Adaptive Scheduling of Data Augmentation for Deep Reinforcement Learning

Byungchan Ko
GSAI
POSTECH
Pohang-si, Republic of Korea
kbc6723@postech.ac.kr

Jungseul Ok
CSE / GSAI
POSTECH
Pohang-si, Republic of Korea
jungseul@postech.ac.kr

Abstract

We consider data augmentation technique to improve data efficiency and generalization performance of reinforcement learning (RL). Our empirical study on Open AI Procgen shows that the timing of augmentation is critical, and that to maximize test performance, an augmentation should be applied either during the entire RL training, or after the end of RL training. More specifically, if the regularization imposed by augmentation is helpful only in testing, then augmentation is best used after training than during training, because augmentation often disturbs the training process. Conversely, an augmentation that provides regularization that is useful in training should be used during the whole training period to fully utilize its benefit in terms of both generalization and data efficiency. Considering our findings, we propose a mechanism to fully exploit a set of augmentations, which automatically identifies the best augmentation (or no augmentation) in terms of RL training performance, and then utilizes all the augmentations by network distillation after training to maximize test performance. Our experiment empirically justifies the proposed method compared to other automatic augmentation mechanism.

1 Introduction

Reinforcement Learning (RL) from visual observations is a fundamental problem, because visual data are among the most common form; e.g., video games [23], board games [29, 30] , and robots [33, 17]. However, images are high-dimensional, so RL from vision often suffers from poor sample efficiency and poor generalization capability. due to the high-dimensional nature of images. To overcoming these problems, regularization by data augmentation has been widely considered [19, 18]; in this process, visual data are augmented by transformations that preserve the meaning or context, e.g., by cropping out unimportant parts of images, or by randomizing colors. Transformations resolve the data scarcity, and also provide an explicit implementation of inductive bias for generalization performance.

Use of the appropriate type of data augmentation significantly improves both data efficiency and generalization performance [25]. However, the correct data augmentation scheme is highly task-dependent: a poor choice can degenerate the generalization and destabilize the training [19, 25]. Hence, a variety of transformations have been developed to enlarge the set of augmentation methods [20, 12]. Meanwhile, numerous regularization methods that use data augmentation e.g., self-supervised learning [25] and representation learning [31, 12] have been proposed to stabilize training process with data augmentation by reducing the interference between RL training and regularization. Previous work addressed *what* data augmentation to use and *how* to use it, but the understanding of *when* to apply it in the training process is limited. We test the hypothesis that applying augmentation method at different epochs can have different effects. This is a non-trivial question, because the timing of data

Presented at Deep RL Workshop, NeurIPS 2021.

augmentation is not critical in supervised learning (SL) [1, 10], whereas a curriculum learning can accelerate RL training [26].

To address our main question, we devise two frameworks with different timings of augmentation: **In**tra **D**istillation with **A**ugmented observations (InDA) and **Ex**tra **D**istillation with **A**ugmented observations (ExDA) (Section 3). Implementing the regularization by augmentation in a form of distillation to minimize interference in RL training, InDA interleave the distillation with RL training, whereas ExDA applies the distillation at the end of RL training. From experiments with InDA and ExDA, we find that: *Time does matter when using augmentation in RL* in contrast to the case of SL [10], in which the effect of data augmentation is relatively insensitive to timing. The difference mainly comes from the fact that RL agent collects samples when training, whereas SL uses a fixed data set. To be specific, the main findings from experiments are:

(i) If augmentation can accelerate RL training, then it *must* be applied as *early* as possible for sample efficiency and generalization; e.g., cropping out an unnecessary part of an image induces an efficient attention mechanism. To maximize sample efficiency, RL training must be accelerated from the beginning. However, we observe that this kind of augmentation often connotes generalization that is transferable only by a diverse experience in training process, i.e., InDA fully exploits generalization gain whereas ExDA does not. Hence, to gain generalization, this augmentation must be applied during training.

(ii) If the regularization imposed by augmentation is helpful only in testing, then to ensure sample efficiency, augmentation *must* be *postponed to the end* of RL training; e.g., augmentation by changing colors is useless when the training task shows a single background, but the testing task has multiple backgrounds. This type of augmentation may interfere with RL training, but delay of augmentation does not degrade sample efficiency and increase generalization ability. Hence, in this case, ExDA, which never disturbs RL training, is better than InDA.

(iii) The optimal time to apply augmentation for each task can be *determined* automatically by the upper confidence bound [3] (UCB) based auto augmentation [25] algorithm. We show that the choice of *no augmentation* is necessary, because augmentation can disturb the training. Thus, auto augmentation with a 'no change' ('identity') function can be used as a discriminator to identify the benefit of augmentation during training.

The above findings suggest effective timings of augmentation in RL. Our contribution also includes the InDA and ExDA algorithms, in particular, which are equipped with the distillation augmentation (DA), which address the independent interest in developing a regularization method that uses augmentation with minimal interference with RL training. The potential advantages of the proposed method over existing methods DrAC [25], RAD [19], Rand-FM [20], are discussed in Section 3.

2 Related Works

Augmented experience in RL. To solve the problem of poor generalization and sparse data, a popular approach is to generate diverse (virtual) experiences and let the RL agent learn from them. Domain randomization is a technique to produce such experiences from a simulator of a targeted system [33, 24, 26]. Accurate simulators of practical systems are difficult to obtain, and this problem limits the spectrum of applications. 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 design of an appropriate image transformer. A method of a curriculum learning for domain randomization, in which the difficulty is gradually increasing [26] provided insights that coincide with some of our findings. However, we provide further understanding of the types of visual augmentation that should 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 [12, 32], self-supervised [25], and contrast [31]. One proposed algorithm [25] 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 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 [25], 'not

augmenting' is not an option, whereas our findings indicate that it should be. In addition, [25] does not consider post augmentation followed by RL training, as in ExDA.

Different time-sensitivity of augmentation than SL. During deep learning, the early state of training often has a significant effect [6, 1]. Therefore, we devised time-sensitive methods that adapt to the progress of training, such as learning rate decay [36] and curriculum learning [34]. Golatkar *et al*. [10] 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 [14, 4, 22, 8, 31, 35, 13], a thorough study should be conducted before introducing a method from different learning framework, because we find the contrasting time-sensitivities of data augmentation. This spirit is also shared with an application [15] of implicit bias in SL [11, 2, 9] to RL.

3 Method

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 reward r_t and an image $o_{t+1} = O(s_{t+1}) \in \mathbb{R}^{k \times k}$ as an observation(possibly partial) of the next state s_{t+1}. To augment observations, we consider image transformation function $\phi : \mathbb{R}^{k \times k} \mapsto \mathbb{R}^{k \times k}$ which maintains the dimension.

Baseline RL algorithm. As the baseline deep-RL algorithm, we use Proximal Policy Optimization (PPO) [27] which is an on-policy actor-critic RL algorithm to learn policy $\pi_\theta(a \mid o)$ and value function V_θ with network parameter θ. Storing a set of recent transitions $\tau_t := (o_t, a_t, r_t, o_{t+1})$ in experience buffer $\mathcal{D}$, the network parameter θ is updated to maximize the following objective function:

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

where α is a hyperparameter and some 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 $\tau \sim \mathcal{D}$, θ_{old} is the network parameter before the update, $\rho_t(\theta)$ is the importance ratio $\frac{\pi_\theta(a_t|o_t)}{\pi_{\theta_{\text{old}}}(a_t|o_t)}$, $\hat{A}_t$ is advantage from Generalized Advantage Estimator [27].

Overall framework. We propose two frameworks: **In**tra **D**istillation with **A**ugmented observations (InDA) and **Ex**tra **D**istillation with **A**ugmented observations (ExDA). To be specific, both of them use PPO for RL and the **D**istillation with **A**ugmented observation (DA) (Section 3.1), for regularization, although our frameworks can use other RL algorithms and augmentation-based regularization. InDA (Section 3.2), interleaves PPO and DA, whereas ExDA (Section 3.3), performs PPO first then DA. We design InDA and ExDA to conduct either DA or PPO in each epoch (Figure 1).

3.1 Distillation with Augmented observations (DA)

DA regularizes reinforcement learning by using distillation with data augmentation, in which we train the network to output the same policies and values for given both original and augmented observations. To do so, we fix the network θ_{old} to be distilled and store observation o_t, which is sampled from $\pi_{\theta_{\text{old}}}$, in $\mathcal{D}$. Their augmented observations are represented as $\phi(o_t)$, where $\phi : \mathbb{R}^{n \times n} \mapsto \mathbb{R}^{n \times n}$ is a

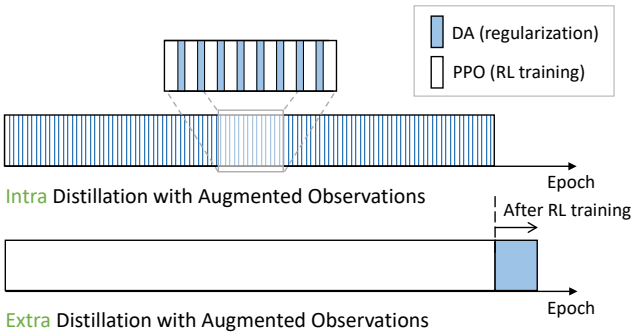


Figure 1: An illustration comparing InDA and ExDA

transformation function. For given $\mathcal{D}$, θ_{old} and ϕ, we then train a network θ to minimize the following distillation loss function:

$$L_{\text{DA}}(\theta) = L_{\text{PD}}(\theta) + L_{\text{VD}}(\theta) \ , \tag{4}$$

where L_{PD} is Kullback–Leibler divergence between policies $\pi_{\theta_{\text{old}}}$ and π_θ and L_{VD} is the mean-squared deviation between value functions $V_{\theta_{\text{old}}}$ and V_θ, i.e.,

$$L_{\text{PD}}(\theta) = \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot|o_t), \pi_\theta(\cdot|o'_t)] \right] \ , \tag{5}$$

$$L_{\text{VD}}(\theta) = \hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[(V_{\theta_{\text{old}}}(o_t) - V_\theta(o'_t))^2 \right] \ . \tag{6}$$

Here o'_t is either the original observation o_t or the augmented one $\phi(o_t)$ with equal probability. The proposed method not only matches the outputs of θ for o_t and $\phi(o_t)$ and also conserves the behavior of θ for o_t to be identical to that of θ_{old} for o_t. This behavior can reduce the interference between reinforcement learning and distillation. Indeed, the performance of RL training can be degraded by distillation without careful consideration on the interference; e.g., [25] with distillation loss of $\hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[\text{KL}[\pi_\theta(\cdot|o_t), \pi_\theta(\cdot|\phi(o_t))] \right]$ and $\hat{\mathbb{E}}_{o_t \sim \mathcal{D}} \left[(V_\theta(o_t) - V_\theta(\phi(o_t)))^2 \right]$ can change the behavior learned in RL training. Separating distillation from RL training provides substantial performance gain (Table 1) compared to other existing methods e.g., [25, 12] (Section 4.1). In addition, the target behaviors ($\pi_{\theta_{\text{old}}}(\cdot|o_t)$ and $V_{\theta_{\text{old}}}(o_t)$), which are used several times during the distillation, are fixed in DA, so we can reduce the computational cost by pre-computing them.

3.2 Intra Distillation with Augmented observations

InDA (Algorithm 1), iteratively optimizes PPO and DA, with PPO and DA explicitly separated. Such a separation reduces their interference [12], whereas they are often optimized simultaneously in other methods [25]. This separation increases the robustness of our algorithm, in addition to the conservative distillation loss functions in (5) and (6). We varied the timing of augmentation by adjusting the time S of starting DA and time T of terminating DA. 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 number RL training rounds n is in the interval of $[S', T']$ (or equivalently, the number of timesteps that have been observed is in the range of $[S, T]$). We provide further details on InDA in supplementary material.

3.3 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. Computational cost can be reduced by replacing L_{DA} with L_{PD} in DA, because value function is not necessary after DA. We check empirically that this reduction does not degrade RL performance. We also consider re-initialization after pre-training, because we expect that diminishing of non-stationarity can improve generalization, as mentioned in [15]. However, training performance is not preserved after re-initialization because $\pi_{\theta_{old}}$ is not completely distilled by low data diversity. Thus, we do not use re-initialization for DA. We leave more interesting details in the supplementary material.

Algorithm 1 InDA

Hyperparameter: N, I, ϕ and (S', T') in rounds (or (S, T) in time steps)
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

Hyperparameter: 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

3.4 Auto Augmentation Discriminator

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 [25], where each augmentation is corresponded to an arm in multi-armed bandit problem and assessed its gain in training with upper confidence bound (UCB) [3]. More formally, the set of arms is the set of image transformations $\Phi = \{\phi_1, \ldots, \phi_k\}$ which includes the identity function also. The inclusion of identity function is an important difference than UCB-DrAC [25] since we observe that using augmentation sometimes needs to be postponed after RL training for the sake of better sampling complexity and test performance. Then, the gain of the augmentation $G(s)$ at the sth sampling is the average return during Interval I, where the augmentation is injected via InDA rather than the distillation method in UCB-DrAC [25]. The return is the sum of estimated advantage $\hat{A}$ and predicted value V_θ. The general UCB algorithm uses a mean of rewards from the entire sampling process, but in RL, the distribution of return is non-stationary [25], so we use the window-average gain $\bar{G}_\phi(s)$ as a reward of each transformation ϕ. Hence, inspired by UCB1 algorithm [3], UCB-InDA selects actions each time using the sum of a window average gain $\bar{G}_\phi(s)$ and a degree of exploration:

$$\phi_t = \underset{\phi \in \Phi}{\arg\max} \left[\bar{G}_\phi(s) + c\sqrt{\frac{\log(s)}{N_\phi(s)}} \right] \tag{7}$$

where c is the UCB exploration coefficient and $N_\phi(s)$ is the selected number of each augmentation after the sth sample. We find c with in adaptive manner, because the appropriate c is different for each training as a result of drastic change of return during the transient time (details in supplementary material). We remark that compared to UCB-DrAC [25], 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 augmentation via InDA. The gain of each component is numerically studied in Section 4

4 Experiment

Setups. We evaluate the time-sensitivity of applying augmentation on the OpenAI Procgen benchmark of 16 games, [5], where at each time t, visual observation o_t is given as an image of size 64×64, and contains full or partial information on the system state. A training or testing environment is defined as a pair of game and mode, where mode determines a set of levels and backgrounds shown in the environment. As the training environment, we use one of two modes: *easy* and *easybg*. Easy mode provided by Cobbe *et al.* [5] contains a set of 200 levels, where an agent can learn basic dynamics of game and experience various backgrounds. To see clear advantage from visual augmentation, we further easicate easy mode and devise easybg mode of which only difference from easy mode is showing only a single background. To evaluate generalization capabilities, we use two modes: *test-bg* and *test-lv*, which contain unseen backgrounds and levels, respectively, in addition to the mode that we use for training. The details of modes in our evaluation is provided in supplementary material.

For clarity, we mainly focus on two visual augmentations, each of which has clearly distinguishing inductive bias:

(a) *Random convolution* transforms an image by passing a single convolutional layer initialized randomly [20]. Augmentation with this can impose invariant behavior on color changes, and thus is anticipated to provide strong generalization on background changes.

(b) *Crop* leaves a randomly-selected rectangle and zero-pads the rest of the image [25]. This augmentation is particularly useful in the fully-observable scenarios, because it imposes an efficient attention mechanism.

We also report the result with other visual augmentations including *color jitter, gray and cutout color* in the supplementary material, where the same main messages can be found. All results in the main paper are averages over five runs.

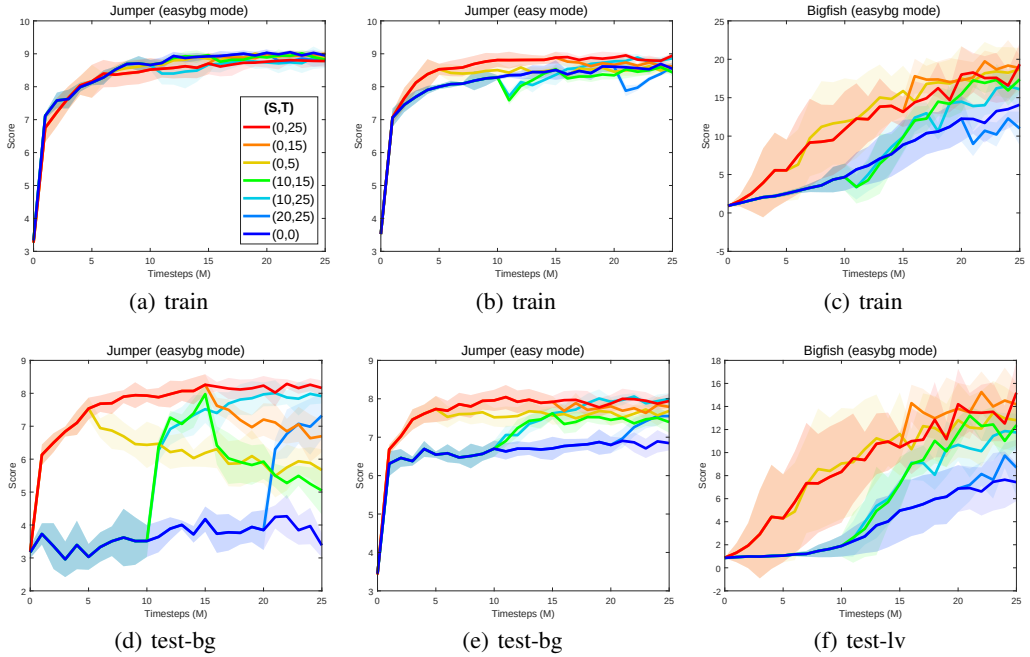
Augmentation		PPO	Oracle	DrAC	RAD	Rand-FM	InDA	ExDA
Rand conv	Train	1.00	0.85	0.88	**0.98**	0.88	0.88	**0.98**
	Test-bg	1.00	2.33	1.86	1.08	1.04	1.92	**2.11**
Color jitter	Train	1.00	0.85	0.95	0.94	-	0.96	**0.98**
	Test-bg	1.00	2.33	1.44	1.37	-	1.43	**1.48**
Grayscale	Train	1.00	0.85	0.93	0.94	-	0.95	**0.99**
	Test-bg	1.00	2.33	1.03	1.04	-	0.97	**1.13**
Cutout color	Train	1.00	0.82	0.82	0.72	-	0.76	**0.94**
	Test-bg	1.00	2.51	1.27	1.33	-	1.19	**1.53**
	Test-lv	1.00	-	0.83	0.69	-	0.69	**0.93**
Crop	Train	1.00	-	1.08	0.28	-	**1.25**	0.91
	Test-lv	1.00	-	1.52	0.46	-	**1.80**	1.09

Table 1: Train and test score of InDA and ExDA on Open AI Procgen, compared to baselines PPO, Oracle, Drac [25], RAD [19], Rand-FM [20]. Oracle is trained with test backgrounds. **Boldface** indicates the best method, and red indicates Oracle. ExDA outperforms other baselines except when we use *crop*, which is evaluated on unseen levels.

4.1 Improving generalization on Procgen

We compare the train and test performances of InDA and ExDA with those of several baselines (Table 1). Every method are trained on 200 levels, using *easybg* mode, which contains a single background. InDA and other baselines are trained for 25M time steps. ExDA is trained with PPO for 20M time steps, then distills for 30 epochs with 0.5M time steps, after training with PPO for 20M time steps. *Test-bg* is used for *random convolution, color jitter, grayscale* and *cutout color*, which give information of color diversity. *Test-lv* is used for *cutout color* and *crop*, which give a consistency of partial observation. Further details about implementation and hyperparameter are described in the supplementary material. ExDA outperforms other baselines with most augmentation

on *test-bg*. Especially, ExDA with *random convolution* has a comparable performance to Oracle, despite being trained on a single background. Moreover, ExDA consumes only 0.5M time steps to inject knowledge from augmented images, whereas the others use all of the training data. We describe the computation issue about both ExDA and InDA in the supplementary material. However, InDA have a better train and test performance with *crop* on *test-lv*. These results demonstrate that each combination of environment and augmentation has a suitable time at which to apply augmentation. Thus, we analyze the proper condition to use InDA or ExDA in the next section.



(a) train (b) train (c) train

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

Figure 2: Comparison according to timing of augmentation with InDA: (S, T) are hyperparameters of InDA for the start and terminal time step of distillation. As an augmentation method, InDA uses random convolution on Jumper, and uses crop on Bigfish. Test results are evaluated on test-bg (unseen backgrounds) and test-lv (unseen levels). Shaded region: standard deviation of five runs. The difference between Jumper (easybg) and Jumper (easy) shows that the benefits of data efficiency and the maintenance of generalization can be changed by the diversity of factors in observations. Augmentation can accelerate the training, such as Bigfish. Furthermore, delayed augmentation commonly improves generalization as much as fully-used cases.

4.2 Time dependency of augmentation in RL

In this section, we study *when* and *how* the agent is particularly helped by augmentation during RL training. For this purpose, we varied DA time (S, T) of InDA to see how generalization's effect depends on the time at which augmentation is used. We evaluate InDA with seven different pairs of start and terminal time of distillation (S, T), where the number of entire timesteps used is 25M: (0, 25), (0, 5), (0, 15), (10, 0), (20, 0), (10, 15), (0, 0). Note that InDA with (S, T) = (0, 0) means RL training only without augmentation, i.e., vanilla PPO. We explain with Jumper and Bigfish in the main paper, and experiments on other environments are described in the supplementary material. Figure 2 represents three environments, Jumper with easybg (Figure 2(a), 2(d)) and easy (Figure 2(b), 2(e)) mode and Bigfish (Figure 2(c), 2(f)) with easybg mode. In the following, we call the curve using a parameter (S,T).

Interrupted augmentation . To determine how generalization would change after regularization stopped, we stop the DA during training, such as (0, 5), (0, 15). Jumper with easybg mode rapidly lost generalization performance (after interruption at both (0, 5) and (0, 15)) (Figure 2(d)), whereas Jumper with easy mode do not (Figure 2(e)). InDA, which uses augmentation throughout training, performs better than PPO during training (Figure 2(b)), but augmentation does not improve the

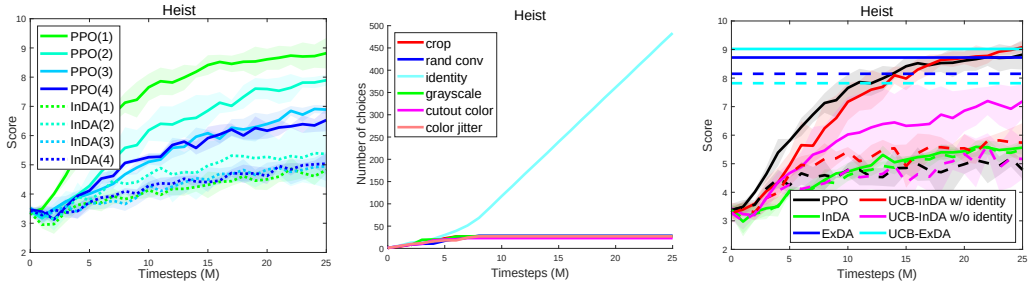
training performance (Figure 2(a)). These results mean that the random convolution alleviates the difficulty by various backgrounds.

On the contrary, random convolution can induce a growing difficulty by increasing the number of factors on a single background. Therefore, the generalization rapidly decreases after augmentation is interrupted during training with a single background because the learning direction toward generalization about various backgrounds is not helpful to train. In contrast, the training can help when their difficulty is solved by augmentation (Figure 2(b), 2(c)). Thus, in deep RL, neural networks maintain the regularization when augmentation helps the training.

Regularization biases toward regions of loss landscape can have several equivalent generalized solutions [10]. For the same reason, augmentation regularizes a neural network model by imposing a bias toward generalization in deep RL. Moreover, in Bigfish, (0, 5) and (0, 15) increase the training performance and generalization, similar to (0, 25), although they use augmented observations only for a while. Therefore, the augmentation may not be necessary during the whole training process in some tasks.

Delayed augmentation . To determine when we start to use augmentation, we delayed its use until after 10M or 20M steps. The generalization rapidly increases after using augmentation at 10M and 20M (Figre 2(d), 2(e)). Although we impose augmentation late, the augmentation helps the generalization regardless of the start timing. In SL, delayed augmentation cannot achieve as much as using augmentation during whole training [10]. However, (10, 25) improves the generalization to be comparable with that of (0, 25), which use augmentation throughout training; this result differs from the case of supervised learning. However, when augmentation noticeably helps the training, the performance achieved using delayed augmentation may not catch up (Figure 2(e)) to the performance achieved using early augmentation (Figure 2(f)), because the RL gradually improves the policy and trajectory, as a result of its Markov property. Furthermore, the number of samples is limited for RL, but not for supervised learning, so using augmentation from the initial time is more critical than supervised learning if augmentation helps the training.

However, we confirm that delayed augmentation can induce bias toward generalization after the interruption, although the augmentation is not used during the initial transient time. For example, curves (10, 15) (Figure 2(e), 2(f)) equivalent results to those of with (10, 25) even after 15M time steps. This result indicates that a bias toward generalization can occur regardless of the timing of augmentation, but usage from the start is essential when the augmentation gives important knowledge during training.



(a) Background diversity (b) Augmentation selection (c) Performance

Figure 3: Comparison of train performance according to background diversity on Heist *easybg* when InDA use random convolution in 3(a). We describe the degree of background diversity in numbers, e.g., PPO(1). As the diversity of the background increases, the performance of the PPO approaches that of InDA. Figure 3(b) 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 3(c). 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.

When ExDA should be used. ExDA have different response than other methods to the timing of applying augmentation. Most methods combined with data augmentation in RL use augmentation throughout RL training. We show that augmentation rapidly increases the generalization performance in spite of the late usage of augmentation (Figure 2). These results motivate suggest that generalization by augmentation after training in RL.

The performance of PPO approached that of InDA as the diversity of the background increased (Figure 3(a)). This result means that the various backgrounds increase the difficulty of training, and also that random convolution has a similar effect to diverse background images. In contrast, ExDA preserves its training score after pure RL training. These gaps in training are quantified by the test performance.

The two analyses suggest that ExDA is appropriate in environments in which training is difficult as a result of various factors such as background, and object color in the image. Thus, for generalization, ExDA should be used when only one background is present.

When InDA should be used. ExDA does not always guarantee improvement of generalization. InDA is better generalized to unseen levels with *crop* than ExDA is (Table 1). ExDA cannot surpass InDA in some environments, for two reasons.

First, the diversity of data has an important ability to generalize about unseen levels [5]. InDA is trained with various observations during training, whereas ExDA applies the augmentation only on a pre-trained policy's trajectories. Thus, augmentation after training may have difficulty overcoming the limitation of data diversity when the generalization needs the diversity of data distribution. Second, InDA can accelerate the training such as 2(c), so ExDA cannot overcome the gap in training performance. Thus, InDA should be used in both cases.

How to choose between InDA and ExDA. InDA is appropriate when the augmentation methods help to train, and ExDA is appropriate when the augmentation methods increase the difficulty of training. However, we cannot know in advance whether certain augmentation helps the training. Thus, we use UCB-InDA to automatically determine the necessity of the augmentation during training. UCB selects the identity function most often (Figure 3(b)). This means that other augmentations are not helpful to train on Heist(*easybg*), so ExDA is more appropriate than InDA. Furthermore, UCB-InDA w/ identity performs better than w/o identity. It suggests that identity should be included in the UCB action, because of the environments do not need augmentation during training. In contrast, PPO is the same as InDA with identity, which is the best transformation on Heist(*easybg*). Thus, we can use UCB-InDA as pre-trained method for ExDA, e.g., UCB-ExDA, because UCB-InDA w/ identity achieves comparable train performance to that of PPO (Figure 3(c)). As the result, we can automatically select InDA or ExDA appropriately for each task.

5 Discussion

We have showed that the timing of visual augmentation affects the performance of RL, although not affect the performance of SL. The difference is a result of non-stationary data generation in RL. If the regularization imposed by augmentation is useful only for testing, then augmentation should be delayed to the end of RL training than being use throughout learning, because sample and computation complexity since it can disturb RL training. However, an augmentation that provides useful regularization in training should be used during the whole training period to fully utilize its benefit in terms of both generalization and data efficiency. We believe that our findings provide useful insights into auto-augmentation to adjust the use of augmentation round-by-round, where DA at the end of RL training would provide substantial gains as ExDA does. However, design of auto-augmentation for RL remains an open problem, because the gain from augmentation has highly non-stationary characteristics, so its evaluation is challenging.

Acknowledgments and Disclosure of Funding

This work was supported by Institute of Information and communications Technology Planning and evaluation (IITP) grant funded by the Korea government (MSIT) (No.2021-0-02068, AI Innovation Hub). This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2021M3E5D2A01023887). The authors would like to thank Kimin Lee and Alexandre Proutiere for fruitful feedback.

References

[1] A. Achille, M. Rovere, and S. Soatto. Critical learning periods in deep networks. In *International Conference on Learning Representations*, 2018.

[2] S. Arora, N. Cohen, W. Hu, and Y. Luo. Implicit regularization in deep matrix factorization. In *Advances in Neural Information Processing Systems*, pages 7411–7422, 2019.

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

[4] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman. Quantifying generalization in reinforcement learning, 2019.

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

[6] D. Erhan, A. Courville, Y. Bengio, and P. 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.

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. *arXiv preprint arXiv:1802.01561*, 2018.

[8] J. Farebrother, M. C. Machado, and M. Bowling. Generalization and regularization in dqn, 2020.

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

[10] A. S. Golatkar, A. Achille, and S. 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.

[11] S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro. Implicit regularization in matrix factorization. In *Advances in Neural Information Processing Systems*, pages 6151–6159, 2017.

[12] N. Hansen and X. Wang. Generalization in reinforcement learning by soft data augmentation. *arXiv preprint arXiv:2011.13389*, 2020.

[13] N. Hansen, Y. Sun, P. Abbeel, A. A. Efros, L. Pinto, and X. Wang. Self-supervised policy adaptation during deployment. *arXiv preprint arXiv:2007.04309*, 2020.

[14] I. Higgins, A. Pal, A. A. Rusu, L. Matthey, C. P. Burgess, A. Pritzel, M. Botvinick, C. Blundell, and A. Lerchner. Darla: Improving zero-shot transfer in reinforcement learning, 2018.

[15] M. Igl, G. Farquhar, J. Luketina, W. Boehmer, and S. Whiteson. The impact of non-stationarity on generalisation in deep reinforcement learning. *arXiv preprint arXiv:2006.05826*, 2020.

[16] M. Ilse, J. M. Tomczak, and P. Forré. Designing data augmentation for simulating interventions. *arXiv preprint arXiv:2005.01856*, 2020.

[17] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, et al. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. *arXiv preprint arXiv:1806.10293*, 2018.

[18] I. Kostrikov, D. Yarats, and R. Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649*, 2020.

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

[20] K. Lee, K. Lee, J. Shin, and H. Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. *arXiv*, pages arXiv–1910, 2019.

[21] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017.

[22] Z. Liu, X. Li, B. Kang, and T. Darrell. Regularization matters in policy optimization – an empirical study on continuous control, 2020.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

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

[25] R. Raileanu, M. Goldstein, D. Yarats, I. Kostrikov, and R. Fergus. Automatic data augmentation for generalization in deep reinforcement learning. *arXiv preprint arXiv:2006.12862*, 2020.

[26] S. C. Raparthy, B. Mehta, F. Golemo, and L. Paull. Generating automatic curricula via self-supervised active domain randomization, 2020.

[27] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550 (7676):354–359, 2017.

[30] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[31] A. Srinivas, M. Laskin, and P. Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning, 2020.

[32] A. Stooke, K. Lee, P. Abbeel, and M. Laskin. Decoupling representation learning from reinforcement learning. *arXiv preprint arXiv:2009.08319*, 2020.

[33] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE, 2017.

[34] X. Wu, E. Dyer, and B. Neyshabur. When do curricula work?, 2020.

[35] D. Yarats, A. Zhang, I. Kostrikov, B. Amos, J. Pineau, and R. Fergus. Improving sample efficiency in model-free reinforcement learning from images, 2020.

[36] K. You, M. Long, J. Wang, and M. I. Jordan. How does learning rate decay help modern neural networks?, 2019.

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 4: 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 4 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 [7] in every experiment.

B.1 InDA

We use PPO [28] 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 [27]. 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:

$$\bar{G}_{\phi_{max}}(s) + c\sqrt{\frac{\log(s)}{N_{\phi_{max}}(s)}} \leq \bar{G}_{\phi_{min}}(s) + c\sqrt{\frac{\log(s)}{N_{\phi_{min}}(s)}} \tag{8}$$

$$c = \frac{\bar{G}_{\phi_{max}} - \bar{G}_{\phi_{min}} + \epsilon}{\sqrt{\log(s)} \times \max(\frac{1}{\sqrt{N_{\phi_{min}}(s)}} - \frac{1}{\sqrt{N_{\phi_{max}}(s)}}, \frac{1}{\sqrt{W-1}} - \frac{1}{\sqrt{W}})} \tag{9}$$

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 [25], Rand-FM [20], RAD [19]. Every baseline is based on PPO [5] and we adopt the implementation of PPO in [5].

- DrAC [25] 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 [20] 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.

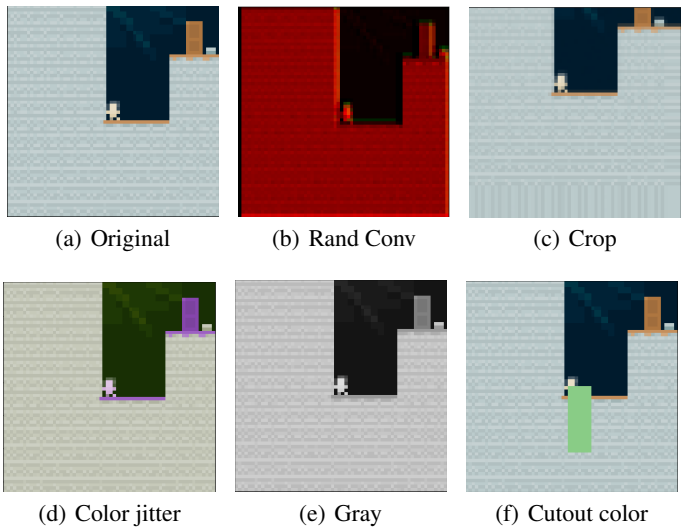
- RAD [19] naively use augmented observations in state distribution. Thus, there are no additional hyperparameters.

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.* [20] (*random convolution*), Laskin *et al.* [19] (*cutout color, color jitter*) and Raileanu *et al.* [25] (*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:



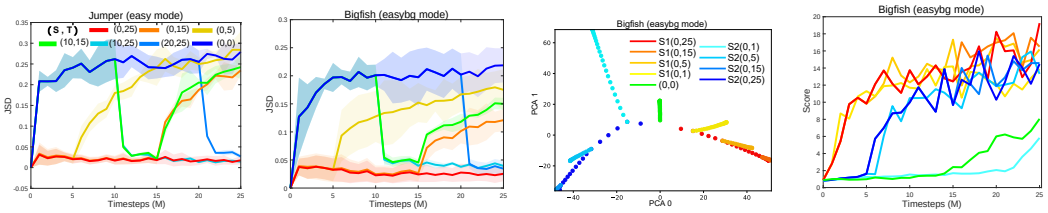
(a) Original (b) Rand Conv (c) Crop

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

Figure 5: Examples of visual augmentations

D Generalization with augmented observations in RL

We mention that data augmentation causes the bias toward generalization in 4.2. Thus, we examine how data augmentation improves the generalization during training. First, We compare the policy distances between augmented observations and non-augmented observations for each usage timing of augmentations in Figure 6(a) and Figure 6(b). We measure the policy distance using Jensen-Shannon



(a) JSD (b) JSD (c) PCA (d) test-lv

Figure 6: Comparison of distance using Jensen-Shannon Divergence (JSD) between policy from augmented observations and non-augmented observations in (a) Jumper(easy mode) with random convolution and (b) Bigfish (easybg mode) with crop. We compare different two samples which trained on Bigfish (easybg mode) about test performance on unseen levels in (c) and optimizing trajectory in (d). We call two samples as S1 and S2 in both (c) and (d). Optimizing trajectory in (c) consists of scattered network parameters after dimension reduction using PCA. The graph of JSD shows that the distance between original and augmented observations is not maintained after augmentation interrupted, in spite of retained generalization in Figure 2(e) and Figure 2(f). Figure (c) represents the learning direction are biased except the S2(0, 1) and (0, 0). This result coincides with the score in (d). Thus, the direction of bias in (c) is toward improving the generalization.

Divergence because it is a lower bound for the joint empirical risk across non-augmented and augmented observations [16, 25]. The policy distances are remarkably reduced when using augmentation in spite of delayed usage at (10, 25), (20, 25) in Figure 6(a) and Figure 6(b). Conversely, the policy distances of (0, 5) and (0, 15) have a rapid increase after interrupted augmentation. Nevertheless, the generalization is similar with fully utilized augmentation such as (0, 25) in Figure 2(e) and Figure 2(f). This shows that the generalization about the change of backgrounds and levels is not relevant to policy consistency about augmented observations after generalization.

We analyze about two samples, which are trained on Bigfish environments, for verifying when generalization occurs. When using augmentation from the initial time, the rising time of performance differs for each sample in Figure 6(d). Sample 1 increases rapidly almost as soon as it starts, while Sample 2 begins to increase after learning for 5M time steps. After 1M, 5M, and 10M, the suspension of the augmentation increases the performance such as non-interrupted one, except when S2(0, 1) at Figure 6(d). Also, Figure 6(c) represent the optimizing trajectories [21] about each sample and PPO using PCA on model parameter space. The method of plotting trajectory is explained in the supplementary material in detail. Each sample learns through different learning paths from the randomly initialized point. We can see that the rest of the agents except S2(0, 1) and PPO are biased toward the generalized orientation. Thus, augmentation causes bias toward improving the generalization, but the learning path's direction differs depending on the initial point. Furthermore, the time step to regularize by augmentation also differs in each random seed.

As a result, the generalization is made by bias toward generalization while matching the policy between augmented observations and non-augmented observations. We need augmentation until the model parameters are biased toward generalization. However, the time to regularize is random according to the learning path. Thus, it is hard to decide when we can stop the augmentation.

E Optimization path plotting

We refer the method from Lie *et al.* [21] and Golatkar *et al.* [10] to represent the optimization trajectories in Figure 6(c). We do PCA analysis with the matrix M, which is combined the weights of the networks for each sample and duration of augmentation. The PCA analysis projects the weights of the networks on the first two principal components as shown in Figure 6(c). We can take a guess about the direction of optimization from the PCA.

F 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 [25], Rand-FM [20], Rad [19] 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 9 and Figure 10. The *crop* method used in our paper do not work well in RAD, because they use a different crop method with [25] in their paper [19]. InDA is more stable than others in training, and it affects generalization performance.

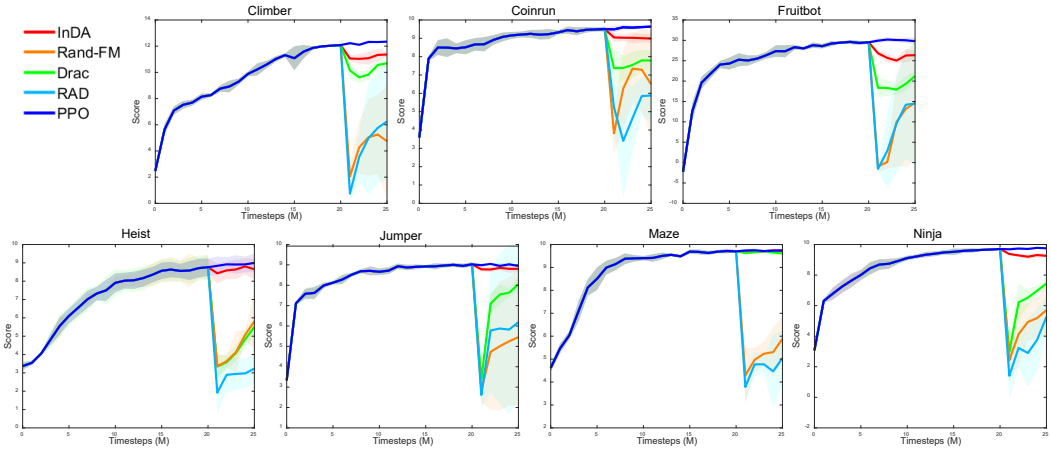


Figure 7: Comparison of the training performance when *random convolution* is applied after 20M timesteps with various augmentation methods.

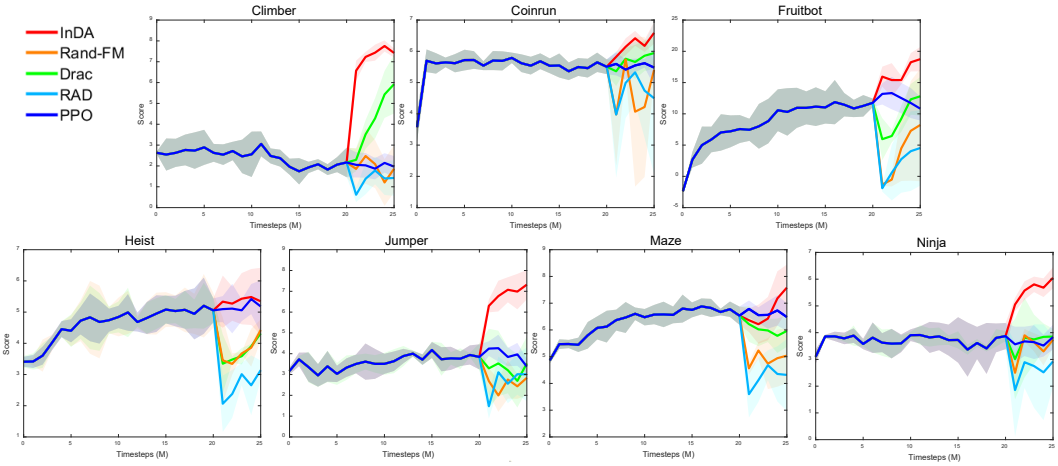


Figure 8: Comparison of the test performance when *random convolution* is applied after 20M timesteps with various augmentation methods.

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 [25] and Rand-FM [20]. 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.

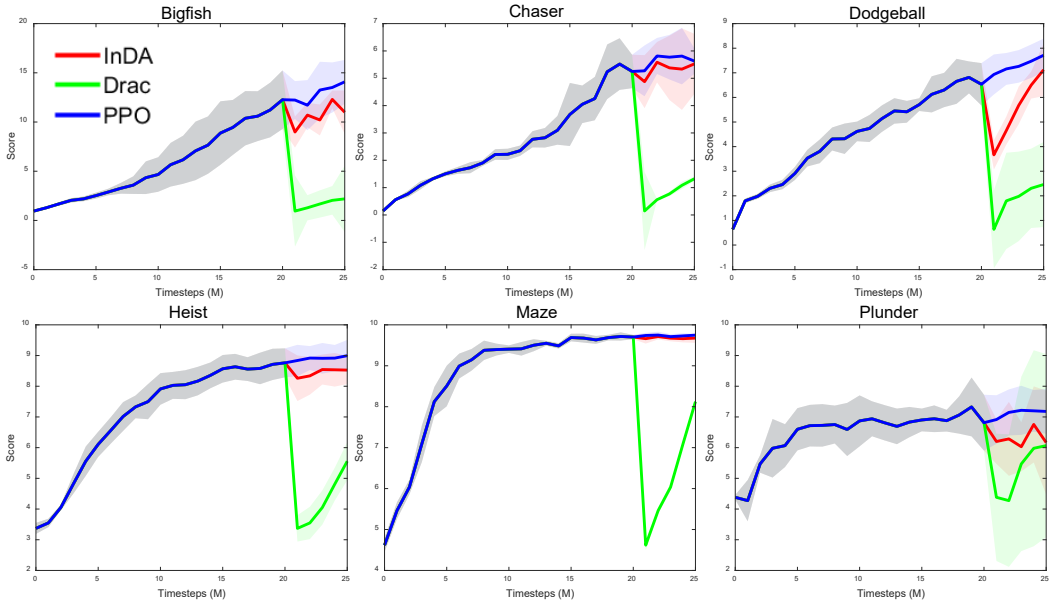


Figure 9: Comparison of the training performance when *crop* is applied after 20M timesteps with InDA and Drac.

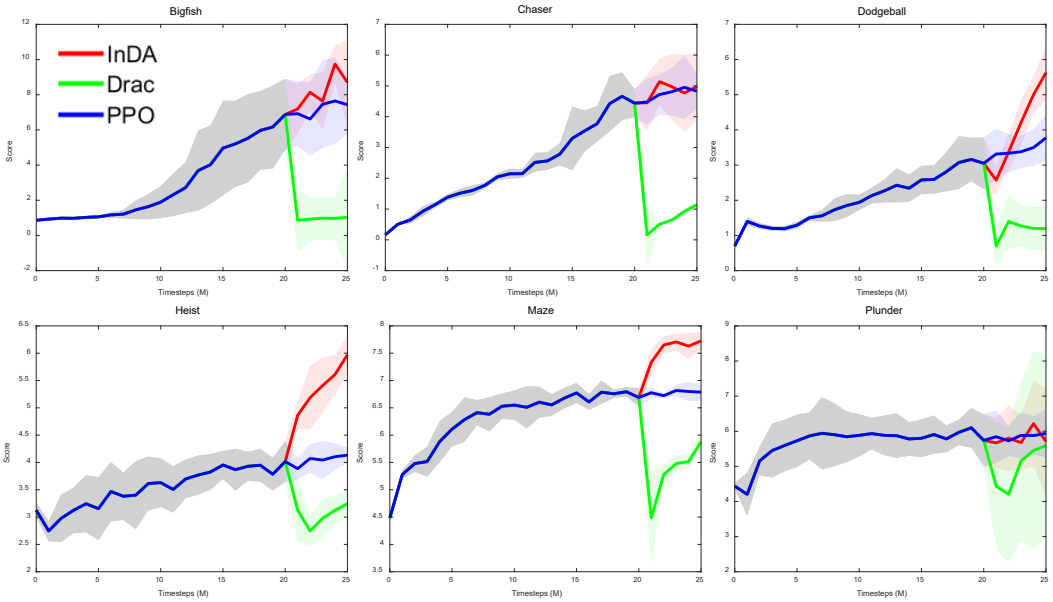


Figure 10: Comparison of the test performance when *crop* is applied after 20M timesteps with InDA and Drac.

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 3.3. 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.* [15] argued that the non-stationarity causes the reduction of generalization. However, the re-initialization is not critical in test performance, as shown in Figure 12. Moreover, sometimes re-initialization makes it difficult to distill training performance such as Fruitbot and Ninja in Figure 11. We use *random convolution* as an augmentation method in here.

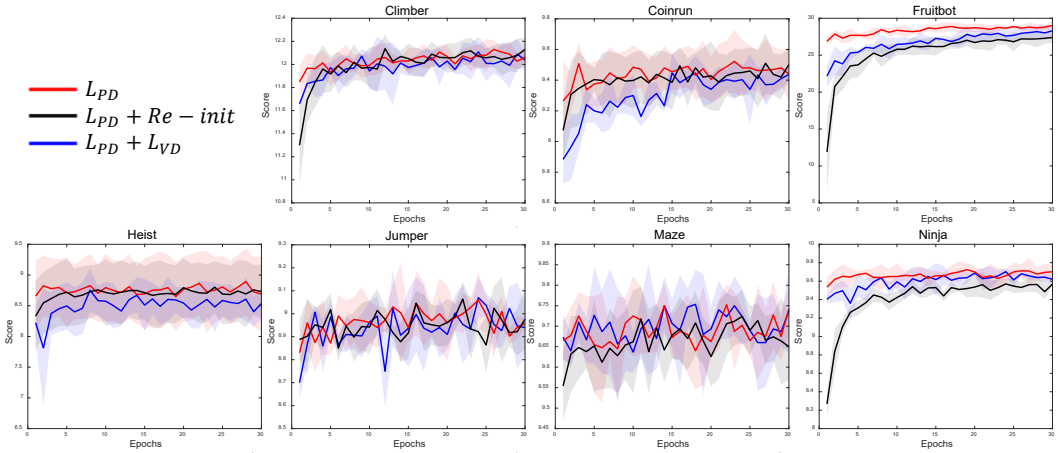


Figure 11: Training performance of ExDA with re-initialization or regularization with value funtion.

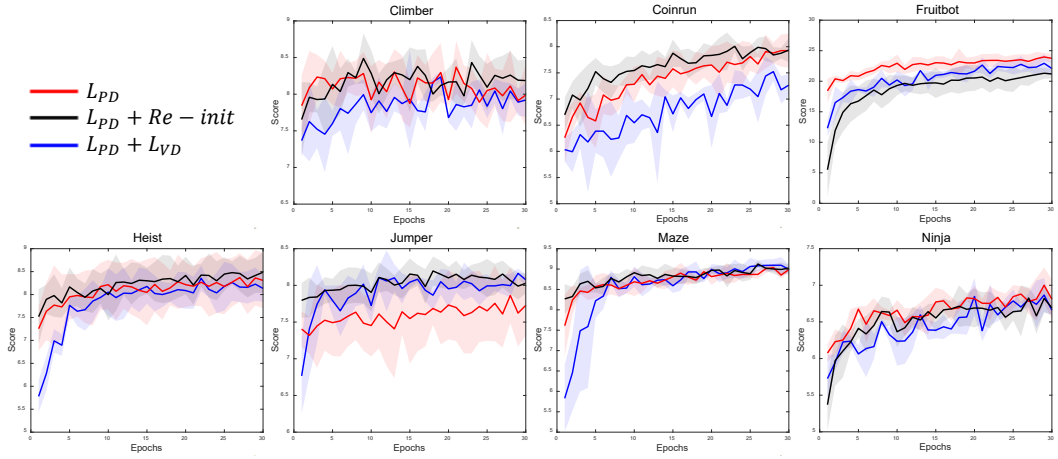


Figure 12: Test performance of ExDA with re-initialization or regularization with value funtion on unseen backgrounds.

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 3, 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 2: 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 3: 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 roughly calculation. Furthermore, the ExDA save the time for augmentation comparing 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 meaningful gain in training performance.

H Time matter in training

This section shows every result of Figure 2 about time dependency with InDA. We experiment with *random convolution, crop, color jitter, gray, cutout color* and evaluate the test on unseen backgrounds (*random convolution, color jitter, gray, cutout color*) and levels (*random crop, cutout color*). However, the effect of generalization is hard to recognize in most cases, as shown in Appendix I. 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, gray, and cutout color. *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 (*random convolution, crop*) and three runs (*color jitter, cutout color, gray*).

H.1 Random convolution

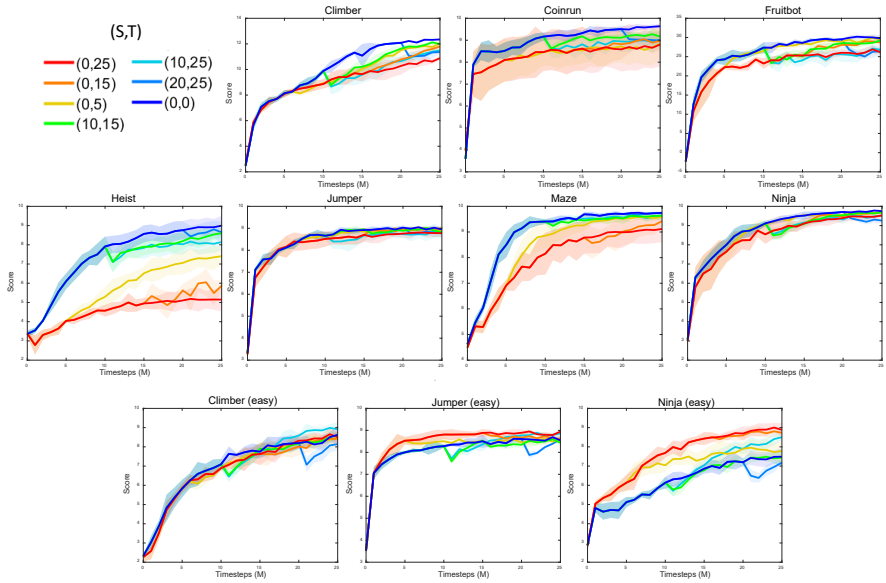


Figure 13: Comparison of training performance according to usage period of augmentation with InDA (*random convolution*): The *easybg* is disturbed by *random convolution*, but, *easy* mode is improved training performance by *random convolution*.

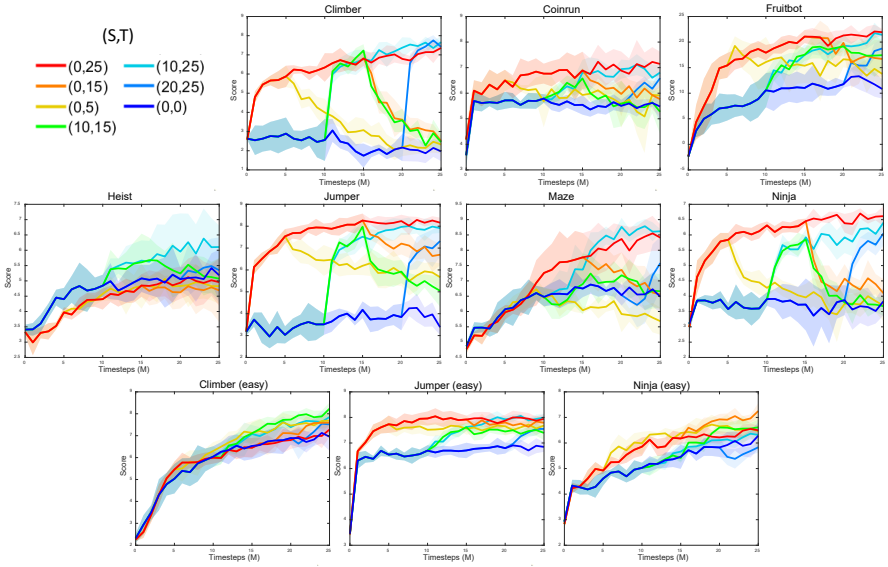


Figure 14: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*random convolution*): Most cases' tendencies are coincidence with the jumper, which is mentioned in the main paper..

H.2 Crop

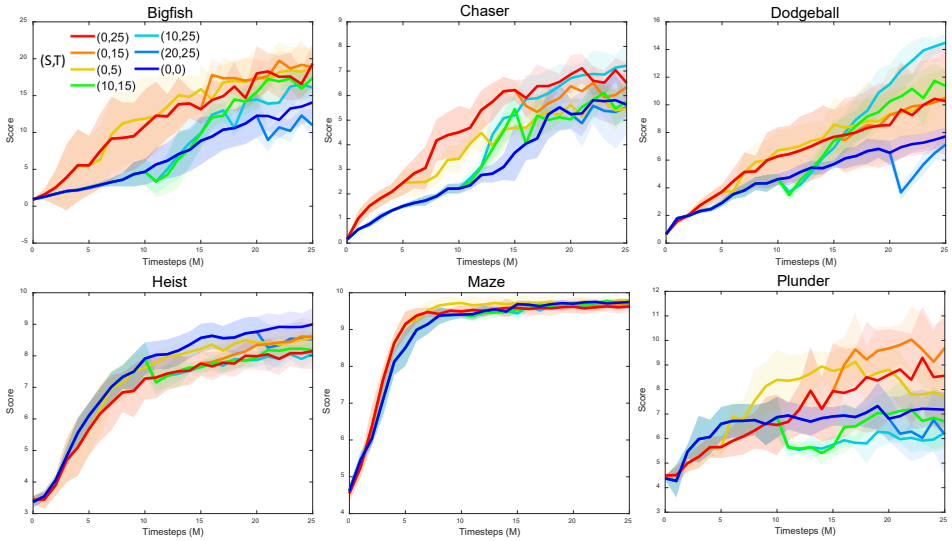


Figure 15: Comparison of training performance according to usage period of augmentation with InDA (*crop*): *Crop* improve the training performance in Bigfish, Chaser, Dodgeball, Plunder. Furthermore, interrupted augmentation is also improved similarly with (0, 25).

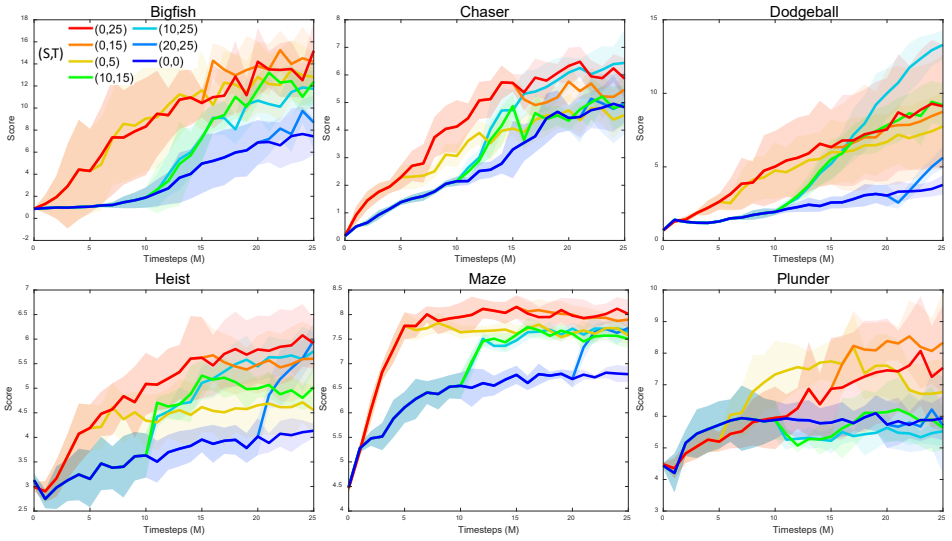


Figure 16: Comparison of generalization on unseen levels according to usage period of augmentation with InDA (*crop*): The generalization is improved by *crop*, and it is conserved after interrupted in Heist and Maze. Bigfish, Chaser, Dodgeball, and Plunder have similar curves with training.

H.3 Color jitter

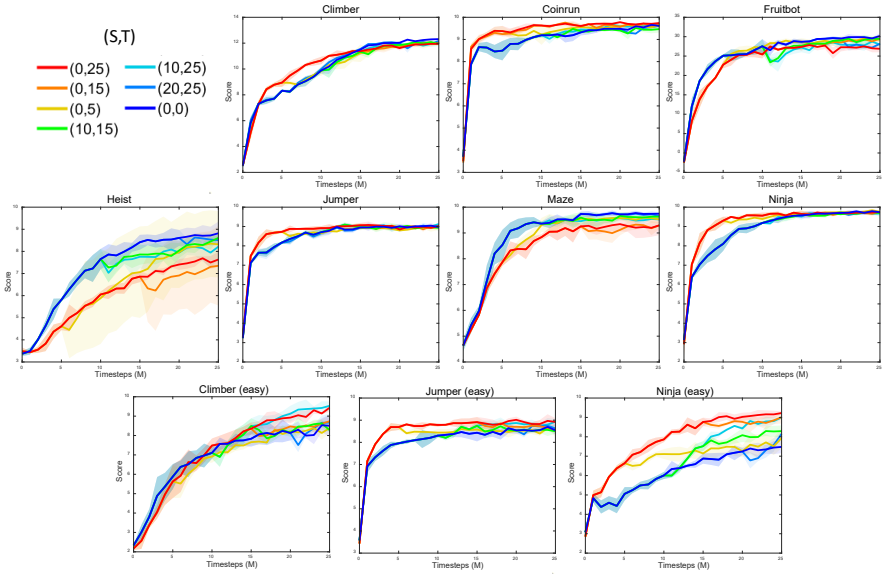


Figure 17: Comparison of training performance according to usage period of augmentation with InDA (*color jitter*): *Color jitter* does not impede the training as much as *random convolution* in most environments. However, *color jitter* helps the training in *easy* mode.

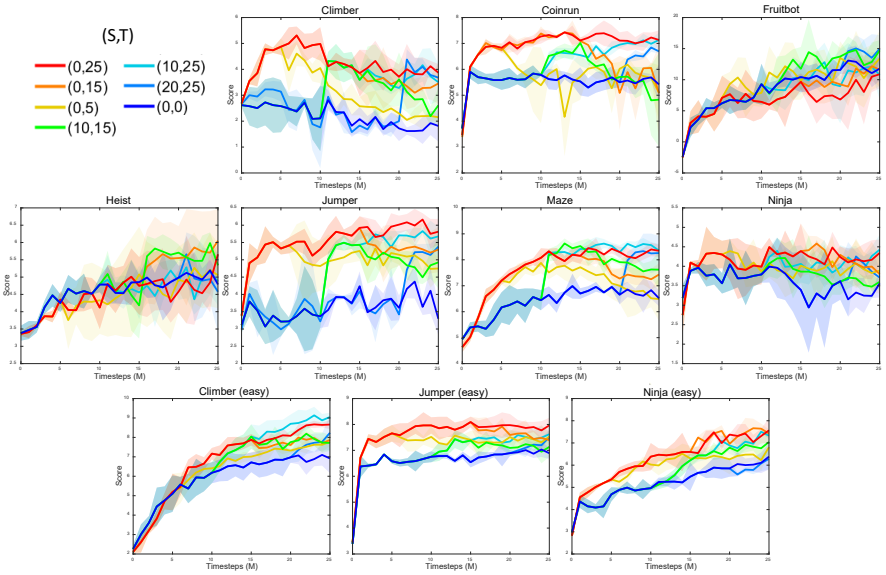


Figure 18: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*color jitter*): Test performance is influenced by *color jitter* as the trend, which is similar to *random convolution*.

H.4 Gray

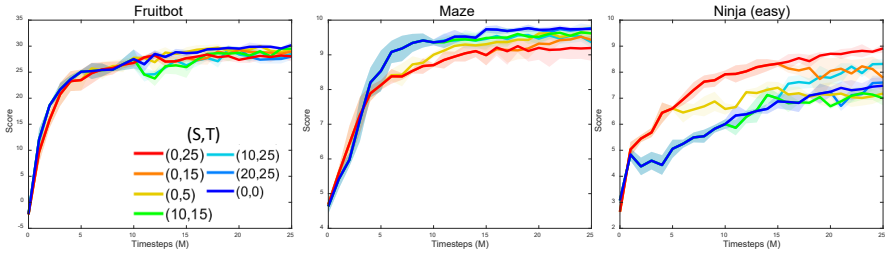


Figure 19: Comparison of training performance according to usage period of augmentation with InDA (*gray*): The effect of *gray* is similar to *color jitter*.

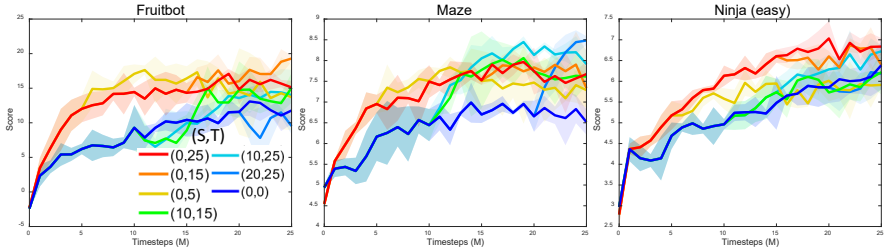


Figure 20: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*gray*): *Gray* improves the generalization, even if the usage of augmentation is delayed. However, it is hard to recognize by low effectiveness of *gray*.

H.5 Cutout color

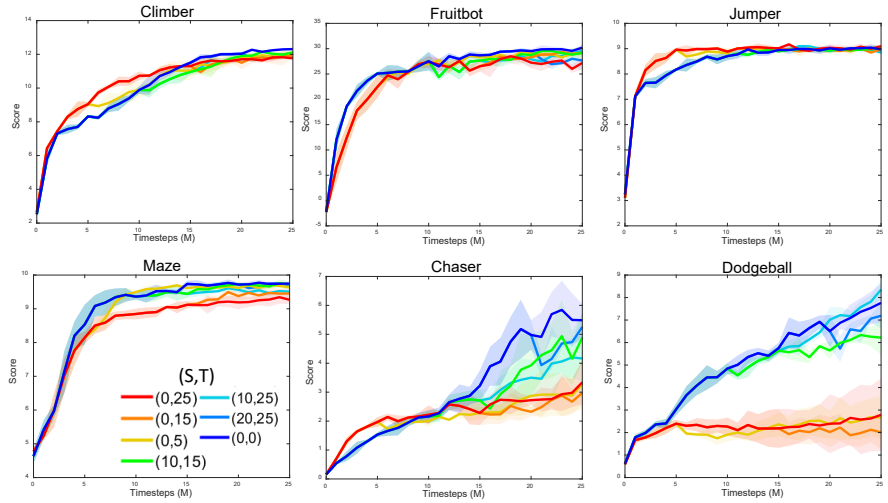


Figure 21: Comparison of training performance according to usage period of augmentation with InDA (*cutout color*): *Cutout color* impedes the training, especially Chaser and Dodgeball are ruined.

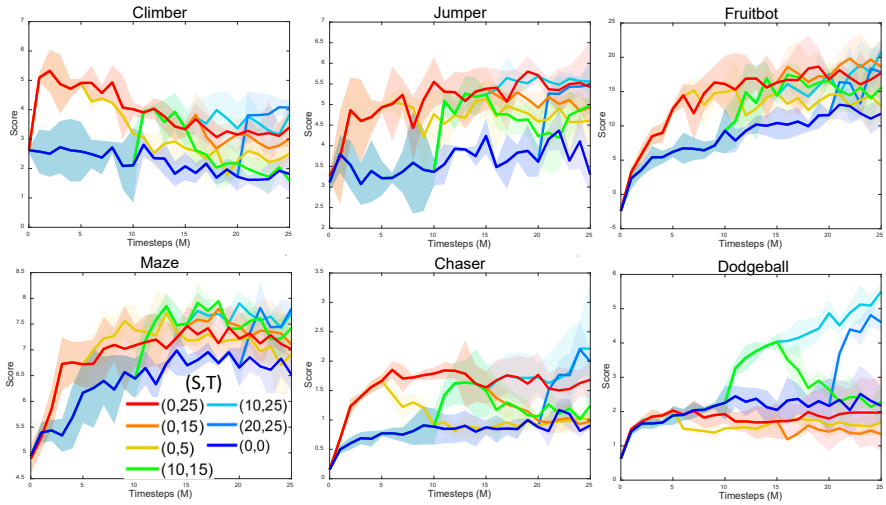


Figure 22: Comparison of generalization on unseen backgrounds according to usage period of augmentation with InDA (*cutout color*): Chaser and Dodgeball have benefited from delayed augmentation because the early used *cutout color* ruins the training.

I Benchmark on Modified Open AI Procgen

We compare the training and test performance on various environments with each augmentation. We also use DrAC [25], RAD [19], Rand-FM [20] as baselines. In every reults, 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 except the Oracle. Red one is the Oracle score, which is trained on test environments such as *easybg-test, easy-test*. Mean and standard deviation is calculated after five runs (*random convolution, crop*) and three runs (*color jitter, cutout color, gray*). For your information, RAD does not work well when using crop, because we use [25]'s crop method which is different with [19].

I.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 4: 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 5: 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 6: 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 7: Test performance benchmark on unseen backgrounds (*easybg, random convolution*).

I.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 8: 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 9: Test performance benchmark on unseen levels (*easybg, crop*).

I.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 10: 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 11: Test performance benchmark on unseen backgrounds (*easy, color jitter*).

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.35** ±0.083	9.78 ±0.306	11.84 ±0.223	12 ±0.256	11.94 ±0.071	12.04 ±0.152
Coinrun	9.64 ±0.07	7.11 ±0.205	8.94 ±0.285	8.62 ±0.091	**9.74** ±0.05	9.45 ±0.09
Fruitbot	29.78 ±0.899	29.74 ±0.443	**30.05** ±0.611	29.48 ±0.507	26.87 ±0.912	29 ±0.878
Heist	**9.00** ±0.513	7.21 ±0.27	7.22 ±0.76	6.89 ±0.348	7.63 ±0.338	8.53 ±0.307
Jumper	8.95 ±0.066	8.72 ±0.119	8.9 ±0.05	8.94 ±0.029	**9.03** ±0.123	**9.03** ±0.086
Maze	**9.75** ±0.513	8.56 ±0.27	9.46 ±0.404	9.46 ±0.184	9.3 ±0.379	9.67 ±0.111
Ninja	9.75 ±0.073	7.81 ±0.422	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 12: Training performance benchmark on *easbg* with *color jitter*.

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	1.97 ±0.51	9.78 ±0.306	**5.05** ±0.407	4.31 ±0.492	4.25 ±0.338	4.34 ±0.856
Coinrun	5.48 ±0.583	7.11 ±0.205	6.46 0±.526	6.47 ±0.194	**7.13** ±0.372	6.53 ±0.375
Fruitbot	10.83 ±1.908	29.74 ±0.443	9.49 ±8.098	8.51 ±1.941	10.88 ±2.263	**18** ±7.442
Heist	5.18 ±0.838	7.21 ±0.27	5.65 ±0.984	5.39 ±0.745	**5.66** ±0.271	5.43 ±0.508
Jumper	3.38 ±0.368	8.72 ±0.119	5.65 ±0.09	5.67 ±0.953	**5.81** ±0.369	5.31 ±0.351
Maze	6.48 ±0.523	8.56 ±0.665	8.22 ±0.455	8.26 ±0.175	8.35 ±0.238	**8.65** ±0.017
Ninja	3.83 ±0.462	7.81 ±0.422	4.22 ±0.487	4.18 ±0.475	**4.34** ±0.345	4.07 ±0.332
Avg	1.00	2.33	1.44	1.37	1.43	**1.48**

Table 13: Test performance benchmark on unseen backgrounds (*easybg, color jitter*).

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.35** ±0.083	9.78 ±0.306	11.12 ±0.26	11.84 ±0.505	11.9 ±0.115	12.06 ±0.03
Coinrun	9.64 ±0.07	7.11 ±0.205	9.53 ±0.135	9.49 ±0.188	**9.74** ±0.046	9.48 ±0.08
Fruitbot	29.78 ±0.899	29.74 ±0.443	**30.01** ±0.572	29.6 ±0.27	28.03 ±0.994	29.32 ±0.937
Heist	**9.00** ±0.513	7.21 ±0.27	6.24 ±0.214	6.53 ±0.474	5.51 ±0.146	8.51 ±0.225
Jumper	8.95 ±0.066	8.72 ±0.119	8.91 ±0.19	8.93 ±.247	**9.18** ±0.18	8.95 ±0.075
Maze	**9.75** ±0.513	8.56 ±0.27	9.46 ±0.192	9.48 ±0.08	9.2 ±0.367	**9.75** ±0.087
Ninja	**9.75** ±0.073	7.81 ±0.422	9.73 ±0.045	9.61 ±0.096	9.6 ±0.081	9.72 ±0.021
Avg	**1.00**	0.85	0.93	0.94	0.95	0.99

Table 14: Training performance benchmark on *easybg* with *gray*.

I.4 Gray

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	1.97 ±0.51	9.78 ±0.306	1.75 ±0.654	1.81 ±0.211	1.24 ±0.502	**2.45** ±0.727
Coinrun	5.48 ±0.583	7.11 ±0.205	5.34 ±0.751	5.31 ±0.501	**6.05** ±0.465	5.79 ±0.061
Fruitbot	10.83 ±1.908	29.74 ±0.443	**17.57** ±0.191	15.47 ±1.449	15.12 ±0.958	15.81 ±0.11
Heist	5.18 ±0.838	7.21 ±0.27	**5.43** ±0.18	5.15 ±0.172	4.32 ±0.112	5.1 ±0.504
Jumper	**3.38** ±0.368	8.72 ±0.119	2.7 ±0.894	4.07 ±0.46	3.55 ±0.992	4.47 ±0.415
Maze	6.48 ±0.523	8.56 ±0.665	7.77 ±0.611	7.93 ±0.104	7.67 ±0.312	**8.33** ±0.119
Ninja	3.83 ±0.462	7.81 ±0.422	3.72 ±0.131	3.91 ±0.62	4.02 ±0.666	**4.03** ±0.071
Avg	1.00	2.33	1.03	1.04	0.97	**1.13**

Table 15: 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 16: 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 17: Test performance benchmark on unseen backgrounds (*easy, gray*).

I.5 Cutout color

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	**12.35** ±0.083	9.78 ±0.306	11.92 ±0.158	8.26 ±0.663	11.76 ±0.027	12.07 ±0.127
Coinrun	9.64 ±0.07	7.11 ±0.205	9.23 ±0.323	8.07 ±0.645	**9.7** ±0.084	9.39 ±0.012
Fruitbot	**29.78** ±0.899	29.74 ±0.443	29.73 ±0.898	29.2 ±0.64	27.18 ±1.302	28.95 ±0.907
Heist	**9.00** ±0.513	7.21 ±0.27	8.47 ±0.397	6.25 ±0.704	6.1 ±0.693	8.65 ±0.21
Jumper	8.95 ±0.066	8.72 ±0.119	8.87 ±0.123	8.75 ±0.131	**9.1** ±0.081	8.91 ±0.053
Maze	**9.75** ±0.513	8.56 ±0.27	9.41 ±0.134	9.17 ±0.118	9.27 ±0.125	9.74 ±0.133
Ninja	**9.75** ±0.073	7.81 ±0.422	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**	0.77	0.82	0.72	0.76	0.94

Table 18: 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 19: Training performance benchmark on *easy* with *cutout color*.

Easybg	PPO	Oracle	DrAC	RAD	InDA	ExDA
Climber	1.97 ±0.51	9.78 ±0.306	3.54 ±0.164	3.97 ±0.999	3.4 ±0.645	**4.29** ±0.154
Coinrun	5.48 ±0.583	7.11 ±0.205	5.87 ±.251	5.93 ±0.061	6.2 ±0.357	**6.41** ±0.131
Fruitbot	10.83 ±1.908	29.74 ±0.443	18.18 ±3.744	**19.24** ±3.385	17.69 ±4.026	17.7 ±0.888
Heist	5.18 ±0.838	7.21 ±0.27	6.6 ±.092	5.76 ±0.551	4.97 ±0.33	**7.51** ±0.119
Jumper	3.8 ±0.368	8.72 ±0.119	4.99 ±0.114	5.48 ±0.28	5.43 ±1.116	**6.02** ±0.235
Maze	6.48 ±0.523	8.56 ±0.665	7.33 ±0.223	7.66 ±.243	7.01 ±0.17	**7.83** ±0.22
Ninja	3.83 ±0.462	7.81 ±0.422	**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	2.44	1.27	1.33	1.19	**1.53**

Table 20: 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 21: 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 22: 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 23: Test performance benchmark on unseen levels (*easy, cutout color*).