Efficient Scheduling of Data Augmentation for Deep Reinforcement Learning

Byungchan Ko*
NALBI
kbc@nalbi.ai

Jungseul Ok
GSAI, POSTECH
jungseul@postech.ac.kr

Abstract

In deep reinforcement learning (RL), data augmentation is widely considered as a tool to induce a set of useful priors about semantic consistency and to improve sample efficiency and generalization performance. However, even when the prior is useful for generalization, distilling it to RL agent often interferes with RL training and degenerates sample efficiency. Meanwhile, the agent is forgetful of the prior due to the non-stationary nature of RL. These observations suggest two extreme schedules of distillation: (i) over the entire training; or (ii) only at the end. Hence, we devise a stand-alone network distillation method to inject the consistency prior at any time (even after RL), and a simple yet efficient framework to automatically schedule the distillation. Specifically, the proposed framework first focuses on mastering train environments regardless of generalization by adaptively deciding which *or no* augmentation to be used for the training. After this, we add the distillation to extract the remaining benefits for generalization from all the augmentations, which requires no additional new samples. In our experiments, we demonstrate the utility of the proposed framework, in particular, that considers postponing the augmentation to the end of RL training. https://github.com/kbc-6723/es-da

1 Introduction

Deep reinforcement learning (RL) aims at finding a neural network to represent policy or value functions taking raw observation as input, of which the most common form in practice is visual data or images of high-dimensionality, e.g., video games [24], board games [31, 30], and robot controls [34, 19]. RL handling high-dimensional input often suffers from poor sample efficiency and generalization capability, mainly due to the curse of dimensionality [4, 16]. To overcome these issues, it has been widely considered to augment data based on prior knowledge that a set of transformations preserve the meaning or context of input observations, e.g., cropping out unimportant parts of images, and changing colors [21, 20, 22, 15]. On one hand, RL agent can be directly fed with the original and augmented data so that it implicitly learns a representation with the prior and improves the sample efficiency and generalization [21]. On the other hand, the prior knowledge in data augmentation can be explicitly distilled via a self-supervised learning, which introduces additional regularization to ensure consistency between responses to original and augmented inputs [26, 15].

However, data augmentation shows highly task-dependent effect in RL, and it is prone to generate severe interference with the training even when it truly conveys a useful prior to train and test environments [21, 26, 15]. We address the problem of alleviating the interference between data augmentation and RL training to improve *sample efficiency* for acquainting train tasks, and *generalization capability* for unseen test environments. This problem in (online) RL is more critical and challenging than that in supervised learning (SL) since the objective function and data distribution are time-varying in RL,

*This work was done while Byungchan Ko studied in GSAI, POSTECH.

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

while they are not in SL. Indeed, according to [1, 11], it is sufficient to partly apply data augmentation just for a short period of SL at any time. Meanwhile, we empirically observe that the prior from data augmentation can be easily forgotten in RL of the non-stationary nature (see Section 5.3), i.e., the effect of augmentation is *time-sensitive*.

Based on our observation about the interference and the time-sensitivity, we propose two simple yet effective methods according to timings of data augmentation : **In**tra **D**istillation with **A**ugmented observations (InDA) and **Ex**tra **D**istillation with **A**ugmented observations (ExDA). Data augmentation beneficial for the sample efficiency needs to be applied over the entire RL training, i.e., InDA. Conversely, data augmentation useful only for the generalization should be postponed to the end of RL training, i.e., ExDA, so that we can minimize the interference in the training, while enjoying the benefit in the testing. InDA and ExDA are equipped with **D**istillation with **A**ugmented observation (DA). DA is a stand-alone self-supervised learning which enables us to induce the prior after RL training, and shows a relatively small interference with RL training by explicitly preserving the response of RL agent to the original input.

The best timing (InDA or ExDA) depends on traits of train task and augmentation. We hence suggest a framework of adaptive scheduling, named UCB-ExDA, that (i) first aims at maximizing the sample efficiency by adaptively selecting which or no augmentation to be used *during RL training*; and then, (ii) distill the priors from all the augmentations *after RL training*. Specifically, inspired by [26], we devise UCB-InDA for the adaptive selection in the first part by modifying the upper confidence bound (UCB) algorithm [3] for multi-armed bandit, where differently from [26], the set of arms includes all the augmentations and *the option to not augment*. In summary, UCB-ExDA is nothing but executing UCB-InDA followed by ExDA. Our experiment demonstrates the utility of the proposed framework.

Our contributions are summarized as follows:

(i) We devise DA (Section 4.1) which explicitly preserves the knowledge of RL agent so that enables distilling the consistency prior of augmentation into RL agent not only during but also after RL training, while other methods [21, 26, 15] need to be applied concurrently with RL training and thus show relatively strong interference in our experiments (Section 5.2).

(ii) We identify the simple yet effective timings of data augmentation: either InDA or ExDA (Section 4.2, Section 4.3), based upon the discovery of the time-sensitivity (Section 5.3) that has not been observed in SL [11], i.e., the proposed timings are effective particularly for RL.

(iii) We finally establish UCB-ExDA which automatically decides the best timing of augmentation, and demonstrate its superiority compared to others (Section 5.4). The advantage of UCB-ExDA is particularly substantial when the best strategy is ExDA postponing augmentation to the end of RL training.

2 Related Works

Augmented experience in RL. To solve the problem of poor generalization and sparse data, a popular approach is to fabricate virtual experiences and train the RL agent to learn with them. Domain randomization is a technique to produce such experiences from a simulator of a targeted system, [34, 25, 27]. Accurate simulators of practical systems are difficult to obtain, and it has limited applicability. However, visual augmentation has no such limit because the method uses simple image transformations such as cropping, tilting, and color jitter, although applications require a careful understanding of the targeted system to guide the design of an appropriate image transformer. A method of curriculum learning for domain randomization, in which the difficulty is gradually increasing [27] provided insights that coincide with some of our findings. However, we provide a further understanding of the types of visual augmentation that should be early or late during training.

Regularization from augmented data in vision-based RL has been implemented in various learning frameworks, including but not limited to representation [15, 33], self-supervised [26], and contrast [32]. One proposed algorithm [26] applies the UCB algorithm [3] to automatically select the most effective augmentations over RL training, where each augmentation is considered as an arm and then evaluate the effectiveness of augmentation by using a sliding window average. The idea of adapting augmentation concurs with our main message regarding the timing of augmentation. In [26], 'not augmenting' is not an option, whereas our findings indicate that it should be. In addition, [26] does not consider post augmentation followed by RL training, as in ExDA.

Other time-sensitivity in deep learning. During deep learning, the early stage of training often has a significant effect [7, 1]. Therefore, we devised time-sensitive methods that adapt to the progress of training, such as learning rate decay [37] and curriculum learning [35]. Golatkar *et al*. [11] studied such a time-sensitivity of regularization techniques for SL, where the effect of data augmentation in different time does not change much. We find that the time-sensitivity of augmentation can be significant in RL. This contrast may occur because of the non-stationary nature of RL, which SL does not have. Although a set of techniques originally developed for SL such as convolutional neural network, weight decay, batch normalization, dropout and self-supervised learning improve deep RL [17, 6, 23, 9, 32, 36, 14], a thorough study should be conducted before introducing a method from different learning frameworks, because we find the contrasting time-sensitivities of data augmentation. This spirit is also shared with an application [18] of implicit bias in SL [13, 2, 10] to RL.

3 Background

Notation. We consider a standard agent-environment interface of vision-based reinforcement learning in a discrete Markov decision process of state space $\mathcal{S}$, action space $\mathcal{A}$, and kernel $P = P(s_{t+1}, r_t | s_t, a_t)$ which determines the state transition and reward distribution. The goal of the RL agent is to find a policy that maximizes the expectation of cumulative reward $\sum_{t=0}^{t'-1} \gamma^t r_t$, where t' is terminating time and $\gamma \in [0, 1]$ is discount factor. At each timestep t, the agent selects an action $a_t \in \mathcal{A}$ and receives a reward r_t and an image $o_{t+1} = O(s_{t+1}) \in \mathbb{R}^{k \times k'}$ as an observation of the next state s_{t+1}. Data augmentation can be described by an image transformation function $\phi : \mathbb{R}^{k \times k'} \mapsto \mathbb{R}^{k \times k'}$ of which output is presumed to have the same or similar semantics of the input.

Baseline RL algorithm. Throughout this paper, we use Proximal Policy Optimization (PPO) [28] as a baseline, although we believe our findings and methods can be easily adjusted to others. PPO is a representative on-policy RL algorithm to learn policy $\pi_\theta(a \mid o)$ and value function V_θ of neural agent parameterized by θ. Storing a set of recent transitions $\tau_t := (o_t, a_t, r_t, o_{t+1})$ in experience buffer $\mathcal{D}$, the agent is updated to minimize the following loss function:

$$L_{\text{PPO}}(\theta) = -L_\pi(\theta) + \alpha L_V(\theta) \ , \tag{1}$$

where α is a hyperparameter and canonical regularization terms are omitted. The clipped policy objective function L_π and value loss function L_V are defined as:

$$L_\pi(\theta) = \hat{\mathbb{E}}\Big[\min(\rho_t(\theta)\hat{A}_t, \text{clip}(\rho_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{2}$$

$$L_V(\theta) = \hat{\mathbb{E}}\Big[\big(V_\theta(o_t) - V_t^{\text{targ}}\big)^2\Big] \ , \tag{3}$$

where the expectation $\hat{\mathbb{E}}$ is taken with respect to $\tau_t \sim \mathcal{D}$, we denote by θ_{old} the parameter before the update, $\rho_t(\theta)$ is the importance ratio $\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$, and $\hat{A}_t$ is the generalized advantage estimation [28].

4 Method

In what follows, we present our methods: **D**istillation with **A**ugmented observations (DA; Section 4.1), **In**tra **DA** (InDA; Section 4.2), **Ex**tra **DA** (ExDA; Section 4.3), and then the adaptive scheduling frameworks based on UCB (UCB-InDA and UCB-ExDA; Section 4.4). DA is a stand-alone knowledge distillation method, which can be used at any time to instill the underlying prior of augmentation into a given RL agent. InDA and ExDA conduct either DA or PPO in each epoch but have different schedules (Figure 1), where InDA interleaves PPO and DA, whereas ExDA performs PPO first then DA. UCB-InDA adaptively decides which or no augmentation to be used in each DA epoch of InDA based on the UCB of estimated gain from each option. UCB-ExDA performs ExDA preceded by UCB-InDA.

4.1 Distillation with augmented observations (DA)

The underlying prior of augmentation can be infused by minimizing a measure of inconsistency between the agent's responses to original and augmented inputs (resp. o_t and $\phi(o_t)$). For instance,

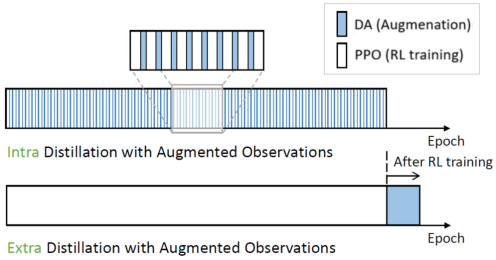


Figure 1: An illustration of InDA and ExDA

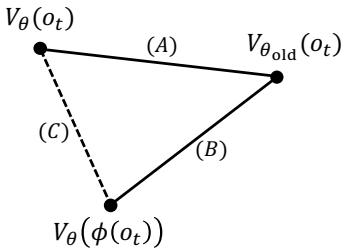


Figure 2: An illustration of distillation losses.

with PPO agent learning policy π_θ and value V_θ, Raileanu *et al*. [26] proposes the following measure:

$$L_{\text{dis}}(\theta, \phi) := \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[\text{KL}[\pi_\theta(\cdot|o_t), \pi_\theta(\cdot|\phi(o_t))] \right] + \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[(V_\theta(o_t) - V_\theta(\phi(o_t)))^2 \right] , \quad (4)$$

which uses Kullback-Leibler divergence (first term) and mean squared deviation (second term) for policy and value inconsistencies, respectively. Noting that $L_{\text{dis}}(\theta; \phi)$ can be minimized to be zero by a constant response to all inputs, the distillation with Eq (4) can distort the RL agent, in particular, when applying it outside of RL training.

We hence devise a network distillation technique (DA) which *explicitly preserves* the RL agent's response to original input and thus is applicable even after RL training. DA distills the knowledge of RL agent θ_{old} into θ using not only original but also augmented observations. More formally, the loss of DA is written as:

$$L_{\text{DA}}(\theta, \phi; \theta_{\text{old}}) := L_{\text{dis}}(\theta, \mathbb{I}; \theta_{\text{old}}) + L_{\text{dis}}(\theta, \phi; \theta_{\text{old}}) \ . \quad (5)$$

We here denote the identity transformation by $\mathbb{I}$ such that $\mathbb{I}(o) = o$ for all o, and extend the definition of L_{dis} in Eq (4) as follows:

$$L_{\text{dis}}(\theta, \phi; \theta_{\text{old}}) = \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|o_t), \pi_\theta(\cdot|\phi(o_t))] \right] + \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[(V_{\theta_{\text{old}}}(o_t) - V_\theta(\phi(o_t)))^2 \right] \ . \quad (6)$$

In Eq (5), the first term ensures that θ and θ_{old} behave identically for the original inputs, and the second one infuses the consistency prior. In Figure 2, $L_{\text{dis}}(\theta, \mathbb{I}; \theta_{\text{old}})$, $L_{\text{dis}}(\theta, \phi; \theta_{\text{old}})$, and $L_{\text{dis}}(\theta, \phi; \theta)$ graphically correspond to (A), (B), and (C), respectively. From this, it follows that minimizing L_{DA} in Eq (5) eventually reduces $L_{\text{dis}}(\theta, \phi; \theta)$ in Eq (4). In addition, the minimization of L_{DA} secures the responses of θ to the original inputs (which can be pre-computed to reduce computation cost) to the those of θ_{old}, while the alternatives (e.g., (A)+(C): $L_{\text{dis}}(\theta, \mathbb{I}; \theta_{\text{old}}) + L_{\text{dis}}(\theta, \phi; \theta)$) does not and thus may generate interference with RL training. Our experiments (Section 5.2; Figure 4) justifies the design of DA by showing substantial advantage compared to the other alternatives such as DrAC [26] using $L_{\text{dis}}(\theta, \phi; \theta)$ in Eq (4). We note that this advantage becomes much clearer when a wrong augmentation is given, c.f., the supplementary material.

4.2 Intra distillation with augmented observations

InDA (Algorithm 1) alternates between minimizing L_{PPO} and L_{DA}, i.e., PPO and DA are explicitly separated, whereas they are often executed simultaneously in other methods [26]. Such a clear separation reduces the interference [15]. We can control the frequency and timing of applying distillation with hyperparameters I, S and T, where we perform DA after each I rounds of RL training only if the current epoch n is in the interval of $[S, T]$, i.e., S and T are the epochs to begin and terminate DA, respectively. We vary S and T to study the time-dependency of data augmentation. We denote InDA$[S, T]$ to indicate the period to apply DA, while we omit the indication when DA is applied over the entire period. We provide further details on InDA in the supplementary material.

Algorithm 1 InDA

Require: N, I, ϕ, S, T

Initialize θ close to origin.
for $n = 1, 2, \ldots, N$ **do**
  // RL training
  Store sampled transitions to $\mathcal{D}$;
  Optimize RL objective $L_{\text{PPO}}(\theta)$ with $\mathcal{D}$;
  // Distillation
  if $n \in [S, T]$ and $\text{mod}(n-1, I) = 0$ **then**
    Store $\theta_{\text{old}} \leftarrow \theta$;
    Minimize $L_{\text{DA}}(\theta)$ for $\mathcal{D}$, θ_{old} and ϕ;
  end if
end for

Algorithm 2 ExDA

Require: N, M, ϕ

Initialize θ close to origin.
//Pre-training phase with RL algorithm
for $n = 1, 2, \ldots, N$ **do**
  Store sampled transitions to $\mathcal{D}$;
  Optimize RL objective $L_{\text{PPO}}(\theta)$ with $\mathcal{D}$;
end for

Store $\theta_{\text{old}} \leftarrow \theta$;
// Distillation at the end of RL training
for $m = 1, 2, \ldots, M$ **do**
  Minimize $L_{\text{DA}}(\theta)$ for $\mathcal{D}$, θ_{old} and ϕ;
end for

4.3 Extra distillation with augmented observations

ExDA (Algorithm 2) performs the distillation after the end of RL training, where the lengths of DA and RL training are parameterized by M and N, respectively. We note that computational cost can be reduced by removing the value inconsistency measure $\hat{\mathbb{E}}_{o_t \sim \mathcal{D}}\left[(V_{\theta_{\text{old}}}(o_t) - V_\theta(\phi(o_t)))^2\right]$ from L_{dis} in Eq (6) because the value consistency is not necessary for RL training nor distillation in the actor-critic framework and including it has a potential risk of generating additional interference. In the supplementary material, it is empirically verified that this reduction does not degrade RL performance. We leave more interesting details in the supplementary. For instance, one can consider re-initializing θ before starting DA as a part of exploiting the implicit bias [13, 18] to improve generalization. However, test performance is often dropped. This is mainly because the dataset to distill $\pi_{\theta_{old}}$ has much less diversity than that observed along the trajectory. Thus, we use no re-initialization for the experiments in the main paper.

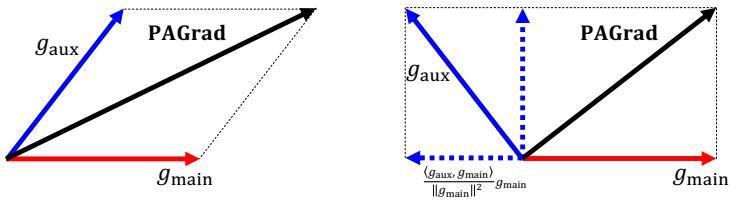
4.4 UCB-based adaptive scheduling frameworks

The training benefit by augmentation differs depending on the task. This dependency complicates the choice of whether to use InDA or ExDA for augmentation. Hence, we devise an auto-augmentation method, called UCB-InDA, inspired by UCB-DrAC [26], where each augmentation is corresponded to an arm in multi-armed bandit (MAB) problem and assessed its potential gain in training with upper confidence bound (UCB) [3]. More formally, in UCB-InDA, the set of arms is the set of augmentations, $\Phi = \{\phi_1, \ldots, \phi_K\}$, which must include the identity function $\mathbb{I}$, i.e., the option not to augment. The inclusion of identity function is small but makes crucial difference than UCB-DrAC [26] since we observe that using augmentation sometimes needs to be postponed after RL training for the sake of better sampling complexity and test performance.

A round of MAB corresponds to every I epoch of InDA, where we let $\phi_{k(s)} \in \Phi$ be the augmentation selected at the s-th round of DA. Let $G(s)$ be the average return, the sum of estimated advantage $\hat{A}$ and predicted value V_θ, over $(I-1)$ epochs of PPO followed by the s-th DA. UCB algorithm assumes that each arm generates reward independently drawn from a fixed distribution, and estimates the empirical mean over the entire sampling process. However, in RL, the return $G(s)$ is non-stationary, so UCB-InDA computes moving average gain $\bar{G}_k(s)$, instead, taken over a certain number (chosen to be 3 in our experiment) of most recent rounds selecting ϕ_k as Raileanu *et al*. [26] proposed. Then, inspired by UCB1 algorithm [3], UCB-InDA selects action $k(s)$ at round s as follows:

$$k(s) = \underset{k \in \{1, \ldots, K\}}{\arg\max}\ \bar{G}_k(s) + c\sqrt{\frac{\log(s)}{N_k(s)}} \tag{7}$$

where c is the UCB exploration coefficient, and $N_k(s)$ is the number of times selecting ϕ_k up to round s. We refer to the supplementary material for the hyperparameter choice. We remark that compared to UCB-DrAC [26], the proposed UCB-InDA has subtle but important differences, summarized in two folds: (i) the inclusion of identity transformation (i.e., no augmentation) and (ii) the distillation with



(a) non-conflicting: $\langle g_{\text{aux}}, g_{\text{main}} \rangle \geq 0$ (b) conflicting: $\langle g_{\text{aux}}, g_{\text{main}} \rangle < 0$

Figure 3: *An illustration of gradient conflicting and PAGrad.* We here let g_{main} and g_{aux} denote the gradients of main (shown in red; e.g., ∇L_{PPO}) and auxiliary (shown in blue; e.g., ∇L_{dis}) losses, respectively. In the computation of PAGrad (shown in black), the component of g_{aux} conflicting to g_{main} only when it exists (i.e., $\langle g_{\text{aux}}, g_{\text{main}} \rangle < 0$).

augmentation via InDA. The gain of each component is numerically studied in Section 5. Finally, we note that UCB-ExDA is nothing but UCB-InDA followed by ExDA.

4.5 PAGrad

Inspired by [38], we devise an alternative approach to reduce the interference by interpreting RL training with data augmentation as a multi-task learning, where L_{PPO} and L_{dis} correspond to the main and auxiliary task losses, respectively. In [38], the degree of conflict from task A to task B is estimated by the inner product of the gradient of task A and the negative gradient of task B, c.f., Figure 3. From this, we propose PAGrad (Projecting Auxiliary Gradient) to compute a modified gradient of L_{PPO} excluding the conflict from the auxiliary gradient ∇L_{dis} to the main one ∇L_{PPO}. Formally, PAGrad computes the gradient given as follows:

$$\nabla L_{\text{PPO}} + \left(\nabla L_{\text{dis}} - \frac{\min\{0, \langle \nabla L_{\text{dis}}, \nabla L_{\text{PPO}} \rangle\}}{\|\nabla L_{\text{PPO}}\|^2} \nabla L_{\text{PPO}} \right) , \tag{8}$$

where $\frac{\min\{0, \langle \nabla L_{\text{dis}}, \nabla L_{\text{PPO}} \rangle\}}{\|\nabla L_{\text{PPO}}\|^2} \nabla L_{\text{PPO}}$ is the components of ∇L_{dis} opposite to ∇L_{PPO} which may disturb optimizing the main objective L_{PPO}. Based on this, we devise DrAC+PAGrad that updates the model parameter toward the negative direction of (8). This is an alternative of InDA, while it concurrently optimizes L_{PPO} and L_{dis} differently from InDA adopting the time-separation of optimizing L_{PPO} and L_{dis}. We also note that it differs from the original method proposed in [38] alternating the main and auxiliary tasks to accomplish every task at equal priority, while we have a clear priority on RL task.

5 Experiment

5.1 Setups

Train and test tasks. We use the OpenAI Procgen benchmark of 16 video games [5], where a main character tries to achieve a specific goal, e.g., finding exit (Maze) or collecting coins (Coinrun), while avoiding enemies given a 2D map. At each time t, visual observation o_t is given as an image of size 64×64. A train or test task is to achieve a high score on a set of environments configured by game and mode, where a mode describes predefined sets of levels (e.g., complexity of map) and backgrounds. Cobbe *et al.* [5] provide *easy* mode for each game, consisting of 200 levels and a certain set of backgrounds.

We simplify *easy* mode and train agents in *easybg* mode, of which the only difference from *easy* mode [5] is showing only a single background. This is useful to investigate the case that using visual augmentation is helpful for testing but not for training. We denote the task configuration by game_name(*mode*), e.g., Coinrun(*easybg*). Then, we evaluate generalization capabilities using two modes: *test-bg* and *test-lv*, which contain unseen backgrounds and levels, respectively, in addition to *easybg* mode that we use for training.

Types of augmentation. For clarity, we mainly focus on two types of visual augmentation, each of which conveys distinguishing inductive bias:

(a) *Random color* transforms an image by passing through either color jitter layer or random convolutional layer proposed in [22]. From this, we can impose the consistency to color changes, which may provide a strong generalization to diverse backgrounds of *test-bg* mode.

(b) *Random crop* leaves a randomly-selected rectangle and pads zeros to the rest of the image [26]. This augmentation is particularly useful in fully-observable games (e.g., Chaser and Heist), because it imposes an efficient attention mechanism.

We also report the result with other augmentations including *color jitter, random convolution, gray, and cutout color* in the supplementary material, from which the same messages can be interpreted.

Baseline methods for RL with data augmentation. We mainly compare the proposed methods (InDA and ExDA) to the following baselines:

(a) *RAD* [21] simply feeds PPO with experiences of original and augmented observations.

(b) *DrAC* [26] is a method to simultaneously minimizing L_{PPO} in Eq (1) and L_{dis} in Eq (4).

(c) *DrAC+PAGrad* is a variant of DrAC, which we devise to investigate another mechanism to relieve the interference between RL training and augmentation in Section 4.5.

The supplementary material presents further details and experiments, which we omitted for simplicity. All results in the main paper are averaged over five runs.

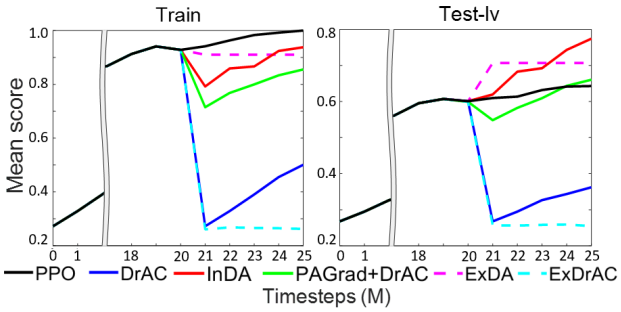
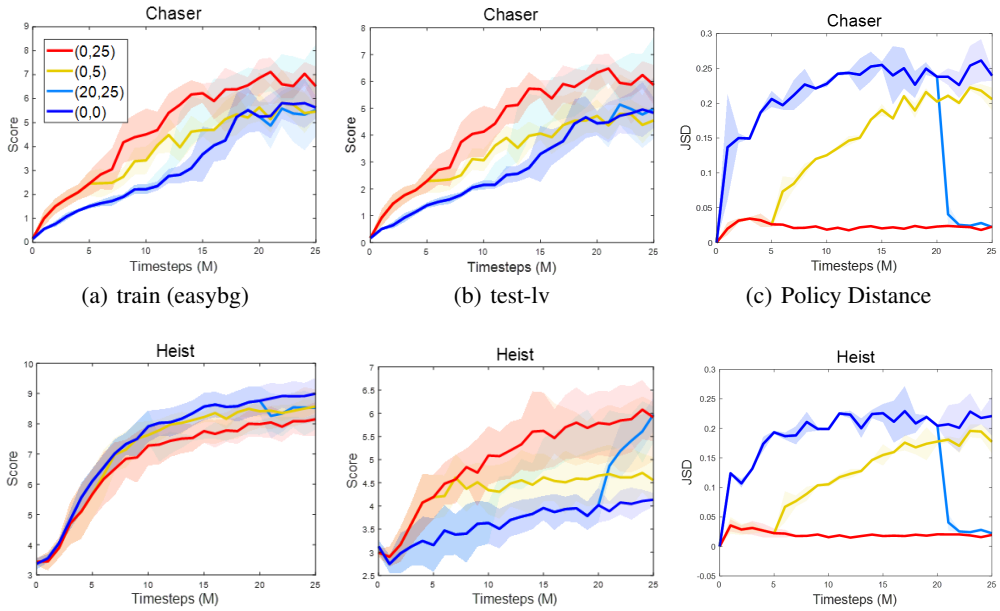


Figure 4: *Benefit of separating distillation from RL training.* We compare InDA, ExDA, DrAC, DrAC+PAGrad and ExDrAC when we start to apply each of them to distill the prior of random crop after 20M timesteps of PPO. ExDrAC is a variant of DrAC without RL training, i.e., minimizing only L_{dis} in Eq (4).

5.2 Benefit of separating distillation from RL training.

In Figure 4, after training PPO agent up to 20M timesteps, we start to suddenly apply one of the different distillation methods with random crop. We report averaged scores over 6 environments (Bigfish, Dodgeball, Plunder, Chaser, Heist, Maze) after normalized by the highest train score of PPO on each environment. After 20M timesteps, ExDA and ExDrAC have no RL training but minimize L_{DA} in Eq (5) and L_{dis} in Eq (4), respectively. The substantial gap between ExDA and ExDrAC justifies the design of DA explicitly preserving the knowledge from RL when distilling the prior. More importantly, it is remarkable that ExDA promptly learns the generalization ability once it starts to distill the prior. This validates the idea of postponing the distillation after RL training.

We now compare the distillation methods (InDA, DrAC, and DrAC+PAGrad) concurrently optimizing the RL objective and distilling the prior in Figure 4. Each method has performance degeneration due to the interference generated by distillation. However, InDA and DrAC+PAGrad have clearly smaller degeneration than DrAC which is the only one without any separation of optimizing the RL objective and distillation loss. We note that DrAC+PAGrad has more interference than InDA, and it seems to fail to impose the prior since there is not much difference to PPO in testing. Hence, this verifies the superiority of InDA which enables distilling the prior while alleviating the interference.



(a) train (easybg) (b) test-lv (c) Policy Distance

(d) train (easybg) (e) test-lv (f) Policy Distance

Figure 5: *Time-sensitivity of applying augmentation.* We compare train and test performance of InDA$[S, T]$ with *random crop*, where the timing of applying DA is governed by starting time S and terminating time T, and we evaluate four different pairs of $[S, T] = [0, 0], [0, 5], [20, 25], [0, 25]$ up to 25M timesteps. Furthermore, we show the change of distance between two policies on an original observation and an augmented observation. Note that InDA with $[S, T] = [0, 0]$ means RL training with no augmentation, i.e., vanilla PPO. We focus on Chaser and Heist since they exemplify two representative time-dependencies. Each train task uses *easybg* mode. We present further details and results with other games in the supplementary material.

5.3 Effective timings of distillation

In what follows, we aim to identifying effective timings of distillation. To this end, in Figure 5, we test several schedules of applying DA on two representative environments of distinguishing traits. The supplementary material presents the result on more environments, which is similar one of the representatives.

An effective timing: InDA. Chaser with random crop (Figure 5(a) and 5(b)) represents the case when augmentation improves the sample efficiency of training and thus the generalization ability in training. To compare the generalization ability, we measure policy distance between original and augmented observations using Jenson-Shannon divergence (Figure 5(c)). InDA[0, 5]'s policy distance is increased after it stops using DA. Thus, the generalization ability is degraded, if we do not continue to use DA. In this case, it is clear that InDA should be applied during the entire RL training, as InDA[0, 25] shows the best performance in both training and testing. In addition, it is also important to apply DA as soon as possible since the effect of InDA[0,5] applying DA in the beginning is relatively prompt and significant comparing to that of InDA[20,25]. This suggests that the automatic framework needs to explore more in the early stage.

An effective timing: ExDA. Heist with *random crop* (Figure 5(d) and 5(e)) shows the opposite use of data augmentation to what Chaser case suggests, i.e., postponing data augmentation as much as possible. Random crop generates a slight interference, although it immediately improves the generalization ability. We remark that the inductive bias from the random crop is easily forgotten, as the test performance of InDA[0, 5] is saturated right after stopping the distillation. This can be explained by the time-varying nature of sample distribution and objective in RL. Interestingly, it is in contrast to the data augmentation in SL, where an early distillation is sufficient to impose the prior [11]. On the other hand, the test performance curve of InDA[20, 25] soars right after DA. Furthermore, Figure 5(f) shows that InDA[20, 25] narrows the gap between two policies on the

original and augmented observation. Recalling the interference between RL training and distillation, this suggests postponing the distillation at the end of RL training, and motivates our ExDA.

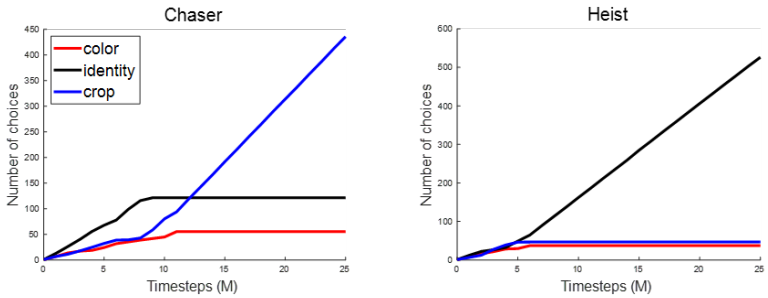
Performance benchmark on InDA and ExDA. In Table 1, we summarize the train and test performances of InDA and ExDA on a set of games with different augmentations and modes. ExDA outperforms other baselines with *random color* on *test-bg*. Moreover, we note that ExDA consumes only 0.5M timesteps to inject the prior at the end of RL training, whereas the others use all the training data. ExDA in both sample efficiency and generalization ability with *random crop* on *test-lv*. It is worth noting that DrAC+PAGrad is slightly better than DrAC, while there is a substantial gap between InDA and each DrAC-based algorithm. This again confirms the benefit from the separated distillation of InDA observed in Figure 4. These results demonstrate that each combination of environment and augmentation has a suitable time at which to apply augmentation, and the gain from using the right distillation timing, i.e., online (InDA, DrAC, or DrAC+PAGrad) or offline (ExDA), is rigid regardless of the choice of algorithms.

Augmentation		Task	PPO	RAD	DrAC	DrAC+PAGrad	InDA	ExDA
Rand color	Rand conv	Train	**1.00**	0.98	0.88	0.89	0.88	0.98
		Test-bg	1.00	1.08	1.86	1.88	1.92	**2.11**
		Test-lv	**1.00**	0.81	0.84	0.84	0.7	0.87
	Color jitter	Train	**1.00**	0.94	0.95	0.95	0.96	0.98
		Test-bg	1.00	1.37	1.44	1.44	1.43	**1.48**
		Test-lv	**1.00**	0.83	0.86	0.86	0.84	0.88
Rand crop		Train	1.00	0.28	1.08	1.09	**1.25**	0.91
		Test-bg	**1.00**	0.64	0.87	0.94	0.94	0.95
		Test-lv	1.00	0.46	1.52	1.53	**1.80**	1.09

Table 1: *Benchmark of InDA and ExDA.* We report normalized train and test scores of InDA, ExDA and DrAC with PAGrad on Open AI Procgen, compared to baselines PPO, DrAC [26], RAD [21]. **Boldface** indicates the best performance. We average the score among several environments [(Rand color: coinrun, ninja, climber, fruitbot, jumper, heist, maze), (Rand crop: Bigfish, Dodgeball, Plunder, Heist, Chaser, Maze)] after normalization considering PPO score to be 1. Every method is trained on 200 levels, using *easybg* mode. We evaluate test performance on both *test-bg* and *test-lv*. The results can be interpreted as an upper bound of potential gain from using data augmentation at perfect timing.

5.4 Adaptive scheduling methods

Adaptive selecting of timings: UCB-InDA and UCB-ExDA. It is hard to know in advance whether a certain augmentation helps the training or not [26]. We hence employ UCB-InDA which estimates the gain or damage from each augmentation from trial and error and identifies the one



(a) Chaser (b) Heist

Figure 6: *Exploration & exploitation to find the most beneficial augmentation*. We show that selected augmentation during training with UCB-InDA for each Chaser and Heist. UCB-InDA automatically selects the augmentation among three options, random color, identity and random crop.

Env	Method	PPO	DrAC	UCB-DrAC	InDA	ExDA	UCB-InDA	UCB-ExDA
Heist	Train	9.2 ± 0.46	3.53 ±0.3	7.41± 2.09	4.9 ± 0.79	9.14 ± 0.56	**9.67± 0.23**	9.45 ± 0.29
	Test-bg	5.18 ± 1.53	3.58 ± 0.31	3.76± 0.54	4.9 ± 0.87	7.05 ± 1.29	6.23± 1.29	**7.86± 0.83**
	Test-lv	4.13 ± 1.39	3.07 ± 0.33	3.49±0.48	1.47± 0.77	5.05 ± 0.98	4.8± 1.24	**5.74± 0.67**
Chaser	Train	5.63± 1.12	0.16±0.02	4.6± 1.22	5.69± 1.42	5.58± 1.33	**7.49±1.27**	7.23± 1.15
	Test-bg	0.87± 0.06	0.1± 0.02	0.57±0.12	3.51± 1.33	1.02± 0.04	1.08± 0.08	**3. 18± 0.79**
	Test-lv	4.83± 0.88	0.14±0.01	4.14±1.01	4.96± 1.21	5.11± 1.05	**6.45± 0.8**	6.43± 1.28

Table 2: *Full exploitation of augmentation to improve generalization on both test-bg and test-lv*. We compare InDA, ExDA, DrAC, UCB-DrAC, UCB-InDA, UCB-ExDA and PPO about train, test-bg and test-lv. **Boldface** indicates the best method. InDA and DrAC use both random color and random crop during RL training. ExDA use both augmentation after RL training. UCB-InDA and UCB-DrAC are trained as automatically selecting the augmentation during training. UCB-ExDA trains ExDA after UCB-InDA with both augmentation.

with most help. Recall Table 2 where PPO (without augmentation) shows much better training performance than InDA in Heist. As shown in Figure 6(b), UCB-InDA is able to identify that no augmentation is best for training in Heist. This implies that ExDA is more appropriate than InDA. Conversely, random crop is selected on Chaser (Figure 6(a)). As the result, we can automatically select InDA or ExDA appropriately for each task.

Fully exploitation of augmentation In Table 2, when both random color and random crop are used to improve generalization on both *test-bg* and *test-lv*, we report numerical evaluation of UCB-InDA and UCB-ExDA with other baselines on train and test tasks. Decreased train performance of DrAC and InDA compare to PPO show the difficulty of simultaneous training with several augmentations. Train performance of UCB-InDA and UCB-DrAC are improved by adaptive selecting, especially, UCB-InDA is better than UCB-DrAC. The gap is made due to the robustness about the change of augmentation during training. In terms of generalization, UCB-ExDA clearly surpasses UCB-InDA thanks to ExDA to extract all the priors from the complete set of data augmentations at the end of RL training.

6 Discussion

We have identified two most effective yet simple timings (InDA and ExDA) of data augmentation for RL, and proposed UCB-ExDA framework to adaptively select the best scheduling augmentations. We note that the effectiveness of this framework is restricted but specialized for RL with the unique non-stationary nature. Indeed, in SL without shift of data distribution and objective, it is sufficient to apply data augmentation at the beginning [11]. Our framework employs the most basic multi-armed bandit algorithm with a finite set of data augmentation. It is interesting to investigate a room to improve by further optimizing continuous parameters of data augmentation for RL, c.f., an auto-augmentation technique to optimize continuous augmentation parameter per sample for SL [12]. Another promising direction is to accelerate the distillation process of DA by data condensation with augmentation [39]. This is possible with our framework clearly separating between RL training and distillation, and may be particularly useful to train distributed RL agents since a condensed data for an agent's distillation is usable for the other.

Acknowledgments and Disclosure of Funding

We thank Kimin Lee for helpful discussions. This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2019-0-01906, Artificial Intelligence Graduate School Program (POSTECH)) and Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, Artificial Intelligence Innovation Hub) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021M3E5D2A01023887).

References

[1] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[2] Sanjeev Arora, Nadav Cohen, Wei Hu, and Yuping Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7411–7422, 2019.

[3] Peter Auer. Using confidence bounds for exploitation-exploration trade-offs. *Journal of Machine Learning Research*, 3(Nov):397–422, 2002.

[4] Richard E Bellman. *Adaptive control processes*. Princeton university press, 2015.

[5] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pages 2048–2056. PMLR, 2020.

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

[7] Dumitru Erhan, Aaron Courville, Yoshua Bengio, and Pascal Vincent. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 201–208. JMLR Workshop and Conference Proceedings, 2010.

[8] 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. *arXiv preprint arXiv:1802.01561*, 2018.

[9] Jesse Farebrother, Marlos C. Machado, and Michael Bowling. Generalization and regularization in dqn, 2020.

[10] Gauthier Gidel, Francis Bach, and Simon Lacoste-Julien. Implicit regularization of discrete gradient dynamics in linear neural networks. In *Advances in Neural Information Processing Systems*, pages 3196–3206, 2019.

[11] Aditya Sharad Golatkar, Alessandro Achille, and Stefano Soatto. Time matters in regularizing deep networks: Weight decay and data augmentation affect early learning dynamics, matter little near convergence. In *Advances in Neural Information Processing Systems*, pages 10678–10688, 2019.

[12] Denis Gudovskiy, Luca Rigazio, Shun Ishizaka, Kazuki Kozuka, and Sotaro Tsukizawa. Autodo: Robust autoaugment for biased data with label noise via scalable probabilistic implicit differentiation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16601–16610, 2021.

[13] Suriya Gunasekar, Blake E Woodworth, Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[14] Nicklas Hansen, Yu Sun, Pieter Abbeel, Alexei A Efros, Lerrel Pinto, and Xiaolong Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[15] Nicklas Hansen and Xiaolong Wang. Generalization in reinforcement learning by soft data augmentation. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13611–13617. IEEE, 2021.

[16] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2019.

[17] Irina Higgins, Arka Pal, Andrei A. Rusu, Loic Matthey, Christopher P Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning, 2018.

[18] Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

[19] 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.

[20] 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.

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

[22] Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv*, pages arXiv–1910, 2019.

[23] Zhuang Liu, Xuanlin Li, Bingyi Kang, and Trevor Darrell. Regularization matters in policy optimization – an empirical study on continuous control, 2020.

[24] 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.

[25] 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.

[26] Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

[27] Sharath Chandra Raparthy, Bhairav Mehta, Florian Golemo, and Liam Paull. Generating automatic curricula via self-supervised active domain randomization, 2020.

[28] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

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

[30] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[31] 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.

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

[33] Adam Stooke, Kimin Lee, Pieter Abbeel, and Michael Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

[34] 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 *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[35] Xiaoxia Wu, Ethan Dyer, and Behnam Neyshabur. When do curricula work?, 2021.

[36] Denis Yarats, Amy Zhang, Ilya Kostrikov, Brandon Amos, Joelle Pineau, and Rob Fergus. Improving sample efficiency in model-free reinforcement learning from images, 2020.

[37] Kaichao You, Mingsheng Long, Jianmin Wang, and Michael I. Jordan. How does learning rate decay help modern neural networks?, 2019.

[38] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[39] Bo Zhao and Hakan Bilen. Dataset condensation with differentiable siamese augmentation. In *International Conference on Machine Learning*, pages 12674–12685. PMLR, 2021.

Checklist

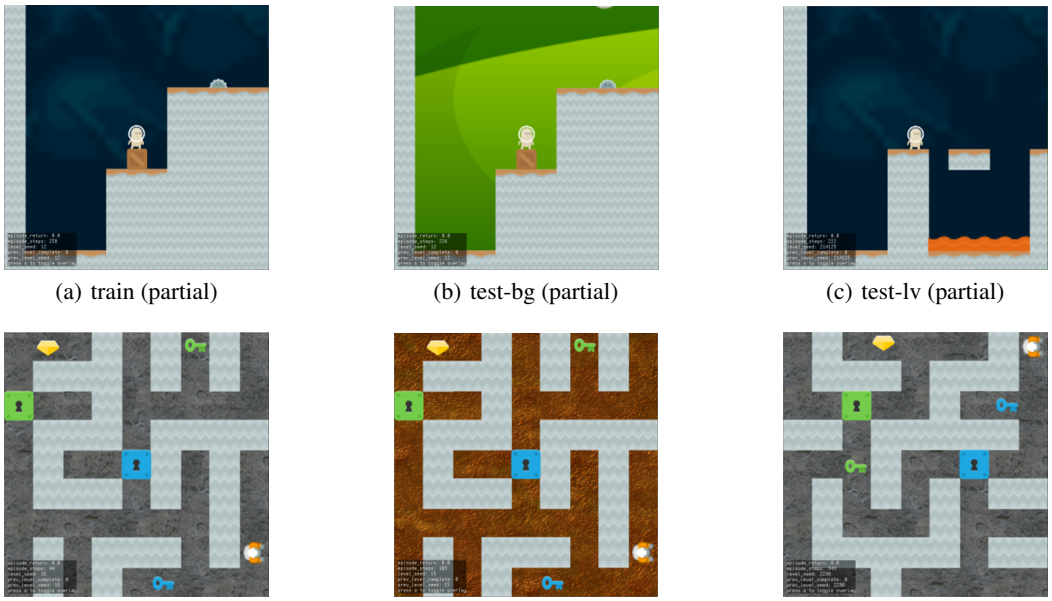
1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] We claim our three main contributions in 1 and target on DeepRL.
   (b) Did you describe the limitations of your work? [Yes] We describe the limitations of our work about a limited set of data augmentation for UCB methods in Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We believe that our research will not have a negative impact on society.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes] We read the ethics review guidelines.
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] Our works are based on experimental results.
   (b) Did you include complete proofs of all theoretical results? [N/A] Our works are based on experimental results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We describe our training environments and include code in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We describe training details in the supplementary material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We indicate standard deviation with score in Figure 5 and Table 2 except normalized score with several environments such as Figure 4 and Table 1.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We explain about training time in the supplementary material.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Our experiment environments are baed on OpenAI Procgen [5].
   (b) Did you mention the license of the assets? [Yes] We include the license of the asssets in the code.
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Modified Procgen Environments



(a) train (partial) (b) test-bg (partial) (c) test-lv (partial)

(d) train (fully) (e) test-bg (fully) (f) test-lv (fully)

Figure 7: An example set of training and testing environments in Procgen benchmark: (upper row) an example of partially observable environment with Coinrun; (lower row) an example of fully observable environment with Heist; (left column) train: a set of levels and backgrounds for training; (center column) test-bg: the same training levels on unseen backgrounds; (right column) test-lv: a set of unseen levels on the same training backgrounds

This section explains Modified Procgen Environments, which is designed to verify different types of generalization, backgrounds, and levels. Open AI Procgen environments [5] share background themes such as *space_backgrounds*, *platform_backgrounds*, *topdown_backgrounds*, *water_backgrounds*, *water_surface_backgrounds*. We create new difficulties as *Easybg*, *Easybg-test*, *Easy-test*. *Easybg* generates environments which contain only one for each background, wall and agent theme. *Easybg-test* and *Easy-test* are for test about background change after trained on *Easybg* and *Easy*. Wall theme in (Climber, Coinrun, Jumper, Ninja) and Agent theme in (Climber, Coinrun) also compose with only one image resource in *Easybg*. Figure 7 presents an example set of modes that we use in evaluation. Furthermore, We fix the exit_wall_choice and enemy theme in Dodgeball. We describe the usage themes in each environment, which are grouped by backgrounds theme as below:

- *space_backgrounds* (Bossfight, Starpilot)
  Background: "space_backgrounds/deep_space_01.png"
- *platform_backgrounds* (Caveflyer, Climber, Coinrun, Jumper, Miner, Ninja)
  Background:"platform_backgrounds/alien_bg.png", Coinrun (Agent color: Beige, Wall themes: Dirt), Climber (Agent color: Blue, Wall themes: tileBlue), Jumper (Wall theme: tileBlue), Ninja (Wall theme: bricksGrey)
- *topdown_backgrounds* (Chaser, Dodgeball, Fruitbot, Heist, Leaper, Maze)
  Background:"topdown_backgrounds/floortiles.png", Dodgeball (Enemy theme: "misc_assets/character1.png", Exit_wall_choice: 0)
- *water_backgrounds* (Bigfish)
  Background:"water_backgrounds/water1.png"
- *water_surface_backgrounds* (Plunder)
  Background:"water_backgrounds/water1.png"

Easybg-test uses backgrounds in each background group, except the one used in *Easybg*. *Easy-test* is only defined for Climber, Jumper, Ninja, and they compose with *topdown_backgrounds*.

B Implementation details

In this section, we explain about InDA, ExDA, UCB-InDA and other baselines. We train the agent with IMPALA-CNN [8] in every experiment.

B.1 InDA

We use PPO [29] as a base RL algorithm, For data efficiency, we store the observations during RL training in buffer $\mathcal{D}_O$. Before DA phase, we also make policy buffer $\mathcal{D}_\Pi$, value function buffer $\mathcal{D}_V$ and augmented observation buffer $\mathcal{D}_\phi$ for distillation, because we only use one network model. We randomly sample pairs of (o, π, V) from buffer, and minimize loss function $L_{\text{DA}}(\theta)$. We reuse the sample three times like PPO, it can be controlled by # Epochs of DA. We did a greed searches for learning rate of DA $l_{DA} \in [1 \times 10^{-3}, 5 \times 10^{-4}, 2 \times 10^{-4}, 1 \times 10^{-4}, 5 \times 10^{-5}]$ and interval $I \in [1, 5, 10]$ and found the best combination $l_{\text{DA}} = 10^{-4}$ and interval $I = 5$. We fix the buffer size $\mathcal{D}_O = 40960$, because we collect the observations during five RL phases $(5 \times 256 \times 32)$. We describe the every hyperparameter as below:

Hyperparamter	Value
γ	0.999
λ	0.95
# Timesteps per rollout	256
# Epochs per rollout	3
# Minibatches per epoch	8
Reward Normalization	Yes
# Workers	1
# Environments per worker	32
Total timesteps	25M
LSTM	No
Frame Stack	No
Optimizer	Adam optimizer
Entropy bonus	0.01
PPO clip range	0.2
Learning rate	5×10^{-4}
Interval I	5
Size of $\mathcal{D}_O$	40960
# Epochs of DA	3
Learning rate of DA l_{DA}	1×10^{-4}
Image transformation ϕ	Any augmentation

B.2 ExDA

In ExDA, we generate and store (o, π, V) using $f_{\theta_{old}}$ in buffer $\mathcal{D}$. The optimal buffer size depends on the episode length of each environment. However, we standardize the buffer size as 0.5M in every environment. We augment the observation with three epochs intervals when using randomized augmentation methods. We did greed searches for # minibatches $[1024, 2048, 4096]$ and learning rate $[5 \times 10^{-4}, 1 \times 10^{-3}, 2 \times 10^{-3}]$. As a result, we select # of minibatches 4096 and a learning rate $1e-3$.We describe every hyperparameter as below:

Hyperparameter	Value
Size of $\mathcal{D}_O$	0.5M
# Epoch	30
# Minibatches per epoch	4096
Learning rate	1×10^{-3}
# Workers	1
Optimizer	Adam optimizer
Image transformation ϕ	Any augmentation

B.3 UCB-InDA

We use UCB-InDA as a discriminator to determine the necessity of augmentation during the training. The gain of an augmentation is a mean of return during interval I, $G(s) = \frac{1}{I}\sum_{i=0}^{j-1} R(s+j)$. The return is computed by the sum of estimated advantage and predicted value, which are expected value of the agent trajectory, $R(s) = \hat{\mathbb{E}}_{(o_t,a_t)\sim\pi_\theta}[\hat{A}_t + V_\theta(o_t)]$. The $\hat{A}_t$ is advantage from Generalized Advantage Estimator [28]. Thus, we can evaluate how augmentation affects the return on the agent trajectory. However, the distribution of return is non-stationary, as the agent policy is changed. Therefore, we use the window average gain $\bar{G}_\phi$ rather than the whole gain from the past evaluation. Furthermore, the drastic change of return causes the gap of gain between the augmentation according to sampling time at the transient time of training and leads to poor exploration about some augmentation methods. For stable exploration, we fix the minimum exploration frequency and use forced exploration method after the minimum exploration as below:

$$\bar{G}_{\phi_{max}}(s) + c\sqrt{\frac{\log(s)}{N_{\phi_{max}}(s)}} \leq \bar{G}_{\phi_{min}}(s) + c\sqrt{\frac{\log(s)}{N_{\phi_{min}}(s)}} \tag{9}$$

$$c = \frac{\bar{G}_{\phi_{max}} - \bar{G}_{\phi_{min}} + \epsilon}{\sqrt{\log(s)} \times \max(\frac{1}{\sqrt{N_{\phi_{min}}(s)}} - \frac{1}{\sqrt{N_{\phi_{max}}(s)}}, \frac{1}{\sqrt{W-1}} - \frac{1}{\sqrt{W}})} \tag{10}$$

where $\phi_{max} = \arg\max_{\phi\in\Phi} \bar{G}_\phi, \phi_{min} = \arg\min_{\phi\in\Phi} \bar{G}_\phi$. We set the hyperparameter as below table:

Hyperparameter	Value
Window size of gain W	3
Minimum exploration frequency	15

B.4 Baselines

We compare ExDA and InDA with PPO [5], DrAC [26], Rand-FM [22], RAD [21]. Every baseline is based on PPO [5] and we adopt the implementation of PPO in [5].

- DrAC [26] regularizes both policy and value function as self-supervised learning. Regularization term have hyperparameter α_r for ratio with PPO objective. We use the hyperparameter recommended by the author.

- Rand-FM [22] is composed with random convolution networks and feature matching. They also need hyperparameter β for ratio between feature matching and PPO objective. We use same β with author.

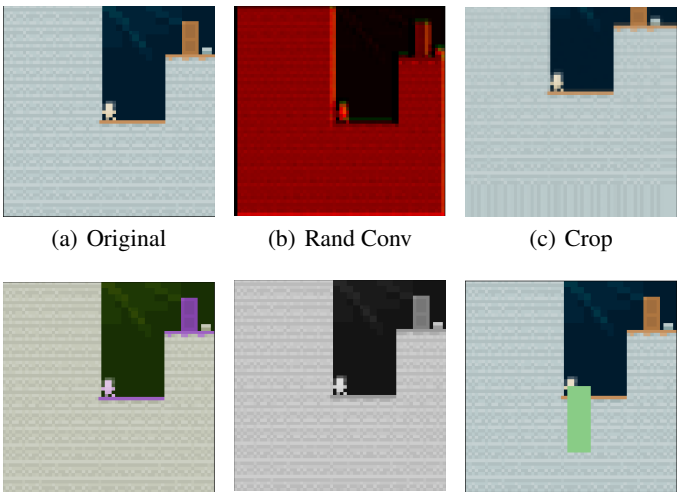
- RAD [21] naively use augmented observations in state distribution. Thus, there are no additional hyperparameters.

We describe the hyperparameter of baselines in below table:

Hyperparameter	Value
γ	0.999
λ	0.95
# of timesteps per rollout	256
# of epochs per rollout	3
# of Minibatches per epoch	8
Reward Normalization	Yes
# of Workers	1
# of environments per worker	64
Total timesteps	25M
LSTM	No
Frame Stack	No
Optimizer	Adam optimizer
Entropy bonus	0.01
PPO clip range	0.2
Learning rate	5×10^{-4}
α_r (DrAC)	0.1
β (Rand-FM)	0.002

C Data augmentation

In our experiments, we use five augmentation methods: *crop, grayscale, cutout color, random convolution* and *color jitter*. We refer the implementation of augmentations from Lee *et al.* [22] (*random convolution*), Laskin *et al.* [21] (*cutout color, color jitter*) and Raileanu *et al.* [26] (*grayscale, crop*). We expect the generalization about background change from *random convolution, color jitter, gray, cutout color*. About the change of levels, we use *crop* and *cutout color* for generalization. Examples of data augmentation are represented below:



(a) Original (b) Rand Conv (c) Crop

(d) Color jitter (e) Gray (f) Cutout color

Figure 8: Examples of visual augmentations

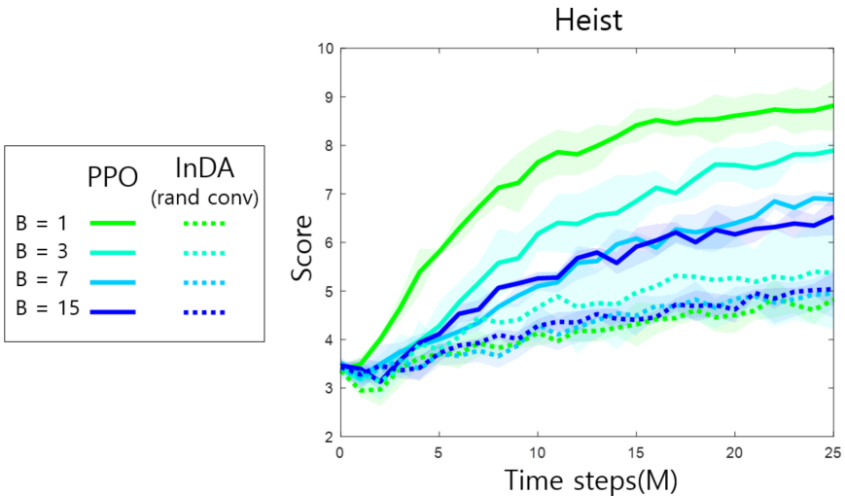


Figure 9: *Interference from increased complexity of learning.* We present train performance when training agent on Heist(*easybg-B*), where *easybg-B* is a variant of *easybg* mode with B backgrounds. We plot the train performance curves of PPO and InDA with random convolution over training epochs in absolute score.

D Interference from increased complexity of learning

Even if the policy for the original observation is fixed and augmentation is used, learning may still be hindered. In Figure 9, we report training curves of the agents on train tasks of different degree of background diversity. InDA with random convolution is anticipated to lead the prior on color consistency, which seems helpful to handle background diversity. However, PPO outperforms InDA in terms of the sample complexity to master train tasks, while the gap is decreasing as the background diversity increases.This implies that diverse backgrounds make hard to train by increased complexity of learning, and also data augmentation can cause similar difficulty even with right prior, when using it during RL training. We further remark that ExDA using random convolution after PPO trained on Heist(*easybg-1*) achieves 8.15 on Heist(*easybg-15*), which is much higher than PPO's 6.53 trained on Heist(*easybg-15*). This suggests the importance of simplifying the train task and the utility of ExDA which completely separates RL training and distillation with augmentation.

E Robustness in loss function change

In ExDA, we transfer the policy after training 20M time steps with PPO. Thus, we explain why other augmentations are not used after pre-training. We compare the results of training and test performance with Drac [26], Rand-FM [22], Rad [21] when we train each method for 5M after training PPO for 20M time steps. We use random convolution and crop as data augmentation methods, and we do not compare with RAD when we use crop in Figure 12 and Figure 13. The *crop* method used in our paper do not work well in RAD, because they use a different crop method with [26] in their paper [21]. InDA is more stable than others in training, and it affects generalization performance.

Every training curves decline immediately after starting training with augmented observations at 20M time steps. The objective function is changed to each baseline, and augmented data is newly added to data distribution. Thus, the optimizer should find a new optimal point for new objective function and data. During find the new optimal points, the agent learns along with the different directions from the optimization direction in pure PPO. Thus, performance can be degraded because the learning direction on loss landscape is different from maximizing rewards on non-augmented data in PPO. In spite of using self-supervised learning or representation learning, the policy is changed because they update the same network's parameter for matching policy or latent features, such as DrAC [26] and Rand-FM [22]. However, InDA is more stable than the others because we distill the fixed policy and value using DA. It does the stable training through conserving the policy on non-augmented observations during optimizing for augmented data.

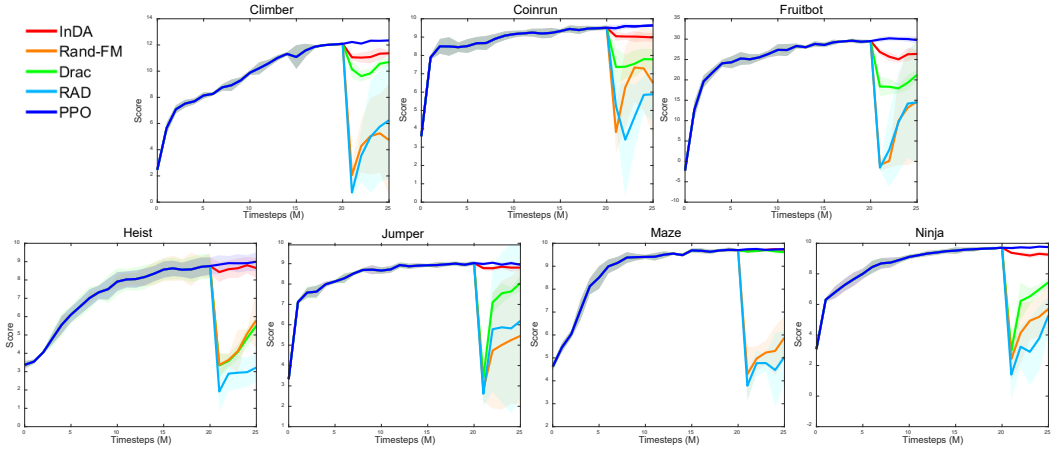


Figure 10: Comparison of the training performance when *random convolution* is applied after 20M timesteps with various augmentation methods.

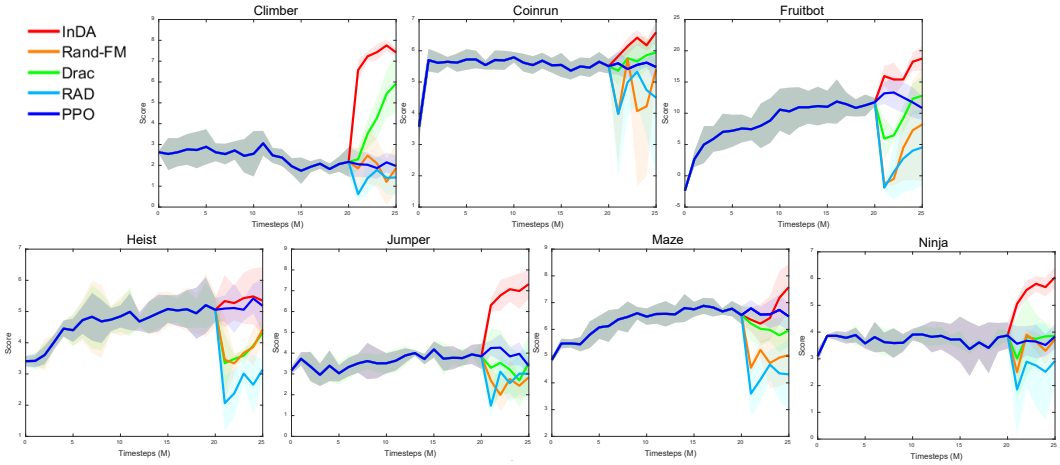


Figure 11: Comparison of the test performance when *random convolution* is applied after 20M timesteps with various augmentation methods.

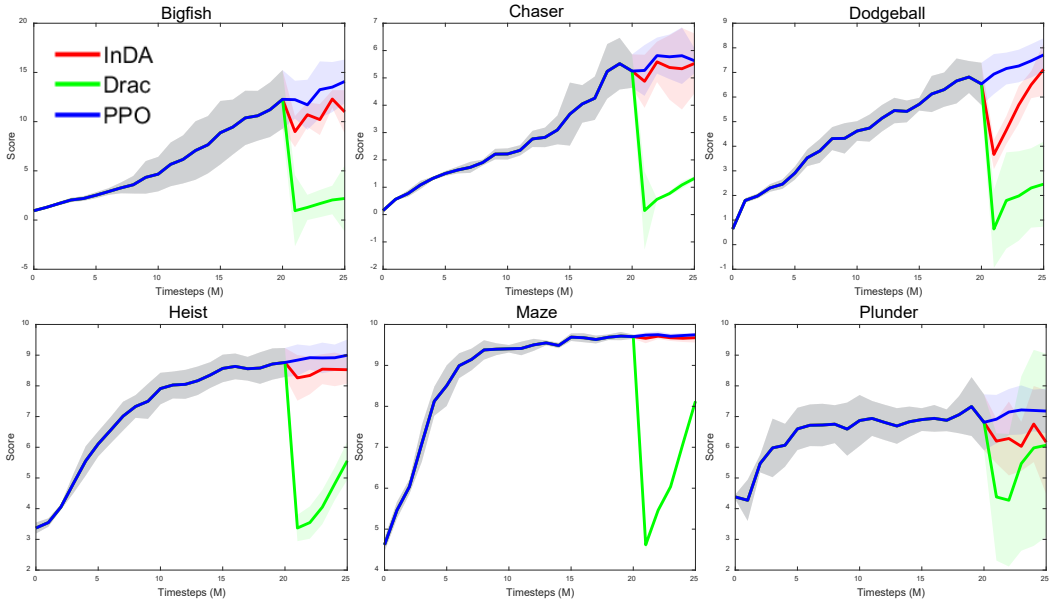


Figure 12: Comparison of the training performance when *crop* is applied after 20M timesteps with InDA and Drac.

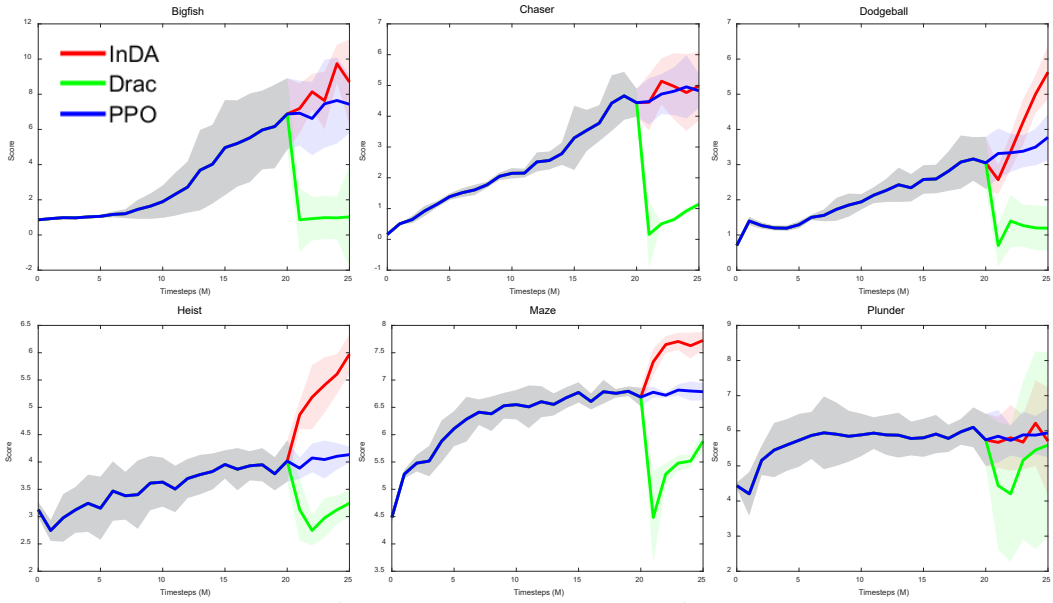


Figure 13: Comparison of the test performance when *crop* is applied after 20M timesteps with InDA and Drac.

F Robustness against wrong augmentation

In this section, we verify the robustness of ExDA from an obvious wrong augmentation: just making an black image, denoted by *black*. Using this clearly interferes with RL training. We compare PPO, InDA, ExDA, UCB-InDA, UCB-ExDA and DrAC. The hyperparameter is almost the same with Section 5.4 except we use *black* instead of *random color* and *random crop*. For UCB-based methods, we use 4 arms: *black*, *random color*, *random crop*, and *no aug*. As expected, ExDA preserves the PPO score, but InDA and DrAC degrade the score. Furthermore, UCB-InDA improves the performance in

Chaser, and also it almost preserves the score in Heist. Lastly, UCB-ExDA also maintains the score from UCB-InDA.

Env	PPO	DrAC	InDA	ExDA	UCB-InDA	UCB-ExDA
Heist	**9.2** $\pm$ **0.46**	7.35 $\pm$ 0.684	6.72 $\pm$0.419	8.6 $\pm$ 0.189	8.84 $\pm$ 0.307	8.64$\pm$ 0.264
Chaser	5.63$\pm$ 1.12	1.49$\pm$0.036	5.43$\pm$ 0.633	5.1$\pm$0.331	**6.74**$\pm$**0.588**	6.28$\pm$ 0.436

Table 3: Robustness from the wrong augmentation

G Ablation study of ExDA

G.1 Initialization and regularization term

In this section, we do an ablation study about the factor of ExDA. We mention the loss function and re-initialization issue in subsection 4.3. ExDA does not have to minimize L_{VD} because the value function is useless after RL training. The below results show that L_{VD} cannot give any benefit in ExDA. Thus, we only use L_{PD} for computational complexity. Furthermore, we also compare to verify the effect of non-stationarity with a re-initialized agent before distillation. Igl *et al.* [18] argued that the non-stationarity causes the reduction of generalization. However, the re-initialization is not critical in test performance, as shown in Figure 15. Moreover, sometimes re-initialization makes it difficult to distill training performance such as Fruitbot and Ninja in Figure 14. We use *random convolution* as an augmentation method in here.

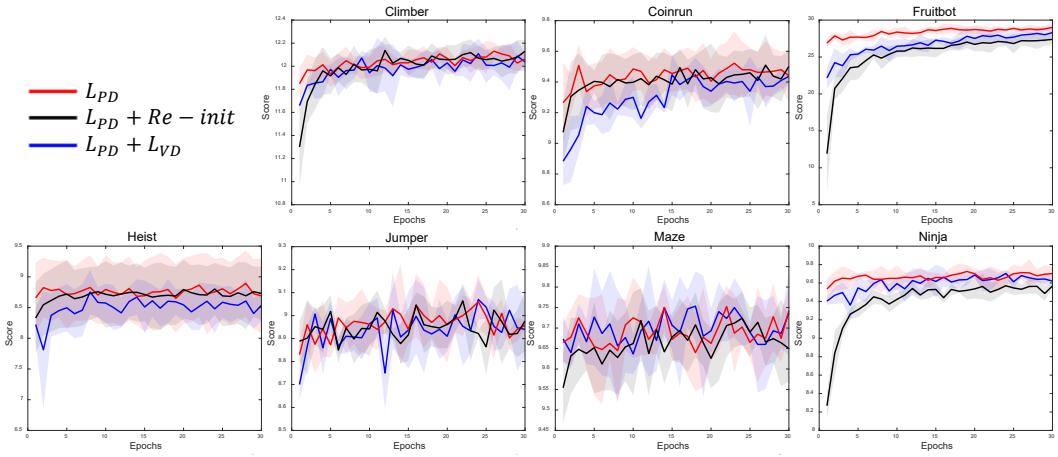


Figure 14: Training performance of ExDA with re-initialization or regularization with value funtion.

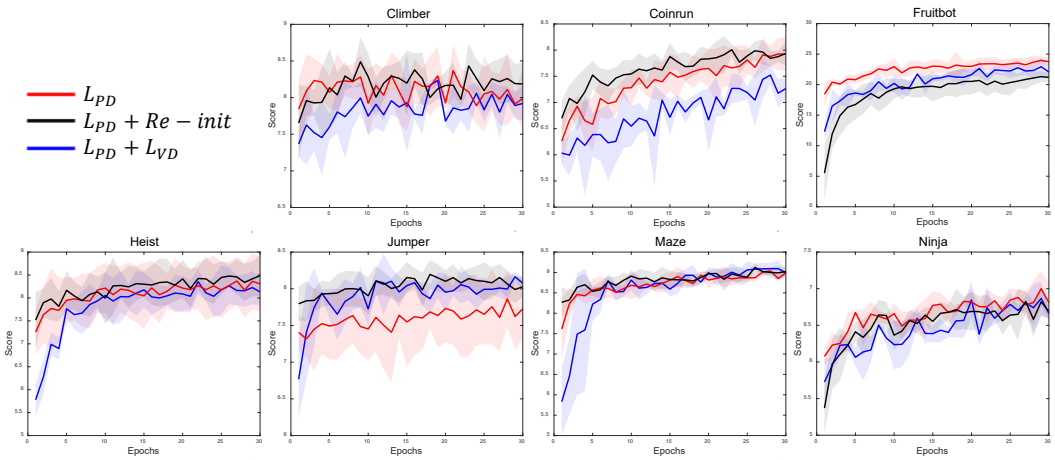


Figure 15: Test performance of ExDA with re-initialization or regularization with value funtion on unseen backgrounds.

G.2 ExDA after InDA with various backgrounds

When augmentation helps the training, ExDA struggle to follow the training performance of InDA because ExDA's training performance is limited by pre-trained agent's policy. Thus, we use InDA for ExDA's pre-training , and call it as ExDA (InDA). As shown in Table 5, ExDA (InDA) is comparable to InDA, but not beyond. Thus, unless there is a meaningful difference in training performance, ExDA has no better generalization than InDA. However, in computational complexity, ExDA is more efficient than others such as InDA and DrAC when they have a similar performance. In the following section, we discuss computational complexity.

Easy	PPO	InDA	ExDA (PPO)	ExDA (PPO) + reinit	ExDA (InDA)	ExDA (InDA) + reinit
Jumper	8.55 ±0.17	8.94 ±0.09	8.5 ±0.183	8.6 ±0.156	8.83 ±0.215	8.83 ±0.126
Ninja	7.49 ±0.42	8.88 ±0.34	7.03 ±0.058	7.23 ±0.159	8.71 ±0.344	8.56 ±0.394
Climber	8.63 ±0.46	8.5 ±0.29	8.1 ±0.268	8.09 ±0.268	8.16 ±0.441	7.99 ±0.383

Table 4: The comparison with diverse agents which are trained with ExDA

Easy	PPO	InDA	ExDA (PPO)	ExDA (PPO) + reinit	ExDA (InDA)	ExDA (InDA) + reinit
Jumper	6.85 ±0.19	7.94 ±0.19	7.54 ±0.158	7.48 ±0.154	7.98 ±0.148	7.67 ±0.155
Ninja	6.29 ±0.19	6.5 ±0.19	5.56 ±0.158	5.73 ±0.154	6.27 ±0.148	5.94 ±0.155
Climber	6.96 ±0.65	7.28 ±0.35	7.06 ±0.541	6.89 ±0.237	6.8 ±0.441	5.45 ±0.383

Table 5: Test performance of agents, which is trained on easy mode with random convolution.

G.3 Computational complexity

We compare the computational complexity with ExDA and InDA. InDA do DA for every 25M observations during training and reuse the sample in three times. However, ExDA only use 0.5M for DA during 30 epochs. Thus, ExDA is almost 5 times more efficient than InDA by rough calculation. Furthermore, the ExDA saves the time for augmentation compared to InDA. When we train with same computational setting (GPU: GeForce RTX 2080 TI), ExDA only consumes 5 hours + 2 hours

(PPO) when using random convolution, but, InDA consumes 18 hours. Thus, we recommend ExDA when InDA cannot give a meaningful gain in training performance.

H Comparison between InDA and ExDA using the same steps of RL training

In every experiment in the paper, we set the evaluation setup to be somewhat unfavorable to ExDA (using 20M time steps of RL training followed by additional 0.5M steps of distillation; denoted by ExDA(20M)) compared to InDA or other baselines (25M time steps of RL training) to clearly avoid potential complaint about the extra 0.5M steps for ExDA, It is obvious that the performance of ExDA is improved if we put more time steps for RL training, and thus the benefit of ExDA compared to InDA becomes more conspicuous if ExDA is the effective timing. Thus, we do an additional experiment evaluating ExDA (25M) using 25M RL time steps and 0.5M distillation time steps as below table:

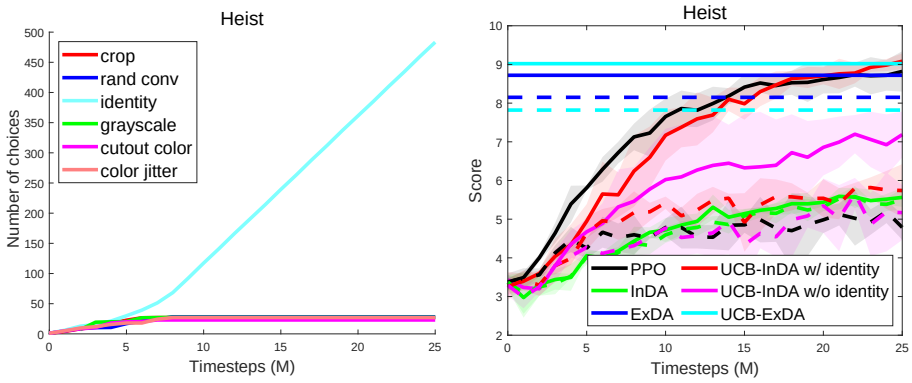
Easybg		PPO	DrAC	RAD	InDA	ExDA(20M)	ExDA(25M)
Heist	Train	**9**	5.95	7.94	5.15	8.72	8.93
	Test-bg	5.18	5.47	4.78	4.96	8.15	**8.26**
	Test-lv	4.13	5.4	3.81	**5.91**	5.35	5.41

Table 6: Additional RL training before ExDA

The result of ExDA(25M) reinforces our main message: postponing data augmentation when it generates a severe interference with RL training in Table 6. ExDA(25M) has a slight drop in train score compared to PPO, but it could be eventually removed if we put (slightly) more steps for distillation.

I UCB with a large set of augmentation

In Figure 6, we only use three arms such as *random color*, *random crop* and *no augmentation*. Thus, we try to select the usuful augmentations at each time among a large set of augmentation, which contains *gray*, *cutout color*, *random convolution*, *color jitter*, *random crop* and *no augmentation*. In Figure 16(a), we show the result of arm selection with UCB-InDA. The *identity* function is most frequently used in the same as Figure 6. On the other hand, we ablate the neccesity of identity function in UCB-InDA, it shows that the *identity* function is needed.



(a) Augmentation selection (b) Performance

Figure 16: Figure 16(a) show the selected number of augmentations by UCB on Heist. We compare two UCB-InDAs, w/ and w/o identity function with PPO, InDA, ExDA, UCB-ExDA on Heist *easybg* in Figure 16(b). UCB-InDA is trained after UCB-InDA w/ identity, we use *random convolution* as a data augmentation in InDA, ExDA. Solid line: train performance; dotted line: test performance. ExDA achieves larger test performance than InDA by preserving train performance. Moreover, UCB-InDA w/ identity outperforms UCB-InDA w/o identity in the training.

J Time matter in training

This section shows every result of Figure 5 about time dependency with InDA. We experiment with *random convolution, crop, color jitter* and evaluate the test on unseen backgrounds (*random convolution, color jitter*) and levels (*random crop*). However, the effect of generalization is hard to recognize in most cases, as shown in Appendix K. Thus, we mainly discuss the most effective augmentation, such as random convolution and crop in the main paper, and only represent some environments that have helped the generalization by color jitter. *easybg* mode is used as default mode with three *easy* mode (Climber, Jumper, Ninja) in our experiments. The shaded regions and solid line represent the standard deviation and mean, across five runs.

J.1 Random convolution

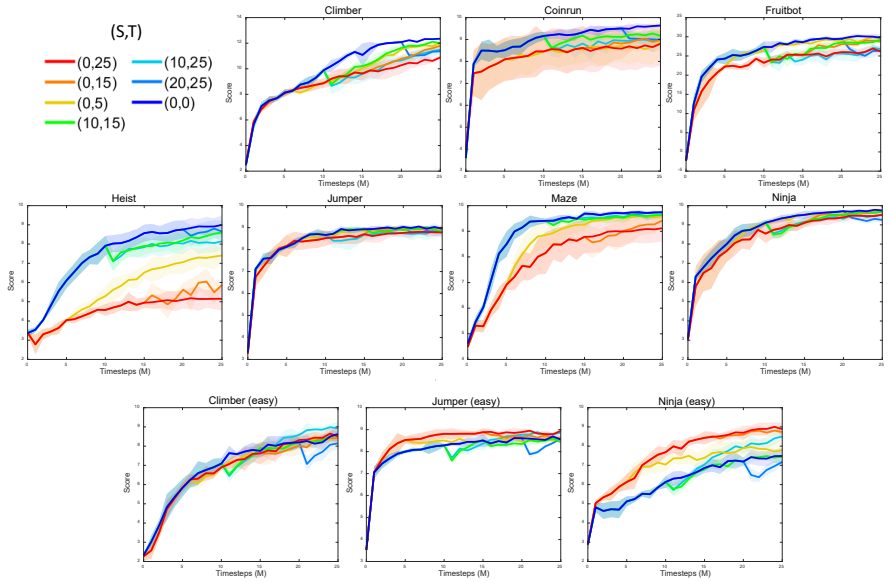


Figure 17: Comparison of training performance according to usage period of augmentation with InDA (*random convolution*): The *easybg* is disturbed by *random convolution*, but, *easy* mode is improved training performance by *random convolution*.

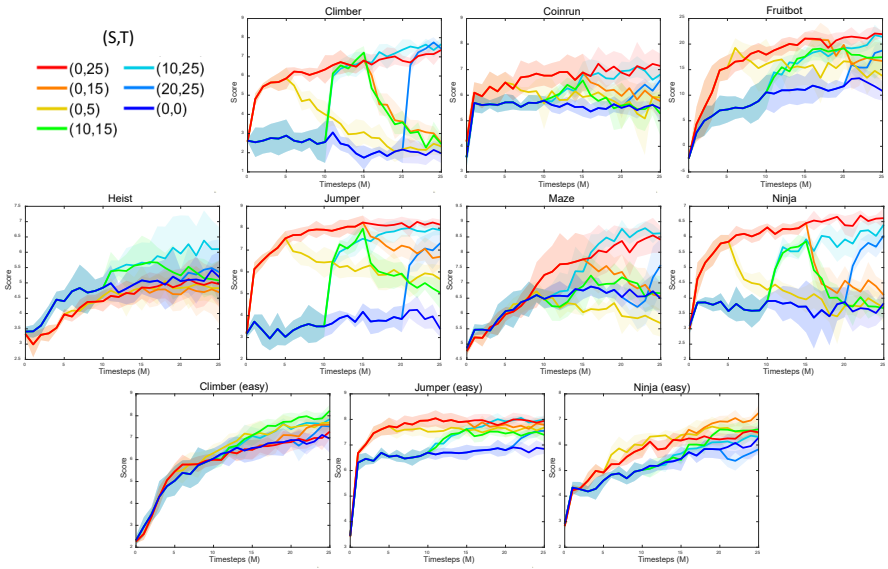


Figure 18: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*random convolution*): Most cases' tendencies are coincidence with the jumper, which is mentioned in the main paper.

J.2 Crop

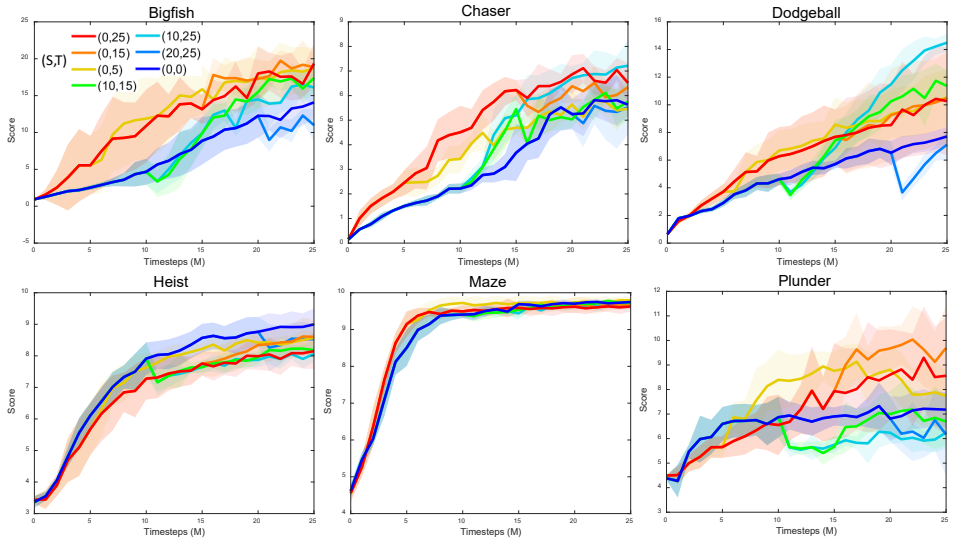


Figure 19: Comparison of training performance according to usage period of augmentation with InDA (*crop*): *Crop* improve the training performance in Bigfish, Chaser, Dodgeball, Plunder. Furthermore, interrupted augmentation is also improved similarly with (0, 25).

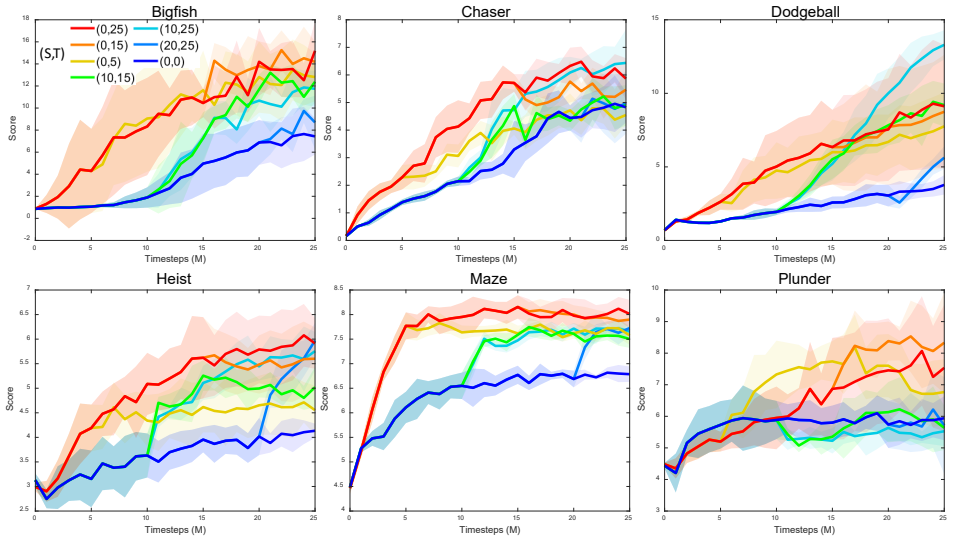


Figure 20: Comparison of generalization on unseen levels according to usage period of augmentation with InDA (*crop*): The generalization is improved by *crop*, and it is conserved after interrupted in Heist and Maze. Bigfish, Chaser, Dodgeball, and Plunder have similar curves with training.

J.3 Color jitter

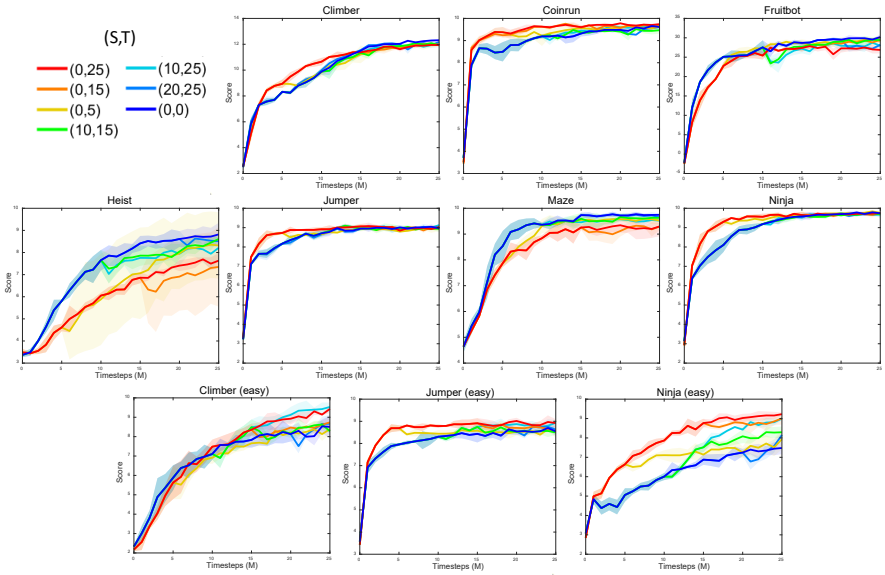


Figure 21: Comparison of training performance according to usage period of augmentation with InDA (*color jitter*): *Color jitter* does not impede the training as much as *random convolution* in most environments. However, *color jitter* helps the training in *easy* mode.

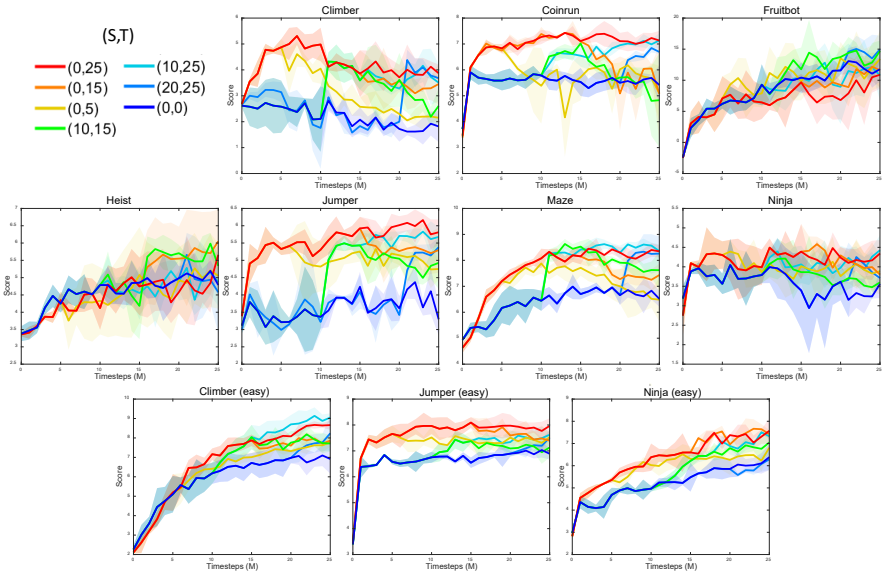


Figure 22: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*color jitter*): Test performance is influenced by *color jitter* as the trend, which is similar to *random convolution*.

K Benchmark on Modified Open AI Procgen

We compare the training and test performance on various environments with each augmentation. We also use DrAC [26], RAD [21], DrAC+PAGrad as baselines. In every result, we train the agent for 25M timesteps, except the ExDA. ExDA is trained with 0.5M after training 20M with PPO. We also compare the average score after normalized by PPO's score and indicate the best score as bold. Mean

and standard deviation is calculated after five runs. We show the result about *random conv, color jitter, random crop* in Table 1. Additionally, we evaluate benchmark with *gray* and *cutout color* in Table 7. For benchmark, we classify Procgen environments with each characteristic as Figure 23. Furthermore, we attach detail results on each environments with Oracle and Rand-FM [22]. Red one is the Oracle score, which is trained on test environments such as *easybg-test, easy-test*. For your information, RAD does not work well when using crop, because we use [26]'s crop method which is different with [21].

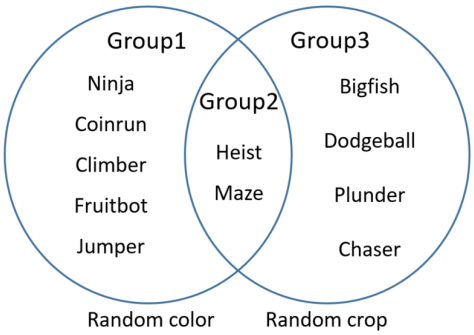


Figure 23: Group of OpenAI Procgen environments: We classify environments according to characteristics about a type of observation window and the importance of color. For Group1, we do not use random crop because they have agent centered window, thus the *random crop* can make hard to find agent. For Group3, we do not use random color, because they have important meaning in the color of objects, thus the transformation of color would make hard to learn information in color. Group2 can apply both of the transformations.

Augmentation	Task	PPO	RAD	DrAC	DrAC+PAGrad	InDA	ExDA
Grayscale	Train	**1.00**	0.94	0.93	0.94	0.95	0.99
	Test-bg	1.00	1.04	1.03	1.04	0.97	**1.13**
	Test-lv	**1.00**	0.81	0.84	0.84	0.84	0.86
Cutout color	Train	**1.00**	0.72	0.82	0.84	0.76	0.94
	Test-bg	1.00	1.33	1.27	1.29	1.19	**1.53**
	Test-lv	**1.00**	0.69	0.83	0.83	0.69	0.93

Table 7: Train and test score of InDA and ExDA on Open AI Procgen, compared to baselines PPO, Drac [26], RAD [21], DrAC+PAGrad. **Boldface** indicates the best method.

K.1 Random convolution

Easy	PPO	DrAC	Rand_FM	RAD	InDA	ExDA
Climber	**8.63** ±0.462	8.33 ±0.407	8.27 ±0.187	7.93 ±0.37	8.5 ±0.291	8.1 ±0.268
Jumper	8.55 ±0.168	8.62 ±0.075	8.47 ±0.13	8.51 ±0.102	**8.94** ±0.09	8.5 ±0.183
Ninja	7.49 ±0.421	8.57 ±0.069	7.69 ±0.529	7.9 ±0.652	**8.88** ±0.343	7.03 ±0.058
Avg	1.00	1.04	0.99	0.99	1.07	0.96

Table 8: Training performance benchmark on *easy* with *random convolution*.

Easy	PPO	DrAC	Rand_FM	RAD	InDA	ExDA
Climber	6.96 ±0.651	7.21 ±0.447	6.63 ±0.39	6.08 ±0.264	**7.28** ±0.341	7.06 ±0.541
Jumper	6.85 ±0.192	**7.97** ±0.128	6.7 ±0.167	6.74 ±0.299	7.94 ±0.185	7.54 ±0.158
Ninja	6.29 ±0.529	6.18 ±0.193	6.22 ±0.57	6.22 ±0.324	**6.5** ±0.191	5.56 ±0.158
Avg	1.00	1.06	0.97	0.95	**1.08**	1

Table 9: Test performance benchmark on unseen backgrounds (*easy, random convolution*).

Easybg	PPO	Oracle	DrAC	Rand-FM	RAD	InDA	ExDA
Climber	**12.35** ±0.083	9.78 ±0.306	11.23 ±0.353	12.2 ±0.128	12.15 ±0.09	10.89 ±0.162	12.07 ±0.073
Coinrun	**9.64** ±0.07	7.11 ±0.205	9.17 ±0.161	9.57 ±0.126	9.56 ±0.107	8.81 ±0.992	9.44 ±0.149
Fruitbot	29.78 ±0.899	29.74 ±0.443	26.07 ±0.658	**30.19** ±0.512	29.92 ±0.623	26.17 ±0.575	28.76 ±0.79
Heist	**9** ±0.513	7.21 ±0.27	5.95 ±0.343	7.7 ±0.6	7.94 ±0.919	5.15 ±0.614	8.72 ±0.533
Jumper	8.95 ±0.066	8.72 ±0.119	8.86 ±0.088	8.91 ±0.13	**9.04** ±0.135	8.78 ±0.172	8.94 ±0.048
Maze	**9.75** ±0.513	8.56 ±0.27	8.1 ±0.343	9.61 ±0.6	9.51 ±0.919	9.12 ±0.614	9.73 ±0.533
Ninja	9.75 ±0.073	7.81 ±0.422	9.43 ±0.109	9.75 ±0.084	**9.78** ±0.03	9.53 ±0.113	9.7 ±0.062
Avg	**1.00**	0.85	0.98	0.98	0.88	0.88	0.98

Table 10: Training performance benchmark on *easybg* with *random convolution*.

Easybg	PPO	Oracle	DrAC	Rand-FM	RAD	InDA	ExDA
Climber	1.97 ±0.51	9.78 ±0.306	7.13 ±0.419	2 ±0.59	2.34 ±1.258	7.36 ±0.273	**8.11** ±0.457
Coinrun	5.48 ±0.583	7.11 ±0.205	7.54 ±0.188	5.65 ±0.216	5.48 ±0.542	7.14 ±0.479	**7.81** ±0.388
Fruitbot	10.83 ±1.908	29.74 ±0.443	19.77 ±0.77	15.19 ±3.363	11.61 ±4.615	21.93 ±0.664	**23.57** ±0.745
Heist	5.18 ±0.838	7.21 ±0.27	5.47 ±0.326	5.03 ±0.6	4.78 ±0.785	4.96 ±0.777	**8.15** ±0.633
Jumper	3.38 ±0.368	8.72 ±0.119	8.14 ±0.17	4.12 ±0.514	3.77 ±0.435	**8.16** ±0.231	7.87 ±0.485
Maze	6.48 ±0.523	8.56 ±0.665	6.4 ±0.419	6.6 ±0.494	6.29 ±0.466	8.41 ±0.436	**8.92** ±0.155
Ninja	3.83 ±0.462	7.81 ±0.422	6.8 ±0.243	3.36 ±0.505	3.98 ±0.44	6.61 ±0.327	**6.85** ±0.25
Avg	1.00	2.33	1.86	1.08	1.04	1.92	**2.11**

Table 11: Test performance benchmark on unseen backgrounds (*easybg, random convolution*).

K.2 Crop

Easybg	PPO	DrAC	RAD	InDA	ExDA
Bigfish	14.08 ±2.229	15.92 ±1.535	5.05 ±3.718	**19.35** ±2.792	11.07 ±3.683
Chaser	5.63 ±0.467	3.97 ±0.642	1.24 ±0.253	**6.52** ±0.825	4.81 ±0.325
Dodgeball	7.71 ±0.678	10.74 ±0.711	1.23 ±0.944	**12.74** ±1.729	6.74 ±0.815
Heist	**9** ±0.513	7.58 ±0.11	4.53 ±0.266	8.15 ±0.57	8.79 ±0.424
Maze	**9.75** ±0.033	9.03 ±0.348	3.95 ±3.418	9.63 ±0.143	9.72 ±0.026
Plunder	7.18 ±0.73	10.73 ±1	0 ±0	**10.29** ±0.285	6.59 ±1.108
Avg	1.00	1.08	0.28	**1.25**	0.91

Table 12: Training performance benchmark on *easybg* with *crop*.

Easybg	PPO	DrAC	RAD	InDA	ExDA
Bigfish	7.43 ±1.65	13.63 ±1.504	4.93 ±3.696	**15.19** ±2.724	6.35 ±2.466
Chaser	4.83 ±0.56	3.59 ±0.519	1.2 ±0.259	**5.86** ±0.745	4.48 ±0.379
Dodgeball	3.78 ±0.659	9.26 ±0.685	1.11 ±0.831	**11.92** ±1.556	3.79 ±0.748
Heist	**4.13** ±0.146	5.4 ±0.448	3.81 ±0.412	**5.91** ±0.516	5.35 ±0.22
Maze	**6.79** ±0.158	7.77 ±0.328	3.9 ±3.377	**8.01** ±0.288	**7.74** ±0.054
Plunder	5.94 ±0.698	**9.49** ±0.605	0 ±0	**8.98** ±0.369	5.98 ±0.944
Avg	1.00	1.519	0.459	**1.798**	1.094

Table 13: Test performance benchmark on unseen levels (*easybg, crop*).

K.3 Color jitter

Easy	PPO	DrAC	RAD	InDA	ExDA
Climber	8.5 ±0.575	**9.33** ±0.212	8.64 ±0.156	9.43 ±0.21	8.18 ±0.45
Jumper	8.54 ±0.22	8.64 ±0.135	8.63 ±0.17	**8.92** ±0.174	8.44 ±0.185
Ninja	7.48 ±0.324	8.69 ±0.331	8.24 ±0.251	**9.23** ±0.081	7.37 ±0.212
Avg	1.00	1.09	1.04	**1.13**	0.98

Table 14: Training performance benchmark on *easy* with *color jitter*.

Easy	PPO	DrAC	RAD	InDA	ExDA
Climber	6.92 ±0.761	8.53 ±0.422	8.37 ±0.023	**8.66** ±0.24	8.14 ±0.477
Jumper	6.89 ±0.223	7.58 ±0.053	7.86 ±0.297	**7.97** ±0.292	7.25 ±0.131
Ninja	6.39 ±0.585	6.79 ±0.32	7.31 ±0.613	**7.57** ±0.555	6.2 ±0.085
Avg	1.00	1.13	1.16	**1.2**	1.07

Table 15: Test performance benchmark on unseen backgrounds (*easy, color jitter*).

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.31** ±0.092	9.85 ±0.298	11.84 ±0.223	12 ±0.256	11.94 ±0.071	12.04 ±0.152
Coinrun	9.61 ±0.074	7.2 ±0.195	8.94 ±0.285	8.62 ±0.091	**9.74** ±0.05	9.45 ±0.09
Fruitbot	30.2 ±0.691	29.69 ±0.619	**30.05** ±0.611	29.48 ±0.507	26.87 ±0.912	29 ±0.878
Heist	**8.82** ±0.523	7.33 ±0.308	7.22 ±0.76	6.89 ±0.348	7.63 ±0.338	8.53 ±0.307
Jumper	8.97 ±0.075	8.67 ±0.132	8.9 ±0.05	8.94 ±0.029	**9.03** ±0.123	**9.03** ±0.086
Maze	**9.75** ±0.035	8.08 ±0.101	9.46 ±0.404	9.46 ±0.184	9.3 ±0.379	9.67 ±0.111
Ninja	9.74 ±0.087	7.56 ±0.286	9.52 ±0.393	9.65 ±0.112	**9.75** ±0.046	9.54 ±0.171
Avg	**1.00**	0.85	0.95	0.94	0.96	0.98

Table 16: Training performance benchmark on *easbg* with *color jitter*.

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	1.82 ±0.605	9.85 ±0.298	**5.05** ±0.407	4.31 ±0.492	4.25 ±0.338	4.34 ±0.856
Coinrun	5.42 ±0.744	7.2 ±0.195	6.46 0±.526	6.47 ±0.194	**7.13** ±0.372	6.53 ±0.375
Fruitbot	11.78 ±1.949	29.69 ±0.619	9.49 ±8.098	8.51 ±1.941	10.88 ±2.263	**18** ±7.442
Heist	4.79 ±0.323	7.33 ±0.308	5.65 ±0.984	5.39 ±0.745	**5.66** ±0.271	5.43 ±0.508
Jumper	3.3 ±0.467	8.6 ±0.132	5.65 ±0.09	5.67 ±0.953	**5.81** ±0.369	5.31 ±0.351
Maze	6.52 ±0.304	8.08 ±0.101	8.22 ±0.455	8.26 ±0.175	8.35 ±0.238	**8.65** ±0.017
Ninja	3.56 ±0.363	7.56 ±0.286	4.22 ±0.487	4.18 ±0.475	**4.34** ±0.345	4.07 ±0.332
Avg	1.00	2.4	1.44	1.37	1.43	**1.48**

Table 17: Test performance benchmark on unseen backgrounds (*easybg, color jitter*).

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.31** ±0.092	9.85 ±0.298	11.12 ±0.26	11.84 ±0.505	11.9 ±0.115	12.06 ±0.03
Coinrun	9.61 ±0.074	7.2 ±0.195	9.53 ±0.135	9.49 ±0.188	**9.74** ±0.046	9.48 ±0.08
Fruitbot	**30.2** ±0.691	29.69 ±0.619	30.01 ±0.572	29.6 ±0.27	28.03 ±0.994	29.32 ±0.937
Heist	**8.82** ±0.523	7.33 ±0.308	6.24 ±0.214	6.53 ±0.474	5.51 ±0.146	8.51 ±0.225
Jumper	8.54 ±0.22	8.67 ±0.132	8.91 ±0.19	8.93 ±.247	**9.18** ±0.18	8.95 ±0.075
Maze	**9.75** ±0.035	8.08 ±0.101	9.46 ±0.192	9.48 ±0.08	9.2 ±0.367	**9.75** ±0.087
Ninja	**9.74** ±0.087	7.56 ±0.286	9.73 ±0.045	9.61 ±0.096	9.6 ±0.081	9.72 ±0.021
Avg	**1**	0.85	0.93	0.94	0.95	0.99

Table 18: Training performance benchmark on *easybg* with *gray*.

K.4 Gray

Easybg	PPO	Oracle	RAD	DrAC	InDA	ExDA
Climber	1.82 ±0.605	9.85 ±0.298	1.75 ±0.654	1.81 ±0.211	1.24 ±0.502	**2.45** ±0.727
Coinrun	5.42 ±0.744	7.2 ±0.195	5.34 ±0.751	5.31 ±0.501	**6.05** ±0.465	5.79 ±0.061
Fruitbot	11.78 ±1.949	29.69 ±0.619	**17.57** ±0.191	15.47 ±1.449	15.12 ±0.958	15.81 ±0.11
Heist	4.79 ±0.323	7.33 ±0.308	**5.43** ±0.18	5.15 ±0.172	4.32 ±0.112	5.1 ±0.504
Jumper	**6.89** ±0.223	8.67 ±0.132	2.7 ±0.894	4.07 ±0.46	3.55 ±0.992	4.47 ±0.415
Maze	6.52 ±0.304	8.08 ±0.101	7.77 ±0.611	7.93 ±0.104	7.67 ±0.312	**8.33** ±0.119
Ninja	3.56 ±0.363	7.56 ±0.286	3.72 ±0.131	3.91 ±0.62	4.02 ±0.666	**4.03** ±0.071
Avg	1	2.2	1.03	1.04	0.97	**1.13**

Table 19: Test performance benchmark on unseen backgrounds (*easybg, gray*).

Easy	PPO	DrAC	RAD	InDA	ExDA
Climber	**8.5** ±0.575	6.95 ±0.547	7.55 ±0.256	7.22 ±0.312	8.05 ±0.461
Jumper	8.54 ±0.22	8.4 ±0.224	8.58 ±0.199	**8.85** ±0.015	8.5 ±0.224
Ninja	7.48 ±0.324	6.67 ±0.435	7.1 ±0.718	**8.91** ±0.165	7.05 ±0.24
Avg	1	0.9	0.95	**1.026**	0.96

Table 20: Training performance benchmark on *easybg* with *gray*.

Easy	PPO	DrAC	RAD	InDA	ExDA
Climber	6.92 ±0.761	4.49 ±0.332	5.57 ±0.307	5.11 ±0.483	**7.24** ±0.721
Jumper	**6.89** ±0.223	5.38 ±0.215	6.59 ±0.055	6.35 ±0.234	6.87 ±0.182
Ninja	6.39 ±0.585	5.67 ±0.318	5.14 ±0.628	**6.84** ±0.206	6.01 ±0.651
Avg	**1**	0.77	0.86	0.91	0.99

Table 21: Test performance benchmark on unseen backgrounds (*easy, gray*).

K.5 Cutout color

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.31** ±0.092	9.85 ±0.298	11.92 ±0.158	8.26 ±0.663	11.76 ±0.027	12.07 ±0.127
Coinrun	9.61 ±0.074	7.2 ±0.195	9.23 ±0.323	8.07 ±0.645	**9.7** ±0.084	9.39 ±0.012
Fruitbot	**30.2** ±0.691	29.69 ±0.619	29.73 ±0.898	29.2 ±0.64	27.18 ±1.302	28.95 ±0.907
Heist	**8.82** ±0.523	7.33 ±0.308	8.47 ±0.397	6.25 ±0.704	6.1 ±0.693	8.65 ±0.21
Jumper	8.97 ±0.075	8.67 ±0.132	8.87 ±0.123	8.75 ±0.131	**9.1** ±0.081	8.91 ±0.053
Maze	**9.75** ±0.035	8.08 ±0.101	9.41 ±0.134	9.17 ±0.118	9.27 ±0.125	9.74 ±0.133
Ninja	**9.74** ±0.087	7.56 ±0.286	9.65 ±0.138	7.17 ±1.993	9.72 ±0.02	9.7 ±0.02
Bigfish	**13.89** ±3.127	13.22 ±1.488	2.54 ±0.13	5.19 ±3.658	1.95 ±0.311	11.22 ±3.66
Chaser	**5.49** ±0.562	3.04 ±0.183	2.88 ±0.699	1.98 ±0.112	3.34 ±0.755	5 ±0.187
Dodgeball	**7.76** ±0.859	5.74 ±1.118	5.71 ±1.008	5.98 ±0.103	2.79 ±1.612	6.57 ±0.693
Plunder	**7.15** ±0.95	6.05 ±0.58	5.43 ±0.082	4.34 ±0.24	4.92 ±0.625	6.87 ±1.255
Avg	**1.00**	0.82	0.82	0.72	0.76	0.94

Table 22: Training performance benchmark on *easybg* with *cutout color*.

Easy	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	8.5 ±0.575	9.85 ±0.298	7.69 ±0.237	6.67 ±0.381	**9.02** ±0.473	8.02 ±0.506
Jumper	7.48 ±0.324	7.56 ±0.286	6.28 ±0.257	5.6 ±0.276	**8.57** ±0.122	7.41 ±0.125
Ninja	8.54 ±0.22	8.67 ±0.132	8.45 ±0.183	8.32 ±0.051	**8.93** ±0.166	8.53 ±0.095
Avg	1.00	1.06	0.91	0.84	**1.08**	0.98

Table 23: Training performance benchmark on *easy* with *cutout color*.

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	1.82 ±0.605	9.85 ±0.298	3.54 ±0.164	3.97 ±0.999	3.4 ±0.645	**4.29** ±0.154
Coinrun	5.42 ±0.744	7.2 ±0.195	5.87 ±.251	5.93 ±0.061	6.2 ±0.357	**6.41** ±0.131
Fruitbot	11.78 ±1.949	29.69 ±0.619	18.18 ±3.744	**19.24** ±3.385	17.69 ±4.026	17.7 ±0.888
Heist	4.79 ±0.323	7.33 ±0.308	6.6 ±.092	5.76 ±0.551	4.97 ±0.33	**7.51** ±0.119
Jumper	3.3 ±0.467	8.67 ±0.132	4.99 ±0.114	5.48 ±0.28	5.43 ±1.116	**6.02** ±0.235
Maze	6.52 ±0.304	8.08 ±0.101	7.33 ±0.223	7.66 ±.243	7.01 ±0.17	**7.83** ±0.22
Ninja	3.56 ±0.363	7.56 ±0.286	**4.29** ±0.245	3.96 ±0.152	3.75 ±0.333	3.76 ±0.348
Bigfish	3.4 ±0.487	13.22 ±1.488	1.29 ±0.08	2.5 ±2.331	1.29 ±0.152	**4.49** ±0.776
Chaser	0.91 ±0.061	3.04 ±0.183	1.08 ±0.038	1.13 ±0.157	1.68 ±0.305	**1.73** ±0.698
Dodgeball	2.17 ±0.53	5.74 ±1.118	3.92 ±0.53	4.02 ±0.345	1.97 ±1.098	**4.37** ±0.527
Plunder	**6.87** ±0.933	6.05 ±0.58	5.27 ±0.208	4.77 ±0.612	4.71 ±0.622	6.45 ±1.232
Avg	1.00	2.51	1.27	1.33	1.19	**1.53**

Table 24: Test performance benchmark on unseen backgrounds (*easybg, cutout color*).

Easy	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	6.92 ±0.761	9.85 ±0.298	6.54 ±0.213	5.24 ±0.417	**7.61** ±0.486	7.25 ±0.325
Jumper	6.39 ±0.585	7.56 ±0.286	5.06 ±0.137	4.9 ±0.382	**6.71** ±0.352	5.78 ±0.488
Ninja	6.89 ±0.223	8.67 ±0.132	6.88 ±0.083	6.79 ±0.278	6.81 ±0.355	**6.92** ±0.212
Avg	1.00	1.29	0.91	0.84	**1.05**	0.99

Table 25: Test performance benchmark on unseen backgrounds (*easy, cutout color*).

Easybg	PPO	DrAC	RAD	InDA	ExDA
Climber	**11.14** ±0.077	10.77 ±0.279	7.26 ±0.843	9.45 ±0.193	10.75 ±0.114
Coinrun	**8.64** ±0.05	8.36 ±0.348	6.89 ±0.503	7.76 ±0.096	8.32 ±0.224
Fruitbot	**28.26** ±0.461	26.88 ±1.276	26.22 ±1.258	23.79 ±0.971	26.33 ±0.894
Heist	**4.07** ±0.07	3.92 ±0.276	2.27 ±0.448	2.15 ±0.553	3.93 ±0.184
Jumper	**7.38** ±0.15	7.32 ±0.195	6.98 ±0.199	6.68 ±0.24	7.25 ±0.117
Maze	6.8 ±0.2	**6.84** ±0.137	6.04 ±0.258	5.91 ±0.03	6.17 ±0.162
Ninja	8.56 ±0.061	**8.63** ±0.132	6.28 ±1.866	7.81 ±0.21	8.34 ±0.119
Bigfish	**7.16** ±2.263	0.91 ±0.037	2.29 ±2.306	0.95 ±0.06	6.04 ±2.783
Chaser	**4.54** ±0.503	2.61 ±0.509	1.8 ±0.12	2.47 ±0.506	4.22 ±0.331
Dodgeball	**3.78** ±0.823	2.71 ±0.362	2.53 ±0.135	1.26 ±0.48	2.82 ±0.593
Plunder	**5.99** ±0.814	5.08 ±0.305	4.07 ±0.479	4.55 ±0.393	5.83 ±1.061
Avg	**1.00**	0.83	0.69	0.69	0.93

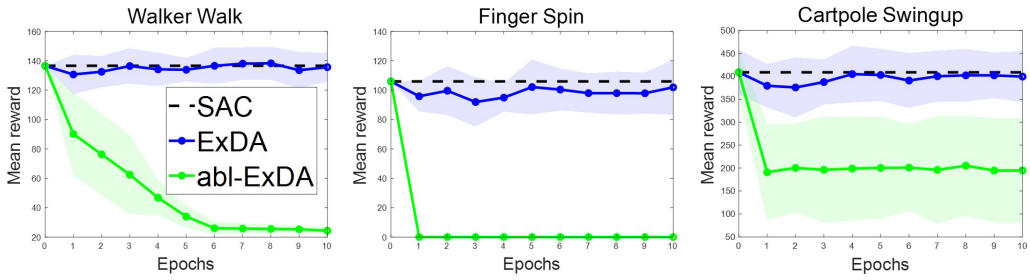
Table 26: Test performance benchmark on unseen levels (*easybg, cutout color*).

Easy	PPO	DrAC	RAD	InDA	ExDA
Climber	5.45 ±0.77	**5.9** ±0.352	5.3 ±0.307	4.26 ±0.122	5.71 ±0.303
Jumper	5.81 ±0.227	**6.01** ±0.389	4.93 ±0.08	4.56 ±0.161	5.43 ±0.333
Ninja	5.77 ±0.09	5.67 ±0.023	5.8 ±0.071	5.65 ±0.166	**5.87** ±0.079
Avg	1.00	**1.03**	0.94	0.85	1

Table 27: Test performance benchmark on unseen levels (*easy, cutout color*).

L Primitive evaluation on DeepMind Control Suite with SAC

We experiment on DeepMind Control Suite (DMC) with a preliminary experiment which provides justification for our proposed method ExDA due to the limited time and computation resources. We first note that [26] has demonstrated that UCB-DrAC, which is similar to UCB-InDA, can accelerate RL training in DMC. Hence, in order to justify our proposed method UCB-ExDA, which combines UCB-InDA and ExDA, it would be sufficient to show the existence of the cases showing a benefit of ExDA in DMC. Furthermore, we use SAC as the base RL algorithm instead of PPO to show the versatility of our method ExDA. SAC is off-policy actor-critic RL algorithm which trains actor and critic networks separately with a replay buffer. Thus, we can use stored data in the replay buffer after RL training, and also we can only distill the output of the actor-network, because the actor-network determines the policy π from observations, while the critic-network computes the value function V from observations. We denote the actor-network parameters using θ same as the notation of PPO in Section 4. We ablate the effect of ExDA to compare w/ and w/o inconsistency loss of policy on original observations. Thus ExDA is trained with Eq 5 after SAC, but ablated ExDA is trained with Eq 11.



(a) (b) (c)

Figure 24: Ablation of ExDA on DMcontrol

$$L_{abl-\text{ExDA}}(\theta, \phi; \theta_{\text{old}}) := L_{\text{dis}}(\theta, \phi; \theta_{\text{old}}) \ . \tag{11}$$

Both methods are trained 10 epochs after 10K time-steps SAC training and detailed hyper-parameters are described below the table. We compare the mean reward of each method SAC, ExDA, abl-ExDA on three environments *Walker-Walk, Finger-Spin, Cartpole-Swingup* and 5 seeds. We evaluate each reward from 50 evaluation episodes in train environments. As below Figure 24, ExDA almost maintains the mean reward of SAC, abl-ExDA degrades the performance of SAC by distilling policy only to augmented observations except the original.

Hyperparameter	Value
# of action repeat	8
# of frame stack	3
Data augmentation	Random Convolution
Learning rate of DA	10^{-2}
Batch size	128
# of train steps	100000
# of distill epochs	10
Replay buffer capacity	100000
Init steps	1000
Learning rate of critic	10^{-3}
β of critic	0.9
τ of critic	0.01
Target update frequency of critic	2
Learning rate of actor	10^{-3}
β of actor	0.9
Log std min of actor	-10
Log std max of actor	2
Update frequency of actor	2
Encoder type	pixel
Feature dimension of encoder	50
Learning rate of encoder	10^{-3}
τ of encoder	0.05
# of layers	4
# of filters	32
Latent dimension	128
Discount factor	0.99
Learning rate of α	10^{-4}
β of α	0.5