, and propose a mechanism for bounding the change in representation from the original environment to a given target environment. Concretely, they propose to add a behavioral cloning (Pomerleau, 1988; Ross et al., 2011) term to the test-time objective of PAD that aims to mitigate catastrophical forgetting by keeping the representation close to the pre-trained network from the original environment. The additional behavioral cloning term is shown to improve consecutive environment transfers of PAD on a number of benchmarks, including the *color_hard* test distribution of DMControl-GB proposed in Section 4.1. This is an encouraging result for self-supervised policy adaptation methods in the truly life-long learning problem setting, where a single policy may continually learn and adapt to new environments without any significant degradation in performance on previously experienced environments.

There is also ample opportunity to apply a PAD-like adaptation mechanism in other RL components. A promising, concurrent line of research (Hanna & Stone, 2017; Karnan et al., 2020; Desai et al., 2020) learns a differentiable action transformation that actively transforms actions taken by a policy trained in simulation into a re-calibrated action space for real-world deployment. In these works, the action transformation is fitted by iteratively collecting data both in simulation and the real world during learning of the policy until the policy behavior is satisfactory in both environments. However, this iterative training process can be both expensive and time consuming when an agent is not easily evaluated in the real world. We hypothesize that an online adaptation mechanism such as PAD could potentially aid such action transformations in continuously adapting to discrepancies in action spaces even after deployment, e.g. to correct minutiae inconsistencies in the learned transformation or adapt to changes over time stemming from tear.

A yet more ambitious direction for future research is to further explore variants of the flexible learning frameworks proposed in PAD and Bodnar et al. (2020) that potentially could extend PAD to also include continual acquisition of related skills, rather than only adapting to changes in the environment. To achieve this objective, it may be necessary to revisit the interaction at test-time, and develop new methods that explore the target environment both efficiently and safely during deployment. For example, it may be useful to learn a policy for probing the environment (Yang et al., 2020a) of a real robot deployment in a way that is safe (e.g. avoids damaging the robot nor its environment by acting conservatively) and sample efficient (e.g. by collecting as diverse data as possible in a given time-frame). While it is unclear exactly how future adaptive agents should be implemented, self-supervised policy adaptation remains a promising step towards more broadly intelligent behavior in agents trained by RL, and we remain optimistic about the future of visual RL.

7.3 On the Importance of Benchmarks

As a final remark, we would like to emphasize the importance of good benchmarks. DMControl is an excellent benchmark for measuring advances in sample efficiency and complexity in continuous control tasks solved by (visual) RL algorithms, but only considers tasks where the agent is trained and deployed in the same environment. Our proposed benchmark, DMControl-GB, extends the tasks in DMControl to include a systematic evaluation of generalization to visually diverse environments, which will enable future research on the topic to accurately measure algorithmic improvements in generalization.

Recent work, Grigsby & Qi (2020), conducts an empirical study on generalization in visual RL based on DMControl tasks and similar test environments as in DMControl-GB. Specifically, they propose to benchmark the same methods as in our empirical analysis, and find that data augmentation techniques (Kostrikov et al., 2020; Laskin et al., 2020a) generally are more effective tools for generalization than auxiliary self-supervised tasks such as CURL (Yarats et al., 2019; Laskin et al., 2020b). While their results are in line with the findings that we discuss in Chapter 6, their study cannot be readily reproduced due to lack of description and open-sourced implementation of test environments. We argue that, while it is important for the community to confirm results independently, we as a community ultimately need diverse, open-sourced, and accessible benchmarks, as well as reproducible implementations of algorithms, to consistently advance the field. Another recent study, Stone et al. (2021), extends our work by proposing and open-sourcing the *Distracting Control Suite*, a new benchmark for visual generalization, also built on DMControl. They propose to both train and test agents in highly configurable environments with randomized colors, video backgrounds, and dynamically changing camera poses, and benchmark recent algorithms based on SAC (Kostrikov et al., 2020; Laskin et al., 2020a) as in Grigsby & Qi (2020) and our work. In this thesis, we release an open-sourced generalization benchmark along with our two proposed methods, as well as implementations of a number of strong baselines from recent literature, including those of other recent benchmarks. We hope that our work will help foster more research on generalization in visual RL, and see the recent interest (Bodnar et al., 2020; Grigsby & Qi, 2020; Stone et al., 2021) in this topic as an encouraging sign for the future of visual RL.

8 Conclusion

Deep RL combines classical RL methods with the expressiveness of deep neural networks, and has been used with tremendous success in a number of application areas of visuomotor control, including game playing, robotic manipulation from visual inputs, and autonomous vehicles. These methods largely rely on joint learning of perception and decision-making in end-to-end optimization frameworks. One of the core challenges in RL is generalization across distributions of related environments: generalizing knowledge learned in a finite set of training environments to a potentially infinite number of related test environments that differ slightly from the training distribution. The problem of generalization is especially important to the field of robotics, where training an agent directly in the real world can be both time consuming and costly. Therefore, practitioners often learn policies in a simulated environment and subsequently deploy the policy in the real world on e.g. a robotic arm or vehicle. This approach is prone to distributional shift from potentially minutiae differences in visuals and dynamics between the simulation and the real world, which is often shown to be catastrophic. In this thesis, we extensively discuss related work that aim to tackle the problem of generalization in RL from visual observations, and we propose a suite of novel benchmarks for systematically evaluating the generalization ability of agents trained by RL. We furthermore identify limitations of current work and in that context propose two distinct frameworks for improving the generalization of current RL algorithms: PAD and SODA.

A natural solution to the problem of environment changes is to keep training after deployment in the new environment, but this cannot be done if the new environment offers no reward signal, as is common in many application areas that deploy agents in the real world. Our proposed PAD method explores the use of self-supervision to allow the policy to continue learning even 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. We do so by optimizing a self-supervised objective during deployment using observations collected directly from the new environment in an online manner. We show that PAD can adapt to a diverse set of environment changes, including a transfer from a simulated environment to a real robotic arm.

Our second method, SODA, addresses the problem of scalability that previous methods based on domain randomization and data augmentation suffer from. As more factors of variation are introduced during training through these techniques, the optimization process becomes increasingly more difficult, leading to poor sample efficiency and unstable learning. Rather than learning policies directly from augmented data, SODA decouples augmentation from policy learning, and instead imposes a soft constraint on the encoder that aims to maximize mutual information between latent representations of augmented and non-augmented data, while the RL optimization process uses strictly non-augmented data. We show that this alternative application of data augmentation improves both sample efficiency, stability, and generalization of recent methods for visual RL.

While our proposed learning frameworks are only initial steps towards general-purpose agents, we ultimately envision general-purpose frameworks in which future agents are learning all the time, with the flexibility to learn both with and without rewards through e.g. self-supervised representation learning methods, and learning both from experience collected by interaction, as well as data sources external to the agent.

Bibliography

Rishabh Agarwal, Dale Schuurmans, and Mohammad Norouzi. An optimistic perspective on offline reinforcement learning. In *International Conference on Machine Learning*, 2020. 66

Pulkit Agrawal, Ashvin Nair, P. Abbeel, Jitendra Malik, and S. Levine. Learning to poke by poking: Experiential learning of intuitive physics. *ArXiv*, abs/1606.07419, 2016. 22

Carlos Bordons Alba. Springer, 1999. ISBN 978-3-540-76241-6. 14

Ferran Alet, Kenji Kawaguchi, Tomas Lozano-Perez, and L. Kaelbling. Tailoring: encoding inductive biases by optimizing unsupervised objectives at prediction time. *ArXiv*, abs/2009.10623, 2020. 21, 66

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Multi-task feature learning. 2006. 35

Jimmy Ba, J. Kiros, and Geoffrey E. Hinton. Layer normalization. *ArXiv*, abs/1607.06450, 2016. 91

Philip Bachman, R. Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. In *NeurIPS*, 2019. 16

J. Bagnell and Brian D. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. 2010. 11

D. Ballard. Modular learning in neural networks. In *AAAI*, 1987. 16

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics Auton. Syst.*, 61:49–73, 2013. 22

David Bau, Hendrik Strobelt, W. Peebles, J. Wulff, B. Zhou, Jun-Yan Zhu, and A. Torralba. Semantic photo manipulation with a generative image prior. *ACM Transactions on Graphics (TOG)*, 38:1 – 11, 2019. 21

J. Baxter. A bayesian/information theoretic model of bias learning. *ArXiv*, abs/1911.06129, 1996. 35

Marc G. Bellemare, Yavar Naddaf, J. Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents (extended abstract). *J. Artif. Intell. Res.*, 47:253–279, 2015. 27

Yoshua Bengio, Eric Thibodeau-Laufer, G. Alain, and J. Yosinski. Deep generative stochastic networks trainable by backprop. *ArXiv*, abs/1306.1091, 2014. 16

Christopher Berner, G. Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, D. Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, R. Józefowicz, Scott Gray, C. Olsson, Jakub W. Pachocki, M. Petrov, Henrique Pond'e de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, J. Schneider, S. Sidor, Ilya Sutskever, Jie Tang, F. Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019. 1

Cristian Bodnar, Karol Hausman, Gabriel Dulac-Arnold, and Rico Jonschkowski. A geometric perspective on self-supervised policy adaptation. *ArXiv*, abs/2011.07318, 2020. 67, 68

T. Brown, Dandelion Mané, Aurko Roy, M. Abadi, and J. Gilmer. Adversarial patch. *ArXiv*, abs/1712.09665, 2017. 1

Y. Carmon, Aditi Raghunathan, L. Schmidt, Percy Liang, and John C. Duchi. Unlabeled data improves adversarial robustness. *ArXiv*, abs/1905.13736, 2019. 20

M. Caron, I. Misra, J. Mairal, Priya Goyal, P. Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. *ArXiv*, abs/2006.09882, 2020. 18

Tianlong Chen, Sijia Liu, S. Chang, Y. Cheng, L. Amini, and Zhangyang Wang. Adversarial robustness: From self-supervised pre-training to fine-tuning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 696–705, 2020a. 20

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *ArXiv*, abs/2002.05709, 2020b. 16, 17, 18, 20, 23, 51

K. Chua, R. Calandra, Rowan McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018. 22

Wesley Chung, V. Thomas, Marlos C. Machado, and Nicolas Le Roux. Beyond variance reduction: Understanding the true impact of baselines on policy optimization. *ArXiv*, abs/2008.13773, 2020. 7

K. Cobbe, O. Klimov, Christopher Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *ICML*, 2019a. 58

K. Cobbe, Christopher Hesse, J. Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *ICML*, 2020. 27

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019b. 1, 30, 57

Will Dabney, A. Barreto, M. Rowland, Robert Dadashi, John Quan, Marc G. Bellemare, and D. Silver. The value-improvement path: Towards better representations for reinforcement learning. *ArXiv*, abs/2006.02243, 2020. 51

Marc Deisenroth and Carl Rasmussen. Pilco: A model-based and data-efficient approach to policy search. pp. 465–472, 01 2011. 22

Siddharth Desai, Haresh Karnan, Josiah P. Hanna, Garrett Warnell, and Peter Stone. Stochastic grounded action transformation for robot learning in simulation, 2020. 21, 24, 66, 67

C. Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2070–2079, 2017. 35, 45

C. Doersch, A. Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1422–1430, 2015. 16

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. Carla: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017. 27

Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *ArXiv*, abs/1604.06778, 2016. 11

Rachit Dubey, Pulkit Agrawal, Deepak Pathak, Thomas L. Griffiths, and Alexei A. Efros. Investigating human priors for playing video games. In *ICML*, 2018. 28

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *ArXiv*, abs/1812.00568, 2018. 23

Abe Fetterman and Josh Albrecht. Understanding self-supervised and contrastive learning withbootstrap your own latent (byol). 2020. URL https://untitled-ai.github.io/understanding-self-supervised-contrastive-learning.html. 53

Chelsea Finn and S. Levine. Deep visual foresight for planning robot motion. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2786–2793, 2017. 23

Dhiraj Gandhi, Lerrel Pinto, and A. Gupta. Learning to fly by crashing. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3948–3955, 2017. 1

Yaroslav Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. *ArXiv*, abs/1409.7495, 2015. 21

Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. *ArXiv*, abs/1803.07728, 2018. 16, 18, 20, 21, 34, 35, 42

Ian J. Goodfellow, M. Mirza, Xia Da, Aaron C. Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211, 2014a. 21

Ian J. Goodfellow, Jean Pouget-Abadie, M. Mirza, B. Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial networks. *ArXiv*, abs/1406.2661, 2014b. 16, 53

Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2015. 1

Evan Greensmith, Peter Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 5, 04 2002. 7

Jake Grigsby and Yanjun Qi. Measuring visual generalization in continuous control from pixels. *ArXiv*, abs/2010.06740, 2020. 68

Jean-Bastien Grill, Florian Strub, Florent Altché, C. Tallec, Pierre H. Richemond, Elena Buchatskaya, C. Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, B. Piot, K. Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning. *ArXiv*, abs/2006.07733, 2020. 16, 17, 18, 20, 50, 51, 53, 54

Alvin Grissom, He He, Jordan L. Boyd-Graber, J. W. Morgan, and Hal Daumé. Don't until the final verb wait: Reinforcement learning for simultaneous machine translation. In *EMNLP*, 2014. 1

Shixiang Gu, Ethan Holly, T. Lillicrap, and S. Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3389–3396, 2017. 23

A. Gupta, Adithyavairavan Murali, Dhiraj Gandhi, and Lerrel Pinto. Robot learning in homes: Improving generalization and reducing dataset bias. *ArXiv*, abs/1807.07049, 2018. 23

David R Ha and J. Schmidhuber. World models. *ArXiv*, abs/1803.10122, 2018. 22

T. Haarnoja, Aurick Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018a. 11, 12, 37, 66

T. Haarnoja, Aurick Zhou, Kristian Hartikainen, G. Tucker, Sehoon Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *ArXiv*, abs/1812.05905, 2018b. 11, 12, 24, 36, 37, 50, 52, 54, 66, 88, 91

Danijar Hafner, T. Lillicrap, Ian S. Fischer, R. Villegas, David R Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. *ArXiv*, abs/1811.04551, 2019. 15, 22, 27

Josiah Hanna and Peter Stone. Grounded action transformation for robot learning in simulation. In *Proceedings of the 31st AAAI Conference on Artificial Intelligence (AAAI)*, February 2017. 21, 67

Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. 2020. 28, 49, 66, 85, 88

Nicklas Hansen, Rishabh Jangir, Yu Sun, Guillem Alenyà, Pieter Abbeel, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. 2020. 33, 58, 66, 88

H. V. Hasselt. Double q-learning. In *NIPS*, 2010. 13

H. V. Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double q-learning. In *AAAI*, 2016. 13

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross B. Girshick. Momentum contrast for unsupervised visual representation learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9726–9735, 2020. 16, 17, 18, 20

Olivier J. Hénaff, A. Srinivas, J. Fauw, Ali Razavi, C. Doersch, S. Eslami, and A. Oord. Data-efficient image recognition with contrastive predictive coding. In *ICML*, 2020. 16, 17

Dan Hendrycks and Thomas G. Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. *ArXiv*, abs/1903.12261, 2019. 2, 19

Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. In *ICML*, 2019a. 20

Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and D. Song. Using self-supervised learning can improve model robustness and uncertainty. *ArXiv*, abs/1906.12340, 2019b. 20, 22, 35, 45

Dan Hendrycks, K. Zhao, Steven Basart, J. Steinhardt, and D. Song. Natural adversarial examples. *ArXiv*, abs/1907.07174, 2019c. 1

Judy Hoffman, E. Tzeng, Trevor Darrell, and Kate Saenko. Simultaneous deep transfer across domains and tasks. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 4068–4076, 2015. 21

Junlin Hu, Jiwen Lu, and Yap-Peng Tan. Deep transfer metric learning. pp. 325–333, 06 2015. doi: 10.1109/CVPR.2015.7298629. 21

S. Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015. 53, 54

Max Jaderberg, V. Mnih, W. Czarnecki, T. Schaul, Joel Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. *ArXiv*, abs/1611.05397, 2017. 22, 35

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. 24

Michael Janner, Justin Fu, Marvin Zhang, and S. Levine. When to trust your model: Model-based policy optimization. *ArXiv*, abs/1906.08253, 2019. 22

Dinesh Jayaraman and K. Grauman. Learning image representations tied to ego-motion. *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1413–1421, 2015. 16

J. Johnson, B. Hariharan, L. V. D. Maaten, Judy Hoffman, Li Fei-Fei, C. L. Zitnick, and Ross B. Girshick. Inferring and executing programs for visual reasoning. *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3008–3017, 2017. 1

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. 2, 23, 66

Arthur Juliani, Ahmed Khalifa, Vincent-Pierre Berges, J. Harper, Hunter Henry, Adam Crespi, J. Togelius, and D. Lange. Obstacle tower: A generalization challenge in vision, control, and planning. *ArXiv*, abs/1902.01378, 2019. 27

Leslie Pack Kaelbling, Michael L. Littman, and Anthony R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99 – 134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. 6

Sham M. Kakade and J. Langford. Approximately optimal approximate reinforcement learning. In *ICML*, 2002. 7

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. 23

Haresh Karnan, Siddharth Desai, Josiah P. Hanna, Garrett Warnell, and Peter Stone. Reinforced grounded action transformation for sim-to-real transfer, 2020. 21, 24, 66, 67

Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, A. Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *ArXiv*, abs/2004.11362, 2020. 16, 18, 20

Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial self-supervised contrastive learning. *ArXiv*, abs/2006.07589, 2020. 20

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. 10, 50, 54

Diederik P. Kingma and M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2014. 16

B. R. Kiran, Ibrahim Sobh, V. Talpaert, Patrick Mannion, A. A. Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *ArXiv*, abs/2002.00444, 2020. 1

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. 2020. 3, 19, 27, 38, 55, 68

A. Krizhevsky. Learning multiple layers of features from tiny images. 2009. 1

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 37

A. Kumar, Justin Fu, G. Tucker, and S. Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. In *NeurIPS*, 2019. 9, 66

A. Kurakin, Ian J. Goodfellow, and S. Bengio. Adversarial examples in the physical world. *ArXiv*, abs/1607.02533, 2017. 1

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. arXiv:2004.14990, 2020a. 3, 19, 27, 37, 38, 39, 55, 56, 57, 58, 59, 60, 61, 68, 88, 94

Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *Proceedings of the 37th International Conference on Machine Learning, Vienna, Austria, PMLR 119*, 2020b. arXiv:2004.04136. 3, 17, 23, 24, 27, 35, 37, 38, 42, 51, 55, 58, 59, 60, 68, 88, 94

Yann LeCun. Self-supervised learning: Could machines learn like humans?, 2018. 14

Kimin Lee, Kibok Lee, Jinwoo Shin, and H. Lee. A simple randomization technique for generalization in deep reinforcement learning. *ArXiv*, abs/1910.05396, 2019. 19, 55, 56

Kimin Lee, M. Laskin, A. Srinivas, and P. Abbeel. Sunrise: A simple unified framework for ensemble learning in deep reinforcement learning. *ArXiv*, abs/2007.04938, 2020a. 3, 23

M. Lee, Matthew Tan, Yuke Zhu, and Jeannette Bohg. Detect, reject, correct: Crossmodal compensation of corrupted sensors. *ArXiv*, abs/2012.00201, 2020b. 51

S. Levine, P. Pastor, A. Krizhevsky, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *The International Journal of Robotics Research*, 37:421 – 436, 2018. 23

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, 23

Sergey Levine, Aviral Kumar, G. Tucker, and Justin Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *ArXiv*, abs/2005.01643, 2020. 66

J. Li, Will Monroe, Alan Ritter, Dan Jurafsky, Michel Galley, and Jianfeng Gao. Deep reinforcement learning for dialogue generation. *ArXiv*, abs/1606.01541, 2016. 1

T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016. 10, 11, 50, 52, 66

X. Lin, H. S. Baweja, G. Kantor, and David Held. Adaptive auxiliary task weighting for reinforcement learning. In *NeurIPS*, 2019. 67

Vincenzo Lomonaco, Karen Desai, Eugenio Culurciello, and Davide Maltoni. Continual reinforcement learning in 3d non-stationary environments. *arXiv preprint arXiv:1905.10112*, 2019. 27, 30, 36, 45, 66

Mingsheng Long, Han Zhu, J. Wang, and Michael I. Jordan. Unsupervised domain adaptation with residual transfer networks. In *NIPS*, 2016. 21

Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, J. Veness, Matthew J. Hausknecht, and Michael H. Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *ArXiv*, abs/1709.06009, 2018. 27

H. Maei, Csaba Szepesvari, S. Bhatnagar, Doina Precup, D. Silver, and R. Sutton. Convergent temporal-difference learning with arbitrary smooth function approximation. In *NIPS*, 2009. 9

Rupam Mahmood, Hado Van Hasselt, and Richard Sutton. Weighted importance sampling for off-policy learning with linear function approximation. volume 4, pp. 3014–3022, 01 2014. 66

Tomas Mikolov, Kai Chen, G. S. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013. 16

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, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 2015. 10, 66

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. 7, 37

R. T. Mullapudi, S. Chen, Keyi Zhang, D. Ramanan, and K. Fatahalian. Online model distillation for efficient video inference. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 3572–3581, 2019. 21

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

D. Nguyen-Tuong and Jan Peters. Using model knowledge for learning inverse dynamics. *2010 IEEE International Conference on Robotics and Automation*, pp. 2677–2682, 2010. 22

M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, 2016. 16, 20

A. Oord, Y. Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *ArXiv*, abs/1807.03748, 2018. 16, 17

C. Packer, Katelyn Gao, J. Kos, Philipp Krähenbühl, V. Koltun, and D. Song. Assessing generalization in deep reinforcement learning. *ArXiv*, abs/1810.12282, 2018. 2, 19, 28, 39, 41, 52, 58

Yunpeng Pan, Ching-An Cheng, Kamil Saigol, Keuntaek Lee, Xinyan Yan, E. Theodorou, and B. Boots. Agile autonomous driving using end-to-end deep imitation learning. In *Robotics: Science and Systems*, 2018. 1

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. 85

Deepak Pathak, Philipp Krähenbühl, J. Donahue, Trevor Darrell, and Alexei A. Efros. Context encoders: Feature learning by inpainting. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2536–2544, 2016. 13, 16, 20, 35

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *ICML*, 2017. 1, 22

Romain Paulus, Caiming Xiong, and R. Socher. A deep reinforced model for abstractive summarization. *ArXiv*, abs/1705.04304, 2018. 1

X. Peng, Marcin 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)*, pp. 1–8, 2018. 1, 2, 19

X. Peng, A. Kumar, Grace Zhang, and S. Levine. Advantage-weighted regression: Simple and scalable off-policy reinforcement learning. *ArXiv*, abs/1910.00177, 2019a. 66

Xingchao Peng, Judy Hoffman, S. Yu, and Kate Saenko. Fine-to-coarse knowledge transfer for low-res image classification. *2016 IEEE International Conference on Image Processing (ICIP)*, pp. 3683–3687, 2016. 21

Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. Moment matching for multi-source domain adaptation. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 1406–1415, 2019b. 21

Lerrel Pinto and A. Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3406–3413, 2016. 1, 23

Lerrel Pinto, Dhiraj Gandhi, Yuanfeng Han, Yong-Lae Park, and A. Gupta. The curious robot: Learning visual representations via physical interactions. In *ECCV*, 2016. 23

Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017. 55

D. Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In *NIPS*, 1988. 67

N. Qian. On the momentum term in gradient descent learning algorithms. *Neural networks : the official journal of the International Neural Network Society*, 12 1:145–151, 1999. 10

Deirdre Quillen, Eric Jang, Ofir Nachum, Chelsea Finn, J. Ibarz, and S. Levine. Deep reinforcement learning for vision-based robotic grasping: A simulated comparative evaluation of off-policy methods. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6284–6291, 2018. 1

Roberta Raileanu, M. Goldstein, Arthur Szlam, and R. Fergus. Fast adaptation via policy-dynamics value functions. *ArXiv*, abs/2007.02879, 2020a. 21, 66

Roberta Raileanu, M. Goldstein, Denis Yarats, Ilya Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *ArXiv*, abs/2006.12862, 2020b. 19, 57

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100, 000+ questions for machine comprehension of text. In *EMNLP*, 2016. 27

F. Ramos, Rafael Possas, and D. Fox. Bayessim: adaptive domain randomization via probabilistic inference for robotics simulators. *ArXiv*, abs/1906.01728, 2019. 2, 19

B. Recht, Rebecca Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? *ArXiv*, abs/1902.10811, 2019. 1

Pierre H. Richemond, Jean-Bastien Grill, Florent Altché, C. Tallec, Florian Strub, A. Brock, S. Smith, Soham De, Razvan Pascanu, B. Piot, and Michal Valko. Byol works even without batch statistics. *ArXiv*, abs/2010.10241, 2020. 53

H. Robbins. A stochastic approximation method. *Annals of Mathematical Statistics*, 22: 400–407, 2007. 10

S. Ross, G. Gordon, and J. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *AISTATS*, 2011. 67

Sebastian Ruder. An overview of gradient descent optimization algorithms. *ArXiv*, abs/1609.04747, 2016. 10

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y. 1, 18, 27

F. Sadeghi and S. Levine. Cad2rl: Real single-image flight without a single real image. *ArXiv*, abs/1611.04201, 2017. 2, 19

Ahmad El Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *ArXiv*, abs/1704.02532, 2017. 1

Andrew M. Saxe, James L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2014. 92

John Schulman, P. Moritz, S. Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2016. 7

John Schulman, P. Abbeel, and Xi Chen. Equivalence between policy gradients and soft q-learning. *ArXiv*, abs/1704.06440, 2017a. 11

John Schulman, F. Wolski, Prafulla Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017b. 11

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-efficient reinforcement learning with momentum predictive representations. *arXiv preprint arXiv:2007.05929*, 2020. 51

Ramanan Sekar, Oleh Rybkin, Kostas Daniilidis, Pieter Abbeel, Danijar Hafner, and Deepak Pathak. Planning to explore via self-supervised world models. In *ICML*, 2020. 3, 22

Pierre Sermanet, Corey Lynch, Yevgen Chebotar, Jasmine Hsu, Eric Jang, S. Schaal, and S. Levine. Time-contrastive networks : Self-supervised learning from pixels. 2017. 16

V. Shankar, Achal Dave, Rebecca Roelofs, D. Ramanan, B. Recht, and L. Schmidt. A systematic framework for natural perturbations from videos. *ArXiv*, abs/1906.02168, 2019. 2, 19

Mahmood Sharif, Sruti Bhagavatula, L. Bauer, and M. Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016. 1

Evan Shelhamer, Parsa Mahmoudieh, Max Argus, and Trevor Darrell. Loss is its own reward: Self-supervision for reinforcement learning. *ArXiv*, abs/1612.07307, 2017. 22

Baifeng Shi, Judy Hoffman, Kate Saenko, Trevor Darrell, and Huijuan Xu. Auxiliary task reweighting for minimum-data learning. *ArXiv*, abs/2010.08244, 2020. 67

Assaf Shocher, Nadav Cohen, and Michal Irani. ”zero-shot” super-resolution using deep internal learning, 2017. 21

Assaf Shocher, S. Bagon, Phillip Isola, and M. Irani. Ingan: Capturing and remapping the ”dna” of a natural image. *arXiv: Computer Vision and Pattern Recognition*, 2018. 21

M. Siam, Sara Elkerdawy, Martin Jägersand, and S. Yogamani. Deep semantic segmentation for automated driving: Taxonomy, roadmap and challenges. *2017 IEEE 20th*

International Conference on Intelligent Transportation Systems (ITSC), pp. 1–8, 2017. 1

D. Silver, G. Lever, N. Heess, T. Degris, Daan Wierstra, and Martin A. Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014. 10

D. Silver, Aja Huang, Chris J. Maddison, A. Guez, L. Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, S. Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. 1

B.F. Skinner. The behavior of organisms: An experimental analysis. *New York, London: D. Appleton-Century Company, Incorporated*, 1938. 5

T. Söderström. *Linear Quadratic Gaussian Control*, pp. 319–365. Springer, 2002. ISBN 978-1-4471-0101-7. 14

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. *arXiv*, abs/1912.02975, 2020. 28, 52

Bradly C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *ArXiv*, abs/1507.00814, 2015. 22

Austin Stone, Oscar Ramirez, Kurt Konolige, and Rico Jonschkowski. The distracting control suite – a challenging benchmark for reinforcement learning from pixels. *ArXiv*, abs/2101.02722, 2021. 68

Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. arXiv:2004.14990, 2020. 3, 23, 27, 35, 38, 51, 59

Y. Sun, E. Tzeng, Trevor Darrell, and Alexei A. Efros. Unsupervised domain adaptation through self-supervision. *ArXiv*, abs/1909.11825, 2019. 21

Y. Sun, X. Wang, Zhuang Liu, J. Miller, Alexei A. Efros, and M. Hardt. Test-time training with self-supervision for generalization under distribution shifts. In *ICML*, 2020. 20, 21, 22, 35, 66, 67

Ilya Sutskever, J. Martens, G. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *ICML*, 2013. 10

R. Sutton. Integrated architectures for learning, planning, and reacting based on approximating dynamic programming. In *ML*, 1990. 17

R. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999. 7

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. 6, 9, 12

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. 37

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. 3, 27, 28, 29, 36, 57, 85, 87

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dm_control: Software and tasks for continuous control, 2020. 27

Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. In *ECCV*, 2020a. 16, 17, 18, 20, 94

Yonglong Tian, C. Sun, Ben Poole, Dilip Krishnan, C. Schmid, and Phillip Isola. What makes for good views for contrastive learning. *ArXiv*, abs/2005.10243, 2020b. 55

Yuandong Tian, Lantao Yu, Xinlei Chen, and S. Ganguli. Understanding self-supervised learning with dual deep networks. *ArXiv*, abs/2010.00578, 2020c. 53

J. Tobin, Rachel H Fong, Alex 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)*, pp. 23–30, 2017. 2, 19, 55

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. 31, 85

J. Tremblay, Aayush Prakash, David Acuna, M. Brophy, V. Jampani, C. Anil, T. To, Eric Cameracci, Shaad Boochoon, and Stan Birchfield. Training deep networks with synthetic data: Bridging the reality gap by domain randomization. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1082–10828, 2018. 19

Sam Verboven, M. H. Chaudhary, Jeroen Berrevoets, and W. Verbeke. Hydalearn: Highly dynamic task weighting for multi-task learning with auxiliary tasks. *ArXiv*, abs/2008.11643, 2020. 67

Pascal Vincent, H. Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *ICML '08*, 2008. 16

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *BlackboxNLP@EMNLP*, 2018. 27

Alex Wang, Yada Pruksachatkun, Nikita Nangia, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Superglue: A stickier benchmark for general-purpose language understanding systems. *ArXiv*, abs/1905.00537, 2019. 27

Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *ICCV*, 2015. 16, 17

Christopher Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8: 279–292, 05 1992. doi: 10.1007/BF00992698. 8, 9

Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 1992. 7

Mitchell Wortsman, Kiana Ehsani, M. Rastegari, Ali Farhadi, and R. Mottaghi. Learning to learn how to learn: Self-adaptive visual navigation using meta-learning. *2019*

IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 6743–6752, 2019. 21, 67

Zhirong Wu, Yuanjun Xiong, S. Yu, and D. Lin. Unsupervised feature learning via non-parametric instance-level discrimination. *ArXiv*, abs/1805.01978, 2018. 16, 18

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018. 27, 30, 36, 45

L. Xiao, Y. Bahri, Jascha Sohl-Dickstein, S. Schoenholz, and Jeffrey Pennington. Dynamical isometry and a mean field theory of cnns: How to train 10, 000-layer vanilla convolutional neural networks. *ArXiv*, abs/1806.05393, 2018. 92

Tete Xiao, Xiaolong Wang, Alexei A. Efros, and Trevor Darrell. What should not be contrastive in contrastive learning. *ArXiv*, abs/2008.05659, 2020. 55

W. Yan, Ashwin Vangipuram, P. Abbeel, and Lerrel Pinto. Learning predictive representations for deformable objects using contrastive estimation. *ArXiv*, abs/2003.05436, 2020. 22

Jiachen Yang, Brenden K. Petersen, H. Zha, and D. Faissol. Single episode policy transfer in reinforcement learning. *ArXiv*, abs/1910.07719, 2020a. 21, 68

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization. *ArXiv*, abs/2003.13661, 2020b. 67

Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. 2019. 3, 23, 27, 35, 36, 38, 68, 88

L. Yen-Chen, A. Zeng, S. Song, P. Isola, and T. Y. Lin. Learning to see before learning to act: Visual pre-training for manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7286–7293, 2020. doi: 10.1109/ICRA40945.2020.9197331. 23, 62, 66

Tianhe Yu, Saurabh Kumar, A. Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *ArXiv*, abs/2001.06782, 2020a. 67

Tianhe Yu, G. Thomas, Lantao Yu, S. Ermon, J. Zou, Sergey Levine, Chelsea Finn, and Tengyu Ma. Mopo: Model-based offline policy optimization. *ArXiv*, abs/2005.13239, 2020b. 22, 66

A. Zamir, Alexander Sax, Bokui Shen, L. Guibas, Jitendra Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018. 20, 35

A. Zamir, Alexander Sax, Teresa Yeo, O. Kar, Nikhil Cheerla, Rohan Suri, Zhangjie Cao, J. Malik, and L. Guibas. Robust learning through cross-task consistency. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11194–11203, 2020. 20

Albert Zhan, P. Zhao, Lerrel Pinto, P. Abbeel, and Michael Laskin. A framework for efficient robotic manipulation. 2020. 23, 62

Marvin Zhang, Henrik Marklund, Abhishek Gupta, Sergey Levine, and Chelsea Finn. Adaptive risk minimization: A meta-learning approach for tackling group shift. *ArXiv*, abs/2007.02931, 2020a. 21, 66

Q. Zhang, Tete Xiao, Alexei A. Efros, Lerrel Pinto, and Xiaolong Wang. Learning cross-domain correspondence for control with dynamics cycle-consistency. 2020b. 24, 66

Richard Zhang. Making convolutional networks shift-invariant again. In *ICML*, 2019. 2

Richard Zhang, Phillip Isola, and Alexei A. Efros. Split-brain autoencoders: Unsupervised learning by cross-channel prediction. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 645–654, 2017. 14, 16, 20

Aurick Zhou and Sergey Levine. Amortized conditional normalized maximum likelihood. *ArXiv*, abs/2011.02696, 2020. 21

Bolei Zhou, Agata Lapedriza, Aditya Khosla, Aude Oliva, and Antonio Torralba. Places: A 10 million image database for scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 55, 86, 89

Tinghui Zhou, Richard Tucker, John Flynn, Graham Fyffe, and Noah Snavely. Stereo magnification: Learning view synthesis using multiplane images. *ACM Trans. Graph. (Proc. SIGGRAPH)*, 37, 2018. 29, 85, 86, 90

H. Zhu, A. Gupta, A. Rajeswaran, S. Levine, and V. Kumar. Dexterous manipulation with deep reinforcement learning: Efficient, general, and low-cost. *2019 International Conference on Robotics and Automation (ICRA)*, pp. 3651–3657, 2019. 1

Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *ICRA*, pp. 3357–3364. IEEE, 2017. 1

Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020. 27

Karl Johan Åström. Optimal control of markov processes with incomplete state information i. 10:174–205, 1965. ISSN 0022-247X. doi: 10.1016/0022-247X(65)90154-X. 6

A DMControl-GB Usage and Implementation Details

The *DMControl Generalization Benchmark* (DMControl-GB) (Hansen & Wang, 2020) is a benchmark for perceptual generalization in continuous control from visual observations, based on DMControl (Tassa et al., 2018). It features a variety of challenging and diverse tasks reflecting real applications such as robotic locomotion, manipulation, and grasping, and measures generalization to diverse visual changes of incremental difficulty. In DMControl-GB, agents are trained in a single, fixed environment and their generalization is evaluated on a wide variety of test environments. DMControl-GB features two distinct test distributions, *randomized colors* and *video backgrounds*, and each distribution comes in *easy* and *hard* variants. The randomized colors distributions are sets of 100 environments with distinct colors of background, floor, and the agent itself, sampled from a truncated Gaussian distribution centered at the original color values. The video distributions consist of 10 (easy) and 100 (hard) unique video backgrounds; *easy* videos are collected in the wild, and *hard* videos are a subset of the RealEstate10K (Zhou et al., 2018) dataset. Samples from each of the test distributions are shown in Figure 4.3, while the documentation and open-sourced implementation of DMControl-GB is available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark. In this appendix, we aim to provide more insight into the usage and design of DMControl-GB.

Dependencies. DMControl-GB is a generalization benchmark for the continuous control tasks proposed in DMControl (Tassa et al., 2018). DMControl itself is built with MuJoCo (Todorov et al., 2012), a proprietary physics engine commonly used in robotics and reinforcement learning research, and a local MuJoCo installation and accompanying license is therefore required in order to use DMControl-GB. All of provided algorithms are implemented in PyTorch (Paszke et al., 2019) and require at least one GPU with CUDA version ≥ 9.2 for policy learning and hardware-accelerated environment rendering. While DMControl and DMControl-GB allow for off-screen software rendering through `mesa` APIs, we strongly advise to use hardware-accelerated rendering tools such as `GLFW` or `EGL` in visual RL, as rendering may otherwise become a significant bottleneck. DMControl-GB additionally support environment monitoring tools that can export renderings to MP4-videos; our monitoring tools require `ffmpeg`. DMControl-GB by default uses the *dcm2gym* OpenAI Gym-like wrapper for DMControl, which is available at https://github.com/denisyarats/dmc2gym.

Setup. Assuming valid local installations of Git, Anaconda and MuJoCo (including license), all dependencies can be installed with the following UNIX commands:

```
$ git clone https://github.com/nicklashansen/ \
    dmcontrol-generalization-benchmark.git
$ cd dmcontrol-generalization-benchmark
$ conda env create -f setup/conda.yml
$ conda activate dmcgen
$ sh setup/install_envs.sh
```

Datasets. Part of DMControl-GB and its provided algorithms rely on external datasets. SODA uses the Places (Zhou et al., 2017) dataset for data augmentation, which can be downloaded by running

```
$ wget http://data.csail.mit.edu/places/places365/ \
    places365standard_easyformat.tar
```

You should familiarize yourself with their terms before downloading. You can access their terms of usage here: http://places2.csail.mit.edu/download.html.
After downloading and extracting the data, add the dataset directory to the `data_dirs` list found in `src/augmentations.py`. All video files required for running the video background experiments of DMControl-GB are included in the repository, namely in the `src/env/` directory. The `video_hard` environment uses a subset of the RealEstate10K (Zhou et al., 2018) dataset for background rendering.

Training and evaluation. The `scripts` directory contains training and evaluation `bash` scripts for all the included algorithms. Alternatively, the python training and evaluation scripts can be called directly, e.g. for training a policy using SODA on the `walker_walk` task with default hyper-parameters call

```
$ python3 src/train.py \
    --domain walker \
    --task walk \
    --algorithm soda \
    --aux_lr 3e-4 \
    --seed 0
```

Note that both the training script and test script require an integer `seed` argument. In our experiments, we use seeds $[0, 9]$ when evaluating across 10 seeds (e.g. in Chapter 5), and seeds $[0, 4]$ when evaluating across 5 seeds (e.g. in Chapter 6). Running the training script should give an output of the form:

```
Working directory: logs/walker_walk/soda/0
Evaluating: logs/walker_walk/soda/0
| eval | S: 0 | ER: 26.2285 | ERTEST: 25.3730
| train | E: 1 | S: 250 | D: 70.1 s | R: 0.0000 | ALOSS: 0.0000 | \
CLOSS: 0.0000 | AUXLOSS: 0.0000
```

where `ER` and `ERTEST` correspond to the mean episodic return in the training and test environments, respectively. By default, the test environment used for continuous evaluation is the `color_hard` distribution. Other distributions can be selected with the `eval_mode` argument, which accepts one of {`train`, `color_easy`, `color_hard`, `video_easy`, `video_hard`}. Training or evaluation can be executed on a specific GPU by providing the identifier for the GPU, e.g.

```
$ CUDA_VISIBLE_DEVICES=0 python3 src/eval.py \
    --algorithm soda \
    --eval_episodes 100 \
    --seed 0
```

to run evaluation on a GPU with the identifier `0`.

Tasks. DMControl-GB is based on DMControl (Tassa et al., 2018) and provides the same tasks. In this thesis work we consider a total of 9 tasks from DMControl, which we describe as follows:

- `walker, walk` ($\mathbf{a} \in \mathbb{R}^6$). A planar walker that is rewarded for walking at a constant pace.
- `walker, stand` ($\mathbf{a} \in \mathbb{R}^6$). A planar walker that is rewarded for standing with an upright torso at a constant minimum height.
- `cartpole, swingup` ($\mathbf{a} \in \mathbb{R}$). Swing up and balance an unactuated pole by applying forces to a cart at its base. The agent is rewarded for balancing the pole within some threshold angle.
- `cartpole, balance` ($\mathbf{a} \in \mathbb{R}$). Balance an unactuated pole by applying forces to a cart at its base. The agent is rewarded for balancing the pole within some threshold angle.
- `ball in cup, catch` ($\mathbf{a} \in \mathbb{R}^2$). An actuated planar receptacle is to swing and catch a ball attached by a string to its bottom. The reward signal is sparse: 1 when the ball is in the cup and 0 otherwise.
- `finger, spin` ($\mathbf{a} \in \mathbb{R}^2$). A toy manipulation problem with a planar 3 degrees-of-freedom finger. The task is to continually spin a free body. The reward signal is sparse: 1 when the finger is rotating and 0 otherwise.
- `finger, turn_easy` ($\mathbf{a} \in \mathbb{R}^2$). A toy manipulation problem with a planar 3 degrees-of-freedom finger. The task is to turn a body to a target rotation indicated by a red target. The reward signal is sparse: 1 when the body is in its target position and 0 otherwise.
- `reacher, easy` ($\mathbf{a} \in \mathbb{R}^2$). A two-link planar reaching task with a randomized target location indicated by a red target. The reward signal is sparse: 1 when the end-effector is at the target location and 0 otherwise.
- `cheetah_run` ($\mathbf{a} \in \mathbb{R}^6$). A planar running biped that is rewarded proportionally to its forward velocity.

Videos of all considered DMControl tasks are available on YouTube at https://www.youtube.com/watch?v=rAai4QzcYbs, and we provide additional videos of DMControl-GB benchmarks at https://nicklashansen.github.io/PAD and https://nicklashansen.github.io/SODA.

Compute requirements. Reinforcement learning from visual observations is computationally costly. Exact training times and memory requirements depend on the specific hardware, algorithm, environment, task, and hyper-parameters. However, one can reasonably expect a training time of 16-48 hours per run (random seed) on most tasks from DMControl-GB using recent hardware and default hyper-parameters for any of the provided algorithms. In general, algorithms that optimize auxiliary objectives during training take longer than those that do not, e.g. SAC and RAD. Additionally, domain randomization baselines are generally slow due to periodic re-initialization of the simulation at the beginning of each episode. Tasks that use a higher frame skip (action repeat) are also faster than those skipping few frames. Because all of the provided algorithms utilize a replay buffer, storing increasingly many image observations throughout training requires a large computer memory, even when storing observations as 8-bit integers (`UINT8`). SODA requires approximately 132 GB of RAM for training and the remaining algorithms require approximately 96 GB of RAM. Although the replay buffer is gradually filled with observations over time, DMControl-GB automatically pre-allocates all required memory to the training process to decrease chance of interference with other running processes and to signal requirements to any system tools that may monitor usage. Evaluation requires approximately 6 GB of RAM, and GPUs used for training and evaluation require at least 4 GB of VRAM.

Algorithms. DMControl-GB provides official implementations of the algorithms proposed in this thesis, PAD and SODA, as well as recent methods for data augmentation, generalization, and representation learning in RL. All algorithms are implemented in a unified framework that uses SAC as base algorithm and a standardized architecture and set of hyper-parameters as shown in Table 6.1. We list the provided algorithms as follows:

- SODA (Hansen & Wang, 2020)
- PAD (Hansen et al., 2020)
- RAD (Laskin et al., 2020a)
- CURL (Laskin et al., 2020b)
- SAC (Haarnoja et al., 2018b)

Implementations of SODA and PAD are provided as results of our thesis work. RAD is implemented using a hardware-accelerated random cropping implementation developed for PAD and SODA, and the CURL implementation is based on the official implementation available at https://github.com/MishaLaskin/curl. The SAC base algorithm that all methods are built off of is based on the implementation of SAC provided by the authors of SAC+AE (Yarats et al., 2019) which is available at https://github.com/denisyarats/pytorch_sac_ae. Implementations of all algorithms are available in DMControl-GB under the `src/algorithms` directory, and we discuss them in detail in Appendix B.

Data augmentations. DMControl-GB provides hardware-accelerated implementations of random cropping and the novel *random overlay* data augmentation using PyTorch. Pseudo-code for the random cropping data augmentation is as follows:

```
def random_crop_cuda(x, size=84):
  """Vectorized CUDA implementation of random crop"""
  assert isinstance(x, torch.Tensor) and x.is_cuda, \
    'input must be CUDA tensor'
  n = x.shape[0]
  img_size = x.shape[-1]
  crop_max = img_size - size
  if crop_max <= 0:
    return x

  x = x.permute(0, 2, 3, 1)
  w1 = torch.LongTensor(n).random_(0, crop_max)
  h1 = torch.LongTensor(n).random_(0, crop_max)

  windows = view_as_windows_cuda(x, (1, size, size, 1))[..., 0,:,:, 0]
  cropped = windows[torch.arange(n), w1, h1]

  return cropped
```

where view_as_windows_cuda is a function that can be written in pseudo-code as

```
def view_as_windows_cuda(x, window_shape):
  """PyTorch CUDA-enabled implementation of view_as_windows"""
  assert isinstance(window_shape, tuple) and \
    len(window_shape) == len(x.shape), \
    'window_shape must be a tuple with same number of dimensions as x'
  slices = tuple(slice(None, None, st) for st in torch.ones(4).long())
  win_indices_shape = [
    x.size(0),
    x.size(1)-int(window_shape[1]),
    x.size(2)-int(window_shape[2]),
    x.size(3)
  ]
  new_shape = tuple(list(win_indices_shape) + list(window_shape))
  strides = tuple(list(x[slices].stride()) + list(x.stride()))

  return x.as_strided(new_shape, strides)
```

Likewise, PyTorch-like pseudo-code for the random overlay data augmentation using the Places (Zhou et al., 2017) dataset is as follows:

```
def random_overlay(x):
  """Randomly overlay an image from Places"""
  alpha = 0.5

  if places_dataloader is None:
    _load_places(batch_size=x.size(0), image_size=x.size(-1))
    imgs = _get_places_batch(batch_size=x.size(0)) \
      .repeat(1, x.size(1)//3, 1, 1)

  return ((1-alpha)*(x/255.) + (alpha)*imgs)*255.
```

PAD additionally creates batches of augmented observations for self-supervised learning at test-time. PyTorch-like pseudo-code for this functionality is as follows:

```
def prepare_pad_batch(obs, next_obs, action, batch_size=32):
  """Prepare batch for self-supervised policy adaptation at test-time"""
  batch_obs = batch_from_obs(torch.from_numpy(obs) \
    .cuda(), batch_size)
  batch_next_obs = batch_from_obs(torch.from_numpy(next_obs) \
    .cuda(), batch_size)
  batch_action = torch.from_numpy(action) \
    .cuda().unsqueeze(0).repeat(batch_size, 1)

  return random_crop_cuda(batch_obs), \
    random_crop_cuda(batch_next_obs), \
    batch_action
```

where random_crop_cuda is the random cropping function that we previously provided pseudo-code for. Our implementations of SAC, PAD, and SODA are discussed in more detail in Appendix B.

Randomized colors. We implement both the randomized colors and video background environments as OpenAI Gym wrappers. We aim to briefly discuss the implementation of the randomized color wrapper in the following. A complete implementation is available in the `src/env/wrappers.py` file of the repository; the randomized color wrapper is named `ColorWrapper`. We have generated two data files, `color_easy.pt` and `color_hard.pt`, that are both located in the `src/env/data` subdirectory, each of which contain a fixed set of 100 colors sampled from a Gaussian distribution centered at the colors of the original environment. At each environment reset, we reinitialize the simulator with modified assets, i.e. the original colors have been replaced with a set of colors sampled uniformly from the corresponding color variant set (i.e. *easy* or *hard*). Because DMControl does not readily support modification of assets, our repository provides custom builds of both DMControl and `dmc2gym` that enables control over assets from within the wrapper. Lastly, we would like to emphasize that the environment sampling is seeded, which ensures reproducibility of test results across different runs and algorithms.

Video backgrounds. As in the randomized color environments, we similarly implement the video background environments as a OpenAI Gym wrapper which we name `VideoWrapper`. We refer the reader to `src/env/wrappers.py` for a complete implementation but briefly discuss the implementation in the following. Because video assets are not native to DMControl nor the underlying MuJoCo physics simulator, we choose to first render observations of the environment and then subsequently manipulate the rendering with a video overlay. To reliably manipulate renderings, we use our color wrapper to initialize the simulation with *green screen*-esque colors on the elements that we wish to replace by video, e.g. the skybox and floor. We perform image manipulation on each frame individually and apply our video overlay on all pixels above a $(100, 80, 70)$ threshold in HSV color space to threshold color and luminance individually, as is common in image processing applications. The `color_easy` set includes 10 videos that we have sourced independently, whereas the `color_hard` set uses a 100-video subset of the RealEstate10K (Zhou et al., 2018) dataset. We include all 110 videos used in our benchmark (located in `src/env/data/video_easy` and `src/env/data/video_hard`, respectively), but only provide them in a pre-processed format to maintain a lightweight repository. As in the randomized color wrapper, our video sampling is seeded to ensure reproducibility.

B Algorithm Implementations

We here aim to provide more insight into the implementation of our proposed algorithms. Specifically, we discuss the implementations that we have made available at https://github.com/nicklashansen/dmcontrol-generalization-benchmark under the `src/algorithms` directory. All methods are implemented using a Soft Actor-Critic (SAC) (Haarnoja et al., 2018b) as base algorithm, which is given in `sac.py`. We implement all network architectures using a common structure of components that we provide in `modules.py` and refer to with the shorthand `m` in the following. Our network architectures can be summarized as the following set of building blocks:

- `m.SharedCNN`: a Convolutional Neural Network (CNN) with parameters shared between the actor-critic and optionally and self-supervised task. By default, it contains 8 convolutional layers in PAD and 11 layers in all other methods.
- `m.HeadCNN`: a CNN that may share parameters between the actor and the critic, but does not share parameters with any self-supervised task. By default, `HeadCNN` is simply the identity function, but in PAD it contains 3 convolutional layers such that each network branch has 11 convolutional layers in total.
- `m.RLProjection`: a projection layer that maps extracted feature maps of previous CNN layers onto a smaller-dimensional latent space. By default, `RLProjection` contains a single linear layer followed by a `LayerNorm` (Ba et al., 2016) normalization layer that does not use batch statistics for normalization.
- `m.Encoder`: a collection of encoder components that defines the `SharedCNN`, `HeadCNN`, and `RLProjection` of a network branch.
- `m.Actor`: the actor (policy) of the actor-critic architecture. It is implemented as an MLP and uses a specified `Encoder` to extract features.
- `m.Critic`: the critic (value functions) of the actor-critic architecture. It is implemented as an MLP and uses a specified `Encoder` to extract features.

Aside from the common components listed above, we additionally provide modules for algorithm-specific components such as `m.InverseDynamics`, `m.CURLHead`, `m.SODAMLP`, and `m.SODAProjection`. We refer readers to the repository for the full source-code, but provide PyTorch-like pseudo-code for the implementation of the `m.Critic` component:

```
class Critic(nn.Module):
  def __init__(self, encoder, action_shape, hidden_dim):
    super().__init__()
    self.encoder = encoder
    self.Q1 = QFunction(
      self.encoder.out_dim, action_shape[0], hidden_dim
    )
    self.Q2 = QFunction(
      self.encoder.out_dim, action_shape[0], hidden_dim
    )
    self.apply(weight_init)

  def forward(self, x, action, detach=False):
    x = self.encoder(x, detach)
    return self.Q1(x, action), self.Q2(x, action)
```

where `QFunction` is also an `m` sub-component defined as

```
class QFunction(nn.Module):
  def __init__(self, obs_dim, action_dim, hidden_dim):
    super().__init__()
    self.trunk = nn.Sequential(
      nn.Linear(obs_dim + action_dim, hidden_dim), nn.ReLU(),
      nn.Linear(hidden_dim, hidden_dim), nn.ReLU(),
      nn.Linear(hidden_dim, 1)
    )
    self.apply(weight_init)

  def forward(self, obs, action):
    assert obs.size(0) == action.size(0)
    return self.trunk(torch.cat([obs, action], dim=1))
```

for a weight initialization function `weight_init`. By default, we apply an orthogonal initialization to all network parameters (Saxe et al., 2014; Xiao et al., 2018). Algorithm-specific update rules are implemented by extending the update rule of SAC given in pseudo-code:

```
def update(self, replay_buffer):
  obs, action, reward, next_obs, not_done = replay_buffer.sample()

  self.update_critic(obs, action, reward, next_obs, not_done)

  if step % self.actor_update_freq == 0:
    self.update_actor_and_alpha(obs)

  if step % self.critic_target_update_freq == 0:
    self.soft_update_critic_target()
```

which corresponds exactly to Algorithm 2 from Section 2.1.3. The function `update_critic` optimizes $J_{Q_1}(\nu;\theta) + J_{Q_2}(\nu;\theta)$ for a batch of transitions ν sampled from a replay buffer $\mathcal{B}$, `update_actor_and_alpha` optimizes $J_\pi(\nu;\theta)$ and $J_\alpha(\nu;\theta)$ sequentially, and finally the function `soft_update_critic_target` then updates the target Q-function parameters $\bar{\theta}_1,\ \bar{\theta}_2$ using exponential moving averages (EMA) of $\theta_1,\ \theta_2$, respectively. Readers are referred to `sac.py` for the complete implementation of SAC. We now aim to provide PyTorch-like pseudo-code for the algorithm-specific update rules of PAD and SODA.

For an inverse dynamics model implemented as a module

```
class InverseDynamics(nn.Module):
  def __init__(self, encoder, action_shape, hidden_dim):
    super().__init__()
    self.encoder = encoder
    self.mlp = nn.Sequential(
      nn.Linear(2*encoder.out_dim, hidden_dim), nn.ReLU(),
      nn.Linear(hidden_dim, hidden_dim), nn.ReLU(),
      nn.Linear(hidden_dim, action_shape[0])
    )
    self.apply(weight_init)
  def forward(self, x, x_next):
    h = self.encoder(x)
    h_next = self.encoder(x_next)
    joint_h = torch.cat([h, h_next], dim=1)
    return self.mlp(joint_h)
```

we define the PAD update rule as

```
def update_inverse_dynamics(self, obs, obs_next, action):
  pred_action = self.pad_head(obs, obs_next)
  pad_loss = F.mse_loss(pred_action, action)

  self.pad_optimizer.zero_grad()
  pad_loss.backward()
  self.pad_optimizer.step()
```

where `pad_head` is an instance of `m.InverseDynamics` and `update_inverse_dynamics` is called at every iteration of the PAD algorithm as specified in Algorithm 3. SODA can similarly be written as the PyTorch-like pseudo-code update rule

```
def update_soda(self, replay_buffer):
  x = replay_buffer.sample_soda(self.soda_batch_size)
  aug_x = x.clone()
  x = augmentations.random_crop(x)
  aug_x = augmentations.random_crop(aug_x)
  aug_x = augmentations.random_overlay(aug_x)
  soda_loss = self.compute_soda_loss(aug_x, x)

  self.soda_optimizer.zero_grad()
  soda_loss.backward()
  self.soda_optimizer.step()
  utils.soft_update_params(
    self.predictor, self.predictor_target,
    self.soda_tau
  )
```

where `compute_soda_loss` is the $\mathcal{L}_{SODA}$ defined in Equation 6.2. In pseudo-code, Equation 6.2 can be defined as

```
def compute_soda_loss(self, x0, x1):
  h0 = self.predictor(x0)
  with torch.no_grad():
    h1 = self.predictor_target.encoder(x1)
  h0 = F.normalize(h0, p=2, dim=1)
  h1 = F.normalize(h1, p=2, dim=1)

  return F.mse_loss(h0, h1)
```

where `predictor` is an instance of `m.SODAPredictor` and `predictor_target` is an EMA of `predictor` updated using the update rule from Equation 6.1. Note that the stop-gradient operation `torch.no_grad` stops gradients from flowing back into the target network during back-propagation, i.e. only the online network defined in `predictor` is updated through gradient descent. The SODA update rule here corresponds exactly to that of Algorithm 4.

Although not explicitly part of our work, we also provide implementations of CURL (Laskin et al., 2020b) and RAD (Laskin et al., 2020a). Our implementation of CURL is based on the official implementation available at https://github.com/MishaLaskin/curl, and our implementation of RAD uses the hardware-accelerated random cropping that we discuss in Appendix A. The instance-based contrastive learning objective that CURL employs is defined in Equation 2.27 and can be implemented efficiently using a multi-class cross-entropy loss with the identity matrix as labels. The authors however ablate the choice of similarity measure f and find a simple bilinear model $f(v_1, v_2) = g(v_1)$ W $g(v_2)$ (using the notation introduced in Section 2.2.2) with learnable parameters W to be more effective than the cosine similarity from Equation 2.29 that Tian et al. (2020a) employs. The CURL update rule can therefore be written as follows in a PyTorch-like pseudo-code:

```
def update_curl(self, x, x_pos):
  z_a = self.curl_head.encoder(x)
  with torch.no_grad():
    z_pos = self.critic_target.encoder(x_pos)
  logits = self.curl_head.compute_logits(z_a, z_pos)
  labels = torch.arange(logits.shape[0]).long().cuda()
  curl_loss = F.cross_entropy(logits, labels)

  self.curl_optimizer.zero_grad()
  curl_loss.backward()
  self.curl_optimizer.step()
```

where `logits` correspond to the output of the bilinear model f as previously described, `curl_head` is an instance of `m.CURLHead`, and `critic_target.encoder` is an EMA of the critic's encoder. Note that CURL uses the same projection parameters for the CURL latent space and the critic latent space, whereas our implementation of SODA uses separate projections for self-supervision and for the actor-critic network. We empirically find separate projections to work better since the projection is a tight bottleneck, but we here implement CURL as proposed in the original implementation. To generate the two views v_1, v_2 that we denote as `x, x_pos` in the above pseudo-code, we apply the same temporally consistent random cropping as in RAD to two copies of an observation, which gives us two views with different crops.

We provide a common API for all the implemented algorithms, where each algorithm can be instantiated through the command-line argument `algorithm`:

```
algorithm = {
  'sac': SAC,
  'rad': RAD,
  'curl': CURL,
  'pad': PAD,
  'soda': SODA
}

def make_agent(obs_shape, action_shape, args):
  return algorithm[args.algorithm](obs_shape, action_shape, args)
```

The reader is referred to `src/arguments.py` for a full list of arguments.

C Pre-print: *Self-Supervised Policy Adaptation during Deployment*

In this appendix, we provide a pre-print of the paper surrounding our PAD method proposed in Chapter 5. The authors and their affiliations are listed below in their entirety. Affiliations are separated by semi-colons.

- Nicklas Hansen (UC San Diego; Technical University of Denmark)
- Rishabh Jangir (UC San Diego)
- Yu Sun (UC Berkeley)
- Guillem Alenya (IRI, CSIC-UPC)
- Pieter Abbeel (UC Berkeley)
- Alexei A. Efros (UC Berkeley)
- Lerrel Pinto (NYU)
- Xiaolong Wang (UC San Diego)

The provided pre-print is dated December 10, 2020, and contains a total of 17 pages including its own references and appendices. Work on this paper was initiated during Nicklas Hansen's stay at UC Berkeley and the Berkeley Artificial Intelligence Research (BAIR) Lab, and was completed as part of this thesis.

We additionally provide supplementary material on the official project webpage: https://nicklashansen.github.io/PAD, and an open-sourced implementation of PAD as well as the complete DMControl-GB benchmark is made available here: https://github.com/nicklashansen/dmcontrol-generalization-benchmark.

Our paper can be cited as follows:

```
@article{hansen2020deployment,
  title={Self-Supervised Policy Adaptation during Deployment},
  author={Nicklas Hansen and Rishabh Jangir and Yu Sun
    and Guillem Alenyà and Pieter Abbeel and Alexei A. Efros
    and Lerrel Pinto and Xiaolong Wang},
  year={2020},
  eprint={2007.04309},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}
```

Correspondence regarding the pre-print should be directed to Nicklas Hansen.

SELF-SUPERVISED POLICY ADAPTATION DURING DEPLOYMENT

Nicklas Hansen[12], **Rishabh Jangir**[13], **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]Project page with code: `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

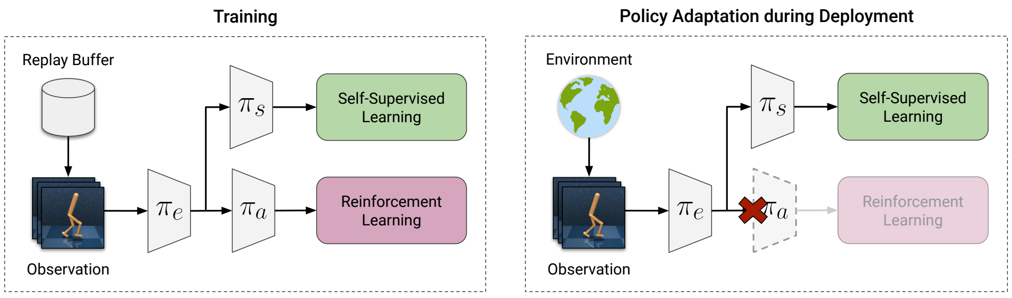


Figure 1. **Left**: Training before deployment. Observations are sampled from a replay buffer for off-policy methods and are collected during roll-outs for on-policy methods. We optimize the RL and self-supervised objectives jointly. **Right**: Policy adaptation during deployment. Observations are collected 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[1]. 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 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. 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 $t = 1...T$:

$$
\begin{aligned}
\mathbf{s}_t &\sim p(\mathbf{s}_t|\mathbf{a}_{t-1}, \mathbf{s}_{t-1}) && (2)\\
\theta_s(t) &= \theta_s(t-1) - \nabla_{\theta_s} L(\mathbf{s}_t; \theta_s(t-1), \theta_e(t-1)) && (3)\\
\theta_e(t) &= \theta_e(t-1) - \nabla_{\theta_e} L(\mathbf{s}_t; \theta_s(t-1), \theta_e(t-1)) && (4)\\
\mathbf{a}_t &= \pi(\mathbf{s}_t; \theta(t)) \quad \text{with} \quad \theta(t) = (\theta_e(t), \theta_a), && (5)
\end{aligned}
$$

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 (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. 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 DMC and CRLMaze environments are shown in Figure 2, and samples from the robot experiments are shown in Figure 4. Code is available at `https://nicklashansen.github.io/PAD/`.

Network details. For DMC 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 hyper-parameters 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 DMC 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 E for implementation details.

[1]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.

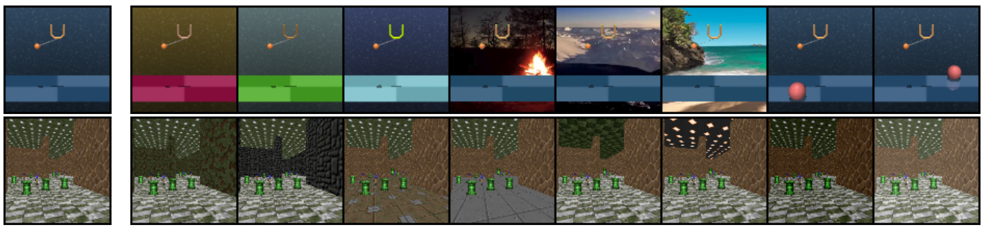


Figure 2. **Left:** Training environments of DMC (top) and CRLMaze (bottom). **Right:** Test environments of DMC (top) and CRLMaze (bottom). Changes to DMC include: randomized colors, video backgrounds, and distracting objects. Changes to CRLMaze include textures and lighting.

Table 1. Cumulative reward in test 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**

4.1 DeepMind Control

DeepMind Control (DMC) (Tassa et al., 2018) is a collection of continuous control tasks where agents only observe raw pixels. Generalization benchmarks on DMC 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 DMC 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 section 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.

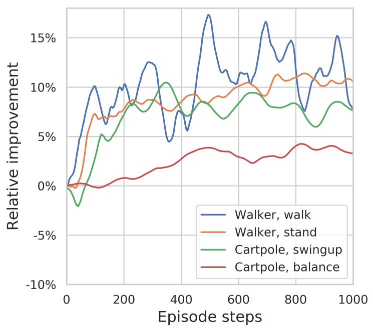


Figure 3. Relative improvement in instantaneous reward over time for PAD on the random color env.

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 section 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. 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**

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 section 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 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

Table 4. Cumulative reward 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 brown 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

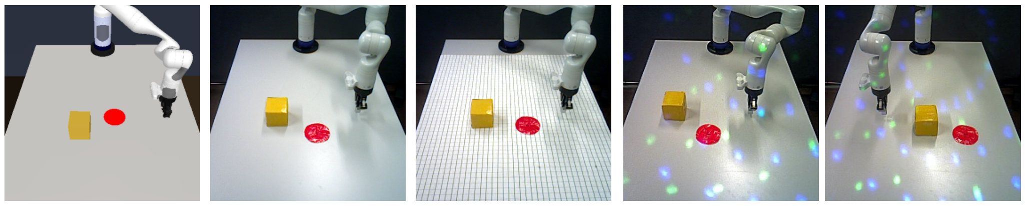

(a) Simulation. *(b)* Default transfer. *(c)* Table cloth. *(d)* Disco lights.

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 D.

Table 5. Success rate of PAD and baselines on a *real* robotic arm. Best method in each environment is in bold and brown 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 brown 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 table 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, a research grant from Qualcomm, and a gift from 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, 16

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

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, 16

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012. 16

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, 16

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, 15, 16

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, 16

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, 16

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. 16

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, 15

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, 15

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, 16

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 DMC and CRLMaze. DMC 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 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 7. 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±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. 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±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 DMC. 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.

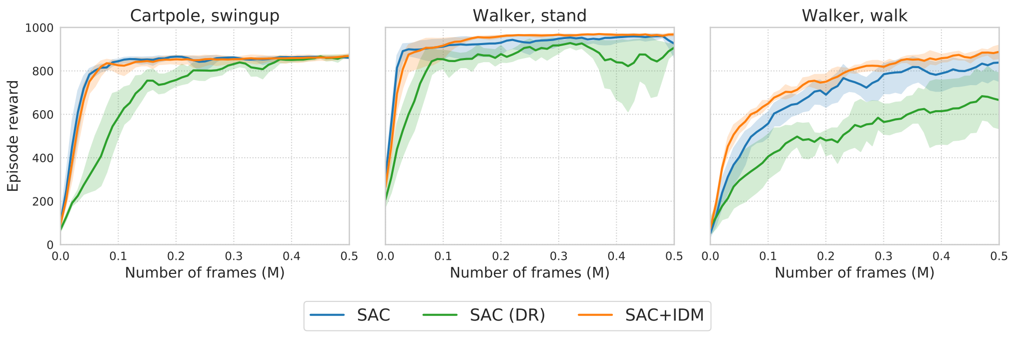


Figure 5. Learning curves for SAC, SAC trained with domain randomization (denoted *SAC (DR)* here), and SAC+IDM on three tasks from the DeepMind Control suite. Episode reward is averaged across 10 seeds and the 95% confidence intervals are visualized as shaded regions. SAC and SAC+IDM exhibit similar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

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. Empirically we 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. Cumulative reward 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 ± 29	452 ± 38	468 ± 47
Walker, stand	743 ± 37	802 ± 41	797 ± 46
Cartpole, swingup	585 ± 73	623 ± 57	630 ± 63

D 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.

E Implementation Details

In this section, we elaborate on implementation details for our experiments on DeepMind Control (DMC) 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 DMC. Code is available at `https://nicklashansen.github.io/PAD/`.

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

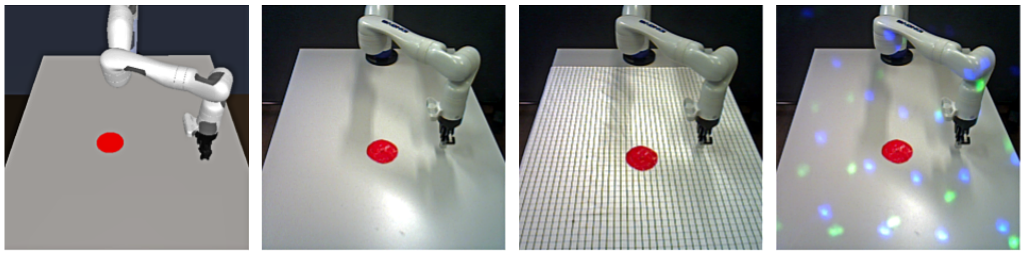

(a) Simulation. *(b)* Default transfer. *(c)* Table cloth. *(d)* Disco.

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.

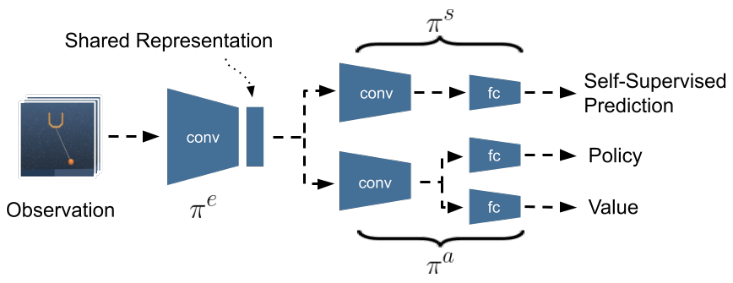


Figure 7. Network architecture for the DMC, CRLMaze, and robotic manipulation experiments. π^s and π^a uses a shared feature extractor π^e. Observations are stacks of 100×100 colored frames. Implementation of policy and value function depends on the learning algorithm.

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 10 and Table 11.

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

Table 10. Hyper-parameters for the DMC tasks.

Hyper-parameter	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 11. Hyper-parameters for the CRLMaze task.

Hyper-parameter	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

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.

D Pre-print: *Generalization in Reinforcement Learning by Soft Data Augmentation*

In this appendix, we provide a pre-print of the paper surrounding our SODA method proposed in Chapter 6. The authors and their affiliations are listed below in their entirety. Affiliations are separated by semi-colons.

- Nicklas Hansen (UC San Diego; Technical University of Denmark)
- Xiaolong Wang (UC San Diego)

The provided pre-print is dated November 26, 2020, and contains a total of 9 pages including its own references and appendices. Work on this paper was initiated and completed in full as part of this thesis.

We additionally provide supplementary material on the official project webpage: https://nicklashansen.github.io/SODA, and an open-sourced implementation of SODA as well as the complete DMControl-GB benchmark is made available here: https://github.com/nicklashansen/dmcontrol-generalization-benchmark.

Our paper can be cited as follows:

```
@article{hansen2020softda,
  title={Generalization in Reinforcement Learning by
    Soft Data Augmentation},
  author={Nicklas Hansen and Xiaolong Wang},
  year={2020},
  eprint={2011.13389},
  archivePrefix={arXiv},
  primaryClass={cs.LG}
}
```

Correspondence regarding the pre-print should be directed to Nicklas Hansen.

Generalization in Reinforcement Learning by Soft Data Augmentation

Nicklas Hansen*, Xiaolong Wang†

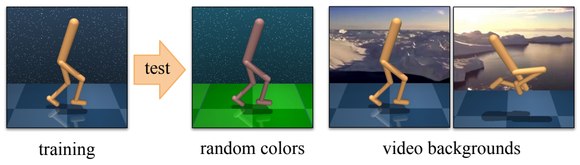
Abstract— **Extensive efforts have been made to improve the generalization ability of Reinforcement Learning (RL) methods via domain randomization and data augmentation. However, as more factors of variation are introduced during training, the optimization process becomes increasingly more difficult, leading to low sample efficiency and unstable training. Instead of learning policies directly from augmented data, we propose SOft Data Augmentation (SODA), a method that decouples augmentation from policy learning. Specifically, SODA imposes a soft constraint on the encoder that aims to maximize the mutual information between latent representations of augmented and non-augmented data, while the RL optimization process uses strictly non-augmented data. Empirical evaluations are performed on diverse tasks from DeepMind Control suite as well as a robotic manipulation task, and we find SODA to significantly advance sample efficiency, generalization, and stability in training over state-of-the-art vision-based RL methods.**[1]

I. INTRODUCTION

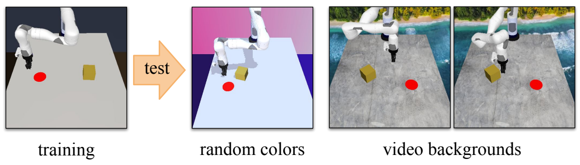
Reinforcement Learning (RL) from visual observations has achieved tremendous success in many areas, including game-playing [1], robotic manipulation [2]–[4], and navigation tasks [5], [6]. While this combination of learning both representation and decision-making jointly with RL has led to great success, numerous studies have shown that RL agents fail to generalize to new environments [7]–[11], which is only exacerbated by the high-dimensional nature of images in practical applications.

To improve generalization in RL, domain randomization [12]–[15] has been widely applied to learn robust representations invariant to visual changes. The assumption made in these approaches is that the distribution of randomized environments sufficiently cover factors of variation encountered at test-time. However, as more factors of variation are introduced during training, the optimization process becomes increasingly more difficult, leading to low sample efficiency, unstable training, and ultimately decrease in performance. If we constrain the randomization to be small, the distribution is unlikely to cover variations in the test environment. Finding the right balance between sample efficiency and generalization therefore requires extensive engineering efforts.

Similarly, recent studies also show that by using the right kinds of data augmentation, both sample efficiency and generalization of RL policies can be improved substantially [9], [16], [17]. Determining which data augmentations to use (and to which degree) is however found to be task-dependent, and a naïve application of data augmentation may lead to unstable training and poor performance [9], [17]. It is



(a) Environments for DeepMind Control tasks. We consider 5 challenging continuous control tasks from this benchmark.



(b) Environments for robotic manipulation. The task is to push the yellow cube to the location of the red disc.

Fig. 1. **Generalization in RL.** Agents are trained in a fixed environment (denoted the *training* environment) and we measure generalization to unseen environments with (i) *random colors* and (ii) *video backgrounds*. To simulate real-world deployment, we additionally randomize camera, lighting, and texture during evaluation in the robotic manipulation task. Additional samples are shown in appendix I.

therefore natural to ask whether we can obtain the benefits of data augmentation while interfering minimally with the RL optimization process.

In this paper, we propose **SO**ft **D**ata **A**ugmentation (SODA), a method that stabilizes training by decoupling data augmentation from policy learning. While previous work attempts to learn policies directly from augmented data, SODA uses strictly *non-augmented* data for policy learning, and instead performs auxiliary representation learning using *augmented* data. Through its auxiliary task, SODA learns to generalize by maximizing the mutual information between latent representations of augmented and non-augmented data. The proposed auxiliary task shares a learned encoder with the policy, and SODA therefore imposes a soft constraint on the learned representation. As the policy is learned only from non-augmented data, the difficulty of RL optimization is greatly reduced, while SODA still benefits substantially from data augmentation through representation learning.

We perform visual representation learning by projecting augmented observations into a compact latent space using the shared encoder and a learned projection, and similarly projecting non-augmented observations using a moving average of the learned network. The SODA objective is then to learn a mapping from latent features of the augmented observation to those of the original observation. With strong and varied data augmentation, SODA learns to ignore factors

*Technical University of Denmark, Denmark
†University of California, San Diego, CA, USA
[1]Project page: https://nicklashansen.github.io/SODA/

of variation that are irrelevant to the RL task, which greatly reduces observational overfitting [10]. Our visual representation learning task is a self-supervised learning task. Our approach is related to recent work on joint learning with self-supervision and RL [18]–[20], where two tasks are trained jointly on the same augmented observations. However, our method is fundamentally different. We decouple the training data flow by using non-augmented data for RL and using augmented data only for representation learning. Besides, instead of learning invariance by contrasting two augmented instances of the same image to a batch of negative samples [19], [21], [22], SODA learns to map augmented images to their non-augmented counterparts in latent space, without the need for negative samples. Because we strictly use data augmentation for representation learning and instead impose a soft constraint on the shared encoder through joint training, we call it *soft* data augmentation.

Empirical evaluations are performed on the novel *DMControl Generalization Benchmark* (DMControl-GB) based on continuous control tasks from DeepMind Control suite (DMControl) [23], as well as a robotic manipulation task. We train in a fixed environment and evaluate generalization to environments that are unseen during training. As shown in Figure 1, agents are trained in a fixed environment, and evaluated on environments with random colors and video backgrounds. Our method improves both sample efficiency and generalization over state-of-the-art vision-based RL methods in 9 out of 10 environments from DMControl-GB and all 3 types of test environments in robotic manipulation.

We highlight our main contributions as follows:

- We present SODA, a stable and efficient representation learning method for vision-based RL.
- We propose the *DMControl Generalization Benchmark*, a new benchmark for generalization in vision-based RL.
- We show that SODA outperforms prior state-of-the-art methods significantly in test-time generalization to unseen and visually diverse environments.

II. RELATED WORK

Learning Visual Invariance with Self-Supervision. Self-supervision for visual representation learning has proven highly successful in computer vision [24]–[28]. Recently, contrastive learning for self-supervised pre-training has achieved very compelling results for many downstream visual recognition tasks [21], [22], [29]–[32]. For example, Chen et al. [21] perform an extensive study on different augmentations (e.g. random cropping, color distortion, rotation) and show that learned representations invariant to these transformations can make good initializations for downstream tasks. Instead of learning with a contrastive objective, Grill et al. [33] show that invariance can also be learned via a prediction error without using any negative samples. Our method also learns invariant feature representations, but rather than aligning two different augmented views of the same instance as in previous work, we instead learn to align the representation of augmented and non-augmented views. As in Grill et al., SODA does not require negative samples.

Self-Supervised Visual Learning for RL. Inspired by the success of self-supervised visual representation learning, researchers have proposed to perform joint learning with self-supervision and RL to improve sample efficiency and performance [18], [19], [34]. For example, Yarats et al. [18] show that jointly training an auto-encoder together with RL improves the sample efficiency of model-free RL. Srinivas et al. [19] further extent this idea by performing auxiliary contrastive learning together with RL, and nearly match state-based RL in sample efficiency. While the proposed contrastive learning task uses data augmentation in training, the same augmented data is used for RL as well. In contrast to these approaches, we separate the training data into two data streams and strictly use the non-augmented data for RL, which helps stabilize training, decrease sensitivity to the choice of data augmentation, and allows us to benefit from stronger data augmentation in the self-supervised learning.

Data Augmentation and Randomization. Our work is inspired by research on generalization by domain randomization [12]–[15], [35]. For example, Tobin et al. [13] render scenes with random textures for object localization in simulation, and show that the learned networks transfer to the real world. Similar generalization across environments have been shown in recent work on RL with data augmentation [9], [16], [17], [36]. In these works, policies are trained to be invariant to certain visual perturbations by augmenting observations. However, it is unpredictable and task-dependent how much randomization can be applied. Instead of training RL policies directly on augmented observations, we propose *soft* data augmentation that decouples training into two objectives with separate data streams.

III. METHOD

We propose **SO**ft **D**ata **A**ugmentation (SODA), a novel approach to data augmentation in RL that exhibits significant improvements in sample efficiency and stability over prior work. SODA is a general framework for data augmentation that can be implemented on top of any standard RL algorithm. It aims to learn representations that effectively encode information shared between an observation and its augmented counterpart, while interfering minimally with the RL objective. In the following, we present the individual components of SODA in detail.

A. Architectural Overview

SODA is implemented as a self-supervised auxiliary task that shares a common encoder f with an RL policy. For a given policy network parameterized by a collection of parameters θ, we split the network and corresponding parameters sequentially into an encoder f_θ and a policy π_θ such that $\mathbf{a} = \pi_\theta(f_\theta(o))$ outputs a distribution over actions (and any other algorithm-specific values) for an input observation o. SODA then consists of the following three components: the shared encoder f_θ, a projection g_θ, and a prediction head h_θ. We additionally consider an architecturally identical encoder f_ψ and projection g_ψ where ψ denotes a collection of parameters separate from θ. We denote f_θ as the *online*

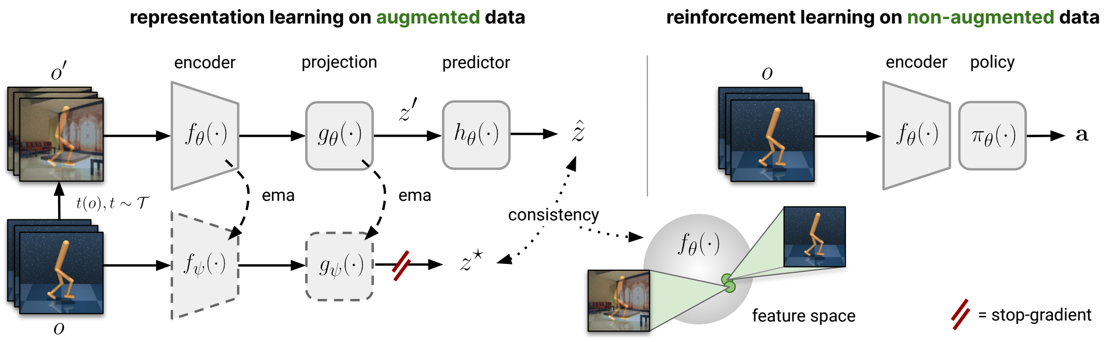


Fig. 2. **SODA architecture.** *Left:* an observation o is augmented to produce a view o', which is then encoded and projected into $z' = g_\theta(f_\theta(o'))$. Likewise, o is encoded by f_ψ and projected by g_ψ to produce features $z^\star$. The SODA objective is then to predict $z^\star$ from z' by h_θ formulated as a consistency loss. *Right:* Reinforcement Learning in SODA. The RL task remains unchanged and is trained directly on the non-augmented observations o. *ema* denotes an exponential moving average.

encoder and f_ψ as the *target* encoder, and similarly for projections g_θ, g_ψ. We then define the parameters ψ as an exponential moving average (EMA) of θ, and update ψ with every iteration of SODA using the update rule,

$$\psi_{n+1} \longleftarrow (1-\tau)\psi_n + \tau\theta_n \tag{1}$$

for an iteration step n and a momentum coefficient $\tau \in (0, 1]$, such that only parameters θ are updated by gradient descent [22], [33], [37]. Figure 2 provides an overview of the architecture, and the SODA task is detailed in the following.

B. Representation Learning by SODA

Given an observation $o \in \mathbb{R}^{C\times H\times W}$, we sample a data augmentation $t \sim \mathcal{T}$ and apply it to produce an augmented observation $o' \triangleq t(o)$. As such, o and o' can be considered different *views* of the same observation, where o is the original observation and o' is corrupted by some noise source. If we assume $f : \mathbb{R}^{C\times H\times W} \rightarrow \mathbb{R}^{C'\times H'\times W'}$ such that $H' \times W'$ is a feature map down-sampled from $H \times W$, then both projections g_θ, g_ψ are learned mappings $g : \mathbb{R}^{C'\times H'\times W'} \rightarrow \mathbb{R}^K$ where $K \ll C' \times H' \times W'$. Given a feature vector $z' \triangleq g_\theta(f_\theta(o'))$, the proposed SODA task is then to learn a mapping $h_\theta : \mathbb{R}^K \rightarrow \mathbb{R}^K$ that predicts the target projection $z^\star \triangleq g_\psi(f_\psi(o))$ of the non-augmented observation o, as shown in Figure 2 (left). We optimize projection g_θ, predictor h_θ, and shared encoder f_θ jointly together with the RL task(s), and employ a consistency loss

$$\mathcal{L}_{SODA}(\hat{z}, z^\star; \theta) = \mathbb{E}_{t\sim\mathcal{T}}\left[\|\hat{z}_\circ - z^\star_\circ\|_2^2\right] \tag{2}$$

where $\hat{z} \triangleq h_\theta(z')$, and $\hat{z}_\circ \triangleq \hat{z}/\|\hat{z}\|_2$, $z^\star_\circ \triangleq z^\star/\|z^\star\|_2$ are ℓ_2-normalizations of $\hat{z}$ and $z^\star$, respectively. We use a predictor h_θ rather than mapping z' directly to $z^\star$ as we find it to improve expressiveness in the learned representations, which is consistent with prior work [21], [33], [34].

The policy π_θ (including any algorithm-specific task heads) uses f_θ to extract features and is optimized with no modifications to architecture nor inputs, i.e. we optimize the RL objective $\mathcal{L}_{RL}$ directly on non-augmented observations o and update π_θ, f_θ by gradient descent, as shown in Figure 2 (right). During training, we continuously alternate between optimizing $\mathcal{L}_{RL}$ and $\mathcal{L}_{SODA}$. The training procedure is summarized in Algorithm 1, and the algorithmic design is further motivated in section III-C.

Algorithm 1 Soft Data Augmentation (SODA)

```
   $\theta, \psi$: randomly initialized network parameters
   $\omega$: RL updates per iteration
   $\tau$: momentum coefficient
1: for every iteration do
2:     for update $= 1, 2, ..., \omega$ do
3:         Sample batch of transitions $\nu \sim \mathcal{B}$
4:         Optimize $\mathcal{L}_{RL}(\nu)$ wrt $\theta$
5:     Sample batch of observations $o \sim \mathcal{B}$
6:     Augment observations $o' = t(o),\ t \sim \mathcal{T}$
7:     Compute online predictions $\hat{z} = h_\theta(g_\theta(f_\theta(o')))$
8:     Compute target projections $z^\star = g_\psi(f_\psi(o))$
9:     Optimize $\mathcal{L}_{SODA}(\hat{z}, z^\star)$ wrt $\theta$
10:    Update $\psi \leftarrow (1-\tau)\psi + \tau\theta$
```

C. Data Augmentation as a Mutual Information Problem

SODA reformulates the problem of generalization as a representational consistency learning problem: the encoder f should learn to map different views of the same underlying state to similar feature vectors. This encourages the encoder to learn features that are shared across views, e.g. physical quantities and object interactions, and discard information that yields no predictive power such as background, lighting, and high-frequency noise. Given an observation o and a data augmentation t, we seek to encode o and $o' = t(o)$ into compact feature vectors $z', z^\star \in \mathbb{R}^K$, respectively, by learned mappings f_θ, g_θ and f_ψ, g_ψ such that the mutual information I between o and o' is maximally preserved. The mutual information between $z^\star$ and z' is then given by

$$I(z^\star; z') = \sum_{z'}\sum_{z^\star} p(z^\star, z') \log \frac{p(z^\star|z')}{p(z^\star)} \tag{3}$$

and is naturally bounded by the mutual information between

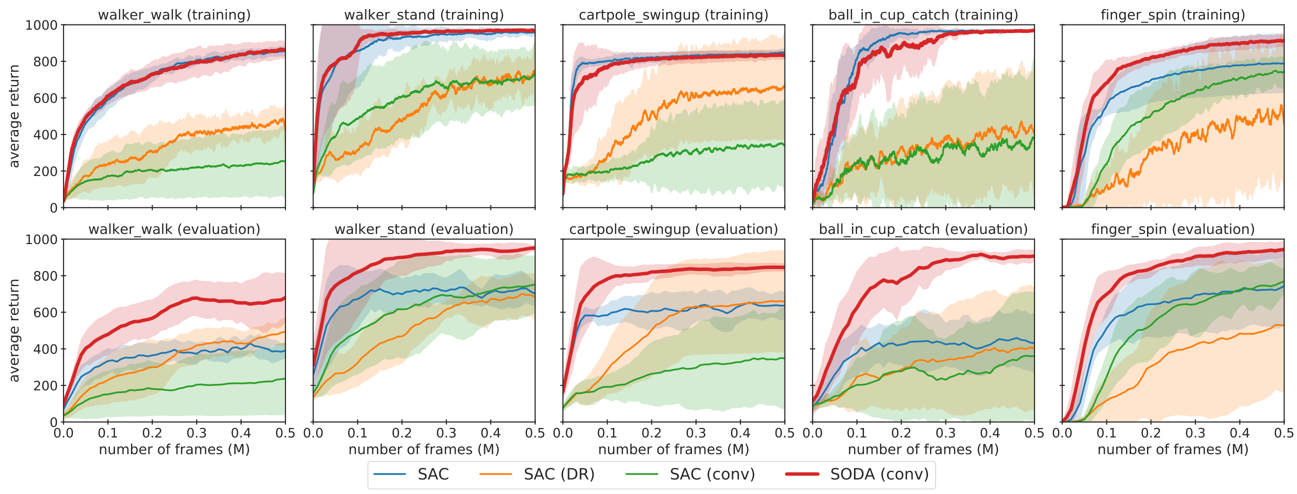


Fig. 3. **Random convolution.** *Top:* average return on the training environment during training. *Bottom:* periodic evaluation of generalization ability measured by average return on the *random color* environment. SODA exhibits sample efficiency and convergence similar to SAC but improves generalization significantly. Average of 5 runs, shaded area is std. deviation.

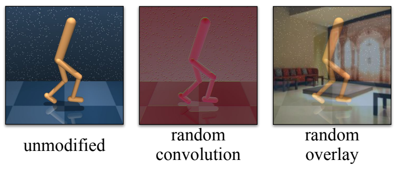


Fig. 4. **Data augmentation.** We consider the following two data augmentations: *random convolution* (as proposed by [9], [16]) and *random overlay* (novel). See appendix III for additional data augmentation samples.

o and o', i.e. $I(z^\star; z') \leq I(o; o')$. As $I(z^\star; z')$ is impractical to optimize directly, we approximate I as a simple consistency loss defined in (2) by the following intuition. If we assume f_ψ, g_ψ to maximally preserve information in o, then $I(z^\star; z') \propto I(o; o')$, and minimizing $\mathcal{L}_{SODA}$ thus maximizes $I(z^\star; z'|o')$. In other words, by learning a mapping from z' to $z^\star$, we maximally preserve information in o by learning to discard noise added by the data augmentation $t(\cdot)$. With strong and varied data augmentation, f_θ learns to ignore factors of variation that are irrelevant to the RL task, and reduces observational overfitting [10], [38]. While the assumption of maximally preserved information may not hold in practice, we find Equation 2 to be a good enough approximation for expressive representation learning. Although the SODA objective does not explicitly prevent trivial solutions, e.g. $f_\theta(\cdot) = \mathbf{0}$, such behavior is implicitly discouraged, and the reason for that is two-fold: (i) trivial equilibria in $\mathcal{L}_{SODA}$ are shown to be unstable due to updates in the targets f_ψ, g_ψ [33]; and (ii) f_θ is jointly optimized between SODA and the RL objective(s), and trivial equilibria are in direct conflict with the RL objective since they entail that f_θ then would preserve no information in o.

IV. EXPERIMENTS

We evaluate SODA on 5 tasks from DeepMind control suite (DMControl) [23], as well as in robotic manipulation. DMControl is comprised of diverse and challenging continuous control tasks and is a widely used benchmark for vision-based RL [9], [18], [19], [36], [39]. To quantify the generalization ability of SODA, we propose *DMControl Generalization Benchmark* (DMControl-GB)[2], a new benchmark for generalization in vision-based RL, based on DMControl. Agents are trained in a fixed environment (denoted the *training* environment), and we measure generalization to two distinct test distributions: (1) environments with randomized colors; and (2) environments with natural videos as background, first proposed by [39]. These test distributions correspond to the `color_hard` and `video_easy` benchmarks from DMControl-GB; we provide results for the full DMControl-GB benchmark in appendix I.

While DMControl-GB provides a good platform for benchmarking algorithms, we are ultimately interested in developing algorithms that solve real-world problems with vision-based RL. To better emulate real-world deployment scenarios, we also consider a robotic manipulation task on a robotic arm (in simulation). As in DMControl-GB, agents are trained in a fixed environment and evaluated on environments with randomized colors and video backgrounds, and we additionally perform random perturbations of camera, lighting, and texture to simulate real-world conditions during testing. Samples from DMControl-GB and robotic manipulation are shown in Figure 1, and additional samples are shown in appendix I.

A. Implementation Details

SODA is a general framework that can be implemented on top of any standard RL algorithm using parameterized policies. In this work we implement SODA using Soft Actor-Critic (SAC) [40] as the base algorithm. We adopt network architecture and hyperparameters from [39], which

[2]SODA and benchmark is open-sourced at: https://github.com/nicklashansen/dmcontrol-generalization-benchmark

implements f_θ as 11 convolutional layers and π_θ as a multi-layer perceptron (MLP). Both g_θ, h_θ are implemented as MLPs with batch normalization [41] and we use a batch size of 256 for the SODA task. Projections are of dimension $K = 100$, and the target components f_ψ, g_ψ are updated with a momentum coefficient $\tau = 0.005$. $\mathcal{L}_{RL}$ and $\mathcal{L}_{SODA}$ are optimized using Adam [42], and in practice we find it sufficient to only make a SODA update after every second RL update, i.e. $\omega = 2$. In all tasks considered, observations are stacks of k consecutive frames ($k = 3$ for DMControl-GB and $k = 1$ for robotic manipulation) of size 100×100 and with no access to state information. See appendix II for further implementation details and a list of hyper-parameters.

B. Data Augmentation

While SODA makes no assumptions about the data augmentation used, it is useful to distinguish between *weak* augmentations that may improve sample efficiency but contribute minimally to generalization, and *strong* augmentations that improve generalization at the cost of sample efficiency [43]. An example of the former is random cropping [9], [36], whereas examples of the latter are domain randomization [12], [13] and random convolution [9], [16]. Consistent with previous work [9], [19], [39], we apply temporally consistent random cropping (down to 84×84) by default in SODA and all baselines, and we refer to the cropped images as non-augmented observations.

We additionally seek to stabilize training with *strong* augmentations for generalization, and consider the following two data augmentations: *random convolution*, where a randomly initialized convolutional layer is applied; and a novel *random overlay* that linearly interpolates between an observation o and an image ε to produce an augmented view

$$t_{overlay}(o) = (1 - \alpha)o + \alpha\varepsilon, \ \varepsilon \sim \mathcal{D} \tag{4}$$

where $\alpha \in [0, 1)$ is the interpolation coefficient and $\mathcal{D}$ is an unrelated dataset. In practice, we use $\alpha = 0.5$ and sample images from Places [44], a dataset containing 1.8M diverse scenes. Both data augmentations are shown in Figure 4.

C. Baselines

We compare SODA to the following baselines: (i) a baseline SAC that is equivalent to RAD [9] as we apply random cropping to all baselines (ii) SAC trained with domain randomization on the *random colors* test distribution (denoted *SAC (DR)*); (iii) SAC using random convolution as data augmentation (denoted *SAC (conv)*; and (iv) SAC using random overlay as data augmentation (denoted *SAC (overlay)*). Likewise, our method is denoted *SODA (conv)* and *SODA (overlay)*. Data augmentation baselines perform RL on augmented observations, whereas SODA does not; we ablate this decision in section IV-D.

D. Stability under Strong Data Augmentation

Domain randomization and data augmentation is known to decrease sample efficiency and can destabilize the RL training. Figure 3 shows training performance and generalization

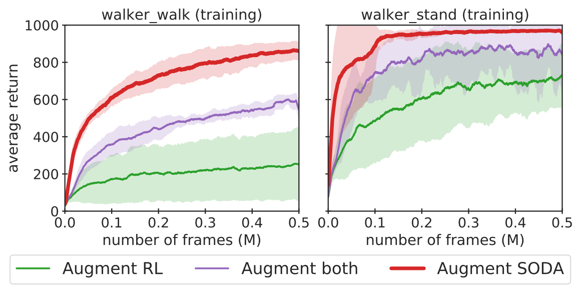


Fig. 5. **Soft data augmentation.** Average return on the training environment for `walker_walk` and `walker_stand` tasks. *Augment RL* corresponds to the SAC (conv) baseline, *Augment both* applies random convolution in both SODA and RL, and *Augment SODA* is the proposed formulation of SODA. Average of 5 runs, shaded area is std. deviation.

of baselines and SODA using random convolution. While SAC converges in training, its generalization remains poor. SODA improves generalization substantially without destabilizing the RL training, and exhibits a sample efficiency similar to the SAC baseline. Both SAC trained with domain randomization and *SAC (conv)* either fail to solve the tasks or converge to sub-optimal policies.

To further investigate the effect of soft data augmentation, we consider a variant of SODA in which random convolution augmentation is applied in both representation learning and RL, and we summarize our findings in Figure 5. While using data augmentation in both objectives (*Augment both*) improves significantly over only applying data augmentation in RL (*Augment RL*), we find the proposed formulation of SODA (*Augment SODA*) superior in both sample efficiency and end-performance.

E. Generalization to Unseen Environments

We evaluate the generalization ability of SODA on the challenging *random colors* and *video backgrounds* benchmarks from DMControl-GB (`color_hard` and `video_easy`, respectively), and additionally compare to a number of recent state-of-the-art methods: (i) CURL [19], a contrastive representation learning method; (ii) RAD [9], a study on data augmentation for RL that uses random cropping and is equivalent to the SAC baseline in section IV-C; and (iii) PAD [39], a method for self-supervised policy adaptation. Results are shown in Table I. We find SODA to outperform previous methods in **9** out of **10** instances, and by as much as **81%** on `ball_in_cup_catch` (video backgrounds). Further, SODA also outperforms the baselines from section IV-C in all tasks except `walker_stand` on video backgrounds.

While SODA with random convolution data augmentation generalizes well to the random colors distribution, it generalizes comparably worse to videos. We conjecture that this is because random convolution captures less factors of variation than overlay (e.g. textures and local color changes), which is in line with the findings of [8], [39]. SODA (overlay) is found to generalize well to both test distributions even though

TABLE I. **Generalization.** Average return of methods trained in a fixed environment and evaluated on: *(left)* DMControl-GB with natural videos as background; and *(right)* DMControl-GB with random colors. Mean and std. deviation of 5 runs.

	video backgrounds								**random colors**			
DMControl-GB (generalization)	CURL [19]	RAD [9]	PAD [39]	SAC (DR)	SAC (conv)	SODA (conv)	SAC (overlay)	SODA (overlay)	CURL [19]	RAD [9]	PAD [39]	SODA (overlay)
`walker, walk`	556 ±133	606 ±63	717 ±79	520 ±107	169 ±124	635 ±48	718 ±47	**768 ±38**	445 ±99	400 ±61	468 ±47	**692 ±68**
`walker, stand`	852 ±75	745 ±146	935 ±20	839 ±58	435 ±100	903 ±56	**960 ±2**	955 ±13	662 ±54	644 ±88	797 ±46	**893 ±12**
`cartpole, swingup`	404 ±67	373 ±72	521 ±76	524 ±184	176 ±62	474 ±143	718 ±30	**758 ±62**	454 ±110	590 ±53	630 ±63	**805 ±28**
`ball_in_cup, catch`	316 ±119	481 ±26	436 ±55	232 ±135	249 ±190	539 ±111	713 ±125	**875 ±56**	231 ±92	541 ±29	563 ±50	**949 ±19**
`finger, spin`	502 ±19	400 ±64	691 ±80	402 ±208	355 ±88	363 ±185	607 ±68	**695 ±97**	691 ±12	667 ±154	**803 ±72**	793 ±128

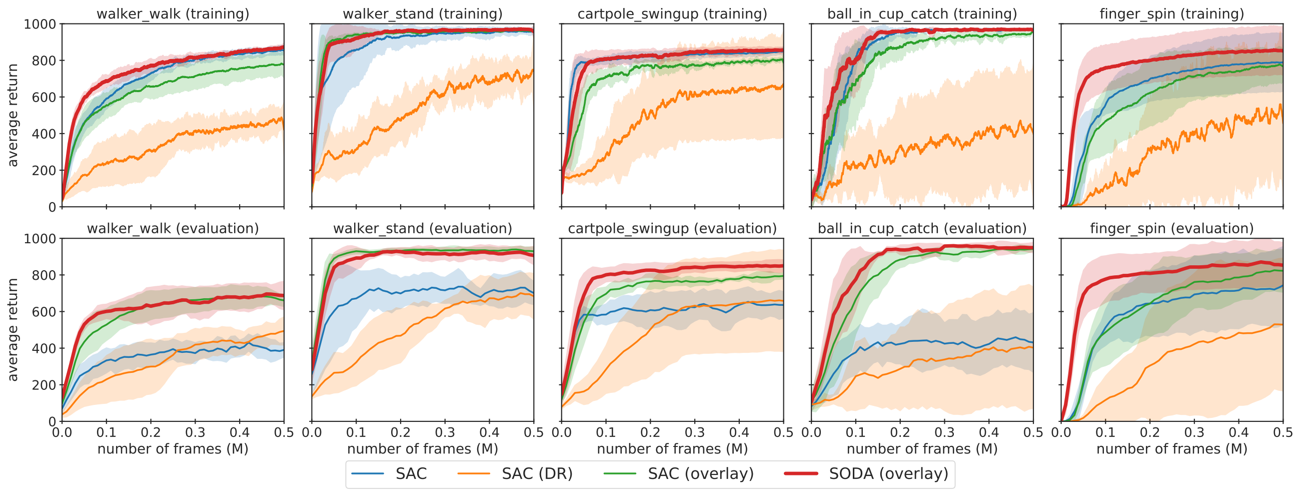


Fig. 6. **Random overlay.** *Top:* average return on the training environment during training. *Bottom:* periodic evaluation of generalization ability measured by average return on the *random color* environment. SODA offers better sample-efficiency than the novel *SAC (overlay)* baseline and similar generalization to *SODA (conv)* even though there is minimal visual similarity between random overlays and the random color environment. Average of 5 runs, shaded area is std. deviation.

there is minimal visual similarity between random overlays and the random color environment. This shows that SODA benefits from strong and varied data augmentation, and we leave it to future work to explore additional (soft) data augmentations. For completeness, training and generalization curves for the random overlay data augmentation are shown in Figure 6, and we provide results and comparisons on two additional benchmarks, `color_easy` and `video_hard`, from DMControl-GB in appendix I.

F. Robotic manipulation

We additionally consider a robotic manipulation task in which the goal is for a robotic arm to push a yellow cube to the location of a red disc (as illustrated in Figure 1). Observations are 100×100 images and agents are trained using dense rewards. Episodes consist of 50 time steps, and at each time step there is a positive reward of 1 for having the cube at the goal location (disc), and a small penalty proportional to the distance between the cube and goal. An average return of 25 therefore means that the cube is in goal position for more than half of all time steps.

TABLE II. **Generalization in robotic manipulation.** Average return of SODA and baselines in the training and test environments. Mean and std. deviation of 5 runs.

`robot, push`	SAC	SAC (DR)	SAC (conv)	SODA (conv)	SAC (overlay)	SODA (overlay)
training	14.5 ±5.2	9.3 ±7.5	0.2 ±1.7	14.4 ±4.6	15.8 ±7.3	**21.3 ±5.0**
random colors	7.2 ±3.4	7.8 ±7.0	−3.1 ±1.4	1.4 ±3.4	13.5 ±4.4	**18.0 ±3.7**
video bg.	0.9 ±6.0	7.0 ±7.4	−4.9 ±1.9	−4.5 ±1.1	7.6 ±7.1	**13.9 ±2.6**

As in DMControl-GB, we consider generalization to environments with (i) random colors; and (ii) video backgrounds. To simulate real-world deployment, we also randomize camera, lighting, and texture during testing. Training and generalization curves are shown in Figure 7, and results are summarized in Table II. We find SODA to outperform baselines in both sample efficiency and generalization to all considered test environments, which suggests that SODA may be a suitable method for reducing observational overfitting in real-world deployment of policies for robotic manipulation.

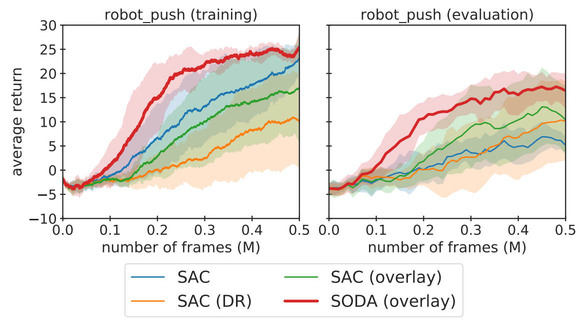


Fig. 7. **Robotic manipulation.** *Left:* average return on the training environment during training. *Right:* periodic evaluation of generalization measured by average return on the *random color* environment. SODA outperforms all baselines in both sample efficiency and generalization, and reduces variance. Average of 5 runs, shaded area is std. deviation.

V. CONCLUSION

We show empirically that SODA improves stability, sample efficiency, and generalization significantly over previous methods and a set of strong baselines. We dissect the success of SODA and find that it benefits substantially from (i) not learning policies on augmented data; (ii) its representational consistency learning formulation; and (iii) strong and varied data augmentation for representation learning.

REFERENCES

[1] 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. 1

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016. 1

[3] 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*, 2018, pp. 9191–9200. 1

[4] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *ArXiv*, vol. abs/1806.10293, 2018. 1

[5] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*. IEEE, 2017, pp. 3357–3364. 1

[6] W. Yang, X. Wang, A. Farhadi, A. Gupta, and R. Mottaghi, "Visual semantic navigation using scene priors," in *ICLR*, 2019. 1

[7] S. Gamrian and Y. Goldberg, "Transfer learning for related reinforcement learning tasks via image-to-image translation," *ArXiv*, vol. abs/1806.07377, 2019. 1

[8] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," 2018. 1, 5

[9] M. Laskin, K. Lee, A. Stooke, L. Pinto, P. Abbeel, and A. Srinivas, "Reinforcement learning with augmented data," *arXiv preprint arXiv:2004.14990*, 2020. 1, 2, 4, 5, 6, 8, 9

[10] X. Song, Y. Jiang, S. Tu, Y. Du, and B. Neyshabur, "Observational overfitting in reinforcement learning," 2019. 1, 2, 4, 8

[11] K. Cobbe, C. Hesse, J. Hilton, and J. Schulman, "Leveraging procedural generation to benchmark reinforcement learning," *ArXiv*, vol. abs/1912.01588, 2019. 1

[12] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," *arXiv preprint arXiv:1710.06542*, 2017. 1, 2, 5

[13] 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. 1, 2, 5

[14] 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. 1, 2

[15] J. Yang, B. Petersen, H. Zha, and D. Faissol, "Single episode policy transfer in reinforcement learning," 2019. 1, 2

[16] K. Lee, K. Lee, J. Shin, and H. Lee, "A simple randomization technique for generalization in deep reinforcement learning," *ArXiv*, vol. abs/1910.05396, 2019. 1, 2, 4, 5, 9

[17] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov, and R. Fergus, "Automatic data augmentation for generalization in deep reinforcement learning," *ArXiv*, vol. abs/2006.12862, 2020. 1, 2

[18] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus, "Improving sample efficiency in model-free reinforcement learning from images," 2019. 2, 4

[19] A. Srinivas, M. Laskin, and P. Abbeel, "Curl: Contrastive unsupervised representations for reinforcement learning," *arXiv preprint arXiv:2004.04136*, 2020. 2, 4, 5, 6, 8, 9

[20] A. Stooke, K. Lee, P. Abbeel, and M. Laskin, "Decoupling representation learning from reinforcement learning," arXiv:2004.14990. 2

[21] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," 2020. 2, 3

[22] K. He, H. Fan, Y. Wu, S. Xie, and R. B. Girshick, "Momentum contrast for unsupervised visual representation learning," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9726–9735, 2020. 2, 3

[23] 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," https://arxiv.org/abs/1801.00690, DeepMind, Tech. Rep., Jan. 2018. 2, 4, 8

[24] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2794–2802. 2

[25] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1422–1430. 2

[26] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros, "Context encoders: Feature learning by inpainting," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2536–2544. 2

[27] M. Noroozi and P. Favaro, "Unsupervised learning of visual representations by solving jigsaw puzzles," in *European Conference on Computer Vision*. Springer, 2016, pp. 69–84. 2

[28] R. Zhang, P. Isola, and A. A. Efros, "Colorful image colorization," in *European conference on computer vision*. Springer, 2016, pp. 649–666. 2

[29] Z. Wu, Y. Xiong, S. X. Yu, and D. Lin, "Unsupervised feature learning via non-parametric instance discrimination," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3733–3742. 2

[30] A. van den Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," 2018. 2

[31] Y. Tian, D. Krishnan, and P. Isola, "Contrastive multiview coding," *arXiv preprint arXiv:1906.05849*, 2019. 2

[32] I. Misra and L. van der Maaten, "Self-supervised learning of pretext-invariant representations," in *CVPR*, 2020. 2

[33] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, B. Piot, K. Kavukcuoglu, R. Munos, and M. Valko, "Bootstrap your own latent: A new approach to self-supervised learning," *ArXiv*, vol. abs/2006.07733, 2020. 2, 3, 4

[34] M. Schwarzer, A. Anand, R. Goel, R. D. Hjelm, A. Courville, and P. Bachman, "Data-efficient reinforcement learning with momentum predictive representations," *arXiv preprint arXiv:2007.05929*, 2020. 2, 3

[35] F. Ramos, R. Possas, and D. Fox, "Bayessim: Adaptive domain randomization via probabilistic inference for robotics simulators," *Robotics: Science and Systems XV*, Jun 2019. 2

[36] I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you

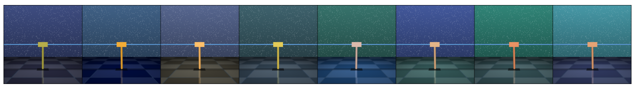
need: Regularizing deep reinforcement learning from pixels," 2020. 2, 4, 5
[37] T. Lillicrap, J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *CoRR*, vol. abs/1509.02971, 2016. 3
[38] C. Packer, K. Gao, J. Kos, P. Krähenbühl, V. Koltun, and D. Song, "Assessing generalization in deep reinforcement learning," 2018. 4
[39] N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang, "Self-supervised policy adaptation during deployment," 2020. 4, 5, 6, 8, 9
[40] 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. 4, 9
[41] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *ArXiv*, vol. abs/1502.03167, 2015. 5, 9
[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *CoRR*, vol. abs/1412.6980, 2015. 5, 9
[43] Y. Tian, C. Sun, B. Poole, D. Krishnan, C. Schmid, and P. Isola, "What makes for good views for contrastive learning," *ArXiv*, vol. abs/2005.10243, 2020. 5
[44] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017. 5, 9

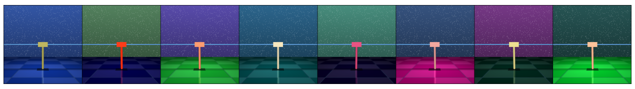
APPENDIX I
DMCONTROL GENERALIZATION BENCHMARK

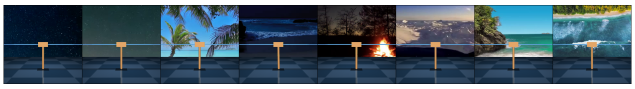
We open-source *DMControl Generalization Benchmark* (DMControl-GB), a benchmark for generalization in vision-based RL, based on the DeepMind Control suite [23]. In DMControl-GB, agents are trained in a fixed environment (denoted the *training* environment), and the generalization ability of agents is evaluated on unseen, visually diverse test distributions. Samples from each of the proposed benchmarks are shown in Figure 8. We benchmark a number of recent, state-of-the-art methods for sample-efficiency and generalization: (i) CURL [19], a contrastive representation learning method; (ii) RAD [9], a study on data augmentation for RL that uses random cropping; and (iii) PAD [39], a method for self-supervised policy adaptation. All algorithms are implemented in a unified framework using standardized architecture and hyper-parameters, wherever applicable. DMControl-GB, SODA, and implementations of the baselines are open-sourced at `https://github.com/nicklashansen/dmcontrol-generalization-benchmark`.

A. Additional results on DMControl-GB

Our main results, Figure 3, Figure 6, and Table I, consider the `color_hard` and `video_easy` benchmarks of DMControl-GB. Results from our experiments on two additional benchmarks from DMControl-GB are shown in Table III. SODA outperforms all previous state-of-the-art methods in 9 out of 10 environments and achieves near-optimal generalization on the `color_easy` benchmark. On the extremely challenging `video_hard` benchmark, SODA shows substantial improvements over baselines, but there is still room for improvement on the majority of tasks. We conjecture that the considerable performance gap between the training environment and `video_hard` is due to observational overfitting [10], and in our experiments we find that our baseline methods tend to rely on visual features such as shadows that may be correlated with the reward signal, but are not necessarily present at test-time.


(a) `color_easy`


(b) `color_hard`


(c) `video_easy`

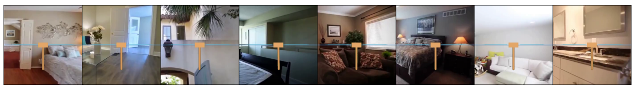

(d) `video_hard`

Fig. 8. **Samples from DMControl-GB.** Agents are trained in a fixed environment (denoted the *training* environment), and the generalization ability of agents is evaluated on the test distributions shown in (a-d). Environments (a-b) randomize the color of background, floor, and the agent itself, while (c-d) replaces the background with natural videos. In (c), only the skybox is replaced, whereas the entirety of the background is replaced in (d). Our main results, Figure 3, Figure 6, and Table I, consider the `color_hard` and `video_easy` benchmarks of DMControl-GB.

TABLE III. Average return of SODA and state-of-the-art methods on the `color_easy` and `video_hard` environments from DMControl-GB. Mean and std. deviation of 5 runs. Results for `color_hard` and `video_easy` are shown in Table I.

DMControl-GB (color_easy)	CURL [19]	RAD [9]	PAD [39]	SODA (overlay)
`walker, walk`	645 ±55	636 ±33	687 ±119	**811 ±41**
`walker, stand`	866 ±46	807 ±67	894 ±39	**960 ±4**
`cartpole, swingup`	668 ±74	763 ±29	812 ±20	**859 ±15**
`ball_in_cup, catch`	565 ±168	727 ±87	775 ±159	**969 ±3**
`finger, spin`	781 ±139	789 ±160	**870 ±54**	855 ±93

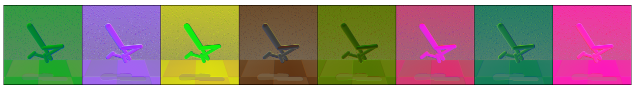
DMControl-GB (video_hard)	CURL [19]	RAD [9]	PAD [39]	SODA (overlay)
`walker, walk`	58 ±18	56 ±9	93 ±29	**381 ±72**
`walker, stand`	45 ±5	231 ±39	278 ±72	**771 ±83**
`cartpole, swingup`	114 ±15	110 ±16	123 ±24	**429 ±64**
`ball_in_cup, catch`	115 ±33	97 ±29	66 ±61	**327 ±100**
`finger, spin`	27 ±21	34 ±11	56 ±18	**302 ±41**

TABLE IV. Hyper-parameters for the DMControl-GB tasks.

Hyperparameter	Value
Frame rendering	$3 \times 100 \times 100$
Frame after crop	$3 \times 84 \times 84$
Stacked frames	3
Number of conv. layers	11
Number of filters in conv.	32
Action repeat	2 (`finger_spin`)
	8 (`cartpole_swingup`)
	4 (otherwise)
Discount factor γ	0.99
Episode time steps	1,000
Learning algorithm	Soft Actor-Critic
Number of training steps	500,000
Replay buffer size	500,000
Optimizer (RL/aux.)	Adam ($\beta_1 = 0.9, \beta_2 = 0.999$)
Optimizer (α)	Adam ($\beta_1 = 0.5, \beta_2 = 0.999$)
Learning rate (RL)	1e-3
Learning rate (α)	1e-4
Learning rate (SODA)	3e-4
Batch size (RL)	128
Batch size (SODA)	256
Actor update freq.	2
Critic update freq.	1
Auxiliary update freq.	1 (CURL)
	2 (PAD/SODA)
	N/A (otherwise)
Momentum coef. τ (SODA)	0.005

Appendix II
Implementation Details

Table IV summarizes the hyper-parameters of SODA and baselines used in our DMControl-GB experiments. We use identical algorithms, architectures, and hyper-parameters for robotic manipulation, but opt for observations as stacks of $k = 3$ frames in DMControl-GB and $k = 1$ in robotic manipulation. We implement SODA and baselines using Soft Actor-Critic (SAC) [40] as the base algorithm, and adopt network architecture and hyperparameters from [39]. We apply the same architecture and hyperparameters in all methods, wherever applicable. As the action spaces of both DMControl-GB and the robotic manipulation task are continuous, the policy learned by SAC outputs the mean and variance of a Gaussian distribution over actions. Frames are RGB images rendered at 100×100 and cropped to 84×84 as in [9], [19], [39], such that inputs to the network are of size $3k \times 84 \times 84$, where the first dimension indicates the color channels and the following dimensions represent spatial dimensions. The same crop is applied to all frames in a stack. The shared encoder consists of 11 convolutional layers and outputs features of size $32 \times 21 \times 21$. In our SODA implementation, both g_θ, h_θ are implemented as MLPs with batch normalization [41] and we use a batch size of 256 for the SODA task. Projections are of dimension $K = 100$, and the target components f_ψ, g_ψ are updated with a momentum coefficient $\tau = 0.005$. $\mathcal{L}_{RL}$ and $\mathcal{L}_{SODA}$ are optimized using Adam [42], and in practice we find it sufficient to only make a SODA update after every second RL update, i.e. $\omega = 2$.


(a) *Random convolution* data augmentation.

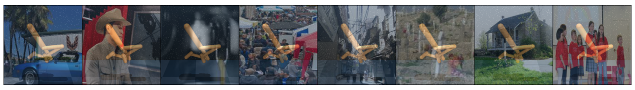

(b) *Random overlay* data augmentation.

Fig. 9. **Samples from data augmentation.** In this work, we consider the *random convolution* data augmentation shown in (a), as well as the novel *random overlay* shown in (b).

Appendix III
Data Augmentations

Section IV-B discusses two data augmentations that we consider in this work: (1) *random convolution* [9], [16], where a randomly initialized convolutional layer is applied; and (2) *random overlay*, where we linearly interpolate between an observation o and an image ε to produce an augmented view $o' = t_{overlay}(o)$, as formalized in Equation 4. In practice, we use an interpolation coefficient of $\alpha = 0.5$ and sample images from Places [44], a dataset containing 1.8M diverse scenes. Samples from the two data augmentations are shown in Figure 9.