A SIMPLE RANDOMIZATION TECHNIQUE FOR GENERALIZATION IN DEEP REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Deep reinforcement learning (RL) agents often fail to generalize to unseen environments (yet semantically similar to trained ones), particularly when they are trained on high-dimensional state spaces such as images. In this paper, we propose a simple technique to improve a generalization ability of deep RL agents by introducing a randomized (convolutional) neural network that randomly perturbs input observations. It enables trained agents to adapt to new domains by learning robust features which are invariant across varied and randomized input observations. Furthermore, we propose an inference method based on Monte Carlo approximation to reduce the variance induced by this randomization. The proposed method significantly outperforms conventional techniques including various regularization and data augmentation techniques across 2D CoinRun, 3D DeepMind Lab exploration, and 3D robotics control tasks.

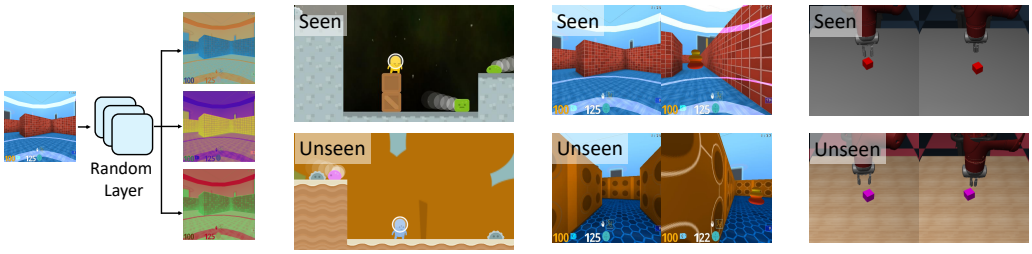
1 INTRODUCTION

Deep reinforcement learning (RL) has been applied to various applications including board games (e.g., Go (Silver et al., 2017) and Chess (Silver et al., 2018)), video games (e.g., Atari games (Mnih et al., 2015) and StarCraft (Vinyals et al., 2017)), and complex robotics control tasks (Tobin et al., 2017; Ren et al., 2019). However, it has been evidenced in recent years that deep RL agents often struggle to generalize to new environments, even when they are semantically similar to the trained ones (Farebrother et al., 2018; Zhang et al., 2018b; Gamrian & Goldberg, 2019; Cobbe et al., 2019). For example, RL agents that learned a near-optimal policy for training levels in a video game fail to perform well in unseen levels (Cobbe et al., 2019), while a human can seamlessly generalize across such similar tasks. Namely, the RL agents often overfit to training environments, thus the lack of generalization ability can raise concerns when deploying them in many applications such as health care (Chakraborty & Murphy, 2014) and finance (Deng et al., 2016).

The generalization of RL agents can be characterized by visual changes (Cobbe et al., 2019; Gamrian & Goldberg, 2019), different dynamics (Packer et al., 2018) and different structures (Beattie et al., 2016; Wang et al., 2016). In this paper, we focus on the generalization across tasks where the trained agents take various unseen visual patterns at test time, e.g., different styles of backgrounds, floors, and other objects (see Figure 1). We also found that RL agents completely fail to perform when facing small visual changes[1] because it is challenging to learn generalizable representations from high-dimensional input observations such as images.

To improve the generalization, several strategies such as regularization (Farebrother et al., 2018; Zhang et al., 2018b; Cobbe et al., 2019) and data augmentation (Tobin et al., 2017; Ren et al., 2019) have been proposed in the literature (see Section 2 for more details). In particular, Tobin et al. (2017) showed that training RL agents in various environments generated by randomizing rendering in a simulator can improve the generalization performance, leading to achieving a good performance in real environments. This implies that RL agents can learn invariant and robust representations if diverse input observations are given during training. However, their method has a limitation that such a physics simulator may not always be available. This motivates our approach of developing a more simple and plausible method applicable to training any deep RL agents.

[1]Examples of the demonstrations on CoinRun and DeepMind Lab exploration task are available at [link-to-video].



(a) Outputs of random layer (b) 2D CoinRun (c) 3D DeepMind Lab (d) 3D Surreal robotics

Figure 1: (a) Examples of randomized inputs (color values in each channel are normalized for visualization) generated by re-initializing the parameters of a random layer. Examples of seen and unseen environments on (b) CoinRun, (c) DeepMind Lab and (d) Surreal robotics control.

The main contribution of this paper is to develop a simple randomization technique for improving the generalization ability across tasks with various unseen visual patterns. Our main idea is to utilize random (convolutional) networks to generate randomized inputs (see Figure 1(a)), and train RL agents (or their policy) by feeding them. Specifically, by re-initializing the parameters of random networks at every iteration, we encourage the agents to be trained under a broad range of perturbed low-level features, e.g., various textures, colors, or shapes. We discover that the proposed simple idea surprisingly helps RL agents to learn more generalizable features which are invariant in unseen environments (see Figure 3) than conventional regularization (Srivastava et al., 2014; Ioffe & Szegedy, 2015) and data augmentation (Cobbe et al., 2019; Cubuk et al., 2019) techniques. Here, we also provide an inference technique based on Monte Carlo approximation, which stabilizes the performance by reducing the variance incurred from our randomization method at test time.

We demonstrate the effectiveness of the proposed method on 2D CoinRun (Cobbe et al., 2019) game, 3D DeepMind Lab exploration task (Beattie et al., 2016), and 3D robotics control task (Fan et al., 2018). For evaluation, we measure the performance of trained agents in unseen environments which have different visual and geometrical patterns (e.g., different styles of backgrounds, objects, and floors). In other words, the trained agents always encounter unseen inputs at test time. We note that learning invariant and robust representations against such changes is essential to generalize to unseen environments. In our experiments, the proposed method significantly reduces the generalization gap in unseen environments unlike conventional regularization and data augmentation techniques. For example, compared to the agents learned with cutout (DeVries & Taylor, 2017) data augmentation methods proposed by Cobbe et al. (2019), ours improves the success rates from 38.0% to 55.7% under 2D CoinRun, the total score from 55.4 to 358.2 for 3D DeepMind Lab, and the total score from 31.3 to 356.8 for Surreal robotics control task.

We think that our finding can also be influential to study other generalization domains such as tasks with different dynamics (Packer et al., 2018), as well as to solve real-world problems such as sim-to-real transfer (Tobin et al., 2017).

2 Related Work

Generalization in deep RL. Recently, the generalization performance of RL agents has been investigated by splitting training and test environments using random seeds (Zhang et al., 2018a) and distinct sets of levels in video games (Machado et al., 2018; Cobbe et al., 2019). A major direction for improving the generalization ability of deep RL algorithms focuses on regularization techniques. Farebrother et al. (2018) and Cobbe et al. (2019) showed that regularization methods can improve the generalization performance of RL agents using different game modes of Atari (Machado et al., 2018) and procedurally generated arcade environments called CoinRun, respectively. On the other hand, data augmentation techniques have also been explored for improving generalization. Tobin et al. (2017) proposed a domain randomization method which generates simulated inputs by randomizing rendering in the simulator. Motivated by this, Cobbe et al. (2019) proposed a data augmentation method by modifying the cutout method (DeVries & Taylor, 2017). We remark that our method enjoys an orthogonal usage with the prior methods, i.e., our method can be combined with the prior methods to further improve the generalization performance.

Random networks for deep RL. Random networks have been utilized in several ways for different purposes in deep RL. Burda et al. (2019) utilized a randomly initialized neural network to define an intrinsic reward for visiting unexplored states in hard exploration problems. By learning to predict the reward from the random network, the agent can recognize unexplored states. Osband et al. (2018) studied a way to improve ensemble-based approaches by adding a randomized network to each ensemble member. They showed that it improves uncertainty estimation and efficient exploration in deep RL. Our method is different from those prior works in that our random network is for improving the generalization ability of RL agents.

Transfer learning. Generalization is also closely related to transfer learning (Parisotto et al., 2016; Rusu et al., 2016a;b), which is used to improve the performance on a target task by transferring the knowledge from a source task. However, unlike supervised learning, it has been observed that simply fine-tuning a model pre-trained on the source task for adapting to the target task is not useful in deep RL (Farebrother et al., 2018; Gamrian & Goldberg, 2019). To handle this issue, Gamrian & Goldberg (2019) proposed a domain transfer method using generative adversarial networks (Goodfellow et al., 2014), and Farebrother et al. (2018) utilized regularization techniques to improve the performance of fine-tuning methods. Higgins et al. (2017) proposed a multi-stage RL, which first learns to extract disentangled representations from the input observation, and then trains the agents on top of the representations. Unlike those prior works, we focus on the zero-shot performance of each agent at test time, without further fine-tuning the agent's parameters.

3 Network randomization technique for generalization

We consider a standard reinforcement learning (RL) framework where an agent interacts with an environment in discrete time. Formally, at each timestep t, the agent receives a state s_t from the environment[2] and chooses an action a_t based on its policy π. The environment returns a reward r_t and the agent transitions to the next state s_{t+1}. The return $R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$ is the total accumulated rewards from timestep t with a discount factor $\gamma \in [0, 1)$. The goal of RL is then to maximize the expected return from each state, i.e., $\mathbb{E}_{(s,a)}\left[R_t\right]$.

3.1 Training agents using randomized input observations

Our main idea is to introduce a random network f where its parameters ϕ are initialized with a prior distribution, e.g., Xavier normal distribution (Glorot & Bengio, 2010). Instead of the original input s, we train an agent using randomized input $\widehat{s} = f(s; \phi)$. For example, in the case of policy-based methods[3] that directly parameterize the policy π, we optimize the parameters of the policy network θ by minimizing the following loss function:

$$\mathcal{L}_{\texttt{policy}}^{\texttt{random}} = -\log \pi\left(a | f\left(s; \phi\right); \theta\right) R, \tag{1}$$

where R is the total accumulated rewards. By re-initializing the parameters ϕ of the random network per iteration, we train our agents using varied and randomized input observations (see Figure 1(a)). Namely, we generate environments with different visual patterns but with the same semantics by randomizing the networks. One can expect that our RL agents can adapt to new environments by learning invariant representation from them (see Figure 3 for supporting experiments).

To learn more invariant features, we also consider the following feature matching (FM) loss between hidden features from clean and randomized observations:

$$\mathcal{L}_{\texttt{FM}}^{\texttt{random}} = ||h\left(f(s; \phi); \theta\right) - h\left(s; \theta\right)||^2, \tag{2}$$

where $h(\cdot)$ denotes the output of the penultimate layer of policy π. Our intuition on the above loss is that the hidden features from clean and randomized inputs should be pulled together to learn more invariant features against the changes in input observations. Namely, the total loss is

$$\mathcal{L}^{\texttt{random}} = \mathbb{E}_{(s,a)}\left[\mathcal{L}_{\texttt{policy}}^{\texttt{random}} + \beta \mathcal{L}_{\texttt{FM}}^{\texttt{random}}\right], \tag{3}$$

where $\beta > 0$ is a hyper-parameter. The full procedure is summarized in Algorithm 1 in Appendix B.

[2]Throughout this paper, we focus on high-dimensional state spaces, e.g., images.

[3]We focus on the policy-based methods for exposition; however, our method is applicable to the value-based methods as well.

Method	Classification accuracy (%)	
	Train (seen)	Test (unseen)
ResNet-18	95.0 ± 2.4	40.3 ± 1.2
ResNet-18 + GR	96.4 ± 1.8	70.9 ± 1.7
ResNet-18 + CO	95.9 ± 2.3	41.2 ± 1.7
ResNet-18 + IV	91.0 ± 2.0	47.1 ± 15.1
ResNet-18 + CJ	95.2 ± 0.6	43.5 ± 0.3
ResNet-18 + ours	95.9 ± 1.6	**84.4** ± 4.5

Table 1: The classification accuracy (%) on dogs and cats datasets. The results show the mean and standard deviation averaged over 3 runs and the best result is indicated in bold.

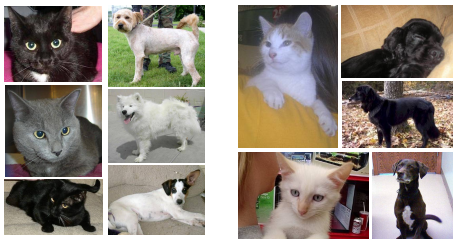


Figure 2: Samples of dogs and cats datasets. The training set consists of bright dogs and dark cats whereas the test set consists of dark dogs and bright cats.

Details of the random networks. We propose to utilize a single-layer convolutional neural network (CNN) as a random network, where its output has the same dimension with the input (see Appendix F for more experimental results about the different types of random networks). To re-initialize the parameters of the random network, we utilize the following mixture of distributions: $P(\phi) = \alpha\mathbb{I}(\phi = \mathbf{I}) + (1-\alpha)\mathcal{N}\left(\mathbf{0}; \sqrt{\frac{2}{n_{\texttt{in}}+n_{\texttt{out}}}}\right)$, where $\mathbf{I}$ is an identity kernel, $\alpha \in [0,1]$ is a positive constant, $\mathcal{N}$ denotes the normal distribution, and $n_{\texttt{in}}, n_{\texttt{out}}$ are the number of input and output channels, respectively. Here, we use clean inputs with probability α because training only randomized inputs would make training too difficult. We use the Xavier normal distribution (Glorot & Bengio, 2010) for randomization because it keeps the variance of the input s and the randomized input $\widehat{s}$ the same. We empirically observe that this distribution makes training more stable.

Removing visual bias. To confirm the desired effects of our method, we conduct an image classification experiment on the dogs and cats database from *Kaggle*.[4] Following the same setup by Kim et al. (2019), we construct datasets with undesirable bias as follows: the training set consists of bright dogs and dark cats while the test set consists of dark dogs and bright cats (see Appendix J for more details). One can expect that a classifier can make a decision based on the undesirable bias, e.g., brightness and color, as it has been observed that CNNs are biased towards texture or color rather than shape (Geirhos et al., 2019). Indeed, Table 1 shows that ResNet-18 (He et al., 2016) struggles to generalize because of overfitting to undesirable bias in training data. To address this issue, several image processing methods (Cubuk et al., 2019) such as grayout (GR), cutout (CO; DeVries & Taylor 2017), inversion (IV) and color jitter (CJ) can be applied (see Appendix E for more details). However, we find that they are not very effective to improve the generalization ability, compared to our method. This confirms that ours changes the visual appearance of attributes and entities in images while effectively keeping the desired and meaningful information. We remark that the prior sophisticated methods (Ganin et al., 2016; Kim et al., 2019) require additional information to eliminate such an undesired bias, while our method does not.[5] Although we mainly focus on RL applications, our idea can also be explorable in this direction.

3.2 Inference methods for small variance

Since the parameter of random networks is drawn from a prior distribution $P(\phi)$, our policy can be interpreted as being modeled by a stochastic neural network: $\pi(a|s;\theta) = \mathbb{E}_{\phi}\left[\pi\left(a|f\left(s;\phi\right);\theta\right)\right]$. Based on this interpretation, one can understand our training procedures (i.e., randomizing the parameters) as training stochastic models using Monte Carlo (MC) approximation (with a single sample per iteration). Inspired by this, at inference or test time, we take an action a by approximating the expectations as follows: $\pi\left(a|s;\theta\right) \simeq \frac{1}{M}\sum_{m=1}^{M}\pi\left(a\middle|f\left(s;\phi^{(m)}\right);\theta\right)$, where $\phi^{(m)} \sim P\left(\phi\right)$ and M is the number of MC samples. In other words, we generate M random inputs for each observation and then aggregate their decisions. We found that this estimator improves the performance of trained agents by approximating the posterior distribution better (see Figure 3(d)).

[4] https://www.kaggle.com/c/dogs-vs-cats

[5] Using the known bias information (i.e., {dark, bright}) and ImageNet pre-trained model, Kim et al. (2019) achieve 90.3%, while our method achieves 84.4% without using both.

4 EXPERIMENTS

In this section, we demonstrate the effectiveness of the proposed method on 2D CoinRun (Cobbe et al., 2019), 3D DeepMind Lab exploration (Beattie et al., 2016), and 3D robotics control task (Fan et al., 2018). To evaluate the generalization ability, we measure the performance of trained agents in unseen environments which consist of different styles of backgrounds, objects, and floors. Due to the space limitation, we provide more detailed experimental setups and results in the appendix.

4.1 BASELINES AND IMPLEMENTATION DETAILS

For CoinRun and DeepMind Lab experiments, similar to Cobbe et al. (2019), we take the CNN architecture used in IMPALA (Espeholt et al., 2018) as the policy network, and use Proximal Policy Optimization (PPO) (Schulman et al., 2017) method to train the agents.[6] At each timestep, agents get an observation frame of size 64×64 as an input (resized from the raw observation of size 320×240 in the case of DeepMind Lab), and we collect the trajectories with the 256-step rollout for training. For Surreal robotics experiments, similar to Fan et al. (2018), we take the hybrid of CNN and long short-term memory (LSTM) architecture as the policy network and use a distributed version of PPO (i.e., actors collect a massive amount of trajectories and centralized learner updates the model parameters using PPO) to train the agents.[7] We measure the performance in the unseen environment for every 10M timesteps and report the mean and standard deviation across 3 runs.

Our proposed method, which augments PPO with random networks and feature matching (FM) loss (denoted PPO + ours), is compared with several regularization and data augmentation methods. As regularization methods, we compare dropout (DO; Srivastava et al. 2014), L2 regularization (L2), and batch normalization (BN; Ioffe & Szegedy 2015). For those methods, we use the hyperparameters suggested in Cobbe et al. (2019), which are empirically shown to be effective: a dropout probability of 0.1 and a coefficient of 10^{-4} for L2 regularization. We also consider various data augmentation methods: a variant of cutout (CO; DeVries & Taylor 2017) proposed in Cobbe et al. (2019), grayout (GR), inversion (IV), and color jitter (CJ) by adjusting brightness, contrast, and saturation (see Appendix E for more details). For our method, we use $\beta = 0.002$ for the weight of the FM loss, $\alpha = 0.1$ for the probability of skipping the random network, $M = 10$ for MC approximation, and a single-layer CNN with the kernel size of 3 as a random network.

4.2 EXPERIMENTS ON COINRUN

Task description. In this task, an agent is located at the leftmost side of the map and the goal is to collect the coin located at the rightmost side of the map within 1000 timesteps. The agent observes its surrounding environment in the third-person view, where the agent is always located at the center of the observation. CoinRun contains an arbitrarily large number of levels which generated deterministically from a given seed. In each level, the style of background, floor, and obstacles is randomly selected from available themes (34 backgrounds, 6 grounds, 5 agents, and 9 moving obstacles). Some obstacles and pitfalls are distributed between the agent and coin, and a collision with them results in the agent's immediate death. For evaluation, we measure the success rates, which correspond to the number of collected coins divided by the number of played levels.

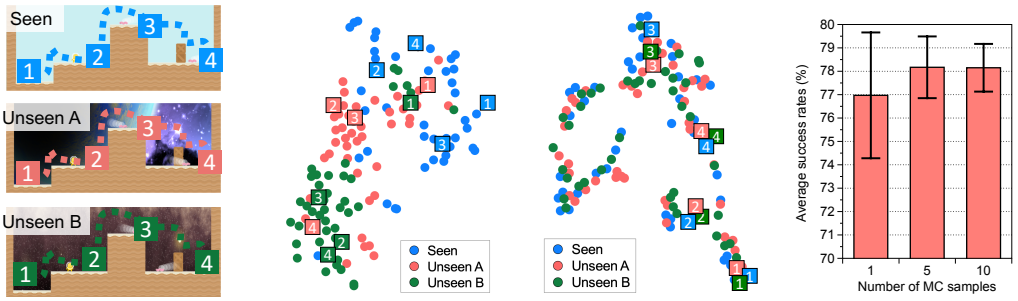
Ablation study on small-scale environments. First, we train agents on one level for 100M timesteps and measure the performance in unseen environments by only changing the style of backgrounds as shown in Figure 3(a). One can note that these visual changes are not significant to the game's dynamics, and the agent should achieve a high success rate if it can generalize well. However, Table 2 shows that all baseline agents fail to generalize to unseen environments, while they achieve a near-optimal performance in the seen environment. This shows that regularization techniques have no significant impact on improving the generalization ability on the contrary to the conventional belief. Even though data augmentation techniques such as cutout (CO) and color jitter (CJ) slightly improve the performance, our proposed method is the most effective one because ours can produce a diverse novelty in attributes and entities. Here, a caveat is that training with randomized inputs might degrade the training performance. However, we found that it is not the

[6] We used a reference implementation in https://github.com/openai/coinrun.

[7] We used a reference implementation with two actors in https://github.com/SurrealAI/surreal.

		PPO	PPO + DO	PPO + L2	PPO + BN	PPO + CO	PPO + IV	PPO + GR	PPO + CJ	PPO + ours Rand	PPO + ours Rand + FM
Success rate	Seen	100 ± 0.0	100 ± 0.0	98.3 ± 2.9	93.3 ± 11.5	100 ± 0.0	95.0 ± 8.6	100 ± 0.0	100 ± 0.0	95.0 ± 7.1	**100** ± 0.0
	Unseen	34.6 ± 4.5	25.3 ± 12.0	34.1 ± 5.4	31.5 ± 13.1	41.9 ± 5.5	37.5 ± 0.8	26.9 ± 13.1	43.1 ± 1.4	76.7 ± 1.3	**78.1** ± 3.5
Cycle-consistency	2-way	18.9 ± 10.9	13.3 ± 2.2	24.4 ± 1.1	25.5 ± 6.6	27.8 ± 10.6	17.8 ± 15.6	17.7 ± 1.1	32.2 ± 3.1	64.7 ± 4.4	**67.8** ± 6.2
	3-way	4.4 ± 2.2	4.4 ± 2.2	8.9 ± 3.8	7.4 ± 1.2	9.6 ± 5.6	5.6 ± 4.7	2.2 ± 3.8	15.6 ± 3.1	39.3 ± 8.5	**43.3** ± 4.7

Table 2: Success rate (%) and cycle-consistency (%) after 100M timesteps in small-scale CoinRun. The results show the mean and standard deviation averaged over 3 runs and the best results are indicated in bold.



(a) Trajectories (b) PPO (c) PPO + ours (d) Effects of MC samples

Figure 3: (a) We collect multiple episodes from different environments by human demonstrators and visualize the hidden representation of trained agents optimized by (b) PPO and (c) PPO + ours constructed by t-SNE, where the colors of points indicate the environments of the corresponding observations. (d) Average success rates for varying number of MC samples.

case due to the high expressive power of DNNs. We also remark that the performance in unseen environments can be further improved by optimizing the FM loss. To verify the effectiveness of MC approximation at test time, we measure the performance in unseen environments by varying the number of MC samples. Figure 3(d) shows the mean and standard deviation across 50 evaluations. The performance and its variance can be improved by increasing the number of MC samples, but the improvement is saturated around 10. Thus, we use 10 samples for the following experiments.

Embedding analysis. We analyze whether the hidden representation of trained RL agents exhibits meaningful abstraction in the unseen environments. To this end, we visualize the features on the penultimate layer of trained agents, which are reduced to 2 dimensions using t-SNE (Maaten & Hinton, 2008). Figure 3 shows the projection of trajectories taken by human demonstrators in seen and unseen environments (see Figure 8 in Appendix C for more results). Here, trajectories from both seen and unseen environments are aligned on the hidden space of our agents, while baselines still yield particularly scattered and disjoint trajectories. This implies that our method makes RL agents be able to learn the invariant and robust representation.

To evaluate the quality of hidden representation quantitatively, we also measure the cycle-consistency proposed in Aytar et al. (2018). Given two trajectories V and U, we first choose $v_i \in V$ and find its nearest neighbor in the other trajectory $u_j = \arg\min_{u \in U} \|h(v_i) - h(u)\|^2$, where $h(\cdot)$ denotes the output of the penultimate layer of trained agents. Then, we find the nearest neighbor of u_j in V, i.e., $v_k = \arg\min_{v \in V} \|h(v) - h(u_j)\|^2$, and we define that v_i is cycle-consistent if $|i - k| \leq 1$, i.e., it can back to the original point. One can note that this cycle-consistency implies that two trajectories are well-aligned in the hidden space. Similar to Aytar et al. (2018), we also evaluate the 3-way cycle-consistency by measuring whether v_i remains cycle-consistent along both paths $V \to U \to J \to V$ and $V \to J \to U \to V$, where J is the third trajectory. Using the trajectories shown in Figure 3(a), Table 2 reports the percentage of input observations in the seen environment (blue curve) that are cycle-consistent with unseen trajectories (red and green curves). Similar to the results shown in Figure 3(c), our method significantly improves the cycle-consistency compared to the vanilla PPO agent.

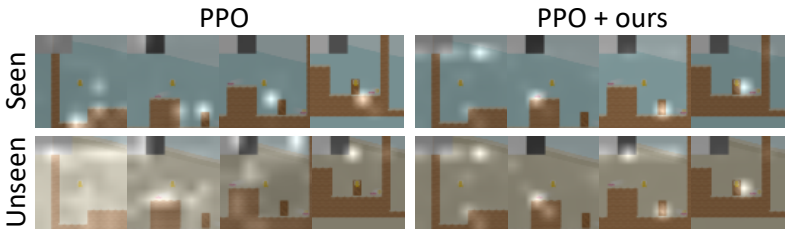
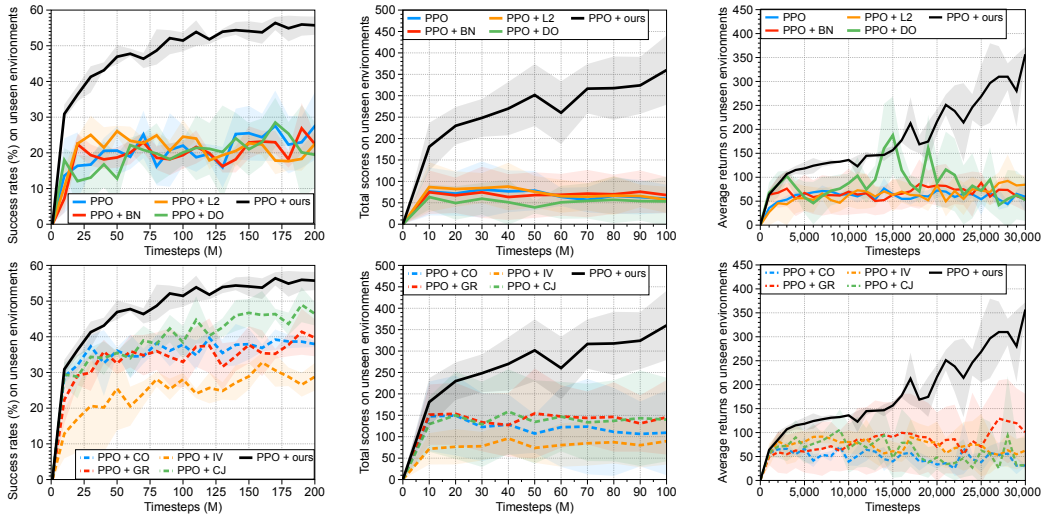


Figure 4: Visualization of activation maps via Grad-CAM in seen and unseen environments in small-scale CoinRun. We aligned images with similar states from different episodes for comparison.



(a) Large-scale CoinRun (b) DeepMind Lab (c) Surreal robotics

Figure 5: The performances of trained agents in unseen environments under (a) large-scale CoinRun, (b) DeepMind Lab and (c) Surreal robotics control. The solid/dashed lines and shaded regions represent the mean and standard deviation, respectively.

Visual interpretation. To verify whether the trained agents can focus on meaningful and high-level information, we visualize the activation maps using Grad-CAM (Selvaraju et al., 2017) by averaging activations channel-wise in the last convolutional layer, weighted by their gradients. As shown in Figure 4, both vanilla PPO and our agents make a decision by focusing on important objects such as obstacles and coin in the seen environment. However, in the unseen environment, the vanilla PPO agent shows widely distributed activation map in some cases, while our agent does not. This implies that our agent is more robust to the environment change. As a quantitative metric, we measure the entropy of normalized activation maps. Specifically, we first normalize activations $\sigma_{t,h,w} \in [0, 1]$ such that it represents a 2D discrete probability distribution at timestep t, i.e., $\sum_{h=1}^{H}\sum_{w=1}^{W}\sigma_{t,h,w} = 1$. Then, we measure the entropy averaged over timesteps as follows: $-\frac{1}{T}\sum_{t=1}^{T}\sum_{h=1}^{H}\sum_{w=1}^{W}\sigma_{t,h,w}\log\sigma_{t,h,w}$. Remark that the entropy of the activation map quantitatively measures how much an agent focuses on salient components in its observation. We observe that our agent produces a low entropy on both seen and unseen environments (i.e., 2.28 and 2.44 for seen and unseen, respectively), whereas vanilla PPO agent does only in the seen environment (2.77 and 3.54 for seen and unseen, respectively).

Results on large-scale experiments. Similar to Cobbe et al. (2019), we also evaluate the generalization ability by training agents on a fixed set of 500 levels of CoinRun. To explicitly separate seen and unseen environments, we only utilize half of the available themes (i.e., style of backgrounds, floors, agents, and moving obstacles) for training, and measure the performances on 1000 different levels which consist of unseen themes. As shown in Figure 5(a), our method still outperforms all baseline methods by a large margin. In particular, our method improves the success rates from 38.0% to 55.7% compared to the PPO with cutout (CO) augmentation proposed in Cobbe et al. (2019). This shows that our agent can learn generalizable representations given only a limited number of seen environments unlike the baseline agents.

	PPO				PPO + ours	
	# of seen environments	Total rewards	# of seen environments	Total rewards	# of seen environments	Total rewards
DeepMind Lab	1	55.4 $\pm$ 33.2	16	218.3 $\pm$ 99.2	1	**358.2** $\pm$ 81.5
Surreal robotics	1	59.2 $\pm$ 31.9	25	168.8 $\pm$ 155.8	1	**356.8** $\pm$ 15.4

Table 3: Comparison with domain randomization. The results show the mean and standard deviation averaged over 3 runs and the best results are indicated in bold.

4.3 Experiments on DeepMind Lab and Surreal robotics control

Results on DeepMind Lab. We also demonstrate the effectiveness of our proposed method on DeepMind Lab (Beattie et al., 2016), which is a 3D game environment in a first-person view with rich visual inputs. We design our task based on the standard exploration task, where a goal object is placed in one of the rooms in a 3D maze. In this task, agents aim to collect the goal object as many as possible within 90 seconds to maximize their rewards. Once the agent collects the goal object, it receives 10 points and is relocated to a random place. Similar to the small-scale CoinRun experiment, we train agents to get the goal object in a fixed map layout and test them on unseen environments by only changing the style of walls and floors. We report the mean and standard deviation of the average scores across 10 different map layouts, which are randomly selected. We provide more details in Appendix I.

We remark that a simple strategy, exploring the map actively and recognizing the goal object, can achieve high scores because the size of the maze is small in this experiment. Even though the baseline agents achieve high scores by learning this simple strategy in the seen environment (see Figure 6(c) in Appendix A for learning curves), Figure 5(b) shows that they fail to adapt to the unseen environments. However, the agent trained by our proposed method achieves high scores in both seen and unseen environments. These results show that our method can learn generalizable representations even from high-dimensional and complex input observations (i.e., 3D environment).

Results on Surreal robotics control. We evaluate our method in the Block Lifting task using Surreal distributed RL framework (Fan et al., 2018): the Sawyer robot gets a reward if it succeeds to lift a block randomly placed on a table. We train agents on a single environment and test on 5 unseen environments with different styles of table and block (see Appendix K for more details). Figure 5(c) shows that our method achieves a significant performance gain compared to all baselines in unseen environments while maintaining its performance in seen environment (see Figure 15 in Appendix K). This implies that our method can maintain important properties such as structural spatial features of the input observation.

Comparison with domain randomization. To verify the effectiveness of our method further, we train the vanilla PPO agents by increasing the number of seen environments generated by randomizing rendering in a simulator, while our agent is still trained in a single environment (see Appendix I and K for more details). Table 3 shows that the performance of baseline agents can be improved similar due to effects of domain randomization (Tobin et al., 2017). However, our method still outperforms the baseline methods, even they are trained with more diverse environments than ours. This implies that our method is more effective in learning generalizable representation than simply increasing the (finite) number of seen environments.

5 Conclusion

In this paper, we explore a type of generalization in RL where the agent is required to generalize to new environments in unseen visual patterns but semantically similar. To improve the generalization ability, we propose to randomize the first layer of CNN to perturb low-level features, e.g., various textures, colors, or shapes. Our method encourages agents to learn invariant and robust representations by producing diverse visual input observations. Such invariant features could be useful for many other related topics like an adversarial defense in RL (see Appendix D for more discussions), sim-to-real transfer (Tobin et al., 2017; Ren et al., 2019), transfer learning (Parisotto et al., 2016; Rusu et al., 2016a;b), and online adaptation (Nagabandi et al., 2019). In future work, we will consider an extension to the generalization on domains with different dynamics and structures.

REFERENCES

Yusuf Aytar, Tobias Pfaff, David Budden, Thomas Paine, Ziyu Wang, and Nando de Freitas. Playing hard exploration games by watching youtube. In *NeurIPS*, 2018.

Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *ICLR*, 2019.

Bibhas Chakraborty and Susan A Murphy. Dynamic treatment regimes. *Annual review of statistics and its application*, 1:447–464, 2014.

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

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

Yue Deng, Feng Bao, Youyong Kong, Zhiquan Ren, and Qionghai Dai. Deep direct reinforcement learning for financial signal representation and trading. *IEEE transactions on neural networks and learning systems*, 28(3):653–664, 2016.

Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

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

Linxi Fan, Yuke Zhu, Jiren Zhu, Zihua Liu, Orien Zeng, Anchit Gupta, Joan Creus-Costa, Silvio Savarese, and Li Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In *CoRL*, 2018.

Jesse Farebrother, Marlos C Machado, and Michael Bowling. Generalization and regularization in dqn. *arXiv preprint arXiv:1810.00123*, 2018.

Shani Gamrian and Yoav Goldberg. Transfer learning for related reinforcement learning tasks via image-to-image translation. In *ICML*, 2019.

Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research*, 17(1):2096–2030, 2016.

Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A Wichmann, and Wieland Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. In *ICLR*, 2019.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *AISTATS*, 2010.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014.

Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *ICLR*, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *ICML*, 2017.

Sandy Huang, Nicolas Papernot, Ian Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. *arXiv preprint arXiv:1702.02284*, 2017.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *ICML*, 2015.

Byungju Kim, Hyunwoo Kim, Kyungsu Kim, Sungjin Kim, and Junmo Kim. Learning not to learn: Training deep neural networks with biased data. In *CVPR*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Yen-Chen Lin, Zhang-Wei Hong, Yuan-Hong Liao, Meng-Li Shih, Ming-Yu Liu, and Min Sun. Tactics of adversarial attack on deep reinforcement learning agents. In *IJCAI*, 2017.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

Marlos C Machado, Marc G Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research*, 61:523–562, 2018.

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

Anusha Nagabandi, Chelsea Finn, and Sergey Levine. Deep online learning via meta-learning: Continual adaptation for model-based rl. In *ICLR*, 2019.

Ian Osband, John Aslanides, and Albin Cassirer. Randomized prior functions for deep reinforcement learning. In *NeurIPS*, 2018.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Emilio Parisotto, Jimmy Lei Ba, and Ruslan Salakhutdinov. Actor-mimic: Deep multitask and transfer reinforcement learning. In *ICLR*, 2016.

Xinyi Ren, Jianlan Luo, Eugen Solowjow, Juan Aparicio Ojea, Abhishek Gupta, Aviv Tamar, and Pieter Abbeel. Domain randomization for active pose estimation. In *ICRA*, 2019.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. In *ICLR*, 2016a.

Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016b.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *ICLR*, 2016.

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

Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, 2017.

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

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

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IROS*, 2017.

Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. *arXiv preprint arXiv:1708.04782*, 2017.

Jane X Wang, Zeb Kurth-Nelson, Dhruva Tirumala, Hubert Soyer, Joel Z Leibo, Remi Munos, Charles Blundell, Dharshan Kumaran, and Matt Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

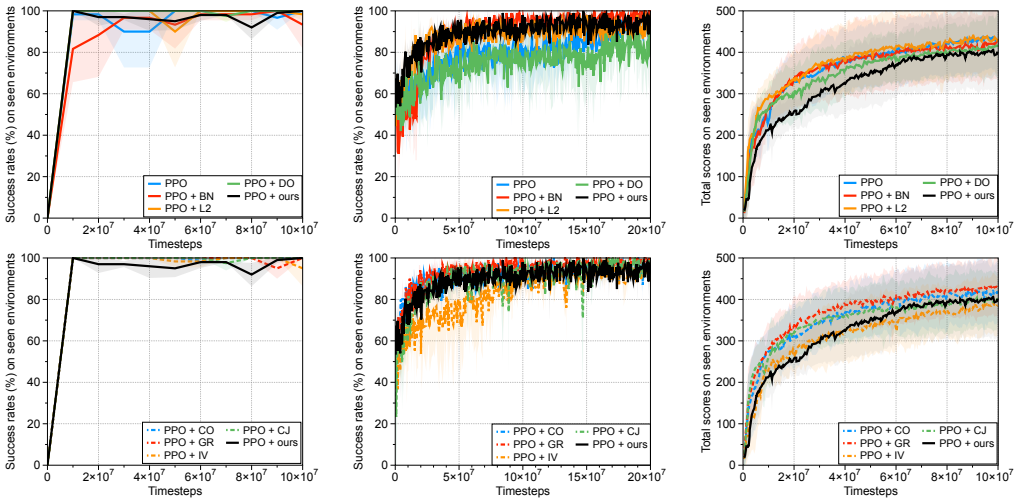
Ronald J Williams and Jing Peng. Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268, 1991.

Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning. *arXiv preprint arXiv:1806.07937*, 2018a.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893*, 2018b.

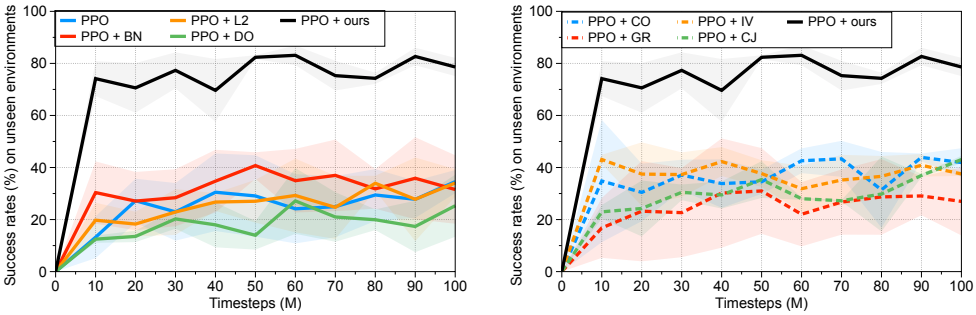
Appendix: A Simple Randomization Technique for Generalization in Deep Reinforcement Learning

A LEARNING CURVES



(a) Small-scale CoinRun (b) Large-scale CoinRun (c) DeepMind lab

Figure 6: Learning curves on (a) small-scale, (b) large-scale CoinRun and (c) DeepMind Lab. The solid line and shaded regions represent the mean and standard deviation, respectively, across 3 runs.



(a) Comparison with regularization techniques (b) Comparison with data augmentation techniques

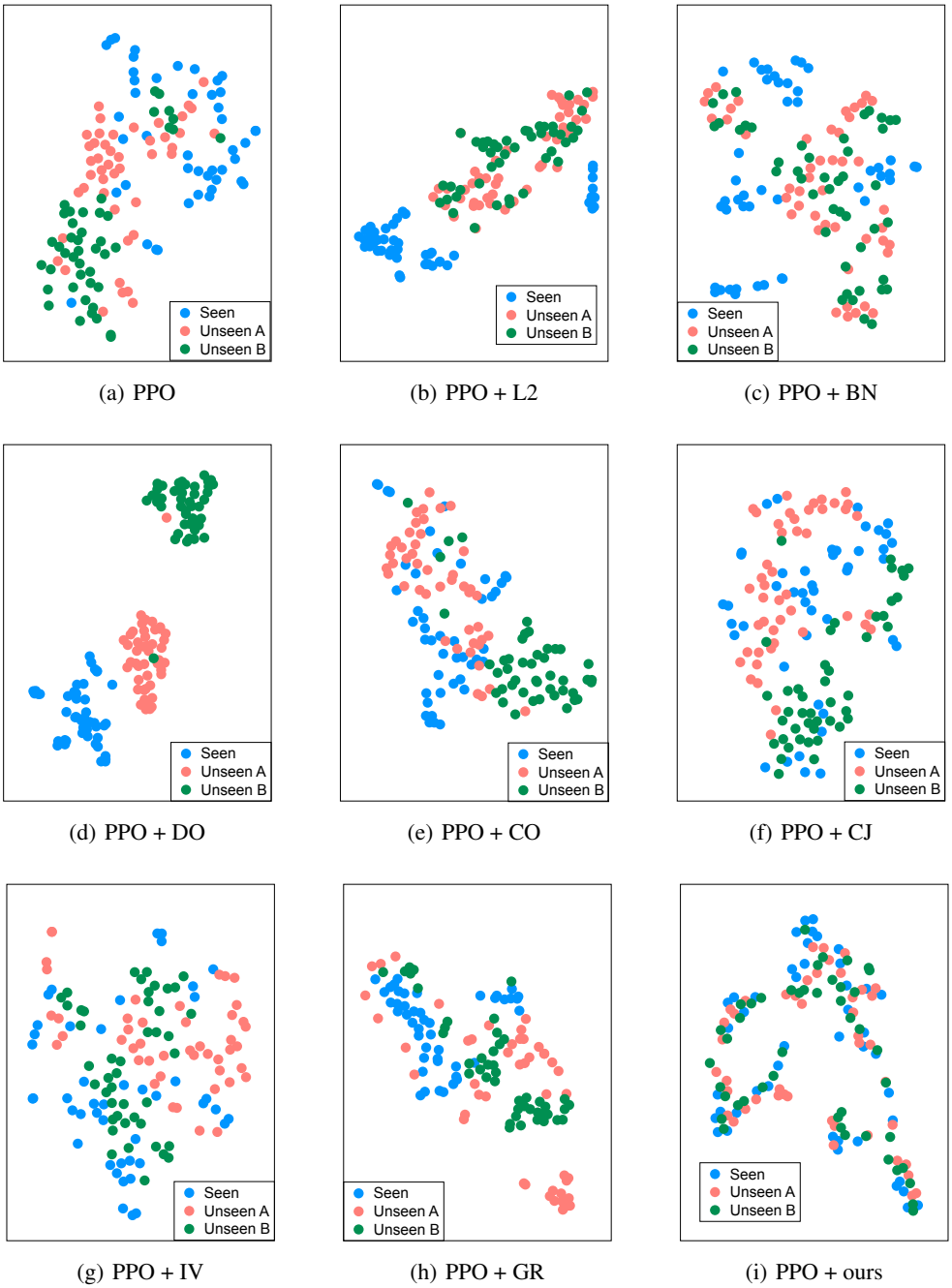
Figure 7: The performance in unseen environments in small-scale CoinRun. The solid/dashed line and shaded regions represent the mean and standard deviation, respectively, across 3 runs.

B TRAINING ALGORITHM

Algorithm 1 PPO + random networks, Actor-Critic Style

for iteration$= 1, 2, \cdots$ **do**
  Sample the parameter ϕ of random networks from prior distribution $P(\phi)$
  for actor$= 1, 2, \cdots, N$ **do**
    Run policy $\pi\left(a|f\left(s;\phi\right);\theta\right)$ in the given environment for T timesteps
    Compute advantage estimates
  end for
  Optimize $\mathcal{L}^{\texttt{random}}$ in equation (3) with respect to θ
end for

C VISUALIZATION OF HIDDEN FEATURES



(a) PPO (b) PPO + L2 (c) PPO + BN

(d) PPO + DO (e) PPO + CO (f) PPO + CJ

(g) PPO + IV (h) PPO + GR (i) PPO + ours

Figure 8: Visualization of the hidden representation of trained agents optimized by (a) PPO, (b) PPO + L2, (c) PPO + BN, (d) PPO + DO, (e) PPO + CO, (f) PPO + GR, (g) PPO + IV, (h) PPO + CJ and (I) PPO + ours using t-SNE. The colors of points indicate the environments of the corresponding observations.

D ROBUSTNESS AGAINST ADVERSARIAL ATTACKS

It is well-known that the adversarial (visually imperceptible) perturbation (Szegedy et al., 2014) to clean input observations can induce the DNN-based policies to make an incorrect decision at test time (Huang et al., 2017; Lin et al., 2017). This undesirable property of DNNs has raised major security concerns. In this section, we evaluate if the proposed method can improve the robustness on adversarial attacks. One can naturally expect that our method improves the robustness against such adversarial attacks because agents are trained with randomly perturbed inputs. To verify that the proposed method can improve the robustness to adversarial attacks, we generate the adversarial samples using FGSM (Goodfellow et al., 2015) by perturbing inputs to the opposite direction to the most probable action initially predicted by the policy:

$$s_{\texttt{adv}} = s - \varepsilon \text{sign}\left(\nabla_s \log \pi(a^*|s;\theta)\right),$$

where ε is the magnitude of noise and $a^* = \arg\max_a \pi(a|s;\theta)$ is the action from the policy. Table 4 shows that our proposed method can improve the robustness against FGSM attacks with $\varepsilon = 0.01$, which implies that hidden representations of trained agents are more robust.

	Small-scale CoinRun		Large-scale CoinRun		DeepMind Lab	
	Clean	FGSM	Clean	FGSM	Clean	FGSM
PPO	100	61.5 (-38.5)	96.2	77.4 (-19.5)	352.5	163.5 (-53.6)
PPO + ours	100	88.0 **(-12.0)**	99.6	84.4 **(-15.3)**	368.0	184.0 **(-50.0)**

Table 4: Robustness against FGSM attacks on training environments. The values in parentheses represent the relative reductions from the clean samples.

E DETAILS FOR TRAINING AGENTS USING PPO

Policy optimization. For all baseline and our methods, we utilize PPO to train their policies. Specifically, we use a discount factor $\gamma = 0.999$, a generalized advantage estimator (GAE) Schulman et al. (2016) parameter $\lambda = 0.95$, and an entropy bonus (Williams & Peng, 1991) of 0.01 to ensure sufficient exploration. We extract 256 timesteps per rollout, and then train the agent for 3 epochs with 8 mini-batches. We use Adam optimizer (Kingma & Ba, 2015) with the starting learning rate 0.0005. We run 32 environments simultaneously during training. As suggested in Cobbe et al. (2019), two boxes are painted in the upper-left corner, where their color represents the x- and y-axis velocity; they help the agents quickly learn to act optimally. In this way, the agent does not need to memorize previous states, such that a simple CNN-based policy without LSTM can perform well in our experimental settings.

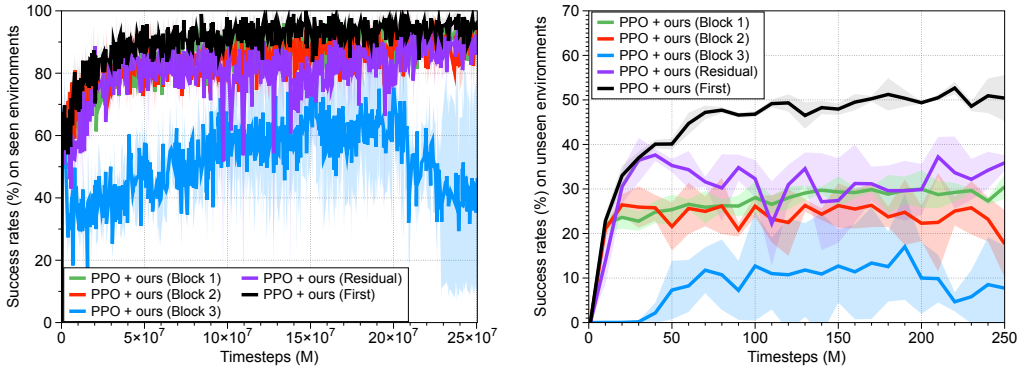
Data augmentation methods. In this paper, we compare a variant of cutout (DeVries & Taylor, 2017) proposed in Cobbe et al. (2019), grayout, inversion, and color jitter (Cubuk et al., 2019). Specifically, the cutout augmentation applies random number of boxes in random size and color to the input, the grayout method averages all three channels of the input, the inversion method inverts pixel values by a 50% chance, and the color jitter changes characteristics of images commonly used for data augmentation in computer vision tasks: brightness, contrast, and saturation. For the cutout augmentation, for every timestep, we first randomly choose the number of boxes from 0 to 5, assign random color and size to them, and place them in the observation. For the color jitter, the parameters for brightness, contrast, and saturation are randomly chosen in [0.5,1.5].[8] We randomize and then fix parameters of these methods for each episode, such that the same image pre-processing is applied within an episode.

F DIFFERENT TYPES OF RANDOM NETWORKS

In this section, we apply random networks to different locations in the network architecture (see Figure 10) and measure the performance in large-scale CoinRun without the feature matching loss.

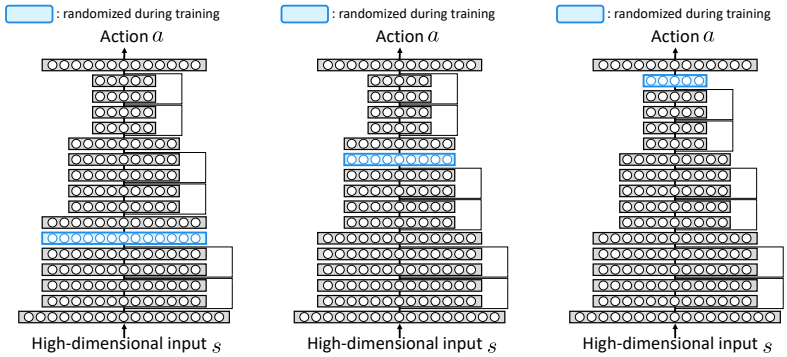
[8]For more details, see https://pytorch.org/docs/stable/_modules/torchvision/transforms/transforms.html#ColorJitter.

For all methods, we use a single-layer CNN with the kernel size of 3, and its output tensor is padded to be in the same dimension with the input tensor. As shown in Figure 9, the performance in unseen environments decreases as the random network is placed in higher layers. On the other hand, the random network in residual connections improves the generalization performance, but it does not outperform the case when a random network is placed at the beginning of the network. This means that randomizing only the local features of inputs can be effective for better generalization performance.

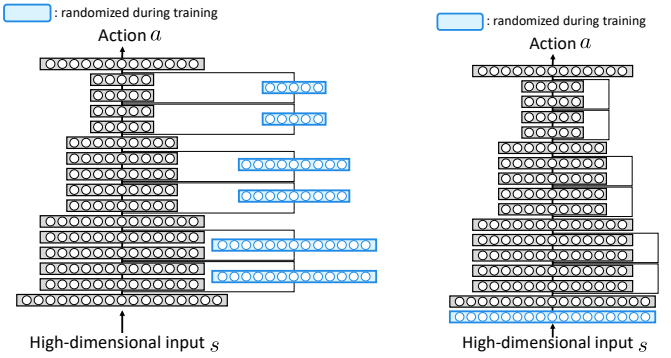


(a) Performance in seen environments (b) Performance in unseen environments

Figure 9: The Performance with random networks in different locations in the network architecture on (a) seen and (b) unseen environments in large-scale CoinRun. We show the mean performances averaged over 3 different runs, and shaded regions represent the standard deviation.

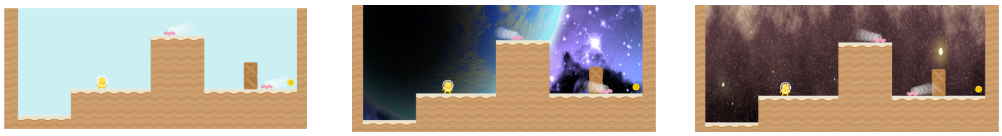


(a) Block 1 (b) Block 2 (c) Block 3



(d) Residual (e) First

Figure 10: Network architectures with random networks in different locations. Only convolutional layers and the last fully connected layer are displayed for conciseness.



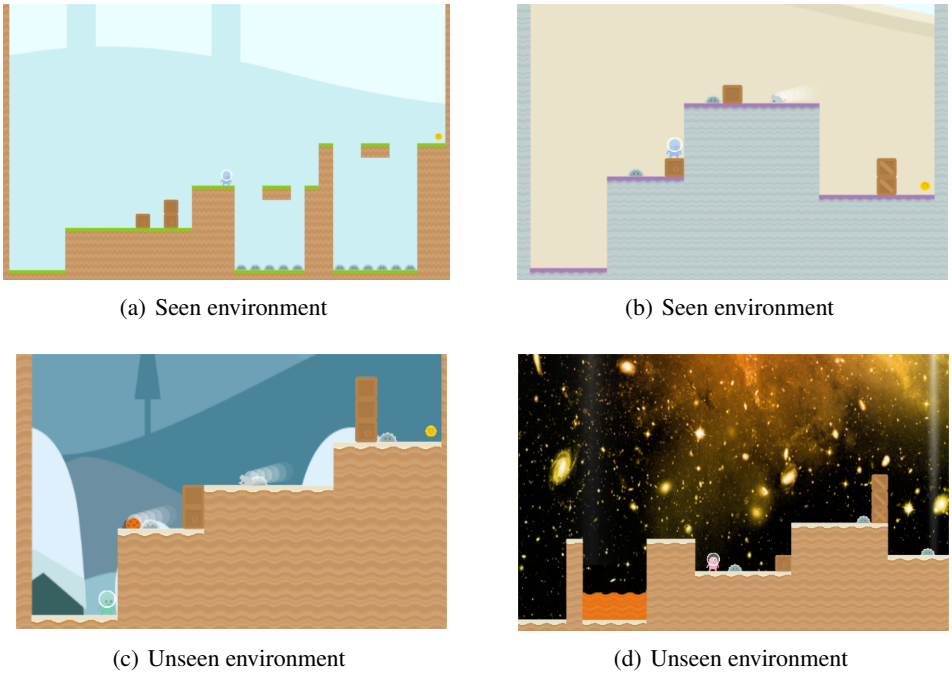
(a) Seen environment (b) Unseen environment (c) Unseen environment

Figure 11: Examples of seen and unseen environments in small-scale CoinRun.

G ENVIRONMENTS IN SMALL-SCALE COINRUN

For small-scale CoinRun environments, we consider a fixed map layout with two moving obstacles and measure the performance of trained agents by changing the style of backgrounds (see Figure 11). Below is the list of seen and unseen backgrounds in this experiment:

- Seen background: `kenney/Backgrounds/blue_desert.png`
- Unseen backgrounds:
- `kenney/Backgrounds/colored_desert.png`
- `kenney/Backgrounds/colored_grass.png`
- `backgrounds/game-backgrounds/seabed.png`
- `backgrounds/game-backgrounds/G049_OT000_002A__background.png`
- `backgrounds/game-backgrounds/Background.png`
- `backgrounds/game-backgrounds/Background (4).png`
- `backgrounds/game-backgrounds/BG_only.png`
- `backgrounds/game-backgrounds/bg1.png`
- `backgrounds/game-backgrounds/G154_OT000_002A__background.png`
- `backgrounds/game-backgrounds/Background (5).png`
- `backgrounds/game-backgrounds/Background (2).png`
- `backgrounds/game-backgrounds/Background (3).png`
- `backgrounds/background-from-glitch-assets/background.png`
- `backgrounds/spacebackgrounds-0/deep_space_01.png`
- `backgrounds/spacebackgrounds-0/spacegen_01.png`
- `backgrounds/spacebackgrounds-0/milky_way_01.png`
- `backgrounds/spacebackgrounds-0/deep_sky_01.png`
- `backgrounds/spacebackgrounds-0/space_nebula_01.png`
- `backgrounds/space-backgrounds-3/Background-1.png`
- `backgrounds/space-backgrounds-3/Background-2.png`
- `backgrounds/space-backgrounds-3/Background-3.png`
- `backgrounds/space-backgrounds-3/Background-4.png`
- `backgrounds/background-2/airadventurelevel1.png`
- `backgrounds/background-2/airadventurelevel2.png`
- `backgrounds/background-2/airadventurelevel3.png`
- `backgrounds/background-2/airadventurelevel4.png`



(a) Seen environment (b) Seen environment

(c) Unseen environment (d) Unseen environment

Figure 12: Examples of seen and unseen environments in large-scale CoinRun.

H ENVIRONMENTS IN LARGE-SCALE COINRUN

In CoinRun, there are 34 themes for backgrounds, 6 for grounds, 5 for agents, and 9 for obstacles. For large-scale CoinRun experiment, we train agents on a fixed set of 500 levels of CoinRun using half of the available themes and measure the performances on 1000 different levels which consist of unseen themes. Specifically, we use the following is a list of seen and unseen themes in this experiment:

- Seen backgrounds:
- `kenney/Backgrounds/blue_desert.png`
- `kenney/Backgrounds/blue_grass.png`
- `kenney/Backgrounds/blue_land.png`
- `kenney/Backgrounds/blue_shroom.png`
- `kenney/Backgrounds/colored_desert.png`
- `kenney/Backgrounds/colored_grass.png`
- `kenney/Backgrounds/colored_land.png`
- `backgrounds/game-backgrounds/seabed.png`
- `backgrounds/game-backgrounds/G049_OT000_002A__background.png`
- `backgrounds/game-backgrounds/Background.png`
- `backgrounds/game-backgrounds/Background (4).png`
- `backgrounds/game-backgrounds/BG_only.png`
- `backgrounds/game-backgrounds/bg1.png`
- `backgrounds/game-backgrounds/G154_OT000_002A__background.png`
- `backgrounds/game-backgrounds/Background (5).png`
- `backgrounds/game-backgrounds/Background (2).png`

- backgrounds/game-backgrounds/Background (3).png
- Unseen backgrounds:
- backgrounds/background-from-glitch-assets/background.png
- backgrounds/spacebackgrounds-0/deep_space_01.png
- backgrounds/spacebackgrounds-0/spacegen_01.png
- backgrounds/spacebackgrounds-0/milky_way_01.png
- backgrounds/spacebackgrounds-0/ez_space_lite_01.png
- backgrounds/spacebackgrounds-0/meyespace_v1_01.png
- backgrounds/spacebackgrounds-0/eye_nebula_01.png
- backgrounds/spacebackgrounds-0/deep_sky_01.png
- backgrounds/spacebackgrounds-0/space_nebula_01.png
- backgrounds/space-backgrounds-3/Background-1.png
- backgrounds/space-backgrounds-3/Background-2.png
- backgrounds/space-backgrounds-3/Background-3.png
- backgrounds/space-backgrounds-3/Background-4.png
- backgrounds/background-2/airadventurelevel1.png
- backgrounds/background-2/airadventurelevel2.png
- backgrounds/background-2/airadventurelevel3.png
- backgrounds/background-2/airadventurelevel4.png

- Seen grounds:
- Dirt
- Grass
- Planet
- Unseen grounds:
- Sand
- Snow
- Stone

- Seen player themes:
- Beige
- Blue
- Unseen player themes:
- Green
- Pink
- Yellow

I ENVIRONMENTS ON DEEPMIND LAB

Dataset. Among the styles (textures and colors) provided for the 3D maze in DeepMind Lab, we take 10 different styles of floors and walls, respectively (see the list below). We construct a training dataset by randomly choosing a map layout and assigning a theme among 10 floors and walls, respectively. For the domain randomization method compared in Table 3, we provide 4 floors and wall themes (16 combinations in total). Trained themes are randomly chosen before training and their combinations are considered as seen environments. To evaluate the generalization ability, we measure the performance of trained agents on unseen environments by only changing the styles of walls and floors. Taking into account that domain randomization has more seen themes than the

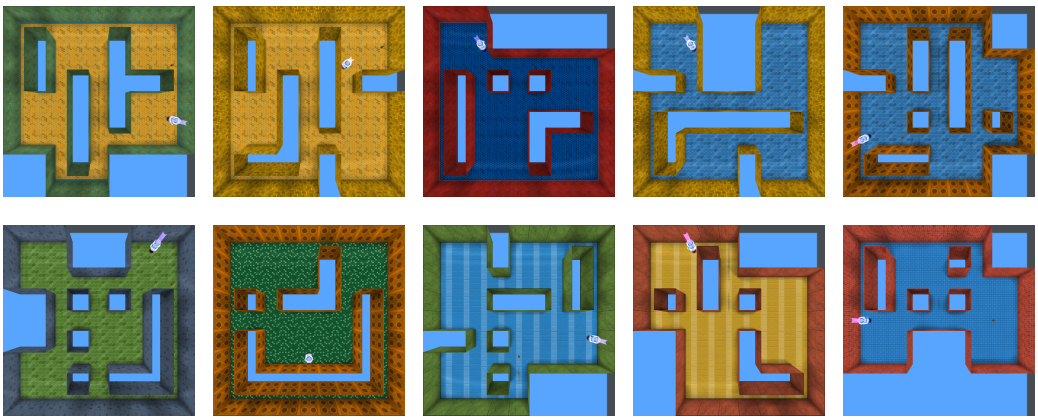


Figure 13: The top-down view of the trained map layouts.

Action	DeepMind Lab native action
Forward	[0, 0, 0, 1, 0, 0, 0]
Backward	[0, 0, 0, -1, 0, 0, 0]
Move Left	[0, 0, -1, 0, 0, 0, 0]
Move Right	[0, 0, 1, 0, 0, 0, 0]
Look Left	[-20, 0, 0, 0, 0, 0, 0]
Look Right	[20, 0, 0, 0, 0, 0, 0]
Forward + Look Left	[-20, 0, 0, 1, 0, 0, 0]
Forward + Look Right	[20, 0, 0, 1, 0, 0, 0]

Table 5: Action set used in the DeepMind Lab experiment. The DeepMind Lab native action set consists of seven discrete actions encoded in integer ([L,U] indicates the lower/upper bound of the possible values): 1) yaw (left/right) rotation by pixel [-512,512], 2) pitch (up/down) rotation by pixel [-512,512], 3) horizontal move [-1,1], 4) vertical move [-1,1], 5) fire [0,1], 6) jump [0,1], and 7) crouch [0,1].

other methods, we compare all methods with 6 floors and walls (36 combinations in total), which are totally unseen for all methods. We report the mean and standard deviation of the average scores across 10 different map layout as shown in Figure 13.

- Floor themes:
- `lg_style_01_floor_orange`
- `lg_style_01_floor_blue`
- `lg_style_02_floor_blue`
- `lg_style_02_floor_green`
- `lg_style_03_floor_green`
- `lg_style_03_floor_blue`
- `lg_style_04_floor_blue`
- `lg_style_04_floor_orange`
- `lg_style_05_floor_blue`
- `lg_style_05_floor_orange`
- Wall themes:
- `lg_style_01_wall_green`
- `lg_style_01_wall_red`

- lg_style_02_wall_yellow
- lg_style_02_wall_blue
- lg_style_03_wall_orange
- lg_style_03_wall_gray
- lg_style_04_wall_green
- lg_style_04_wall_red
- lg_style_05_wall_red
- lg_style_05_wall_yellow

Action space. Similar to IMPALA (Espeholt et al., 2018), the agent can take eight actions which are samples from the DeepMind Lab native actions: {Forward, Backward, Move Left, Move Right, Look Left, Look Right, Forward + Look Left, Forward + Look Right}. Table 5 describes the detailed mapping.

J EXPERIMENTS ON DOGS AND CATS DATABASE

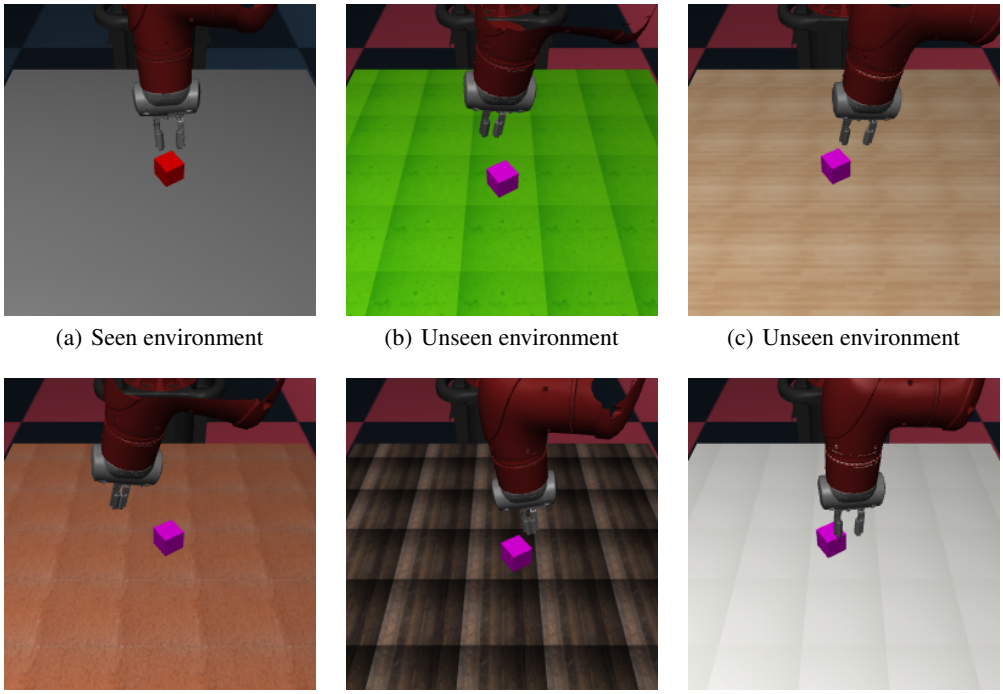
Dataset. The original database is a set of 25K images of dogs and cats for training and 12,500 images for testing. Similar to Kim et al. (2019), we manually categorized the data according to the color of the animal: bright and dark. Based on this, we construct biased datasets such that the training set consists of bright dogs and dark cats while test and validation sets contain dark dogs and bright cats. Specifically, training, validation, and test sets consist of 10,047, 1,000, and 5,738 images, respectively.[9] We train ResNet-18 (He et al., 2016) using the following hyper-parameters: initial learning rate is chosen from $\{0.05, 0.1\}$ and we drop it by 0.1 at 50 epochs with total 100 epochs. We use Nesterov momentum 0.9 for SGD, mini-batch size is chosen from $\{32, 64\}$, and weight decay is set to 0.0001. We report the training and test set accuracies with the hyperparameters chosen by validation. Unlike Kim et al. (2019), we do not use ResNet-18 pre-trained with ImageNet (Russakovsky et al., 2015) to avoid inductive bias from the pre-trained dataset.

K EXPERIMENTAL RESULTS ON SURREAL ROBOT MANIPULATION

We evaluate our method in the Block Lifting task using Surreal distributed RL framework (Fan et al., 2018). In this task, the Sawyer robot gets a reward if it succeeds to lift a block, which is randomly placed on a table. Following the experimental setups in (Fan et al., 2018), we take the hybrid CNN-LSTM architecture (see Figure 15(a)) as the policy network and use a distributed version of PPO (i.e., actors collect massive amount of trajectories and centralized learner updates the model parameters using PPO) to train the agents.[10] Agents take observation frames of 84×84 with proprioceptive features (e.g., robot joint positions and velocities) and output the mean and log of the standard deviation for each action dimension. The actions are then from the Gaussian distribution parameterized by the output. We train agents on a single environment and test on 5 unseen environments with different styles of table, floor, and block as shown in Figure 14. For Surreal robot manipulation experiment, we train the vanilla PPO agent on 25 environments which are generated by changing the styles of tables and boxes. Specifically, we use {blue, gray, orange, white, purple} and {red, blue, green, yellow, cyan} for table and box, respectively.

[9]The code is available at https://github.com/feidfoe/learning-not-to-learn/tree/master/dataset/dogs_and_cats

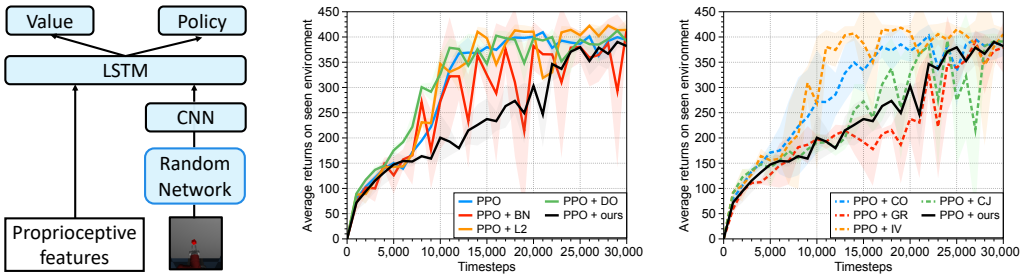
[10]We used a reference implementation with two actors in https://github.com/SurrealAI/surreal.



(a) Seen environment (b) Unseen environment (c) Unseen environment

(d) Unseen environment (e) Unseen environment (f) Unseen environment

Figure 14: Examples of seen and unseen environments in Surreal robot manipulation.



(a) Network architecture (b) Regularization (c) Data augmentation

Figure 15: (a) An illustration of network architectures for the Surreal robotics control experiment, and learning curves with (b) regularization and (c) data augmentation techniques. The solid line and shaded regions represent the mean and standard deviation, respectively, across 3 runs.