Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning

Byungchan Ko\textsuperscript{1} Jungseul Ok\textsuperscript{1,2}

Abstract

Data augmentation technique from computer vision has been widely considered as a regularization method to improve data efficiency and generalization performance in vision-based reinforcement learning. We variate the timing of using augmentation, which is, in turn, critical depending on tasks to be solved in training and testing. According to our experiments on Open AI Progen Benchmark, if the regularization imposed by augmentation is helpful only in testing, it is better to procrastinate the augmentation after training than to use it during training in terms of sample and computation complexity. We note that some of such augmentations can disturb the training process. Conversely, an augmentation providing regularization useful in training needs to be used during the whole training period to fully utilize its benefit in terms of not only generalization but also data efficiency. These phenomena suggest a useful timing control of data augmentation in reinforcement learning.

1. Introduction

Reinforcement Learning (RL) from visual observations is a fundamental problem since visual data is one of the most common observations available in practice, e.g., video game (Mnih et al., 2015), board game (Silver et al., 2017; 2018), and robot (Tobin et al., 2017; Kalashnikov et al., 2018). However, such a vision-based RL often suffers from the problem of sample efficiency and generalization capability due to the high-dimensional nature of images. As a part of overcoming this, regularization by data augmentation has been widely considered (Laskin et al., 2020; Kostrikov et al., 2020), where visual data is augmented by transformation preserving the same meaning or context, e.g., cropping out unimportant part of images or randomizing backgrounds. It not only resolves the data scarcity but also provides an explicit implementation of inductive bias for generalization performance.

Recent works show that using the right types of data augmentation significantly improves both data efficiency and generalization performance (Raileanu et al., 2020). However, choosing the right data augmentation is found to be highly task-dependent, where poor choice can degenerate the generalization and destabilize the training (Laskin et al., 2020; Raileanu et al., 2020). Hence, a variety of transformations have been developed to enlarge the set of data augmentation (Lee et al., 2019; Hansen & Wang, 2020). A number of regularization methods using data augmentation, meanwhile, have been proposed to stabilize training process with data augmentation, e.g., self-supervised learning (Raileanu et al., 2020) and representation learning (Srinivas et al., 2020; Hansen & Wang, 2020) by reducing the interference between RL training and regularization. Although the previous works address what data augmentation to use and how to use it, but the understanding on 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 since the timing of data augmentation is not critical in supervised learning (SL) (Achille et al., 2018; Golatkar et al., 2019), whereas a curriculum learning can accelerate RL training (Raparthy et al., 2020).

To address our main question, we devise two frameworks with different timings of applying augmentation: InDA (Intra Distillation with Augmented observations) and ExDA (Extra Distillation with Augmented observations) (Section 3). Implementing the regularization via augmentation in a form of distillation to minimize interference in RL training, InDA interleave the distillation with RL training, while ExDA applies the distillation at the end of RL training. From experiments with InDA and ExDA, we find that:

\textit{Time does matter when using augmentation in RL}

in contrast to the case of SL (Golatkar et al., 2019) where the effect of data augmentation is relatively insensitive to timing. The difference mainly comes from the fact that RL
agent collects samples when training but SL uses a fixed data set. To be specific, our main findings from experiments are given in two folds:

(i) Applying augmentation needs to be applied as early as possible for sample efficiency and generalization if it can accelerate RL training, e.g., cropping out unnecessary part of image induces an efficient attention mechanism. It is obvious that we need to accelerate RL training from beginning in order to maximize sample efficiency. However, interestingly, we observe that this kind of augmentation often connotes generalization which is transferable only through diverse experience in training process, i.e., ExDA does not fully exploit generalization gain whereas InDA does. Hence, it is indeed necessary to use such an augmentation during training to gain generalization. (Section 4.1)

(ii) Applying augmentation needs to be postponed to the end of RL training for sample efficiency if the regularization imposed by augmentation is helpful only in testing, e.g., augmentation by changing colors is useless when a single background is shown in training, but testing task has multiple backgrounds. We find that this type of augmentation possibly interferes with RL training but there is no harm in generalization from delaying it. Hence, in this case, ExDA, which never disturbs RL training, is better than InDA. (Section 4.2)

In the above findings, we characterize and suggest effective timings of applying augmentation in RL. In addition to this, our contribution can include proposing InDA and ExDA algorithms, in particular, which is equipped with the distillation augmentation (DA) since it is of independent interest that developing a regularization method using augmentation with minimal interference with RL training. Compared to existing methods DrAC (Raileanu et al., 2020), RAD (Laskin et al., 2020), Rand-FM (Lee et al., 2019), a potential advantage of the proposed one is discussed in Section 3.

2. Related Works

Augmented experience in RL. In order to resolve the problem of poor generalization and sparse data, it is a popular approach to generate diverse (virtual) experiences and let RL agent learn from them. Domain randomization is a technique to produce such experiences from a simulator of targeted system (Tobin et al., 2017; Pinto et al., 2017; Raparthy et al., 2020). It is hard to obtain an accurate simulator of practical systems and thus this limits the spectrum of application. Visual augmentation, meanwhile, has no such limit since it is based on simple image transformations such as cropping, tilting, color jitters and so on, although we need a careful understanding on the targeted system to design appropriate image transformer. Raparthy et al. (2020) propose a curriculum learning for domain randomization where the difficulty is gradually increasing and the insight coincides with some of our findings. However, we provide further understandings on which types of visual augmentation need to be used as sooner or later as possible.

The regularization from augmented data in vision-based RL has been implemented in various learning frameworks, including but not limited to representation (Hansen & Wang, 2020; Stooke et al., 2018), self-supervised (Raileanu et al., 2020), and contrast (Srinivas et al., 2020). Raileanu et al. (2020) further proposes an algorithm to automatically select the most effective augmentations over RL training based on UCB (Upper Confidence Bound) algorithm (Auer, 2002), where each augmentation is considered as an arm and is evaluated its effectiveness in sliding window. The idea of adapting augmentation shares with our main message regarding the timing of augmentation. In Raileanu et al. (2020), not augmenting is not counted as an arm, while according to our findings, it needs to be. In addition, the post augmentation followed by RL training of ExDA is not considered.

Different time-sensitivity of augmentation than SL. In deep learning, we often observe that the early state of training impacts significantly (Erhan et al., 2010; Achille et al., 2018). Hence, this motivates us to devise time-sensitive methods adapting to the progress of training such as learning rate decay (You et al., 2019) and curriculum learning (Wu et al., 2020). Golatkar et al. (2019) 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 perhaps is 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 (Higgins et al., 2018; Cobbe et al., 2019; Liu et al., 2020; Farebrother et al., 2020; Srinivas et al., 2020; Yarats et al., 2020; Hansen et al., 2020), a thorough study needs to be in advance of introducing method from different learning framework as we find the contrasting time-sensitivities of data augmentation. This spirit is also shared with an application (Igl et al., 2020) of implicit bias in SL (Gunasekar et al., 2017; Arora et al., 2019; Gidel et al., 2019) to RL.

3. Method

Notation. We consider a standard agent-environment interface of vision-based reinforcement learning in discrete Markov decision process of state space $S$, action space $A$ and kernel $P = P(s_{t+1}, r_t | s_t, a_t)$ which determines the state transition and reward distribution. The goal of RL agent is to find policy maximizing the expectation of cumu-
lative 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 (possibly partial) observation of the next state \( s_{t+1} \). To augment observations, we consider image transformation function \( \phi: \mathbb{R}^{k \times k} \rightarrow \mathbb{R}^{k \times k} \) which maintains the dimension.

**Baseline RL algorithm.** As baseline of deep RL algorithm, we use Proximal Policy Optimization (PPO) (Schulman et al., 2015) which is an on-policy actor-critic RL algorithm to learn policy \( \pi_\theta(a \mid o) \) and value function \( V_\theta \) with network parameter \( \theta \). Storing a set of recent transitions \( \tau_t := (o_t, a_t, r_t, o_{t+1}) \) in experience buffer \( D \), the network parameter \( \theta \) is updated to maximize the following objective function:

\[
L_{\text{PPO}}(\theta) = L_\pi(\theta) - \alpha L_V(\theta),
\]

where \( \alpha \) is a hyperparameter and some regularization terms are omitted. The clipped policy objective function \( L_\pi \) and value loss function \( L_V \) are defined as:

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

\[
L_V(\theta) = \mathbb{E}_t \left[ (V_\theta(o_t) - V_{\hat{t}}(o_t))^2 \right],
\]

where the expectation \( \mathbb{E} \) is taken with respect to \( \rho_t(\theta) \sim D, \theta_{\text{old}} \) is the network parameter before the update, \( \rho_t(\theta) \) is the importance ratio \( \frac{\pi_\theta(a_t \mid o_t)}{\pi_{\text{old}}(a_t \mid o_t)} \). \( \hat{A}_t \) is advantage from Generalized Advantage Estimator (Schulman et al., 2015).

**Overall framework.** We propose two frameworks: InDA (Intra Distillation with Augmented observations) and ExDA (Extra Distillation with Augmented observations). To be specific, both of them use PPO for RL and the DA (Distillation with Augmented observations), described in Section 3.1, for regularization, although our frameworks can employ other RL algorithms and augmentation-based regularization. InDA, described in Section 3.2, interleaves PPO and DA, whereas ExDA, described in Section 3.3, performs PPO first then DA. As shown in Figure 1, we design InDA and ExDA to conduct either DA or PPO in each epoch.

### 3.1. Distillation with Augmented observations (DA)

DA regularizes reinforcement learning using distillation with data augmentation, where 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 \( \theta_{\text{old}} \) to be distilled, and store observation \( o_t \) sampled from \( \pi_{\text{old}} \) and their augmentations \( o'_t = \phi(o_t) \) in augmented buffer \( D_\phi \), where \( \phi: \mathbb{R}^{n \times n} \rightarrow \mathbb{R}^{n \times n} \) is a transformation function. We then train a network \( \theta \) to minimize the following distillation loss function:

\[
L_{\text{DA}}(\theta) = L_{\text{PD}}(\theta) + L_{\text{VD}}(\theta),
\]

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

\[
L_{\text{PD}}(\theta) = \mathbb{E}_o \left[ \frac{KL[\pi_\theta(a_t \mid o_t) \mid \pi_{\text{old}}(a_t \mid o_t)]}{(V_\theta(o_t) - V_{\text{old}}(o_t))^2} \right],
\]

\[
L_{\text{VD}}(\theta) = \mathbb{E}_o \left[ (V_\theta(o_t) - V_{\text{old}}(o_t))^2 \right],
\]

The expectation \( \mathbb{E}_o \) here is taken with respect to \( \tau \sim D_\phi \). Note that the augmented buffer \( D_\phi \) contains not only augmented observations but also original ones, and we freeze \( \theta_{\text{old}} \) as trained in reinforcement learning. Hence, the distilled network \( \theta \) behaves the same with \( \theta_{\text{old}} \) for the original input images. This reduces the degeneration of training performance right after distillation. To our best knowledge, there is no prior work applying augmentation at this level of conservation of RL model. Another advantage of freezing \( \theta_{\text{old}} \) is the reduced computational cost by pre-computing outcomes of \( \theta_{\text{old}} \) in advance of DA, which is not available in other existing methods matching outputs of original and augmented inputs online (Raileanu et al., 2020).

### 3.2. Intra Distillation with Augmented observations

As described in Algorithm 1, InDA iteratively optimize PPO and DA, where we explicitly separate PPO and DA by phase. Such a separation reduces their interference (Hansen & Wang, 2020), while they are often optimized simultaneously in other methods (Raileanu et al., 2020). This phasic separation further robustifies our algorithm in addition to the conservative distillation of DA. We variate the timing of augmentation by \( S \) and \( T \), which are the times of, respectively, starting and terminating DA. Also, we adjust the interval of distillation with \( I \). Each DA phase, we compute \( \pi_{\text{old}} \) and \( V_{\text{old}} \) using \( \theta_{\text{old}} \) which is trained until current DA phase. We provide further details on InDA in the supplementary material.

### 3.3. Extra Distillation with Augmented observations

As described in Algorithm 2, ExDA performs the distillation followed by the end of RL training, where the lengths of...
we expect that diminishing of non-stationarity can improve we consider the re-initialization after pre-training, because

Algorithm 1 InDA

1: Hyperparameter: $S, T, N, I, \phi$
2: Initialize $\theta$ close to origin.
3: for $n = 1, 2, \ldots, N$ do
4: // RL training
5: for $i = 1, 2, \ldots, I$ do
6: Store sampled transitions to $D$;
7: Optimize RL objective $L_{\text{PPO}}(\theta)$ with $D$;
8: end for
9: // Distillation phase
10: if $n \in [S, T]$ then
11: Store $\theta_{\text{old}} \leftarrow \theta$;
12: Generate $D_{\phi}$ by augmenting $D$ with $\phi$;
13: Minimize $L_{\text{DA}}(\theta)$ with $D_{\phi}$;
14: end if
15: end for

Algorithm 2 ExDA

1: Hyperparameter: $N, M, \phi$
2: Initialize $\theta$ close to origin.
3: // Pre-training phase with RL algorithm
4: for $n = 1, 2, \ldots, N$ do
5: Store sampled transitions to $D$;
6: Optimize RL objective $L_{\text{PPO}}(\theta)$ with $D$;
7: end for
8: Store $\theta_{\text{old}} \leftarrow \theta$;
9: // Distillation at the end of RL training
10: for $m = 1, 2, \ldots, M$ do
11: Generate $D_{\phi}$ by augmenting $D$ with $\phi$;
12: Minimize $L_{\text{DA}}(\theta)$ with $D_{\phi}$;
13: end for

da and RL training are parameterized by $M$ and $N$, respectively. It is worth to note that one can lower computational cost by replacing $L_{\text{DA}}$ with $L_{\text{PD}}$ in DA as there is no need of value function after DA. We empirically check that this reduction wouldn’t degenerate RL performance. Besides, we consider the re-initialization after pre-training, because we expect that diminishing of non-stationarity can improve generalization as mentioned in (Igl et al., 2020). However, training performance is not preserved after re-initialization because $\pi_{\theta_{\text{old}}}$ is not completely distillation by low data diversity. Thus, we do not use re-initialization for DA. We leave more interesting details in the supplementary material.

4. Experiment

Setups. We evaluate the time-sensitivity of applying augmentation on OpenAI Procgen benchmark of 16 games, (Cobbe et al., 2020), where at each time $t$, visual observation $o_t$ is given as an image of size $64 \times 64$, and contains full or partial information on the system state. Training or testing environment is defined by a pair of game and mode, where mode determines a set of levels and backgrounds shown in the environment. As training environment, we use one of two modes: easy and easybg. Easy mode provided by Cobbe et al. (2020) 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. Fig 2 presents an example set of modes that we use in evaluation. The details of modes in our evaluation is provided in the supplementary material.

For the sake of 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 (Lee et al., 2019). 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 puts zero-pads to the outside (Raileanu et al., 2020). This
Figure 3. Comparison according to usage period of augmentation with InDA: \((S, T)\) is hyper-parameters of InDA for a start and terminal time step of distillation. InDA uses random convolution on Jumper and crop on Bigfish as an augmentation method. Test results are evaluated on test-bg (unseen backgrounds) and test-lv (unseen levels). Shade region is a 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 differed by the diversity of factors in observations. For instance, when an agent is trained on Jumper (easybg mode), augmentation decreases the data efficiency at the curve (0, 25) of (a). Moreover, they lose the generalization after the interruption at curves (0, 5), (0, 15) and (10, 15) in (d). On the contrary, augmentation boosts the training [(0, 25) of (b)] and maintains the generalization [(0, 5), (0, 15), (10, 15) of (e)] despite augmentation interrupted on Jumper (easy mode). In addition, the delayed augmentation can improve data generalization as much as the curve (0, 25) in the curves (10, 25), (20, 25). On the other hand, when the augmentation significantly accelerates the training, the early start of augmentation is essential for data efficiency and generalization because of limited data. Furthermore, the short-term augmentation can contribute to data efficiency and generalization such as the curves (0, 5), (10, 15) in Bigfish and Jumper (easy mode).

We also report 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 reported as averages over five runs.

4.1. Augmentation during RL training

Reinforcement learning has a non-stationarity, where the distribution of agent’s trajectory and policy are changing over time, but recently proposed data augmentation frameworks use augmented data in the whole training time. Furthermore, augmentation does not always help train and test simultaneously, as the test environment has unseen challenges from training. Thus, we consider the necessity of using augmentation whole training time, even though RL has non-stationary characteristics. In this section, we want to find out when and how the agent is particularly helped by augmentation in RL training.

We variate the times to start and terminate distillation, denoted by \(S\) and \(T\), respectively. We experiment by changing the distillation start time \(S\) and distillation terminal time \(T\) of InDA to see how generalization’s effect depends on the time to use augmentation. Compared to PPO, we evaluate InDA with six different pairs of start and terminal time of distillation \((S, T)\). When the training time steps are 25M, the six pairs \((S, T)\) consist of \((0, 25), (0, 5), (0, 15), (10, 0), (20, 25), (0, 0)\).
Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning

Figure 4. Comparison distance using Jensen-Shannon Divergence 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). Optimizing trajectory in (c) consists of scattered network parameters after dimension reduction using PCA.

(20, 0), (10, 15) and also PPO can be considered InDA with \((S, T) = (0, 0)\). We explained with Jumper and Bigfish in the main paper, and experiments on other environments are described in the supplementary material. Fig. 3 represents three environments, Jumper with easy and easybg mode and Bigfish with easybg mode. In the following, we call the curve using a parameter \((S, T)\).

Interrupted augmentation. We wonder how generalization would change after regularization stopped. Thus, we stop the DA during training, such as \((0, 5), (0, 15)\). When we compare the graph Fig 3(d) and Fig 3(e), generalization performance in Fig 3(d) rapidly decrease after interrupted on both \((0, 5)\) and \((0, 15)\). In contrast, the curves \((0, 5)\) and \((0, 15)\) in Fig 3(e) maintain the generalization in spite of interrupting augmentation. In training performance, InDA which uses augmentation throughout training, performs better than PPO in Fig 3(b); however, in Fig 3(a), augmentation does not improve the training performance. 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 in a single background. Therefore, the generalization rapidly decreases after augmentation interrupted when training with a single background. On the other hand, the training can have help when their difficulty is solved by augmentation, such as Fig 3(b), Fig 3(c). Thus, in deep reinforcement learning, neural networks maintain the regularization when augmentation helps the training. Similarly, Golatkar et al. (2019) argued that the regularization biases toward regions of loss landscape have several equivalent generalized solutions. For the same reason, augmentation regularizes a neural network model by the bias toward generalization in deep reinforcement learning. Moreover, \((0, 5), (0, 15)\) increase the training performance and generalization, similar to \((0, 25)\), although they use augmented observations only for a while in Bigfish.

Delayed augmentation. We experiment on the generalization when we start to use augmentation lately as 10M, 20M. As shown in Fig 3(d) and Fig 3(e), the generalization rapidly increase after using augmentation at 10M and 20M. Although we use augmentation lately, augmentation helps the generalization regardless of the usage timing. (Golatkar et al., 2019) show that delayed augmentation cannot achieve as much as using augmentation during whole training in supervised-learning. However, \((10, 25)\) improves generalization comparable with \((0, 25)\), which use augmentation throughout training, unlike supervised learning. However, when augmentation largely helps the training, such as Fig 3(e), delayed augmentation struggle to follow earlier in Fig 3(f), because the reinforcement learning has the Markov property. Furthermore, unlike supervised learning, reinforcement learning has a limited number of samples, so using augmentation from the initial time is more critical than supervised learning if augmentation helps the training. We confirm that delayed augmentation can induce bias toward generalization after interrupted, although it is not used in initial transient time from \((10, 15)\) at Fig 3(e), Fig 3(f). Through this result, augmentation inducing bias toward generalization can occur regardless of timing.

Generalization with augmented observations in RL. We mention that data augmentation causes the bias toward generalization. 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 Fig 4(a) and Fig 4(b). We measure the policy distance using Jensen-Shannon Divergence because it is a lower bound for the joint empirical risk across non-augmented and augmented observations (Ilse et al., 2020; Raileanu et al., 2020). The policy distances are remarkably reduced when using augmentation in spite of delayed usage at \((10, 25), (20, 25)\) in Fig 4(a) and Fig 4(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 Fig 3(e) and Fig 3(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 Fig 4(d). Sample 1 increases rapidly almost as soon as it starts, while Sample 2 begins to increase after learning about 5M. After 1M, 5M, and 10M, stopping augmentation increases the performance like non-interrupted, except when discontinued in 1M of Sample 2. Also, we draw the optimizing trajectories (Li et al., 2017) about each sample and PPO using PCA in Fig 4(e). 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 Sample2’s (0, 1) and (0, 0) are biased toward the direction of generalization. Thus, augmentation causes bias toward 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 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 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.

### 4.2. Augmentation after RL training

Most methods using data augmentation in reinforcement learning use augmentation throughout the RL training. We showed that augmentation rapidly increases the generalization performance despite the late usage of augmentation in Fig 3. From these results, we consider that the possibility to improve generalization after training in reinforcement learning. To confirm the effect of augmentation after RL training, we compare ExDA with InDA, PPO (Schulman et al., 2017) and Oracle that trained on test environments.

We train 200 levels of single background Procgen environments during 25M time-steps. In contrast, we train ExDA during 30 epochs with 0.5M time steps, after training with PPO for 20M time-steps. Further detail about implementation and hyper-parameter is described in the supplementary material.

Table 2 shows that ExDA outperforms InDA because after training can preserve the training performance while augmentation impedes the training when using it during training. ExDA also exceed the train and test performance of other baselines such as DrAC (Raileanu et al., 2020), Rand-FM (Lee et al., 2019), Rand (Laskin et al., 2020) on most environments which have help the generalization by random convolution. Especially in Table 1, Heist struggles to train the policy when augmentation is used during training. Moreover, ExDA is comparable with Oracle despite it trained with a single background. This result shows that various backgrounds are critical factors to hinder training. Thus, it is possible to choose training in a single background rather than in a different background in an environment where the difficulty rise due to various backgrounds is large. Furthermore, computation complexity is distinctly decreased when using ExDA rather than InDA, because InDA update for 25M augmented data while ExDA computes only 30 epochs with 0.5M. We describe the computation issue about both ExDA and InDA in the supplementary material.

However, using augmentation after training does not always

| Test  | PPO   | Oracle | InDA  | ExDA  |
|-------|-------|--------|-------|-------|
| Coinrun | 5.48  | 7.11   | 7.14  | 7.8   |
| ±0.583 | ±0.205| ±0.479 | ±0.388|
| Fruitbot| 10.83 | 29.74  | 21.93 | 23.57 |
| ±1.908 | ±0.443| ±0.664 | ±0.745|
| Clumber| 1.97  | 9.78   | 7.36  | 8.11  |
| ±0.51  | ±0.306| ±0.273 | ±0.457|
| Heist  | 5.18  | 7.21   | 4.96  | 8.15  |
| ±0.838 | ±0.269| ±0.777 | ±0.633|

Table 1. Comparison training performance: Training with random convolution in both ExDA and InDA. Oracle is trained using PPO on test environments. ExDA preserve the training performance of PPO rather than InDA, this shows that the augmentation can disturb the training. In addition, diverse backgrounds increase the difficulty of training as shown in Oracle.

Table 2. Comparison generalization on unseen backgrounds: Both ExDA and InDA improve the generalization, however, ExDA outperforms InDA because of gap of training performance.
provide a guarantee of improving generalization. In Table 3, ExDA improve generalization than PPO. However, InDA is better than ExDA despite low training performance in Table 3. The number of data is important for generalization of the level (Cobbe et al., 2020), and augmentation during training can increase observations’ diversity. Thus, augmentation after training is hard to get over the limitation of data diversity when the generalization needs the diversity of state distribution.

|     | PPO   | InDA  | ExDA  |
|-----|-------|-------|-------|
| Env | Train | Test  | Train | Test  | Train | Test |
| Maze| 9.75  | 6.79  | 9.63  | 8.01  | 9.72  | 7.74 |
|     | ±0.0327 | ±0.158 | ±0.143 | ±0.288 | ±0.026 | ±0.054 |
| Heist| 9.00  | 4.13  | 8.15  | 5.91  | 8.79  | 5.35 |
|     | ±0.513 | ±0.146 | ±0.570 | ±0.516 | ±0.424 | ±0.220 |

Table 3. Comparison the performance: Training with crop in both ExDA and InDA and testing on unseen levels. InDA generalize better than ExDA, despite ExDA almost transfer the training performance of PPO.

Preserving the training performance after regularization. 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. In Fig 5, we compare the results of training and test performance with Drac (Raileanu et al., 2020), Rand-FM (Lee et al., 2019), Rad (Laskin et al., 2020) when training 5M with each method after training 20M with PPO.

Every training curves decline immediately after starting the train at 20M in Fig 5(a). 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 in a different direction from the optimization direction in 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 (Raileanu et al., 2020) and SODA (Hansen & Wang, 2020). However, InDA is more stable than the others because we distill the fixed policy and value in InDA. It does the stable training through conserving the policy on non-augmented observations during optimizing for augmented data. We confirm similar results on the other environments and compare them in the supplementary materials.

5. Discussion

We have studied the time-sensitivity of applying visual augmentation for RL, which is in contrast to that for SL, where such a difference can be explained with non-stationary data generation in RL. In particular, if the regularization imposed by augmentation is useful only for testing, it is better to procrastinate the augmentation to the end of RL training than to use it for the entire learning in terms of sample and computation complexity since it can disturb the RL training. However, an augmentation providing regularization useful in training obviously needs to be used during the whole training period to fully utilize its benefit in terms of not only generalization but also data efficiency. We believe that our findings provide useful insights not only for auto-augmentation adjusting the use of augmentation round-by-round, where DA at the end of RL training would provide substantial gains as ExDA does. However, it is still open to design auto-augmentation for RL as the gain from augmentation has highly non-stationary, and thus its evaluation is challenging.

References

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

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

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

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

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

Erhan, D., Courville, A., Bengio, Y., and Vincent, P. Why does unsupervised pre-training help deep learning? In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 201–208. JMLR Workshop and Conference Proceedings, 2010.

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

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

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

Golatkar, A. S., Achille, A., and Soatto, S. 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*, pp. 10678–10688, 2019.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.
Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

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

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. 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)*, pp. 23–30. IEEE, 2017.

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

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

You, K., Long, M., Wang, J., and Jordan, M. I. How does learning rate decay help modern neural networks?, 2019.
A. Modified Procgen Environments

This section explains Modified Procgen Environments, which is designed to verify different types of generalization, backgrounds, and levels. Open AI Procgen environments (Cobbe et al., 2020) 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`. Furthermore, We fix the exit wall choice and enemy theme in Dodgeball. We describe the usage themes in each environments, 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 and other baselines. We train the agent with IMPALA-CNN (Espeholt et al., 2018) in every experiment.

B.1. InDA

We use PPO (Schulman et al., 2017) as base RL algorithms, For data efficiency, we store the observations during RL training in buffer $D_O$. Before DA phase, we also make policy buffer $D_\Pi$, value function buffer $D_V$ and augmented observation buffer $D_\phi$ for distillation, because we only use one network model. We randomly sample pairs of $(o, \pi, V)$ from buffer, and minimize loss function $L_{DA}(\theta)$. 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 $l_{DA} = 10^{-4}$ and interval $I = 5$. We fix the buffer size $D_O = 40960$, because we collect the observations during five RL phases $(5 \times 256 \times 32)$. We describe the every hyperparameter in Table 4.

B.2. ExDA

In ExDa, we generate and store $(o, \pi, V)$ using $f_{\theta_{da}}$ in buffer $D$. 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 in Table 5:
Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning

| Hyperparamter               | Value       |
|-----------------------------|-------------|
| $\gamma$                   | 0.999       |
| $\lambda$                  | 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              | $5 \times 10^{-4}$ |
| Interval $I$               | 5           |
| Size of $D_O$              | 40960       |
| # Epochs of DA             | 3           |
| Learning rate of DA $l_{DA}$ | $1 \times 10^{-4}$ |
| Image transformation $\phi$ | Any augmentation |

*Table 4. Hyperparameter for InDA*

| Hyperparamter               | Value       |
|-----------------------------|-------------|
| Size of $D_O$               | 0.5M        |
| # Epoch                     | 30          |
| # Minibatches per epoch     | 4096        |
| Learning rate               | $1 \times 10^{-3}$ |
| # Workers                   | 1           |
| Optimizer                   | Adam optimizer |
| Image transformation $\phi$ | Any augmentation |

*Table 5. Hyperparameter for ExDA*

B.3. Baselines

We compare ExDA and InDA with PPO (Cobbe et al., 2020), DrAC (Raileanu et al., 2020), Rand-FM (Lee et al., 2019), RAD (Laskin et al., 2020). Every baseline are based on PPO (Cobbe et al., 2020) and we adopt the implementation of PPO in (Cobbe et al., 2020).

- DrAC (Raileanu et al., 2020) regularizes both policy and value function as self-supervised learning. Regularization term have hyperparameter $\alpha_r$ for ratio with PPO objective. We use the hyperparameter recommended by the author.

- Rand-FM (Lee et al., 2019) is composed with random convolution networks and feature matching. They also need hyperparameter $\beta$ for ratio between feature matching and PPO objective. We use same $\beta$ with author.

- RAD (Laskin et al., 2020) naively use augmented observations in state distribution. Thus, there are no additional hyperparameters.

We describe the hyperparameter of baselines in Table 6.
| Hyperparameter          | Value  |
|------------------------|--------|
| $\gamma$               | 0.999  |
| $\lambda$              | 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          | $5 \times 10^{-4}$ |
| $\alpha_r$ (DrAC)      | 0.1    |
| $\beta$ (Rand-FM)      | 0.002  |

Table 6. Hyperparameter for baseline algorithms

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. (2019) (*random convolution*), Laskin et al. (2020) (*cutout color*, *color jitter*) and Raileanu et al. (2020) (*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:

![Examples of visual augmentations](image-url)
D. Robustness in loss function change

In this sections, we show the training and test performance of delayed augmentations with other baselines such as DrAC (Raileanu et al., 2020), Rand-FM (Lee et al., 2019), RAD (Laskin et al., 2020). We use random convolution and crop as data augmentation methods, and we do not compare with RAD when we use crop in Fig 9 and Fig 10. The crop method used in our paper do not work well in RAD, because they use a different crop method with (Raileanu et al., 2020) in their paper (Laskin et al., 2020). As mentioned in Fig 5, InDA is more stable than others in training, and it affects generalization performance.

![Figure 7](image1.png)

*Figure 7. Comparison training performance of delayed augmentation with other baselines when using random convolution as an augmentation method.*

![Figure 8](image2.png)

*Figure 8. Comparison test performance of delayed augmentation with other baselines when using random convolution as an augmentation method.*
Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning

Figure 9. Comparison training performance of delayed augmentation with DrAC when using *crop* as an augmentation method.

Figure 10. Comparison test performance of delayed augmentation with DrAC when using *crop* as an augmentation method.
E. Ablation study of ExDA

E.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 Sec 3.3. The ExDA does not have to minimize $L_{VD}$ because the value function is useless after RL training. The below results show that the $L_{VD}$ 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. (2020) argued that the non-stationarity causes the reduction of generalization. However, the re-initialization is not critical in test performance, as shown in Fig 12. Moreover, sometimes re-initialization makes it difficult to distill training performance such as Fruitbot and Ninja in Fig 11. We use random convolution as an augmentation method in here.

![Graphs showing the training performance of ExDA with re-initialization or regularization with value function.](image)

*Figure 11. Training performance of ExDA with re-initialization or regularization with value function.*

![Graphs showing the test performance of ExDA with re-initialization or regularization with value function on unseen backgrounds.](image)

*Figure 12. Test performance of ExDA with re-initialization or regularization with value function on unseen backgrounds.*
E.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 8, 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 about computational complexity.

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

Table 7. 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 | 7.94 | 7.54 | 7.48 | 7.98 | 7.67 |
|       | ±0.19 | ±0.19 | ±0.158 | ±0.154 | ±0.148 | ±0.155 |
| Ninja | 6.29 | 6.5 | 5.56 | 5.73 | 6.27 | 5.94 |
|       | ±0.19 | ±0.19 | ±0.158 | ±0.154 | ±0.148 | ±0.155 |
| Climber | 6.96 | 7.28 | 7.06 | 6.89 | 6.8 | 5.45 |
|       | ±0.65 | ±0.35 | ±0.541 | ±0.237 | ±0.441 | ±0.383 |

Table 8. Test performance of agents, which is trained on easy mode with random convolution.

E.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.

F. Time matter in training

This section shows every result of Fig 3 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 G. 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).
F.1. Random convolution

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

Figure 14. Comparison 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.
F.2. Crop

Figure 15. Comparison 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 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.
F.3. Color jitter

Figure 17. Comparison 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 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.
F.4. Gray

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

Figure 20. Comparison 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.

F.5. Cutout color

Figure 21. Comparison training performance according to usage period of augmentation with InDA (cutout color): Cutout color impedes the training, especially Chaser and Dodgeball are ruined.
G. Benchmark on Modified Open AI Procgen

We compare the training and test performance on various environments with each augmentation. We also use DrAC (Raileanu et al., 2020), RAD (Laskin et al., 2020), Rand-FM (Lee et al., 2019) as baselines. In every results, 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 (Raileanu et al., 2020)’s crop method which is different with (Laskin et al., 2020).

G.1. Random convolution

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

Table 9. Training performance benchmark on easy with random convolution.
Table 10. Test performance benchmark on unseen backgrounds (easy, random convolution).

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

Table 11. Training performance benchmark on easybg with random convolution.

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

Table 12. Test performance benchmark on unseen backgrounds (easybg, random convolution).
## G.2. Crop

| Easybg  | PPO     | DrAC   | RAD   | InDA   | ExDA   |
|---------|---------|--------|-------|--------|--------|
| Bigfish | 14.08   | 15.92  | 5.05  | **19.35** | 11.07  |
|         | ±2.229  | ±1.535 | ±3.718| ±2.792 | ±3.683 |
| Chaser  | 5.63    | 3.97   | 1.24  | **6.52** | 4.81   |
|         | ±0.467  | ±0.642 | ±0.253| ±0.825 | ±0.325 |
| Dodgeball | 7.71   | 10.74  | 1.23  | **12.74** | 6.74   |
|         | ±0.678  | ±0.711 | ±0.944| ±1.729 | ±0.815 |
| Heist   | 9       | 7.58   | 4.53  | 8.15   | 8.79   |
|         | ±0.513  | ±0.11  | ±0.266| ±0.37  | ±0.424 |
| Maze    | **9.75**| 9.03   | 3.95  | 9.63   | 9.72   |
|         | ±0.033  | ±0.348 | ±3.418| ±0.143 | ±0.026 |
| Plunder | 7.18    | 10.73  | 0     | **10.29** | 6.59   |
|         | ±0.73   | ±1     | ±0    | ±0.285 | ±1.108 |
| Avg     | 1.00    | 1.08   | 0.28  | **1.25** | 0.91   |

*Table 13. Training performance benchmark on easybg with crop.*

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

*Table 14. Test performance benchmark on unseen levels (easybg, crop).*

## G.3. Color jitter

| Easy    | PPO     | DrAC   | RAD   | InDA   | ExDA   |
|---------|---------|--------|-------|--------|--------|
| Climber | 8.5     | **9.33**| 8.64  | 9.43   | 8.18   |
|         | ±0.573  | ±0.212 | ±0.156| ±0.21  | ±0.45  |
| Jumper  | 8.54    | 8.64   | 8.63  | **8.92** | 8.44   |
|         | ±0.22   | ±0.135 | ±0.17 | ±0.174 | ±0.185 |
| Ninja   | 7.48    | 8.69   | 8.24  | **9.23** | 7.37   |
|         | ±0.324  | ±0.331 | ±0.251| ±0.081 | ±0.212 |
| Avg     | 1.00    | 1.09   | 1.04  | **1.13** | 0.98   |

*Table 15. Training performance benchmark on easy with color jitter.*
| Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning |
|---------------------------------------------------------------|
| **Easy** | **PPO** | **DrAC** | **RAD** | **InDA** | **ExDA** |
|-----------|---------|----------|---------|----------|----------|
| Climber   | 6.92    | 8.53     | 8.37    | **8.66** | 8.14     |
|           | ±0.76   | ±0.42    | ±0.02   | ±0.24    | ±0.477   |
| Jumper    | 6.89    | 7.58     | 7.86    | **7.97** | 7.25     |
|           | ±0.22   | ±0.053   | ±0.297  | ±0.292   | ±0.131   |
| Ninja     | 6.39    | 6.79     | 7.31    | **7.57** | 6.2      |
|           | ±0.58   | ±0.32    | ±0.613  | ±0.555   | ±0.085   |
| Avg       | 1.00    | 1.13     | 1.16    | **1.2**  | 1.07     |

*Table 16. Test performance benchmark on unseen backgrounds (easy, color jitter).*

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

*Table 17. Training performance benchmark on easbg with color jitter.*

| Easybg    | **PPO** | **Oracle** | **DrAC** | **RAD** | **InDA** | **ExDA** |
|-----------|---------|------------|----------|---------|----------|----------|
| Climber   | 1.82    | **9.85**   | **5.05** | 4.31    | 4.25     | 4.34     |
|           | ±0.605  | ±0.298     | ±0.407   | ±0.492  | ±0.338   | ±0.856   |
| Coinrun   | 5.42    | 7.2        | 6.46     | 6.47    | **7.13** | 6.53     |
|           | ±0.744  | ±0.195     | ±0.526   | ±0.194  | ±0.372   | ±0.375   |
| Fruitbot  | **11.78**| **29.69**  | **9.49** | 8.51    | 10.88    | **18**   |
|           | ±1.949  | ±0.619     | ±8.098   | ±1.941  | ±2.263   | ±7.442   |
| Heist     | **4.79** | **7.33**   | **5.65** | **5.39**| **5.66** | **5.43** |
|           | ±0.323  | ±0.308     | ±0.984   | ±0.745  | ±0.271   | ±0.508   |
| Jumper    | **3.3** | **8.6**    | **5.65** | **5.67**| **5.81** | **5.31** |
|           | ±0.467  | ±0.132     | ±0.09    | ±0.953  | ±0.369   | ±0.351   |
| Maze      | **6.52**| **8.08**   | **8.22** | **8.26**| 8.35     | **8.65** |
|           | ±0.304  | ±0.101     | ±0.455   | ±0.175  | ±0.238   | ±0.017   |
| Ninja     | **3.56**| **7.56**   | **4.22** | **4.18**| **4.34** | **4.07** |
|           | ±0.363  | ±0.286     | ±0.487   | ±0.475  | ±0.345   | ±0.332   |
| Avg       | 1.00    | **2.4**    | **1.44** | **1.37**| 1.43     | **1.48** |

*Table 18. Test performance benchmark on unseen backgrounds (easybg, color jitter).*
G.4. Gray

| Easybg  | PPO   | Oracle | DrAC  | RAD  | InDA | ExDA |
|---------|-------|--------|-------|------|------|------|
| Climber | 12.31 | 9.85   | 11.12 | 11.84| 11.9 | 12.06|
| ±0.092  | ±0.298| ±0.26  | ±0.505| ±0.115| ±0.03|
| Coinrun | 9.61  | 7.2    | 9.53  | 9.49 | 9.74 | 9.48 |
| ±0.074  | ±0.195| ±0.135 | ±0.188| ±0.046| ±0.08|
| Fruitbot| 30.2  | 29.69  | 30.01 | 29.6 | 28.03| 29.32|
| ±0.691  | ±0.619| ±0.572 | ±0.27 | ±0.994| ±0.937|
| Heist   | 8.82  | 7.33   | 6.24  | 6.53 | 5.51 | 8.51 |
| ±0.523  | ±0.308| ±0.214 | ±0.474| ±0.146| ±0.225|
| Jumper  | 8.54  | 8.67   | 8.91  | 8.93 | 9.18 | 8.95 |
| ±0.022  | ±0.132| ±0.19  | ±0.247| ±0.18  | ±0.075|
| Maze    | 9.75  | 8.08   | 9.46  | 9.48 | 9.2  | 9.75 |
| ±0.035  | ±0.101| ±0.192 | ±0.08 | ±0.367| ±0.087|
| Ninja   | 9.74  | 7.56   | 9.73  | 9.61 | 9.6  | 9.72 |
| ±0.087  | ±0.286| ±0.045 | ±0.096| ±0.081| ±0.021|
| Avg     | 1     | 0.85   | 0.93  | 0.94 | 0.95 | 0.99 |

Table 19. Training performance benchmark on easybg with gray.

| Easybg  | PPO   | Oracle | RAD  | DrAC  | InDA | ExDA |
|---------|-------|--------|------|-------|------|------|
| Climber | 1.82  | 9.85   | 1.75 | 1.81  | 1.24 | 2.45 |
| ±0.605  | ±0.298| ±0.654 | ±0.211| ±0.502| ±0.727|
| Coinrun | 5.42  | 7.2    | 5.34 | 5.31  | 6.05 | 5.79 |
| ±0.744  | ±0.195| ±0.751 | ±0.501| ±0.465| ±0.061|
| Fruitbot| 11.78 | 29.69  | 17.57| 15.47 | 15.12| 15.81|
| ±1.949  | ±0.619| ±0.191 | ±1.449| ±0.958| ±0.11|
| Heist   | 4.79  | 7.33   | 5.43 | 5.15  | 4.32 | 5.1  |
| ±0.323  | ±0.308| ±0.18  | ±0.172| ±0.112| ±0.504|
| Jumper  | 6.89  | 8.67   | 2.7  | 4.07  | 3.55 | 4.47 |
| ±0.223  | ±0.132| ±0.894 | ±0.46 | ±0.992| ±0.415|
| Maze    | 6.52  | 8.08   | 7.77 | 7.93  | 7.67 | 8.33 |
| ±0.304  | ±0.101| ±0.611 | ±0.104| ±0.312| ±0.119|
| Ninja   | 3.56  | 7.56   | 3.72 | 4.91  | 4.02 | 4.03 |
| ±0.363  | ±0.286| ±0.131 | ±0.062| ±0.666| ±0.071|
| Avg     | 1     | 2.2    | 1.03 | 1.04  | 0.97 | 1.13 |

Table 20. Test performance benchmark on unseen backgrounds (easybg, gray).

| Easy  | PPO   | DrAC  | RAD  | InDA | ExDA |
|-------|-------|-------|------|------|------|
| Climber | 8.5   | 6.95  | 7.55 | 7.22 | 8.05 |
| ±0.575 | ±0.347| ±0.256| ±0.312| ±0.461|
| Jumper | 8.5   | 8.4   | 8.58 | 8.85 | 8.5  |
| ±0.224 | ±0.224| ±0.199| ±0.015| ±0.242|
| Ninja  | 7.48  | 6.67  | 7.1  | 8.91 | 7.05 |
| ±0.324 | ±0.435| ±0.718| ±0.165| ±0.24|
| Avg    | 1     | 0.95  | 1.026| 0.96 |

Table 21. Training performance benchmark on easybg with gray.
In Table 22, we see the test performance benchmark on unseen backgrounds for various games using different data augmentation techniques.

### G.5. Cutout color

#### Table 23. Training performance benchmark on easybg with cutout color.

| Easybg | PPO | Oracle | DrAC | RAD | InDA | ExDA |
|--------|-----|--------|------|-----|------|------|
| Climber | 12.31 | 9.85 | 11.92 | 8.26 | 11.76 | 12.07 |
| ±0.092 | ±0.298 | ±0.158 | ±0.663 | ±0.027 | ±0.127 |
| Coinrun | 9.61 | 7.2 | 9.23 | 8.07 | 9.7 | 9.39 |
| ±0.074 | ±0.195 | ±0.323 | ±0.645 | ±0.084 | ±0.012 |
| Fruitbot | 30.2 | 29.69 | 29.73 | 29.2 | 27.18 | 28.95 |
| ±0.691 | ±0.619 | ±0.898 | ±0.64 | ±1.302 | ±0.907 |
| Heist | 8.82 | 7.33 | 8.47 | 6.25 | 6.1 | 8.65 |
| ±0.523 | ±0.308 | ±0.397 | ±0.704 | ±0.693 | ±0.21 |
| Jumper | 8.97 | 8.67 | 8.87 | 8.75 | 9.1 | 9.91 |
| ±0.075 | ±0.132 | ±0.123 | ±0.131 | ±0.081 | ±0.053 |
| Maze | 9.75 | 8.08 | 9.41 | 9.17 | 9.27 | 9.74 |
| ±0.035 | ±0.101 | ±0.134 | ±0.118 | ±0.125 | ±0.133 |
| Ninja | 9.74 | 7.56 | 9.65 | 7.17 | 9.72 | 9.7 |
| ±0.087 | ±0.286 | ±0.138 | ±1.993 | ±0.02 | ±0.02 |
| Bigfish | 13.89 | 13.22 | 2.54 | 5.19 | 1.95 | 11.22 |
| ±3.127 | ±1.488 | ±0.13 | ±3.658 | ±0.311 | ±3.66 |
| Chaser | 5.49 | 3.04 | 2.88 | 1.98 | 3.34 | 5 |
| ±0.562 | ±0.183 | ±0.699 | ±0.112 | ±0.755 | ±0.187 |
| Dodgeball | 7.76 | 5.74 | 5.71 | 5.98 | 2.79 | 6.57 |
| ±0.859 | ±1.118 | ±1.008 | ±0.103 | ±1.612 | ±0.693 |
| Plunder | 7.15 | 6.05 | 5.43 | 4.34 | 4.92 | 6.87 |
| ±0.95 | ±0.38 | ±0.082 | ±0.24 | ±0.625 | ±1.255 |
| Avg | 1.00 | 0.82 | 0.82 | 0.72 | 0.76 | 0.94 |

#### Table 24. Training performance benchmark on easy with cutout color.

| Easy | PPO | Oracle | DrAC | RAD | InDA | ExDA |
|------|-----|--------|------|-----|------|------|
| Climber | 8.5 | 9.85 | 7.69 | 6.67 | **9.02** | 8.02 |
| ±0.575 | ±0.298 | ±0.337 | ±0.381 | ±0.473 | ±0.506 |
| Jumper | 7.48 | 7.56 | 6.28 | 5.6 | **8.57** | 7.41 |
| ±0.324 | ±0.286 | ±0.257 | ±0.276 | ±0.122 | ±0.125 |
| Ninja | 8.54 | 8.67 | 8.45 | 8.32 | **8.93** | 8.53 |
| ±0.22 | ±0.132 | ±0.183 | ±0.051 | ±0.166 | ±0.095 |
| Avg | 1.00 | 1.06 | 0.91 | 0.84 | **1.08** | 0.98 |

### Footnotes

1. **Easy PPO Oracle DrAC RAD InDA ExDA**
2. **Climber 6.92 4.49 5.57 5.11 7.24**
3. ±0.761 ±0.332 ±0.307 ±0.483 ±0.721
4. **Jumper 6.89 5.38 6.59 6.35 6.87**
5. ±0.223 ±0.215 ±0.055 ±0.234 ±0.182
6. **Ninja 6.39 5.67 5.14 6.84 6.01**
7. ±0.585 ±0.318 ±0.628 ±0.206 ±0.651
8. **Avg 1 0.77 0.86 0.91 0.99**
## Time Matters in Using Data Augmentation for Vision-Based Deep Reinforcement Learning

| Easybg | PPO | Oracle | DrAC | RAD | InDA | ExDA |
|--------|-----|--------|------|-----|------|------|
| Climber | 1.82 | 9.85 | 3.54 | 3.97 | 3.4 | 4.29 |
| ±0.605 | ±0.298 | ±0.164 | ±0.999 | ±0.645 | ±0.154 |
| Coinrun | 5.42 | 7.2 | 5.87 | 5.93 | 6.2 | 6.41 |
| ±0.744 | ±0.195 | ±0.251 | ±0.061 | ±0.357 | ±0.131 |
| Fruitbot | 11.78 | 29.69 | 18.18 | 19.24 | 17.69 | 17.7 |
| ±1.949 | ±0.619 | ±3.744 | ±3.385 | ±4.026 | ±0.888 |
| Heist | 4.79 | 7.33 | 6.6 | 5.76 | 4.97 | 7.51 |
| ±0.323 | ±0.308 | ±0.092 | ±0.551 | ±0.33 | ±0.119 |
| Jumper | 3.3 | 8.67 | 4.99 | 5.48 | 5.43 | 6.02 |
| ±0.467 | ±0.132 | ±0.114 | ±0.28 | ±1.116 | ±0.235 |
| Maze | 6.52 | 8.08 | 7.33 | 7.66 | 7.01 | 7.83 |
| ±0.304 | ±0.101 | ±0.223 | ±0.243 | ±0.17 | ±0.22 |
| Ninja | 3.56 | 7.56 | 4.29 | 3.96 | 3.75 | 3.76 |
| ±0.363 | ±0.286 | ±0.245 | ±0.152 | ±0.333 | ±0.348 |
| Bigfish | 3.4 | 13.22 | 1.29 | 2.5 | 1.29 | 4.49 |
| ±0.487 | ±1.488 | ±0.08 | ±2.331 | ±0.152 | ±0.776 |
| Chaser | 0.91 | 3.04 | 1.08 | 1.13 | 1.68 | 1.73 |
| ±0.061 | ±0.183 | ±0.038 | ±0.157 | ±0.305 | ±0.698 |
| Dodgeball | 2.17 | 5.74 | 3.92 | 4.02 | 1.97 | 4.37 |
| ±0.53 | ±1.118 | ±0.53 | ±0.345 | ±1.098 | ±0.527 |
| Plunder | 6.87 | 6.05 | 5.27 | 4.77 | 4.71 | 6.45 |
| ±0.933 | ±0.58 | ±0.208 | ±0.612 | ±0.622 | ±1.232 |
| Avg | 1.00 | 2.51 | 1.27 | 1.33 | 1.19 | 1.53 |

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

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

*Table 26. Test performance benchmark on unseen backgrounds (easy, cutout color).*
| Easybg | PPO   | DrAC  | RAD  | InDA  | ExDA  |
|-------|-------|-------|------|-------|-------|
| Climber | 11.14 | 10.77 | 7.26 | 9.45  | 10.75 |
| ±0.077 | ±0.279| ±0.843| ±0.193| ±0.114|
| Coinrun | 8.64  | 8.36  | 6.89 | 7.76  | 8.32  |
| ±0.05  | ±0.348| ±0.503| ±0.096| ±0.224|
| Fruitbot | 28.26 | 26.88 | 26.22| 23.79 | 26.33 |
| ±0.461| ±1.276| ±1.258| ±0.971| ±0.894|
| Heist | 4.07 | 3.92 | 2.27 | 2.15 | 3.93 |
| ±0.07 | ±0.276| ±0.448| ±0.553| ±0.184|
| Jumper | 7.38 | 7.3 | 6.98 | 6.68 | 7.25 |
| ±0.15 | ±0.195| ±0.199| ±0.24 | ±0.117|
| Maze | 6.8 | 6.84 | 6.04 | 5.91 | 6.17 |
| ±0.2 | ±0.137| ±0.258| ±0.03 | ±0.162|
| Ninja | 8.56  | 8.63  | 6.28 | 7.81  | 8.34  |
| ±0.061| ±0.132| ±1.866| ±0.21 | ±0.119|
| Bigfish | 7.16 | 0.91 | 2.29 | 0.95 | 6.04 |
| ±2.263| ±0.037| ±2.306| ±0.06 | ±2.783|
| Chaser | 4.54 | 2.61 | 1.8 | 2.47 | 4.22 |
| ±0.503| ±0.509| ±0.12 | ±0.506 | ±0.331|
| Dodgeball | 3.78 | 2.71 | 2.53 | 1.26 | 2.82 |
| ±0.823| ±0.362| ±0.135| ±0.48 | ±0.593|
| Plunder | 5.99 | 5.08 | 4.07 | 4.55 | 5.83 |
| ±0.814| ±0.305| ±0.479| ±0.393| ±1.061|
| Avg | 1.00 | 0.83 | 0.69 | 0.69 | 0.93 |

Table 27. Test performance benchmark on unseen levels (easybg, cutout color).

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

Table 28. Test performance benchmark on unseen levels (easy, cutout color).