Reinforcement Learning with Augmented Data

Michael Laskin* UC Berkeley
Kimin Lee* UC Berkeley
Adam Stooke UC Berkeley
Lerrel Pinto UC Berkeley, NYU
Pieter Abbeel UC Berkeley
Aravind Srinivas UC Berkeley

Abstract

Learning from visual observations is a fundamental yet challenging problem in Reinforcement Learning (RL). Although algorithmic advances combined with convolutional neural networks have proved to be a recipe for success, current methods are still lacking on two fronts: (a) data-efficiency of learning and (b) generalization to new environments. To this end, we present Reinforcement Learning with Augmented Data (RAD), a simple plug-and-play module that can enhance most RL algorithms. We perform the first extensive study of general data augmentations for RL on both pixel-based and state-based inputs, and introduce two new data augmentations - *random translate* and *random amplitude scale*. We show that augmentations such as random translate, crop, color jitter, patch cutout, random convolutions, and amplitude scale can enable simple RL algorithms to outperform complex state-of-the-art methods across common benchmarks. RAD sets a new state-of-the-art in terms of data-efficiency and final performance on the DeepMind Control Suite benchmark for pixel-based control as well as OpenAI Gym benchmark for state-based control. We further demonstrate that RAD significantly improves test-time generalization over existing methods on several OpenAI ProcGen benchmarks. Our RAD module and training code are available at https://www.github.com/MishaLaskin/rad.

1 Introduction

Learning from visual observations is a fundamental problem in reinforcement learning (RL). Current success stories build on two key ideas: (a) using expressive convolutional neural networks (CNNs) [1] that provide strong spatial inductive bias; (b) better credit assignment [2–4] techniques that are crucial for sequential decision making. This combination of CNNs with modern RL algorithms has led to impressive success with human-level performance in Atari [2], super-human Go players [5], continuous control from pixels [3, 4] and learning policies for real-world robot grasping [6].

While these achievements are truly impressive, RL is notoriously plagued with poor data-efficiency and generalization capabilities [7, 8]. Real-world successes of reinforcement learning often require *months* of data-collection and (or) training [6, 9]. On the other hand, biological agents have the remarkable ability to learn quickly [10, 11], while being able to generalize to a wide variety of unseen tasks [12]. These challenges associated with RL are further exacerbated when we operate on pixels due to high-dimensional and partially-observable inputs. Bridging the gap of data-efficiency and generalization is hence pivotal to the real-world applicability of RL.

Supervised learning, in the context of computer vision, has addressed the problems of data-efficiency and generalization by injecting useful priors. One such often ignored prior is *Data Augmentation*. It was critical to the early successes of CNNs [1, 13] and has more recently enabled better supervised [14,

*Equal contribution.

15], semi-supervised [16–18] and self-supervised [19–21] learning. By using multiple augmented views of the same data-point as input, CNNs are forced to learn consistencies in their internal representations. This results in a visual representation that improves generalization [17, 19–21], data-efficiency [16, 19, 20] and transfer learning [19, 21].

Inspired by the impact of data augmentation in computer vision, we present RAD: **R**einforcement Learning with **A**ugmented **D**ata, a technique to incorporate data augmentations on input observations for reinforcement learning pipelines. Through RAD, we ensure that the agent is learning on multiple views (or augmentations) of the same input (see Figure 1). This allows the agent to improve on *two key capabilities*: (a) **data-efficiency**: learning to quickly master the task at hand with drastically fewer experience rollouts; (b) **generalization**: improving transfer to unseen tasks or levels simply by training on more diversely augmented samples. To the best of our knowledge, we present the first extensive study of the use of data augmentation for reinforcement learning with *no changes* to the underlying RL algorithm and no additional assumptions about the domain other than the knowledge that the agent operates from image-based or proprioceptive (positions & velocities) observations.

We highlight the main contributions of RAD below:

- We show that *RAD outperforms prior state-of-the-art baselines* on both the widely used pixel-based DeepMind control benchmark [22] as well as state-based OpenAI Gym benchmark [23]. On both benchmark, RAD sets a new state-of-the-art *in terms data-efficiency and asymptotic performance* on the majority of environments tested.
- We show that RAD significantly *improves test-time generalization* on several environments in the OpenAI ProcGen benchmark suite [24] widely used for generalization in RL.
- We *introduce two new data augmentations*: **random translation** for image-based input and **random amplitude scaling** for proprioceptive input that are utilized to achieve state-of-the-art results. To the best of our knowledge, these augmentations were not used in prior work.

2 Related work

2.1 Data augmentation in computer vision

Data augmentation in deep learning systems for computer vision can be found as early as LeNet-5 [1], an early implementation of CNNs on MNIST digit classification. In AlexNet [13] wherein the authors applied CNNs to image classification on ImageNet [25], data augmentations, such as random flip and crop, were used to improve the classification accuracy. These data augmentations inject the priors of invariance to translation and reflection, playing a significant role in improving the performance of supervised computer vision systems. Recently, new augmentation techniques such as AutoAugment [14] and RandAugment [15] have been proposed to further improve the performance. For unsupervised and semi-supervised learning, several unsupervised data augmentation techniques have been proposed [18, 16, 26]. In particular, contrastive representation learning approaches [19–21] with data augmentations have recently dramatically improved the label-efficiency of downstream vision tasks like ImageNet classification.

2.2 Data augmentation in reinforcement learning

Data augmentation has also been investigated in the context of RL though, to the best of our knowledge, there was no extensive study on a variety of widely used benchmarks prior to this work. For improving generalization in RL, domain randomization [27–29] was proposed to transfer policies from simulation to the real world by utilizing diverse simulated experiences. Cobbe et al. [30] and Lee et al. [31] showed that simple data augmentation techniques such as cutout [30] and random convolution [31] can be useful to improve generalization of agents on the OpenAI CoinRun and ProcGen benchmarks.

To improve the data-efficiency, CURL [32] utilized data augmentations for learning contrastive representations in the RL setting. While the focus in CURL was to make use of data augmentations jointly through contrastive and reinforcement learning losses, RAD attempts to directly use data augmentations for reinforcement learning without any auxiliary loss (see Section I for discussions on tradeoffs between CURL and RAD). Concurrent and independent to our work, DrQ [33] utilized random cropping and regularized Q-functions in conjunction with the off-policy RL algorithm

SAC [34]. On the other hand, RAD can be plugged into any reinforcement learning method (on-policy methods like PPO [4] and off-policy methods like SAC [34]) *without making any changes to the underlying algorithm*.

For a more detailed and comprehensive discussion of prior work, we refer the reader to Appendix A.

3 Background

RL agents act within a Markov Decision Process, defined as the tuple $(\mathcal{S}, \mathcal{A}, P, \gamma)$, with the following components: states $s \in \mathcal{S} = \mathbb{R}^n$, actions $a \in \mathcal{A}$, and state transition distribution, $P = P(s_{t+1}, r_t | s_t, a_t)$, which defines the task mechanics and rewards. Without prior knowledge of P, the RL agent's goal is to use experience to maximize expected rewards, $R = \sum_{t=0}^{\infty} \gamma^t r_t$, under discount factor $\gamma \in [0, 1)$. Crucially, in RL from pixels, the agent receives image-based observations, $o_t = O(s_t) \in \mathbb{R}^k$, which are a high-dimensional, indirect representation of the state.

Soft Actor-Critic. SAC [34] is a state-of-the-art off-policy algorithm for continuous controls. SAC learns a policy $\pi_\psi(a|o)$ and a critic $Q_\phi(o, a)$ by maximizing a weighted objective of the reward and the policy entropy, $\mathbb{E}_{s_t, a_t \sim \pi} \left[\sum_t r_t + \alpha \mathcal{H}(\pi(\cdot|o_t)) \right]$. The critic parameters are learned by minimizing the squared Bellman error using transitions $\tau_t = (o_t, a_t, o_{t+1}, r_t)$ from an experience buffer $\mathcal{D}$,

$$\mathcal{L}_Q(\phi) = \mathbb{E}_{\tau \sim \mathcal{D}} \left[\left(Q_\phi(o_t, a_t) - (r_t + \gamma V(o_{t+1})) \right)^2 \right]. \tag{1}$$

The target value of the next state can be estimated by sampling an action using the current policy:

$$V(o_{t+1}) = \mathbb{E}_{a' \sim \pi} \left[Q_{\bar{\phi}}(o_{t+1}, a') - \alpha \log \pi_\psi(a'|o_{t+1}) \right], \tag{2}$$

where $Q_{\bar{\phi}}$ represents a more slowly updated copy of the critic. The policy is learned by minimizing the divergence from the exponential of the soft-Q function at the same states:

$$\mathcal{L}_\pi(\psi) = -\mathbb{E}_{a \sim \pi} \left[Q_\phi(o_t, a) - \alpha \log \pi_\psi(a|o_t) \right], \tag{3}$$

via the reparameterization trick for the newly sampled action. α is learned against a target entropy.

Proximal policy optimization. PPO [4] is a state-of-the-art on-policy algorithm for learning a continuous or discrete control policy, $\pi_\theta(a|o)$. PPO forms policy gradients using action-advantages, $A_t = A^\pi(a_t, s_t) = Q^\pi(a_t, s_t) - V^\pi(s_t)$, and minimizes a clipped-ratio loss over minibatches of recent experience (collected under $\pi_{\theta_{old}}$):

$$\mathcal{L}_\pi(\theta) = -\mathbb{E}_{\tau \sim \pi} \left[\min \left(\rho_t(\theta) A_t,\ \text{clip}(\rho_t(\theta), 1 - \epsilon, 1 + \epsilon) A_t \right) \right], \quad \rho_t(\theta) = \frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{old}}(a_t|o_t)}. \tag{4}$$

Our PPO agents learn a state-value estimator, $V_\phi(s)$, which is regressed against a target of discounted returns and used with Generalized Advantage Estimation [4]:

$$\mathcal{L}_V(\phi) = \mathbb{E}_{\tau \sim \pi} \left[\left(V_\phi(o_t) - V_t^{targ} \right)^2 \right]. \tag{5}$$

4 Reinforcement learning with augmented data

We investigate the utility of data augmentations in model-free RL for both off-policy and on-policy settings by processing image observations with stochastic augmentations before passing them to the agent for training. For the base RL agent, we use SAC [34] and PPO [4] as the off-policy and on-policy RL methods respectively. During training, we sample observations from either a replay buffer or a recent trajectory and augment the images within the minibatch. In the RL setting, it is common to stack consecutive frames as observations to infer temporal information such as object velocities. Crucially, augmentations are applied randomly across the batch but consistently across the frame stack [32] as shown in Figure 1.[2] This enables the augmentation to retain temporal information present across the frame stack.

[2] For on-policy RL methods such as PPO, we apply the different augmentations across the batch but consistently across time.

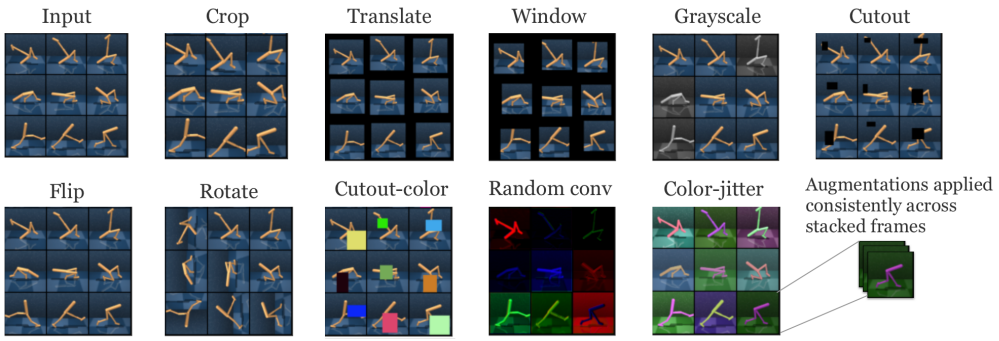


Figure 1: We investigate ten different types of data augmentations - crop, translate, window, grayscale, cutout, cutout-color, flip, rotate, random convolution, and color-jitter. During training, a minibatch is sampled from the replay buffer or a recent trajectory randomly augmented. While augmentation across the minibatch is stochastic, it is consistent across the stacked frames.

Augmentations of image-based input: Across our experiments, we investigate and ablate crop, translate, window, grayscale, cutout, cutout-color, flip, rotate, random convolution, and color jitter augmentations, which are shown in Figure 1. Of these, the *translate and window are novel augmentations* that we did not encounter in prior work.

Crop: Extracts a random patch from the original frame. For example, in DMControl we render 100×100 pixel frames and crop randomly to 84×84 pixels. As our experiments will confirm, the intuition behind random cropping is primarily to imbue the agent with additional translation invariance. **Translate**: Renders the full image within a larger frame and translates the image randomly across the larger frame. For example, a 100×100 pixel image could be randomly translated within a 108×108 empty frame. **Window**: Selects a random window from an image by masking out the cropped part of the image. **Grayscale**: Converts RGB images to grayscale with some random probability p. **Cutout**: Randomly inserts a small black occlusion into the frame, which may perceived as cutting out a small patch from the originally rendered frame. **Cutout-color**: Another variant of cutout where instead of rendering black, the occlusion color is randomly generated. **Flip**: Flips an image at random across the vertical axis. **Rotate**: Randomly samples an angle from the following set $\{0°, 90°, 180°, 270°\}$ and rotates the image accordingly. **Random convolution**: First introduced in [31], augments the image color by passing the input observation through a random convolutional layer. **Color jitter**: Converts RGB image to HSV and adds noise to the HSV channels, which results in explicit color jittering.

Extension to state-based inputs: We also consider an extension to state-based inputs such as proprioceptive features (e.g., positions and velocities). Specifically, we propose two data augmentation techniques: (a) **random amplitude scaling** which multiplies the uniform random variable, i.e., $s' = s * z$, where $z \sim \text{Uni}[\alpha, \beta]$, and (b) **Gaussian noise** which adds Gaussian random variable, i.e., $s' = s + z$, where $z \sim \mathcal{N}(0, I)$. Here, s', s and z are all vectors (see Appendix J for more details). Similar to image inputs, augmentations are applied randomly across the batch but consistently across the time, i.e., same randomization to current and next input states. Of these, *random amplitude scaling is a novel augmentation*. The intuition behind these augmentations is that *random amplitude scaling* randomizes the amplitude of input states while maintaining their intrinsic information (e.g., sign of inputs). We also remark that *Gaussian noise* is related to offset invariance.

5 Experimental results

5.1 Setup

DMControl: First, we focus on studying the data-efficiency of our proposed methods on pixel-based RL. To this end, we utilize the DeepMind Control Suite (DMControl) [22], which has recently become a common benchmark for comparing efficient RL agents, both model-based and model-free. DMControl presents a variety of complex tasks including bipedal balance, locomotion, contact forces,

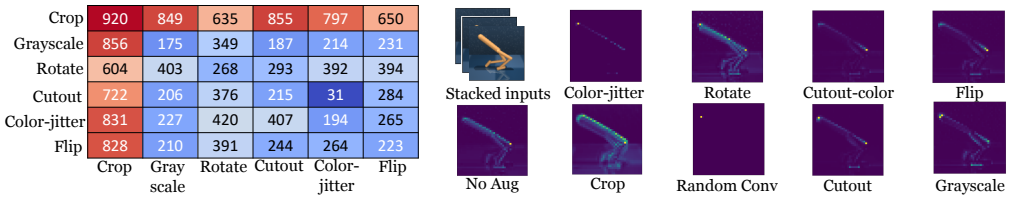
Table 1: We report scores for RAD and baseline methods on DMControl100k and DMControl500k. In both settings, RAD achieves state-of-the-art performance on all (**6** out of **6**) environments. We selected these 6 environments for benchmarking due to availability of baseline performance data from CURL [32], PlaNet [35], Dreamer [36], SAC+AE [37], and SLAC [38]. We also show performance data on 15 environments in total in the Appendix D. Results are reported as averages across 10 seeds for the 6 main environments. A full list of hyperparameters is provided in Table 4 of Appendix E.

500K STEP SCORES	RAD	CURL	PLANET	DREAMER	SAC+AE	SLACV1	PIXEL SAC	STATE SAC
FINGER, SPIN	**947** ± 101	926 ± 45	561 ± 284	796 ± 183	884 ± 128	673 ± 92	192 ± 166	923 ± 211
CARTPOLE, SWING	**863** ± 9	845 ± 45	475 ± 71	762 ± 27	735 ± 63	-	419 ± 40	848 ± 15
REACHER, EASY	**955** ± 71	929 ± 44	210 ± 44	793 ± 164	627 ± 58	-	145 ± 30	923 ± 24
CHEETAH, RUN	**728** ± 71	518 ± 28	305 ± 131	570 ± 253	550 ± 34	640 ± 19	197 ± 15	795 ± 30
WALKER, WALK	**918** ± 16	902 ± 43	351 ± 58	897 ± 49	847 ± 48	842 ± 51	42 ± 12	948 ± 54
CUP, CATCH	**974** ± 12	959 ± 27	460 ± 380	879 ± 87	794 ± 58	852 ± 71	312 ± 63	974 ± 33
100K STEP SCORES								
FINGER, SPIN	**856** ± 73	767 ± 56	136 ± 216	341 ± 70	740 ± 64	693 ± 141	224 ± 101	811 ± 46
CARTPOLE, SWING	**828** ± 27	582 ± 146	297 ± 39	326 ± 27	311 ± 11	-	200 ± 72	835 ± 22
REACHER, EASY	**826** ± 219	538 ± 233	20 ± 50	314 ± 155	274 ± 14	-	136 ± 15	746 ± 25
CHEETAH, RUN	**447** ± 88	299 ± 48	138 ± 88	235 ± 137	267 ± 24	319 ± 56	130 ± 12	616 ± 18
WALKER, WALK	**504** ± 191	403 ± 24	224 ± 48	277 ± 12	394 ± 22	361 ± 73	127 ± 24	891 ± 82
CUP, CATCH	**840** ± 179	769 ± 43	0 ± 0	246 ± 174	391 ± 82	512 ± 110	97 ± 27	746 ± 91

and goal-reaching with both sparse and dense reward signals. For DMControl experiments, we evaluate the data-efficiency by measuring the performance of our method at 100k (i.e., low sample regime) and 500k (i.e., asymptotically optimal regime) *simulator or environment* steps[3] during training by following the setup in CURL [32]. These benchmarks are referred to as *DMControl100k* and *DMControl500k*. For comparison, we consider six powerful recent pixel-based methods: CURL [32] learns contrastive representations, SLAC [38] learns a forward model and uses it to shape encoder representations, while SAC+AE [37] minimizes a reconstruction loss as an auxiliary task. All three methods use SAC [34] as their base algorithm. Dreamer [36] and PlaNet [35] learn world models and use them to generate synthetic rollouts similar to Dyna [39]. Pixel SAC is a vanilla Soft Actor-Critic operating on pixel inputs, and state SAC is an *oracle* baseline that operates on the proprioceptive state of the simulated agent, which includes joint positions and velocities. We also provide learning curves for longer runs and examine how RAD compares to state SAC and CURL across a more diverse set of environments in Appendix D.

ProcGen: Although DMControl is suitable for benchmarking data-efficiency and performance, it evaluates the performance on the same environment in which the agent was trained and is thus not applicable for studying generalization. For this reason, we focus on the OpenAI ProcGen benchmarks [24] to investigate the generalization capabilities of RAD. ProcGen presents a suite of game-like environments where the train and test environments differ in visual appearance and structure. For this reason, it is a commonly used benchmark for studying the generalization abilities of RL agents [30]. Specifically, we evaluate the zero-shot performance of the trained agents on the full distribution of unseen levels. Following the setup in ProcGen [24], we use the CNN architecture found in IMPALA [40] as the policy network and train the agents using the Proximal Policy Optimization (PPO) [4] algorithm for 20M timesteps. For all experiments, we use the *easy environment difficulty* and the hyperparameters suggested in [24], which have been shown to be empirically effective.

[3] *environment steps* refers to the number of times the underlying simulator is stepped through. This measure is independent of policy heuristics such as action repeat. For example, if action repeat is set to 4, then 100k *environment* steps corresponds to 25k *policy* steps.



(a) Scores on DMControl500k for Walker, walk. (b) Spatial attention map of augmentations for Walker, walk.

Figure 2: (a) We ablate six common data augmentations on the walker, walk environment by measuring performance on DMControl500k of each permutation of any two data augmentations being performed in sequence. For example, the *crop* row and *grayscale* column correspond to the score achieved after applying random crop and then random grayscale to the input images (entries along the main axis use only one application of the augmentation). (b) Spatial attention map of an encoder that shows where the agent focuses on in order to make a decision in Walker Walk environment. Random crop enables the agent to focus on the robot body and ignore irrelevant scene details compared to other augmentations as well as the base agent that learns without any augmentation.

OpenAI Gym. For OpenAI Gym experiments with proprioceptive inputs (e.g., positions and velocities), we compare to PETS [41], a model-based RL algorithm that utilizes ensembles of dynamics models; POPLIN-P [42], a state-of-the-art model-based RL algorithm which uses a policy network to generate actions for planning; POPLIN-A [42], variant of POPLIN-P which adds noise in the action space; METRPO [43], model-based policy optimization based on TRPO [44]; and two state-of-the-art model-free algorithms, TD3 [45] and SAC [34]. In our experiments, we apply RAD to SAC. Following the experimental setups in POPLIN [42], we report the mean and standard deviation across four runs on Cheetah, Walker, Hopper, Ant, Pendulum and Cartpole environments.

5.2 Improving data-efficiency on DeepMind Control Suite

Data-efficiency: Mean scores shown in Table 1 and learning curves in Figure 7 show that data augmentation significantly improves the data-efficiency and performance across the six extensively benchmarked environments compared to existing methods. We summarize the main findings below:

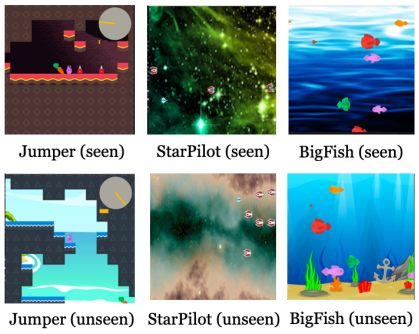
- RAD is the **state-of-the-art algorithm** on all (**6** out of **6**) environments on both DMControl100k and DMControl500k benchmarks.
- RAD **improves** the performance of **pixel SAC by 4x** on both DMControl100k and DMControl500k *solely through data augmentation* without learning a forward model or any other auxiliary task.
- RAD **matches** the performance of **state-based SAC** on the majority of (**11** out of **15**) DMControl environments tested as shown in Figure 7.
- **Random translation** or **random crop**, stand-alone, have the highest impact on final performance relative to all other augmentations as shown in in Figure 2(a).

Which data augmentations are the most effective? To answer this question for DMControl, we ran RAD with permutations of two data augmentations applied in sequence (e.g., crop followed by grayscale) on the *Walker Walk* environment and report the scores at 500k environment steps. We chose this environment because SAC without augmentation fails at this task, resulting in scores well below 200. Our results, shown in Figure 2(a), indicate that most data augmentations improve the performance of the base policy, and that random crop by itself was the most effective by a large margin.

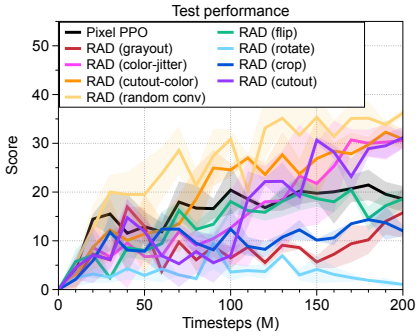
Why is random crop so effective? To analyze the effects of random crop, we decompose it into its two component augmentations: (a) *random window*, which masks a random boundary region of the image, exactly where crop would remove it, and (b) *random translate*, which places the full image entirely within a larger frame of zeros, but at a random location. In Appendix C, Figure 6 shows resulting learning curves from each augmentation. The benefit of translations is clearly demonstrated, whereas the random information hiding due to windowing produced little effect. Table 1 reports

Table 2: We present the generalization results of RAD with different data augmentation methods on the three OpenAI ProcGen environments: BigFish, StarPilot and Jumper. We report the test performances after 20M timesteps. The results show the mean and standard deviation averaged over three runs. We see that RAD is able to outperform the baseline PPO trained on two times the number of training levels benefitting from data augmentations such as random crop, cutout and color jitter.

	# of training levels	Pixel PPO	RAD (gray)	RAD (flip)	RAD (rotate)	RAD (random conv)	RAD (color-jitter)	RAD (cutout)	RAD (cutout-color)	RAD (crop)
BigFish	100	1.9 ± 0.1	1.5 ± 0.3	2.3 ± 0.4	1.9 ± 0.0	1.0 ± 0.1	1.0 ± 0.1	2.9 ± 0.2	2.0 ± 0.2	**5.4** ± 0.5
	200	4.3 ± 0.5	2.1 ± 0.3	3.5 ± 0.4	1.5 ± 0.6	1.2 ± 0.1	1.5 ± 0.2	3.3 ± 0.2	3.5 ± 0.3	**6.7** ± 0.8
StarPilot	100	18.0 ± 0.7	10.6 ± 1.4	13.1 ± 0.2	9.7 ± 1.6	7.4 ± 0.7	15.0 ± 1.1	17.2 ± 2.0	**22.4** ± 2.1	20.3 ± 0.7
	200	20.3 ± 0.7	20.6 ± 1.0	20.7 ± 3.9	15.7 ± 0.7	11.0 ± 1.5	20.6 ± 1.1	24.5 ± 0.1	**24.5** ± 1.6	24.3 ± 0.1
Jumper	100	5.2 ± 0.5	5.2 ± 0.1	5.2 ± 0.7	5.7 ± 0.6	5.5 ± 0.3	**6.1** ± 0.2	5.6 ± 0.1	5.8 ± 0.6	5.1 ± 0.2
	200	**6.0** ± 0.2	5.6 ± 0.1	5.4 ± 0.3	5.5 ± 0.1	5.2 ± 0.1	5.9 ± 0.1	5.4 ± 0.1	5.6 ± 0.4	5.2 ± 0.7



(a) ProcGen



(b) Test performance on modified CoinRun

Figure 3: (a) Examples of seen and unseen environments on ProcGen. (b) The test performance under the modified CoinRun. The solid/dashed lines and shaded regions represent the mean and standard deviation, respectively.

scores using *random translate*, a new SOTA method, for all environments except for *Walker Walk*, where random crop sometimes reduced variance. In Figure 5 of Appendix C, we ablate the size of the final translation image, finding in some cases that random placement within as little as two additional pixels in height and width is sufficient to reap the benefit.

What are the effects on the learned encoding? To understand how the augmentations affect learned representations encoder, we visualize a spatial attention map from the output of the last convolutional layer. Similar to [46], we compute the spatial attention map by mean-pooling the absolute values of the activations along the channel dimension and follow with a 2-dimensional spatial softmax. Figure 4 visualizes the spatial attention maps for the augmentations considered. Without augmentation, the activation is highly concentrated at the point of the forward knee, whereas with random crop/translate, entire edges of the body are prominent, providing a more complete and robust representation of the state.

5.3 Improving generalization on OpenAI ProcGen

Generalization: We evaluate the generalization ability on three environments from OpenAI Procgen: BigFish, StarPilot, and Jumper (see Figure 3(a)) by varying the number of training environments and ablating for different data augmentation methods. We summarize our findings below:

Table 3: Performance on OpenAI Gym. The training timestep varies from 50,000 to 200,000 depending on the difficulty of the tasks. The results show the mean and standard deviation averaged over four runs and the best results are indicated in bold. For baseline methods, we report the best number in POPLIN [42].

	Cheetah	Walker	Hopper	Ant	Pendulum	Cartpole
PETS	2288.4 ± 1019.0	282.5 ± 501.6	114.9 ± 621.0	1165.5 ± 226.9	155.7 ± 79.3	**199.6 ± 4.6**
POPLIN-A	1562.8 ± 1136.7	-105.0 ± 249.8	202.5 ± 962.5	1148.4 ± 438.3	**178.3 ± 19.3**	**200.6 ± 1.3**
POPLIN-P	4235.0 ± 1133.0	597.0 ± 478.8	2055.2 ± 613.8	**2330.1 ± 320.9**	167.9 ± 45.9	**200.8 ± 0.3**
METRPO	2283.7 ± 900.4	-1609.3 ± 657.5	1272.5 ± 500.9	282.2 ± 18.0	174.8 ± 6.2	138.5 ± 63.2
TD3	3015.7 ± 969.8	-516.4 ± 812.2	1816.6 ± 994.8	870.1 ± 283.8	168.6 ± 12.7	-409.2 ± 928.8
SAC	4035.7 ± 268.0	-382.5 ± 849.5	2020.6 ± 692.9	836.5 ± 68.4	162.1 ± 12.3	**199.8 ± 1.9**
RAD	**4554.3 ± 1209.0**	**806.4 ± 706.7**	**2149.1 ± 249.9**	1150.9 ± 494.6	167.4 ± 9.7	**199.9 ± 0.8**
Timesteps	200000	200000	200000	200000	50000	50000

- As shown in Table 2, various data augmentation methods such as random crop and cutout **significantly improve the generalization performance** on the BigFish and StarPilot environments (see Appendix F for learning curves).
- In particular, **RAD with random crop** achieves **55.8%** relative gain over pixel-based PPO on the BigFish environment.
- RAD trained with **100** training levels outperforms the pixel-based PPO trained with **200** training levels on both BigFish and StarPilot environments. *This shows that data augmentation can be more effective in learning generalizable representations compared to simply increasing the number of training environments.*
- In the case of Jumper (a navigation task), the gain from data augmentation is not as significant because the task involves structural generalization to different map layouts and is likely to require recurrent policies [24].
- To verify the effects of data augmentations on such environments, we consider a modified version of CoinRun [30] which corresponds to a simpler version of Jumper. By following the set up in [31], we train agents on a fixed set of 500 levels with half of the available themes (style of backgrounds, floors, agents, and moving obstacles) and then measure the test performance on 1000 different levels consisting of unseen themes to evaluate the generalization ability across the visual changes. As shown in Figure 3(b), data augmentation methods, such as random convolution, color-jitter, and cutout-color, improve the generalization ability of the agent to a greater extent than random crop suggesting the need to further study data augmentations in these environments.

5.4 Improving state-based RL on OpenAI Gym

Data-efficiency on state-based inputs. Table 3 shows the average returns of evaluation rollouts for all methods (see Figure 12 in Appendix J for learning curves). We report results for state-based RAD using the best performing augmentation - random amplitude scaling; for details regarding performance of other augmentations we refer the reader to Appendix J. Similar to data augmentation in the visual setting, RAD is the state-of-the-art algorithm on the majority (**4** out of **6**) of benchmarked environments. RAD consistently improves the performance of SAC across all environments, and outperforms a competing state-of-the-art method - POPLIN-P - on most of the environments. It is worth nothing that RAD improves the average return compared to POPLIN-P by **1.7x** in Walker, an environment where most prior RL methods fail, including both model-based and model-free ones. We hypothesize that random amplitude scaling is effective because it forces the agent to be robust to input noise while maintaining the intrinsic information of the state, such as sign of inputs and relative differences between them.

These results showcase the generality of incorporating inductive biases through augmentation (e.g., amplitude invariance) by showing that improving RL with data augmentation is not specific to pixel-based inputs but also applies RL from state. By achieving state-of-the-art performance across both visual and proprioceptive inputs, RAD sets a powerful new baseline for future algorithms.

6 Conclusion

In this work, we proposed RAD, a simple plug-and-play module to enhance any reinforcement learning (RL) method using data augmentations. For the first time, we show that data augmentations *alone* can significantly improve the data-efficiency and generalization of RL methods operating from pixels, *without any changes to the underlying RL algorithm*, on the DeepMind Control Suite and the OpenAI ProcGen benchmarks respectively. Our implementation is simple, efficient, and has been open-sourced. We hope that the performance gains, implementation ease, and wall clock efficiency of RAD make it a useful module for future research in data-efficient and generalizable RL methods; and a useful tool for facilitating real-world applications of RL.

7 Broader Impact

While there has been a trend in growing complexity and compute requirements to achieve state-of-the-art results in Computer Vision [20], NLP [47], and RL [48], there are two negative long-term consequences of these trends: (i) the energy demands of these large models are harmful to the environment due to increased carbon emissions if not powered by renewable energy sources (ii) they make AI research inaccessible to researchers without access to tremendous compute and engineering resources. RAD shows that, by incorporating powerful inductive biases, state-of-the-art results can be achieved with simpler methods that require less compute and model complexity than complex competing methods. RAD is therefore accessible to a broad range of researchers (even those without access to GPUs) and leaves a much smaller carbon footprint than competing methods.

While it's fair to say that even with the result from this paper, we are far removed from making Deep RL practical for solving real-world-complexity robotics problems, we believe this work provides progress towards that goal. Robots being able to learn through RL in the real world opens up opportunities for better elderly care, autonomous cleaning and disinfecting, more reliable / resilient supply chain and manufacturing operations (especially when humans might not be available due to a pandemic). On the flipside, an RL agent will optimize whatever reward one specifies. If the person in charge of the system specifies a reward that's bad for the world (or perhaps mistakenly even for themselves), the more powerful the RL, the worse the outcome. For this reason, in addition to developing better algorithms that achieve new state-of-the-art performance, it is also important to pursue complementary research on safety [49, 50].

References

[1] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *ICLR*, 2016.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[6] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[7] Yan Duan, John Schulman, Xi Chen, Peter L. Bartlett, Ilya Sutskever, and Pieter Abbeel. Rl^2: Fast reinforcement learning via slow reinforcement learning. *arXiv:1611.02779*, 2016.

[8] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI*, 2018.

[9] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[10] Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and brain sciences*, 40, 2017.

[11] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. In *ICLR*, 2020.

[12] Zoubin Ghahramani, Daniel M Wolpert, and Michael I Jordan. Generalization to local remappings of the visuomotor coordinate transformation. *Journal of Neuroscience*, 16(21):7085–7096, 1996.

[13] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *NeurIPS*, 2012.

[14] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *CVPR*, 2019.

[15] Ekin D. Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V. Le. Randaugment: Practical automated data augmentation with a reduced search space. *arXiv:1909.13719*, 2019.

[16] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V. Le. Unsupervised data augmentation for consistency training. *arXiv:1904.12848*, 2019.

[17] Qizhe Xie, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *CVPR*, 2020.

[18] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin Raffel. Mixmatch: A holistic approach to semi-supervised learning. In *NeurIPS*, 2019.

[19] Olivier J Hénaff, Aravind Srinivas, Jeffrey De Fauw, Ali Razavi, Carl Doersch, SM Eslami, and Aaron van den Oord. Data-efficient image recognition with contrastive predictive coding. *arXiv preprint arXiv:1905.09272*, 2019.

[20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv:2002.05709*, 2020.

[21] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *CVPR*, 2020.

[22] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[23] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[24] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *arXiv preprint arXiv:1912.01588*, 2019.

[25] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[26] Kihyuk Sohn, David Berthelot, Chun-Liang Li, Zizhao Zhang, Nicholas Carlini, Ekin D. Cubuk, Alex Kurakin, Han Zhang, and Colin Raffel. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *arXiv:2001.07685*, 2020.

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

[28] 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. In *IROS*, 2017.

[29] 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.

[30] Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *ICML*, 2019.

[31] Kimin Lee, Kibok Lee, , Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *ICLR*, 2020.

[32] Aravind Srinivas, Michael Laskin, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. *arXiv:2004.04136*, 2020.

[33] Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

[34] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *ICML*, 2018.

[35] Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

[36] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination. In *ICLR*, 2020.

[37] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images. *arXiv preprint arXiv:1910.01741*, 2019.

[38] Alex X Lee, Anusha Nagabandi, Pieter Abbeel, and Sergey Levine. Stochastic latent actor-critic: Deep reinforcement learning with a latent variable model. *arXiv preprint arXiv:1907.00953*, 2019.

[39] Richard Sutton. Dyna, an integrated architecture for learning, planning, and reacting. In *ACM SIGART Bulletin*. 1991.

[40] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *ICML*, 2018.

[41] Kurtland Chua, Roberto Calandra, Rowan McAllister, and Sergey Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In *NeurIPS*, 2018.

[42] Tingwu Wang and Jimmy Ba. Exploring model-based planning with policy networks. In *ICLR*, 2020.

[43] Thanard Kurutach, Ignasi Clavera, Yan Duan, Aviv Tamar, and Pieter Abbeel. Model-ensemble trust-region policy optimization. In *ICLR*, 2018.

[44] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, 2015.

[45] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *ICML*, 2018.

[46] Sergey Zagoruyko and Nikos Komodakis. Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer. In *ICLR*, 2017.

[47] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

[48] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov 2019.

[49] Dario Amodei, Chris Olah, Jacob Steinhardt, Paul F. Christiano, John Schulman, and Dan Mané. Concrete problems in AI safety. *CoRR*, abs/1606.06565, 2016.

[50] Geoffrey Irving and Amanda Askell. Ai safety needs social scientists. *Distill*, 2019.

[51] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NeurIPS*, 2017.

[52] David Ha and Jürgen Schmidhuber. World models. In *NeurIPS*, 2018.

A Extended related work

A.1 Data augmentation in supervised learning

Since our focus is on image-based observations, we cover the related work in computer vision. Data augmentation in deep learning systems for computer vision can be found as early as LeNet-5 [1], an early implementation of CNNs on MNIST digit classification. In AlexNet [13] wherein the authors applied CNNs to image classification on ImageNet, data augmentations were used to increase the size of the original dataset by a factor of 2048 by randomly flipping and cropping 224×224 patches from the original image. These data augmentations inject the priors of invariance to translation and reflection, playing a significant role in improving the performance of supervised computer vision systems. Recently, new augmentation techniques such as AutoAugment [14] and RandAugment [15] have been proposed to further improve the performance of these systems.

A.2 Data augmentation for data-efficiency in semi & self-supervised learning

Aside from improving supervised learning, data augmentation has also been widely utilized for unsupervised and semi-supervised learning. MixMatch [18], FixMatch [26], UDA [16] use unsupervised data augmentation in order to maximize label agreement without access to the actual labels. Several contrastive representation learning approaches [19, 21, 20] have recently dramatically improved the label-efficiency of downstream vision tasks like ImageNet classification. Contrastive approaches utilize data augmentations and perform patch-wise [19] or instance discrimination (MoCo, SimCLR) [21, 20]. In the instance discrimination setting, the contrastive objective aims to maximize agreement between augmentations of the same image and minimize it between all other images in the dataset [20, 21]. The choice of augmentations has a significant effect on the quality of the learned representations as demonstrated in SimCLR [20].

A.3 Prior work in reinforcement learning related to data augmentation

A.3.1 Data augmentation with domain knowledge

While not directly known for data augmentation in reinforcement learning, the following ideas can be viewed as techniques to diversify the data used to train an RL agent:

Domain randomization [27–29] is a simple data augmentation technique primarily used for transferring policies from simulation to the real world where one takes advantage of the simulator's access to information about the rendering and physics and thus can train transferable policies from diverse simulated experiences.

Hindsight experience replay [51] applies the idea of re-labeling trajectories with terminal states as fictitious goals, improving the ability of goal-conditioned RL to learn quickly with sparse rewards. This, however, makes assumptions about the goal space matching with the state space and has had limited success with pixel-based observations.

A.3.2 Synthetic rollouts using a learned world model

While usually not viewed as a data augmentation technique, the idea of generating fake or synthetic rollouts to improve the data-efficiency of RL agents has been proposed in the Dyna framework [39]. In recent times, these ideas have been used to improve the performance of systems that have explicitly learned world models of the environment and generated synthetic rollouts using them [52, 11, 36].

A.3.3 Data augmentation for data-efficient reinforcement learning

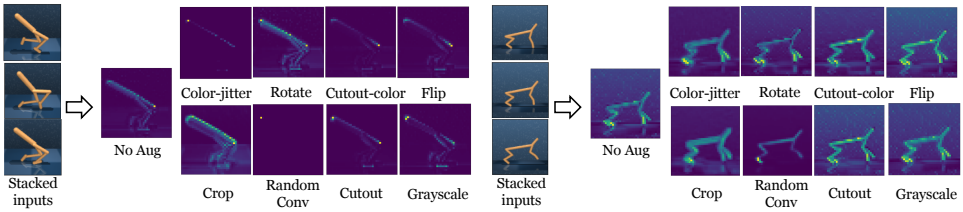
Data augmentation is a key component for learning contrastive representations in the RL setting as shown in the CURL framework [32], which learns representations that improve the data-efficiency of pixel-based RL by enforcing consistencies between an image and its augmented version through instance contrastive losses. Prior to our work, CURL was the state-of-the-art model for data-efficient RL from pixel inputs. While the focus in CURL was to make use of data augmentations jointly through contrastive and reinforcement learning losses, RAD attempts to directly use data augmentations for reinforcement learning without any auxiliary loss. We refer the reader to a discussion on tradeoffs between CURL and RAD in Section I. Concurrent and independent to our work, DrQ [33] uses data

augmentations and weighted Q-functions in conjunction with the off-policy RL algorithm SAC [34] to achieve state-of-the-art data-efficiency results on the DeepMind Control Suite. On the other hand, RAD can be plugged into any reinforcement learning method (on-policy methods like PPO [4] and off-policy methods like SAC [34]) *without making any changes to the underlying algorithm*. We further demonstrate the benefits of data augmentation to generalization on the OpenAI ProcGen benchmarks in addition to data-efficiency on the DeepMind Control Suite.

A.4 Data augmentation for generalization in reinforcement learning

Cobbe et al. [30] and Lee et al. [31] showed that simple data augmentation techniques such as cutout [30] and random convolution [31] can be useful to improve generalization of agents on the OpenAI CoinRun and ProcGen benchmarks. In this paper, we extensively investigate more data augmentation techniques such as random crop and color jitter on a more diverse array of tasks. With our efficient implementation of these augmentations, we demonstrate their utility with the on-policy RL algorithm PPO [4] for the first time.

B Attention maps for various data augmentations



(a) Walker (b) Cheetah

Figure 4: Spatial attention map of an encoder that shows where the agent focuses on in order to make a decision in (a) Walker Walk and (b) Cheetah Run environments. Random crop enables the agent to focus on the robot body and ignore irrelevant scene details compared to other augmentations as well as the base agent that learns without any augmentation. In addition to the agent, the base cheetah encoder focuses on the stars in the background, which are irrelevant to the task and likely harm the agent's performance. Random crop enables the encoder to capture the agent's state *much more clearly* compared to other augmentations. The quality of the attention map with random crop suggests that RAD improves the *contingency-awareness* of the agent (recognizing aspects of the environment that are under the agent's control) thereby improving its data-efficiency.

C Random translate ablations

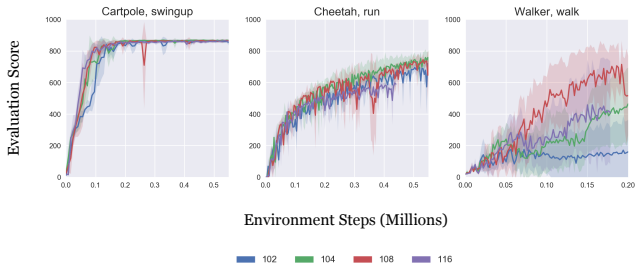


Figure 5: Ablations for different sizes of the larger frame in which the image is randomly translated. The original image is always rendered at a resolution of 100x100 pixels, and then translated within a larger frame of size 102,104,108, and 116. We note that augmentation significantly improves performance compared to SAC with no augmentation on Cartpole and Cheetah environments even for the smallest frame size of 102 where minimal augmentation is happening.

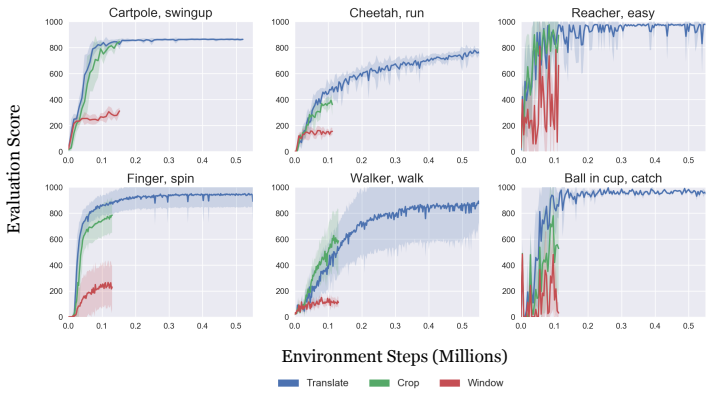


Figure 6: We ablate random translation, cropping, and windowing. Since cropping can be viewed as a simultaneous translation and window operation, we wish to understand which component is responsible for the most gains. We find that the gains come primarily from the translation operation and that, on most environments, RAD with translation results in more stable and efficient learning.

D Learning curves for RAD on DMControl

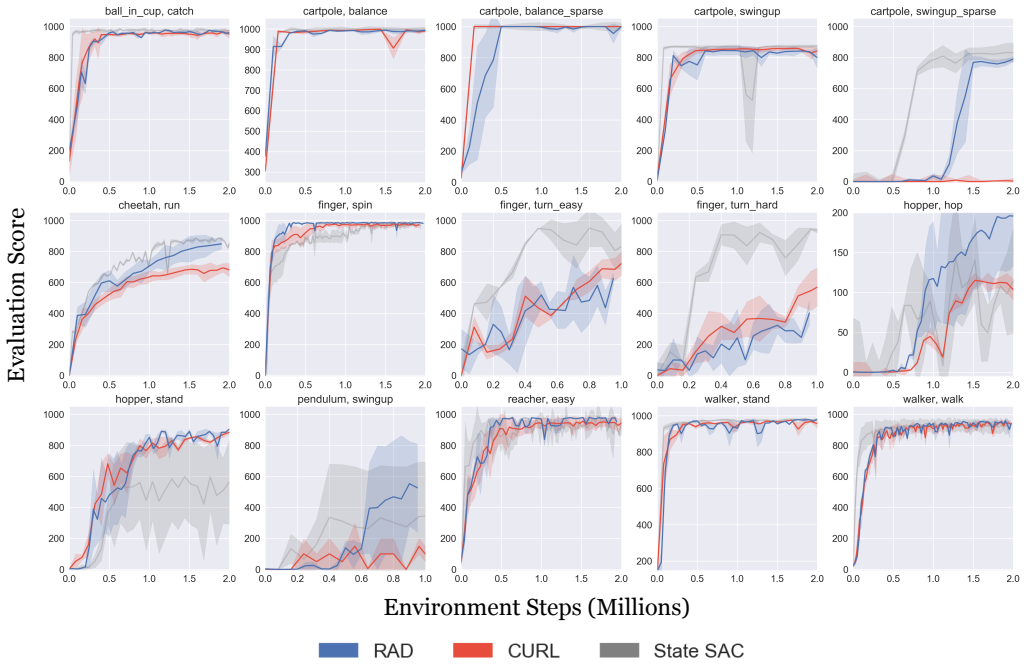


Figure 7: We benchmark the performance of RAD relative to the best performing pixel-based baseline (CURL) as well as SAC operating on state input on 15 environments in total. RAD matches state SAC performance on the majority (**11** out of **15** environments) and performs comparably or better than CURL on all of the environments tested. Results are average values across 3 seeds.

E Implementation details for DMControl

For DMControl experiments, we utilize the same encoder architecture as in [32] which is similar to the architecture in [37]. We show a full list of hyperparameters for DMControl experiments in Table 4.

Table 4: Hyperparameters used for DMControl experiments. Most hyperparameters values are unchanged across environments with the exception for action repeat, learning rate, and batch size.

Hyperparameter	**Value**
Augmentation	Crop - walker, walk; Translate - otherwise
Observation rendering	$(100, 100)$
Observation down/upsampling	$(84, 84)$ (crop); $(108, 108)$ (translate)
Replay buffer size	100000
Initial steps	1000
Stacked frames	3
Action repeat	2 finger, spin; walker, walk 8 cartpole, swingup 4 otherwise
Hidden units (MLP)	1024
Evaluation episodes	10
Optimizer	Adam
$(\beta_1, \beta_2) \rightarrow (f_\theta, \pi_\psi, Q_\phi)$	$(.9, .999)$
$(\beta_1, \beta_2) \rightarrow (\alpha)$	$(.5, .999)$
Learning rate $(f_\theta, \pi_\psi, Q_\phi)$	$2e-4$ cheetah, run $1e-3$ otherwise
Learning rate (α)	$1e-4$
Batch Size	512
Q function EMA τ	0.01
Critic target update freq	2
Convolutional layers	4
Number of filters	32
Non-linearity	ReLU
Encoder EMA τ	0.05
Latent dimension	50
Discount γ	.99
Initial temperature	0.1

F Implementation details and additional results for ProcGen

F.1 Environment descriptions

BigFish. In this environment, the agent starts as a small fish and the goal is to eat fish smaller than itself. The agent can receive a small reward for eating fish and a large reward is given when it becomes bigger that all other fish. The spawn timing, position of all fish, and style of background change throughout the level.

StarPilot. A simple side scrolling shooter game, where the agent receive the reward by avoiding enemy. The spawn timing of all enemies and obstacles, along with their corresponding types, are changing throughout the level.

Jumper. An open world environment, where the goal is to find the carrot which is randomly located in the map. Style of background, location of enemy and map structure are changing throughout the level.

Modified CoinRun. In this task, an agent is located at the leftmost side of the map and the goal is to collect the coin located at the rightmost side of the map within 1,000 timesteps. The agent observes its surrounding environment in the third-person point of view, where the agent is always located at the center of the observation. Similar to [31], half of the available themes are utilized (i.e., style of backgrounds, floors, agents, and moving obstacles) for training.

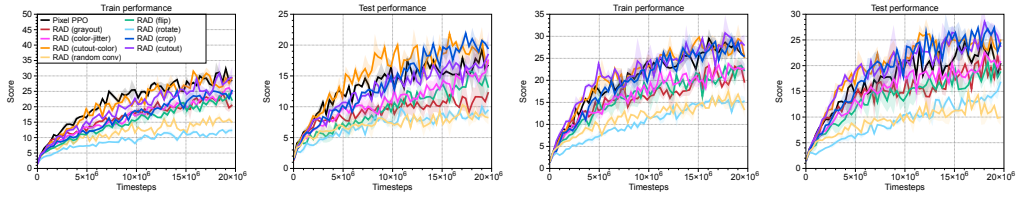
F.2 Implementation details

For ProcGen experiments, we follow the hyperparameters proposed in [24], which are empirically shown to be effective. Specifically, we use the CNN architecture found in IMPALA [40] as the policy network, and train the agents using the Proximal Policy Optimization (PPO) with following hyperparameters:

Table 5: Hyperparameters used for ProcGen experiments.

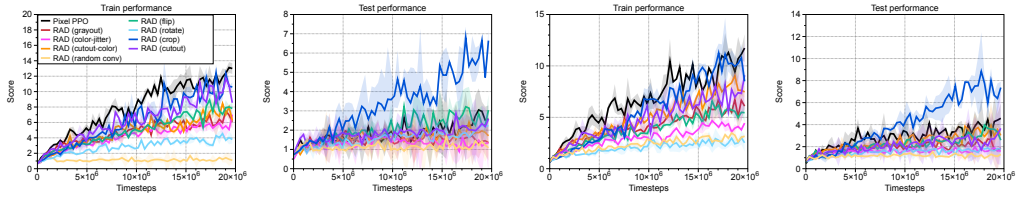
Hyperparameter	**Value**
Observation rendering	$(64, 64)$
Discount γ	.99
GAE parameter λ	0.95
# of timesteprs per rollout	256
# of minibatches per rollout	8
Entropy bonus	0.1
PPO clip range	0.2
Reward Normalization	Yes
# of Workers	1
# of environments per worker	64
Total timesteps	20M
LSTM	No
Frame Stack	No
Optimizer	Adam
Learning rate (α)	$5e-4$

F.3 Learning curves



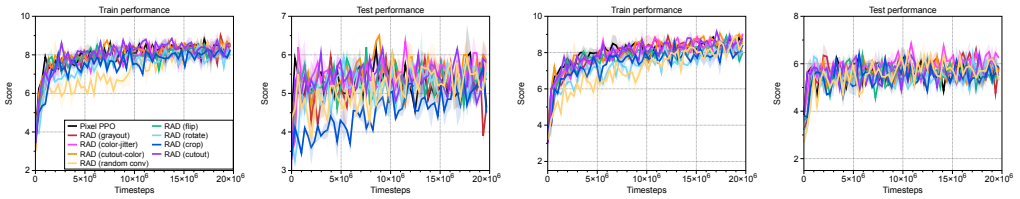
(a) Train (100) (b) Test (100) (c) Train (200) (d) Test (200)

Figure 8: Learning curves of PPO and RAD agents trained with (a/b) 100 and (c/d) 200 training levels on StarPilot. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.



(a) Train (100) (b) Test (100) (c) Train (200) (d) Test (200)

Figure 9: Learning curves of PPO and RAD agents trained with (a/b) 100 and (c/d) 200 training levels on Bigfish. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.



(a) Train (100) (b) Test (100) (c) Train (200) (d) Test (200)

Figure 10: Learning curves of PPO and RAD agents trained with (a/b) 100 and (c/d) 200 training levels on Jumper. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.

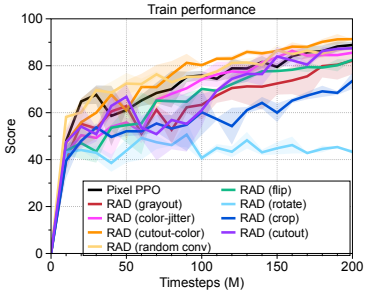


Figure 11: Learning curves of PPO and RAD in the modified Coinrun. The solid line and shaded regions represent the mean and standard deviation, respectively, across three runs.

G Code for select augmentations

```
def random_crop(imgs, size=84):
    n, c, h, w = imgs.shape
    w1 = torch.randint(0, w - size + 1, (n,))
    h1 = torch.randint(0, h - size + 1, (n,))
    cropped = torch.empty((n, c, size, size),
        dtype=imgs.dtype, device=imgs.device)
    for i, (img, w11, h11) in enumerate(zip(imgs, w1, h1)):
        cropped[i][:] = img[:, h11:h11 + size, w11:w11 + size]
    return cropped

def random_cutout(imgs, min_cut=4, max_cut=24):
    n, c, h, w = imgs.shape
    w_cut = torch.randint(min_cut, max_cut + 1, (n,)) # random size cut
    h_cut = torch.randint(min_cut, max_cut + 1, (n,)) # rectangular shape
    fills = torch.randint(0, 255, (n, c, 1, 1)) # assume uint8.
    for img, wc, hc, fill in zip(imgs, w_cut, h_cut, fills):
        w1 = torch.randint(w - wc + 1, ()) # uniform over interior
        h1 = torch.randint(h - hc + 1, ())
        img[:, h1:h1 + hc, w1:w1 + wc] = fill
    return imgs

def random_flip(imgs, p=0.5):
    n, _, _, _ = imgs.shape
    flip_mask = torch.rand(n, device=imgs.device) < p
    imgs[flip_mask] = imgs[flip_mask].flip([3]) # left-right
    return imgs
```

H Time-efficiency of data augmentation

The primary gain of our data augmentation modules is enabling efficient augmentation of stacked frame inputs in the minibatch setting. Since the augmentations must be applied randomly across the batch but consistently across the frame stack, traditional frameworks like Tensorflow and PyTorch that focus on augmenting single-frame static datasets, are unsuitable for this task. We further show wall-clock efficiency relative to the PyTorch API in Table 6.

Table 6: We compare the data augmentation speed between the RAD augmentation modules and performing the same augmentations in PyTorch. We calculate the number of additional minutes required to perform 100k training steps. On average, the RAD augmentations are nearly 2x faster than augmentations accessed through the native PyTorch API. Additionally, since the PyTorch API is meant for processing single-frame images, it is not designed to apply augmentations consistently across the frame stack but randomly across the batch. Cutout and random convolution augmentations are not present in the PyTorch API.

	OURS	PYTORCH
CROP	31.8	33.5
GRAYSCALE	15.6	51.2
CUTOUT	36.6	-
CUTOUT COLOR	45.2	-
FLIP	4.9	37.0
ROTATE	46.5	62.4
RANDOM CONV.	45.8	-

I Discussion

I.1 CURL vs RAD

Both CURL and RAD improve the data-efficiency of RL agents by enforcing consistencies in the input observations presented to the agent. CURL does this with an explicit instance contrastive loss between an image and its augmented version using the MoCo [21] mechanism. On the other hand, RAD does not employ any auxiliary loss and directly trains the RL objective on multiple augmented views of the observations, thereby ensuring consistencies on the augmented views implicitly. The performance of RAD matches that of CURL and surpasses CURL on some of the environments in the DeepMind Control Suite (refer to Figure 7). This suggests the potential conclusion that data augmentation is sufficient for data-efficient reinforcement learning from pixels. We argue that the conclusion requires a bit more nuance in the following subsection.

I.2 Is data augmentation sufficient for RL from pixels?

The improved performance of RAD over CURL can be attributed to the following line of thought: While both methods try to improve the data-efficiency through augmentation consistencies (CURL explicitly, RAD implicitly); RAD outperforms CURL because *it only optimizes for what we care about, which is the task reward*. CURL, on the other hand, jointly optimizes the reinforcement and contrastive learning objectives. If the metric used to evaluate and compare these methods is the score attained on the task at hand, a method that purely focuses on reward optimization is expected to be better as long as it implicitly ensures similarity consistencies on the augmented views (in this case, just by training the RL objective on different augmentations directly).

However, we believe that a representation learning method like CURL is *arguably* a more general framework for the usage of data augmentations in reinforcement learning. CURL can be applied *even without any task (or environment) reward* available. The contrastive learning objective in CURL that ensures consistencies between augmented views is disentangled from the reward optimization (RL) objective and is therefore capable of learning-rich semantic representations from high dimensional observations gathered from random rollouts. Real-world applications of RL might involve performing plenty of interactions (or rollouts) with sparse reward signals, and tasks presented to the agent as image-based goals. In such scenarios, CURL and other representation learning methods are *likely*

to be more important even though current RL benchmarks are primarily about single or multi-task reward optimization.

Given these subtle considerations, we believe that both RAD and representation learning methods like CURL will be useful tools for an RL practitioner in future research encompassing data-efficient and generalizable RL.

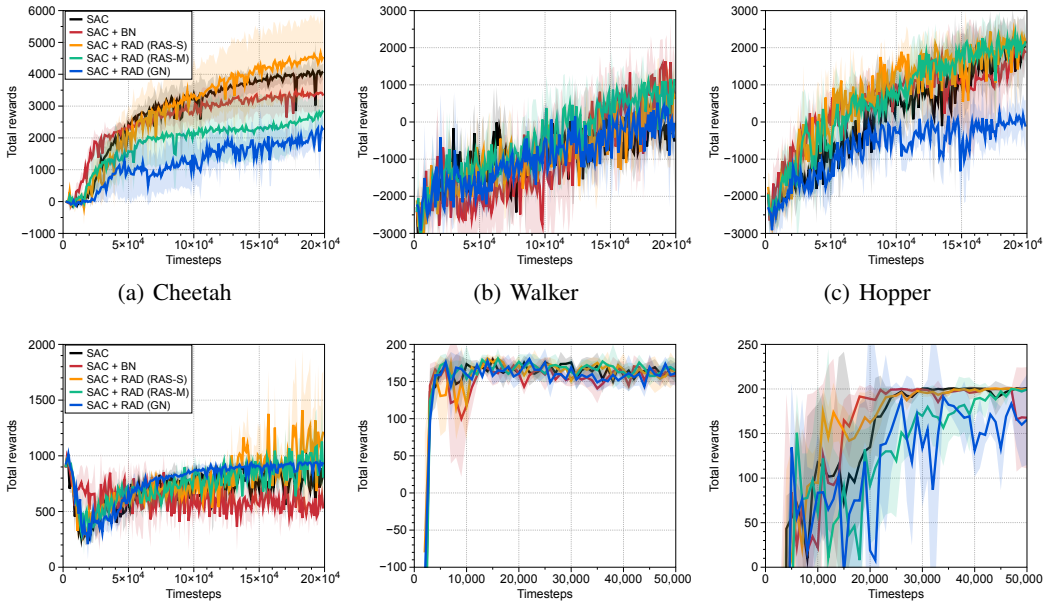
J Implementation details and additional results for OpenAI Gym

J.1 Implementation details

We consider a combination of SAC and RAD using the publicly released implementation repository (https://github.com/vitchyr/rlkit) without any modifications on hyperparameters and architectures. For random amplitude scaling, we consider two variants: (a) random amplitude scaling with a single variable (RAS-S) that multiplies the one-dimensional uniform random variable, i.e., $\mathbf{s}' = \mathbf{s} * z$, where $z \sim \text{Uni}[\alpha, \beta]$, and (b) random amplitude scaling with a multivariate variable (RAS-M) that multiplies the multivariate uniform random variable, i.e., $\mathbf{s}' = \mathbf{s} * \mathbf{z}$, where $\mathbf{z} \sim \text{Uni}[\alpha, \beta]$. The minimum value of uniform distribution is chosen from $\alpha \in \{0.6, 0.8\}$ and the maximum value of uniform distribution is chosen from $\beta \in \{1.2, 1.4\}$. For Gaussian noise, we add Gaussian random variable, i.e., $\mathbf{s}' = \mathbf{s} + \mathbf{z}$, where $\mathbf{z} \sim \mathcal{N}(0, I)$. The optimal parameters are chosen to achieve the best performance on training environments.

J.2 Experimental results on OpenAI Gym

As shown in Table 7 and Figure 12, random amplitude scaling is effective in almost all environments. We expect that this is because random amplitude scaling forces the agent to be robust to input noise while maintaining the intrinsic information of the state. We also remark that a simple normalization technique such as batch normalization is not very effective compared to RAD, which implies that the gains from RAD can not be achieved by normalization.



(a) Cheetah (b) Walker (c) Hopper

(d) Ant (e) Pendulum (f) Cartpole

Figure 12: Learning curves of variants of RAD, i.e., random amplitude scaling with a single variable (RAS-S), random amplitude scaling with multivariate variables (RAS-M), and Gaussian noise (GN), on OpenAI Gym. The solid line and shaded regions represent the mean and standard deviation, respectively, across four runs.

Table 7: Performance of variants of RAD, i.e., random amplitude scaling with a single variable (RAS-S), random amplitude scaling with a multivariate variable (RAS-M), and Gaussian noise (GN), on OpenAI Gym. The training timestep varies from 50,000 to 200,000 depending on the difficulty of the tasks. The results show the mean and standard deviation averaged over four runs and the best results are indicated in bold. For baseline methods, we report the best number in POPLIN [42].

	Cheetah	Walker	Hopper
PETS	2288.4 ± 1019.0	282.5 ± 501.6	114.9 ± 621.0
POPLIN-A	1562.8 ± 1136.7	-105.0 ± 249.8	202.5 ± 962.5
POPLIN-P	4235.0 ± 1133.0	597.0 ± 478.8	2055.2 ± 613.8
METRPO	2283.7 ± 900.4	-1609.3 ± 657.5	1272.5 ± 500.9
TD3	3015.7 ± 969.8	-516.4 ± 812.2	1816.6 ± 994.8
SAC	4035.7 ± 268.0	-382.5 ± 849.5	2020.6 ± 692.9
SAC + BN	3386.9 ± 549.3	751.2 ± 1017.9	1854.1 ± 329.3
SAC + RAD (RAS-S)	**4554.3 ± 1209.0**	370.4 ± 579.3	**2149.1 ± 249.9**
SAC + RAD (RAS-M)	2787.4 ± 466.8	**806.4 ± 706.7**	2096.1 ± 442.7
SAC + RAD (GN)	2222.3 ± 418.3	-121.6 ± 664.6	19.0 ± 619.6
Timesteps	200000	200000	200000
	Ant	Pendulum	Cartpole
PETS	1165.5 ± 226.9	155.7 ± 79.3	**199.6 ± 4.6**
POPLIN-A	1148.4 ± 438.3	**178.3 ± 19.3**	**200.6 ± 1.3**
POPLIN-P	**2330.1 ± 320.9**	167.9 ± 45.9	**200.8 ± 0.3**
METRPO	282.2 ± 18.0	174.8 ± 6.2	138.5 ± 63.2
TD3	870.1 ± 283.8	168.6 ± 12.7	-409.2 ± 928.8
SAC	836.5 ± 68.4	162.1 ± 12.3	**199.8 ± 1.9**
SAC + BN	580.8 ± 70.2	158.4 ± 10.2	178.1 ± 38.2
SAC + RAD (RAS-S)	1150.9 ± 494.6	158.9 ± 16.8	**199.9 ± 0.8**
SAC + RAD (RAS-M)	966.6 ± 321.0	167.4 ± 9.7	198.8 ± 1.3
SAC + RAD (GN)	932.9 ± 12.9	163.4 ± 12.6	169.9 ± 31.0
Timesteps	200000	50000	50000