Pair-Based Data Augmentations for Reinforcement Learning in ProcGen

Jia Shao[*]
McGill University, Mila

Thang Doan
McGill University, Mila

Joelle Pineau
McGill University, Mila

Abstract

We begin with the hypothesis that more refined data augmentations, namely mixup using embedding layer representations, saliency-based mixup and similarity-based mixup, can improve generalization in the context of pixel-based reinforcement learning. Since pixel-based learning is sensitive to visual perturbations, we are interested in studying robustness also. In this work, we introduce four approaches that use pair-based data augmentations during the training phase: *Embedding Mixup, Co-Mixreg, L2-distance based Mixreg,* and *L2-distance based Embedding Mixup*. Embedding Mixup manipulates the embedding layer of the trained convolutional neural network (CNN), whereas the other models perturb observations using pair mixup then feeds the result as input to the model. Co-Mixreg is a saliency-based approach that, similar to the remaining two models, introduces a similarity metric for training observation mixup. While these methods do not beat state-of-the-art results on the ProcGen benchmark, they learn a smooth policy and generalize well to unseen environments. Additionally, they are simple to implement and fast to train.

1 Introduction

Reinforcement learning (RL) has come a long way since its early inception, where its agents learned from states (e.g. TD3 [Fujimoto et al., 2018], SAC [Haarnoja et al., 2018]). More recently, deep reinforcement learning from pixels has become a popular research direction, since we are rarely fortunate enough to have access to complete or even accurate state information. For example, in robot navigation tasks, the robot will be equipped with cameras to collect environment observations [Tobin et al., 2017]. We do not have full access to state information and must interpret camera footage to understand the environment. Thus, researchers are interested in designing pixel-based RL agents that match the performance of state-based RL agents, both in terms of policy effectiveness and learning efficiency (better wall-clock training time, better sample efficiency). Challenges in pixel-based reinforcement learning include but are not limited to poor generalization to unseen environments, brittleness in the face of visual distractors, and a tendency to overfit to task-irrelevant details [Wang et al., 2020, Raileanu and Fergus, 2021].

A pixel-based RL model's tendency to overfit is analogous to memorizing trajectories and task-irrelevant visual details instead of actually learning how to complete a task. Data augmentation is a regularizer that reduces overfitting as it increases the number and diversity of images used during training [Kostrikov et al., 2021]. The augmentations in Kostrikov et al. [2021], namely random crop, translation and cutout, are ones commonly used in computer vision's classification tasks [Cireşan et al., 2011, 2012, Simard et al., 2003, Krizhevsky et al., 2012, Chen et al., 2020]: they augment each training observation individually and independently of others. Wang et al. [2020] show that these augmentations only minimally perturb observations, which leads to limited generalization

[*]Work done during an internship at McGill University and Mila, Montréal, QC, Canada. Correspondence to: jia.r.shao@mail.mcgill.ca

McGill University Summer Research Internship 2021.

performance gain. Unlike Kostrikov et al. [2021]'s DrQ, the Mixreg algorithm [Wang et al., 2020] uses a data augmentation technique that combines observation pairs: Mixup [Zhang et al., 2018b]. Seeing the success of Mixup in Mixreg, we propose variations of Mixup to address its tendency to produce invalid training scenarios. Our motivation is to use these Mixup variations to increase sample diversity while ensuring that augmented results are sensible to the agent.

The first part of our work is based on Mixreg, a reinforcement learning algorithm that uses Mixup in its data augmentation step [Wang et al., 2020]. Mixreg's algorithm steps and objectives are borrowed from OpenAI's Proximal Policy Optimization (PPO) algorithm [Schulman et al., 2017]. The second part of our work is inspired by Co-Mixup [Kim et al., 2021] and injects the notion of saliency-based detection into CutMix [Yun et al., 2019]. Overall, we adapt supervised classification techniques from computer vision to the realm of reinforcement learning.

In this paper:

(i) We introduce four types of algorithm, each using different mixing strategies at the embedding or input layers: Embedding Mixup, Co-Mixreg, L2-distance based Mixreg, and L2-distance based Embedding Mixup. These can be used for both policy and value-based RL algorithms;

(ii) We show that, much like Mixreg, our four algorithms outperform the Procgen benchmark. However, our algorithms do not surpass Mixreg's performance;

(iii) We analyze the differences between these algorithms and identify what sets Mixreg apart from the other algorithms.

2 Related Works

Pixel-based reinforcement learning agents often fail to generalize to new environments even when these are similar to training images in terms of dynamics and semantics. These agents overfit to training environments, meaning that they are memorizing trajectories and task-irrelevant visual cues instead of learning robust representations [Raileanu and Fergus, 2021]. Strong generalization is especially important in high-dimensional tasks, namely when learning from pixel-based input – without it, agents are sensitive to small visual changes [Wang et al., 2020].

Poor generalization is often attributed to a lack of sample diversity during training [Zhang et al., 2018a]. Computer vision techniques are a natural family of solutions to address this issue, as supervised image classification tasks use and benefit greatly from data augmentation [Perez and Wang, 2017, Kostrikov et al., 2021]. In particular, RAD [Laskin et al., 2020] and DrQ Kostrikov et al. [2021] use data augmentations to diversify training observations, namely random crop, translation, cutout, rotation, flipping, color-jitter and random convolutions. Laskin et al. [2020] found that random crop and cutout greatly improve generalization performance on certain ProcGen games. However, the choice of an appropriate augmentation is task-specific, and an inappropriate choice can tamper with an agent's learning. AutoDRAC (Automatic Data-Regularized Actor-Critic) [Raileanu et al., 2021] aims to algorithmically determine the optimal data augmentation to use for a given task.

Data augmentations provide inconsistent performance across different game environments: on certain games, they score worse than the PPO baseline [Wang et al., 2020]. These augmentation methods only locally perturb observations, meaning they introduce limited sample diversity, and create training observations that differ greatly from possible test-time scenarios [Wang et al., 2020]. These concerns gave rise to a new class of data augmentation that use observation pairs instead of independently augmenting observations: image classification algorithms such as Mixup and CutMix study generalization in the convex hull of training examples [Zhang et al., 2018b, Yun et al., 2019]. Unlike RAD and DrQ, Mixup and CutMix use image pairs for training. Mixup performs a linear combination of an image pair's RGB values, thus learning a smoother, linear function between training examples [Zhang et al., 2018b]. CutMix uses cutout on an image patch then replaces it with a patch from the second mixing image. This algorithm significantly improves model robustness [Yun et al., 2019].

3 Background

Reinforcement Learning in ProcGen. We formally define a discrete-time partially observable Markov decision process (POMDP) as a tuple $(S, A, T, R, O, \Omega, \gamma)$, where S is the state space, A is

the set of actions, $T : S \times A \times S \to [0, 1]$ is the probability distribution describing state transitions, $R : S \times A \to \mathbb{R}$ is the reward function, O is a set of observations, $\Omega : S \times A \times O \to [0, 1]$ is the observation probability function, and γ is the discount factor. ProcGen has 16 game environments, and each game has its own distribution q of levels that can expressed as POMDPs. For each game, we sample n POMDPs for training, $\mathcal{M}_{train} = \{m_1, \ldots, m_n\}$ where each $m_i \sim q$. We want to find a policy π over parameters θ which maximizes the discounted reward over all POMDPs for a game:

$$J(\pi_\theta) = \mathbf{E}_{q,\pi,T_m}\left[\sum_{t=0}^{T} \gamma^t R_m(s_t, a_t)\right] \tag{1}$$

In the training phase, we sample from a subset of the full level distribution: by default for the "easy" level distribution, only 200 unique levels can be generated. During the test phase, we sample from the entire distribution, using all levels generated from any integer seed.

PPO: Proximal Policy Optimization. ProcGen's PPO baseline uses a particular variant of the algorithm named PPO-Clip (which we will refer to as PPO henceforth). PPO is a policy gradient method for reinforcement learning. It alternates between two phases: the environment interaction phase and the optimization phase. At each update PPO collects transition samples using the rollout policy $\pi_{\theta_{old}}(a_t|s_t)$ then optimizes the surrogate loss $L^{CLIP}(\theta)$:

$$L^{CLIP}(\theta) = \hat{\mathbf{E}}_t\left[\min\left(r_t(\theta)\hat{A}_t, \textit{clip}\left(r_t(\theta), 1-\varepsilon, 1+\varepsilon\right)\hat{A}_t\right)\right] \tag{2}$$

where θ is the policy parameter, r_t is the ratio between log probabilities of the old and new policies, $\hat{A}_t$ is the estimated advantage at time t, and ε is a clipping hyperparameter. PPO is algorithmically simple yet highly sample efficient.

Mixture Regularization. Mixture regularization, dubbed Mixreg, performs a linear combination of RGB values between image pairs using a mixup coefficient from the beta distribution. The mixup is expressed as follows:

$$\tilde{s} = \lambda s_i + (1-\lambda)s_j \tag{3}$$

where $\lambda \sim \mathrm{Beta}(0.2, 0.2)$ is the mixup coefficient, s_i, s_j are different environment observations and $\tilde{s}$ is the resulting image fed as input to the model. With the same coefficient, further interpolation is introduced on the associated reward signals of each pair:

$$\tilde{r} = \lambda r_i + (1-\lambda)r_j \tag{4}$$

where λ uses the same value as the one sampled from the Beta distribution in 3 and r_i, r_j are s_i, s_j's respective supervision signals. As a result, the mixed supervision signals are proper signals for the mixed observation. Since coefficients chosen from a beta distribution are constrained to the interval $(0, 1)$, each interpolated result resides in the convex hull of the mixing pairs. With Mixreg, the learned state value function exhibits greater smoothness than PPO's function. This result agrees with Mixup's findings on the learned function's linear behaviour in the convex hull of its training examples. Thus, Mixreg excels in terms of generalization and outperforms the PPO baseline.

A key weakness of this algorithm is that the mixup process can generate invalid training images since observations are mixed randomly (see Figures 5a, 5d). In ProcGen, different levels have distinct color schemes and backgrounds, as well as different spawn positions of the agent and other game entities, among other game details. Since image are paired randomly, they are not guaranteed to belong to similar game contexts. Thus, interpolation results do not resemble possible generated test scenarios.

Saliency in Deep Learning. Deep learning models are also known as "black box" models because humans cannot understand the intermittent prediction steps. In particular, it is difficult to understand how a CNN arrives at a class prediction in the realm of computer vision [Simonyan et al., 2014]. To improve the interpretability of these CNNs, we create visualizations, such as saliency maps [Adebayo et al., 2020]. Saliency maps detect parts of an input image that contribute the most to the network's learning process. They can be created using the backpropagation algorithm: in classification tasks, for a given observation and class label, we compute the gradient of the class score with respect to the network's image input [Simonyan et al., 2014]. Then with the gradients at the input layer, we can create a mask to extract relevant pixels and ignore pixels with task-irrelevant information.

On the other hand, pixel-based reinforcement learning does not belong in the realm of image classification. Rather, it is a form of self-supervised learning – class labels for observations are not

available and the model must generate its own supervisory signals for training. Instead of outputting a vector of class prediction scores, the reinforcement learning network returns a loss value to describe the quality of the action chosen for a given state observation. Still, it is possible to generate a saliency map by computing the gradient of the loss value with respect to the input, as demonstrated in Co-Mixup [Kim et al., 2021].

4 Method

Since Mixreg uses a naive linear combination of random image pairs, results are not guaranteed to be sensible training observations. To address this flaw, we experiment with new mixup strategies, namely embedding mixup and saliency-guided patch mixup. We also use distance metrics to optimize the choice of mixup image pairs. We aim to produce an algorithm that surpasses Mixreg's results while retaining Mixreg's generalization capabilities.

4.1 ProcGen Benchmark

Among popular reinforcement learning benchmarks, OpenAI's ProcGen [Cobbe et al., 2020] is ideal for evaluating generalization for pixel-based learning. ProcGen's 16 game environments are similar to those from its predecessor ALE (the Arcade Learning Environment) [Bellemare et al., 2013], and some are even directly inspired from Atari games. Unlike ALE, game levels in ProcGen are procedurally generated: for each level there are different layouts, spawn locations of the agent and other entities, and visual aesthetics, among other details. Additionally there is a clear distinction between training and test sets.

4.2 Embedding Mixup on Latent space

Embedding mixup performs linear interpolation at the embedding layer instead of feeding mixed observations to the neural network. By combining pairs of low dimensional representations, the idea is to produce meaningful representations devoid of task-irrelevant information. Upon collecting rollout trajectories from the current policy, embedding mixup retrieves pairs of embedding level representations then linearly combines them:

$$\tilde{s} = \alpha\phi(s_i) + (1-\alpha)\phi(s_j) \tag{5}$$

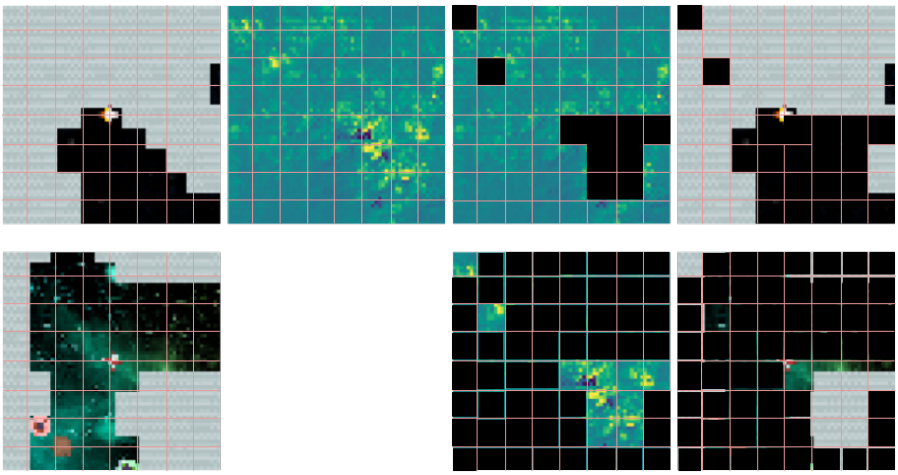
where $\alpha \sim \mathrm{Beta}(0.2, 0.2)$ is the mixup coefficient, $\phi(s)$ corresponds to the embedding level representation of the image s, s_i, s_j are different environment observations. The mixup result, a representation denoted $\tilde{s}$, is then fed back into the network and a loss value is computed. Embedding mixup uses the same objective as PPO.

4.3 Saliency-Guided Patch Mixup

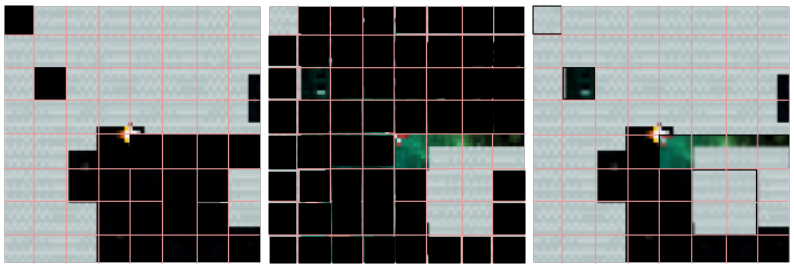
Saliency-guided patch mixup (Figure 1) seperates each observation of size 64 by 64 pixel into 64 square patches, where each patch is 8 by 8 pixels. This algorithm relies on the use of saliency maps to create masks. Since these masks highlight salient parts of an image input, they determine which parts of the image are preserved during mixup. Patches that have a high concentration of salient pixels will be colored entirely and kept, whereas patches with less salient information will be discarded. In saliency-guided patch mixup, we retain the Mixreg algorithm steps and change only the image mixup strategy. Instead of a naive linear interpolation of RGB values, we use saliency masks to extract patches from one image and inject the patches into a second image. The resulting image retains salient information from the game, and the background elements that are not in focus are altered so the agent may learn to dissociate task-relevant information from background elements. Unlike Mixreg and Embedding Mixup, the coefficient used to scale the supervision signals is not drawn randomly from the beta distribution, but instead it is determined based on the proportion of extracted patches.

4.4 Distance-Based Mixup

Wang's Mixreg performs random pairings of observations for its mixup step. Since Mixreg uses random observation pairs for mixup, resulting training observations are not guaranteed to be sensible



(a) Columns from left to right: image pair used for Co-Mixreg's mixup; saliency map produced from the first image of the input pair; image masks produced from the saliency map, for each input image; image masks laid over each respective input image.



(b) The right-most image is the additive result of the masked image pair (first and second images).

Figure 1: Co-Mixreg's mixup procedure, demonstrated on the `Caveflyer` game environment.

and to look like possible test-time scenarios. We hypothesized that using a similarity metric to choose mixup pairs will positively impact mixup model performance. The chosen metric needs to assess the degree of similarity of two observation backgrounds, because improper Mixreg mixups occur with dissimilar backgrounds. Although procedurally generated game levels have different spawn positions for the agent and enemies, these occupy few pixels relative to the background and have a negligible effect on mixup quality. We used Euclidean distance (2-norm) because small non-background elements have a negligible effect on this distance:

$$D(x, y) = \sqrt{(x_R - y_R)^2 + (x_G - y_G)^2 + (x_B - y_B)^2} \tag{6}$$

where x, y are two observations, R, G, B represent the red, green and blue values for each RGB image.

We hypothesize that when Euclidean distance between two images is minimized, then those two images are very likely to share similar level contexts and background colors. When Euclidean distance is too small however, the mixup observations will be too similar and the mixup result will not differ significantly from the original observations; in this case, the algorithm will perform similarly to PPO.

5 Experimental Results

We evaluate our methods on the ProcGen benchmark [Cobbe et al., 2020]. ProcGen has 16 different game environments total. Much like in Wang et al. [2020]'s Mixreg, we are evaluating all our models on six of these games (`Caveflyer, Climber, Dodgeball, Fruitbot, Jumper, Starpilot`) due to high computation costs. Wang et al. [2020] chose these six games because they are semantically different from one another. Thus, they are sufficient for testing generalization, and they allow us to

perform comparative studies between Mixreg and our models. Additionally, unlike ALE [Bellemare et al., 2013], each game level is procedurally generated, thus we can safely evaluate generalization.

In ProcGen, models are only given access to pixel-based observations. Specifically, the observations are of size 64×64 pixels over three color channels. The action space for all games is discrete: the agent can choose one among 15 actions for each observation frame. The creators of ProcGen offer "easy" and "hard" settings to accommodate those who have insufficient access to computing resources. "Easy" and "hard" do not refer to level difficulty; they indicate how many steps are required to complete a level, with easy mode requiring less steps than hard mode. These modes have the same game semantics. We chose easy mode for our experiments since we have limited computing resources. In easy mode, training is performed on a set of 200 levels (as opposed to hard mode's 500 training levels) for 25M timesteps (as opposed to hard mode's 200M timesteps). For each game and for each model, we average our results over three runs (three seeds).

Reported results are run on 3 different seeds for each of the six chosen games, for each model.

We begin by discussing results from easy and hard mode Mixreg with respect to the PPO baseline. Then, we will interpret results from our models: Embedding Mixup, Co-Mixreg, and their L2 distance-based variants. We will compare these results with the Mixreg baseline, as well as the PPO benchmark. Finally, we will show supporting experimental results and discussion.

5.1 How do Mixreg and PPO perform in easy mode and in hard mode?

It is important to note that Wang et al. [2020] reported the Mixreg results on "hard" mode, using the hard mode default number of training levels and timesteps. As such, we ran Mixreg in "easy" mode (with the corresponding default arguments) so we may make sensible comparisons between all presented models.

	PPO	MIXREG
CAVEFLYER	**5.833**	4.4
CLIMBER	7.233	**7.267**
DODGEBALL	**2.667**	1.533
FRUITBOT	25.4	**26.6**
JUMPER	6.333	**7.333**
STARPILOT	27.467	**34.067**

(a) Test results over 3 runs, easy mode.

[HARD]	PPO	MIXREG
CAVEFLYER	3.733	**6.1**
CLIMBER	4.0	**5.933**
DODGEBALL	1.333	**6.733**
FRUITBOT	11.133	**16.667**
JUMPER	3.333	**4.667**
STARPILOT	4.033	**8.567**

(b) Test results over 3 runs, hard mode.

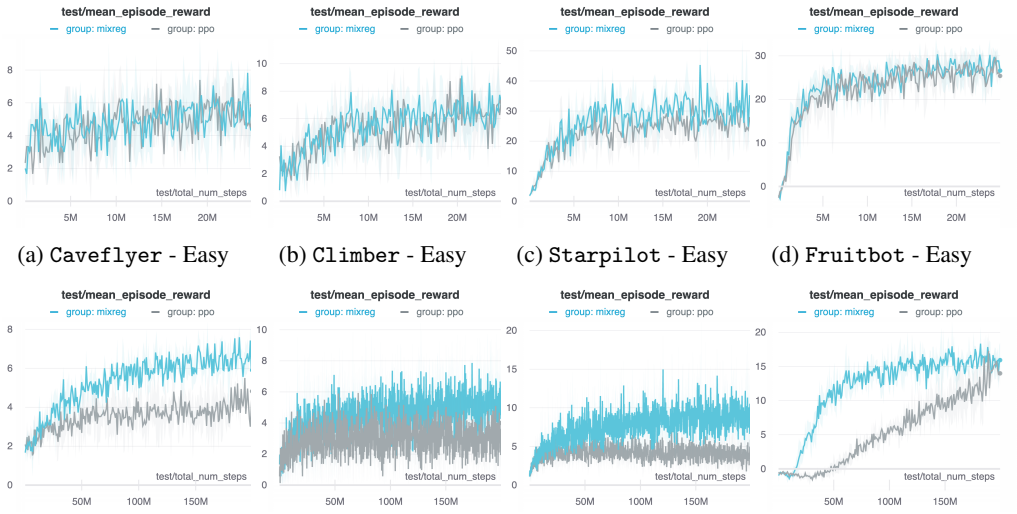
In all games run in hard mode, Mixreg surpasses PPO scores by nearly 120% on average. In easy mode, Mixreg outperforms PPO in four out of six game environments. Figure 2 shows that in hard mode, Mixreg surpasses PPO greatly, whereas in easy mode, Mixreg and PPO perform similarly. From easy to hard mode, PPO exhibits a dramatic drop in performance, whereas Mixreg's drop is minimal. The only difference between easy and hard ProcGen games is the number of timesteps required to complete a level. Since hard mode levels require more steps to complete, we can expect a drop in performance if a model is not sample efficient, i.e. it requires more training experience in order to reach the same level of performance as in easy mode. PPO's relatively inferior performance in hard mode could suggest that it is not as sample efficient as Mixreg. Next, we see how our methods compare to Mixreg.

5.2 Results

5.2.1 Embedding Mixup and Generalization

Upon first glance of Table 2, we notice that Mixreg has the most number of best scores among all models (top score in three out of six games, where one is a tie with Co-Mixreg). Every other model holds one highest score each.

Mixreg scores higher in `Climber`, `Jumper` and `Starpilot`, and Embedding Mixup holds higher scores in the other three game environments. Mixreg and Embedding Mixup also achieved the highest



(a) `Caveflyer` - Easy (b) `Climber` - Easy (c) `Starpilot` - Easy (d) `Fruitbot` - Easy

(e) `Caveflyer` - Hard (f) `Climber` - Hard (g) `Starpilot` - Hard (h) `Fruitbot` - Hard

Figure 2: Mixreg and PPO in easy mode (top) and hard mode (bottom), for four of the ProcGen games.

	CAVEFLYER	CLIMBER	DODGEBALL	FRUITBOT	JUMPER	STARPILOT	Average
PPO	5.833	7.233	**2.667**	25.4	6.333	27.467	12.489
MIXREG	4.4	**7.267**	1.533	26.6	**7.333**	**34.067**	13.533
CO-MIXREG	7.2	6.7	2.0	25.233	**7.333**	21.467	11.656
MIXREGL2	**7.333**	5.067	2.4	25.9	7	31.8	13.25
EMBDMIX	7.2	6.467	**2.667**	27.2	5.667	30.767	13.328
EMBDMIXL2	5.167	5.633	1.333	**29.833**	7	22.667	11.939
Average per Game	6.189	6.394	2.1	26.694	6.778	28.039	
Std Dev per Game	1.243	0.884	0.575	1.706	0.655	5.105	
Median per Game	6.517	6.583	2.2	26.25	7.0	29.117	

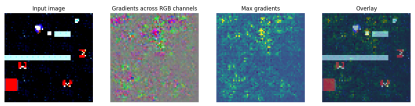
Table 2: Test results over 3 seeds in easy mode. Runs are performed over 25M timesteps. Best results for each game is bolded. The right-most column describes the average score of each algorithm achieved over all six games.

average score across all games. The same goes for PPO and Embedding Mixup: PPO surpasses Embedding Mixup in 2 games and ties once. Further analysis of scores relative to the average shows surprising results: Mixreg scores above the average on three games, whereas Embedding Mixup scores above the average in five out of six games. This could suggest that Embedding Mixup has better overall performance and better generalization abilities.

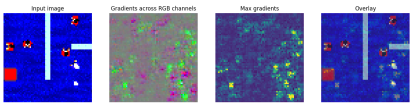
From the trained models' saliency maps (Figure 3), we notice that Embedding Mixup manages to identify specific clusters of "relevant" pixels: task-relevant details are brightly illuminated in contrast to the dull, dark task-irrelevant pixel clusters. The delimitations between bright and dull pixels are sharp. On the other hand, Mixreg tends to illuminate most pixels in each observation, regardless of its importance in each task. Although some hotspots are identified and are lighter in color, the edges between task-relevant and task-irrelevant pixel clusters are not as clearly defined. This means that the Mixreg model learns from both task-relevant details from task-irrelevant visual elements.

5.2.2 Comparing Mixreg's Naive Mixup and Co-Mixreg's Saliency-Guided Patch Mixup

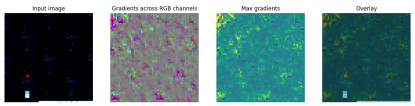
Of the six games we evaluated our models on, Mixreg scores higher on three games, Co-Mixreg scores higher on two others, and both are tied for top score in the `Jumper` game. Although Mixreg



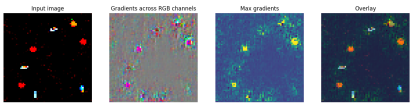
(a) Dodgeball - Mixreg



(b) Dodgeball - Embedding Mixup



(c) Fruitbot - Mixreg



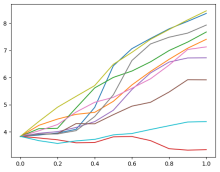
(d) Fruitbot - Embedding Mixup

Figure 3: Trained Mixreg, Embedding Mixup models' saliency maps.

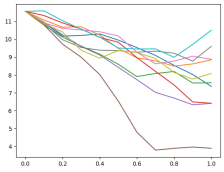
and Co-Mixreg final test scores differ marginally, Mixreg scores higher on more games and learns a smoother policy. Co-Mixreg's weakness is its lack of ability to generalize to unseen environments.

Figure 4 compares Mixreg (top row) and Co-Mixreg (bottom row) in terms of their ability to generalize to unseen environments. Keep in mind that we only use 10 Mixup coefficients, meaning none of our plots are smooth by the mathematical definition. Instead, we refer to relative smoothness when we compare two models (closer to linear means smoother). Most runs in the Mixreg plots show monotonic functions; all runs produce a relatively smooth state-value function. In the Co-Mixreg plots, we observe no monotonicity. These runs show kinks as well as steep changes in state-value.

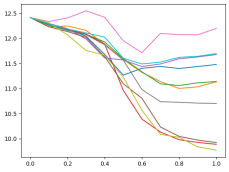
We can justify this result by considering the differences between Mixreg and Co-Mixreg's mixup algorithm. Mixreg's mixup result is the product of a linear mixup, so it follows that a linear interpolation between two observations produces a smooth, linear function. Co-Mixreg's mixup is not linear: it depends on the model's trained network to determine salient pixels, which are then used to extract patches of pixels for mixup. The model's current parameters at each timestep determine the number of patches involved in the mixup as well as the location of said patches. From our experiments, it follows that Mixreg's linear mixup produces smoother state-value functions for unseen environments. Co-Mixreg, while inventive in its nature, can hardly surpass Mixreg's smooth interpolation to unseen environments. Thus, we conclude that Mixreg is superior to Co-Mixreg in terms of generalization.



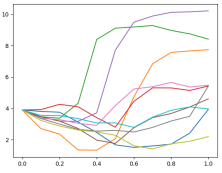
(a) `Caveflyer` - Mixreg



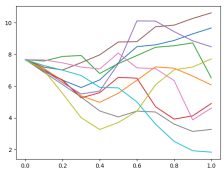
(b) `Climber` - Mixreg



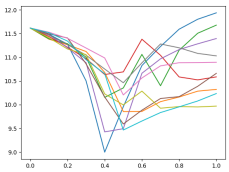
(c) `Fruitbot` - Mixreg



(d) `Caveflyer` - Co-Mixreg



(e) `Climber` - Co-Mixreg



(f) `Fruitbot` - Co-Mixreg

Figure 4: Interpolation of the value function through two different states. Comparison of Mixreg (top) and Co-Mixreg (bottom) models' learned state-value function on unseen, linearly combined observations, for three game environments. The x-axis holds mixup coefficients in the range $[0, 1]$. The y-axis is the state-value for each mixup result, determined by a trained model.

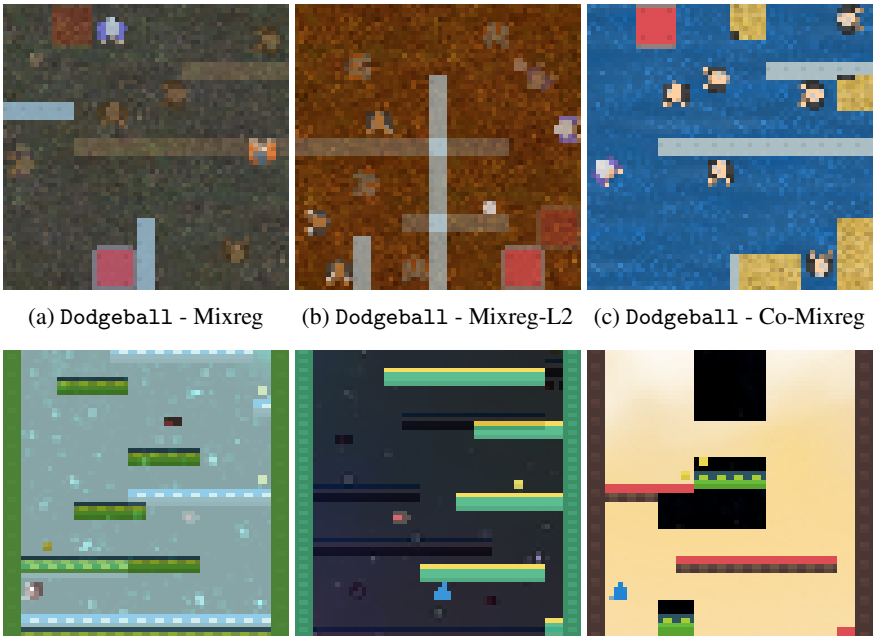
5.2.3 How Does Mixing Distance Affect the Outcome of Learning?

We initially hypothesized that using a similarity metric for mixup pairing can posivitely affect performance, specifically that a smaller distance creates more cohesive pairings similar to training examples. However, when we ran ablations to compare model performance for different mixing distances, we find that a greater mixing distance produces better results. Although the mixup result is more sensible to the human eye, forcing a smaller mixing distance leads to less diversity during the training phase. The gain from using realistic test-time examples during training is overshadowed by the loss in sample diversity.

	CO-MIXREG	MIXREG-L2	EMBDMIX-L2
CAVEFLYER	1000	1000	1500
CLIMBER	2000	2000	2000
DODGEBALL	500	1500	500
FRUITBOT	2000	500	2000
JUMPER	2000	1500	500
STARPILOT	2000	500	2000

Table 3: Optimal L2 mixing distance for each of Co-Mixreg, Mixreg-L2 and Embedding Mixup-L2 for all games. Table values are one of 500, 1000, 1500 or 2000. Given a batch observation, the L2 distance to other batch observations is computed then these images are ranked from most similar to least similar. Table values represent the similarity rank chosen for mixup.

As shown in Table 3, there are eight instances where a greater mixup rank of 2000 is optimal, as opposed to five instances where a lesser rank of 500 is preferred. Thus, contrary to our initial hypothesis, our L2-distance based methods Mixreg-L2, Co-Mixreg and Embedding Mixup-L2 must choose image pairings that have dissimilar backgrounds, i.e. they maximise distance (visual dissimilarity) between image pairs. Most games benefit from larger mixing distances (`Climber`) but some perform better with smaller mixing distances (`Dodgeball`).



(a) `Dodgeball` - Mixreg (b) `Dodgeball` - Mixreg-L2 (c) `Dodgeball` - Co-Mixreg

(d) `Climber` - Mixreg (e) `Climber` - Mixreg-L2 (f) `Climber` - Co-Mixreg

Figure 5: Training mixup observations for models using L2-distance mixup, using L2-distance rank 2000 (i.e. choosing dissimilar image pairings).

`Dodgeball` is a particularly difficult game to learn using mixup techniques, since it is a game where the agent moves across the screen to reach an objective (a red square) while avoiding enemies and walls. dissimilar mixups tend to produce insensible images. In Figure 5b, we see that Mixreg-L2's mixup has the following flaws:

(i) There are two objectives (red squares at the bottom right);

(ii) Since walls from both mixup images appear, they sometimes block off entire sections of the environment, meaning there are no possible paths to the correct objective;

(iii) The agent appears twice (purple character).

Figure 5a shows that Mixreg shares some of the flaws above. However, since Mixreg's mixup is random, it will not consistenly pick image pairs that are visually different. Thus, the chances of producing a flawed training image are lower. Figure 5c shows that Comixreg produces results that are valid scenarios in ProcGen and that look sensible to the human eye. The downside of this mixup is that it can entirely remove important information from the environment or falsify it, namely the position of platforms in Figure 5f. Although Mixreg is less sensible to the human eye, it does not lose any information from the game environment. Instead, it produces images with extraneous detail.

Even after choosing the most optimal L2-distance, Mixreg and Embedding Mixup still perform better than their respective L2-distance counterparts on four out of six game environments (Table 2). As dicussed in §5.2.2, Co-Mixreg does not bring a significant improvement relative to our Mixreg benchmark.

6 Discussion

In this work, we propose pairwise image augmentation techniques to increase sample diversity for reinforcement learning tasks. We find that Mixreg's data augmentation produces invalid training images that differ greatly from test-time scenarios. In particular, Mixreg uses a naive linear combination of image pairs, so extraneous details are included in training inputs. We introduce methods that linearly combine embedding level representations, that use saliency detection to combine image patches and that mix dissimilar image pairs to increase sample diversity. Our work compares the proposed methods with Mixreg and PPO to highlight the advantages and disadvantage of each algorithm. We demonstrate that although Mixreg's data augmentation creates invalid game scenarios, it learns smoother policies than Co-Mixreg since Mixreg's mixup is linear and Co-Mixreg's isn't. While Embedding Mixup does not score as high as Mixreg, it consistently achieves above average performance for each game (compared to other algorithms) and produces more pertinent saliency maps; this means that mixup at the embedding layer can learn good representations and improve generalization. Finally, Mixreg trumps the performance of all L2 distance-based algorithms. Unlike the algorithms that use L2-norm for mixup, Mixreg's mixup results are widely distributed since it uses random pairings: some results are minimally perturbed because they are produced from similar pairings, whereas others are discordant because they are produced from dissimilar pairings. In the future, we can explore more sophisticated embedding layer manipulations since it shows promise in terms of generalization. We also propose to study why Mixreg retains its performance in both easy and hard distributions in ProcGen, whereas PPO exhibits a dramatic performance drop. Since game levels from the hard distribution require more steps to complete, understanding the performance gaps between easy and hard distributions can prove vital to our understanding of how reinforcement learning agents learn. We hope these ideas will be explored in future work.

References

Julius Adebayo, Justin Gilmer, Michael Muelly, Ian Goodfellow, Moritz Hardt, and Been Kim. Sanity checks for saliency maps, 2020.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, Jun 2013. ISSN 1076-9757. doi: 10.1613/jair.3912. URL http://dx.doi.org/10.1613/jair.3912.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

Dan Cireşan, Ueli Meier, and Juergen Schmidhuber. Multi-column deep neural networks for image classification, 2012.

Dan C. Cireşan, Ueli Meier, Jonathan Masci, Luca M. Gambardella, and Jürgen Schmidhuber. High-performance neural networks for visual object classification, 2011.

Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning, 2020.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

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

Jang-Hyun Kim, Wonho Choo, Hosan Jeong, and Hyun Oh Song. Co-mixup: Saliency guided joint mixup with supermodular diversity, 2021.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels, 2021.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data, 2020.

Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning, 2021.

Roberta Raileanu, Max Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in deep reinforcement learning, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

P.Y. Simard, D. Steinkraus, and J.C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *Seventh International Conference on Document Analysis and Recognition, 2003. Proceedings.*, pages 958–963, 2003. doi: 10.1109/ICDAR.2003.1227801.

Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps, 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, 2017.

Kaixin Wang, Bingyi Kang, Jie Shao, and Jiashi Feng. Improving generalization in reinforcement learning with mixture regularization, 2020.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019.

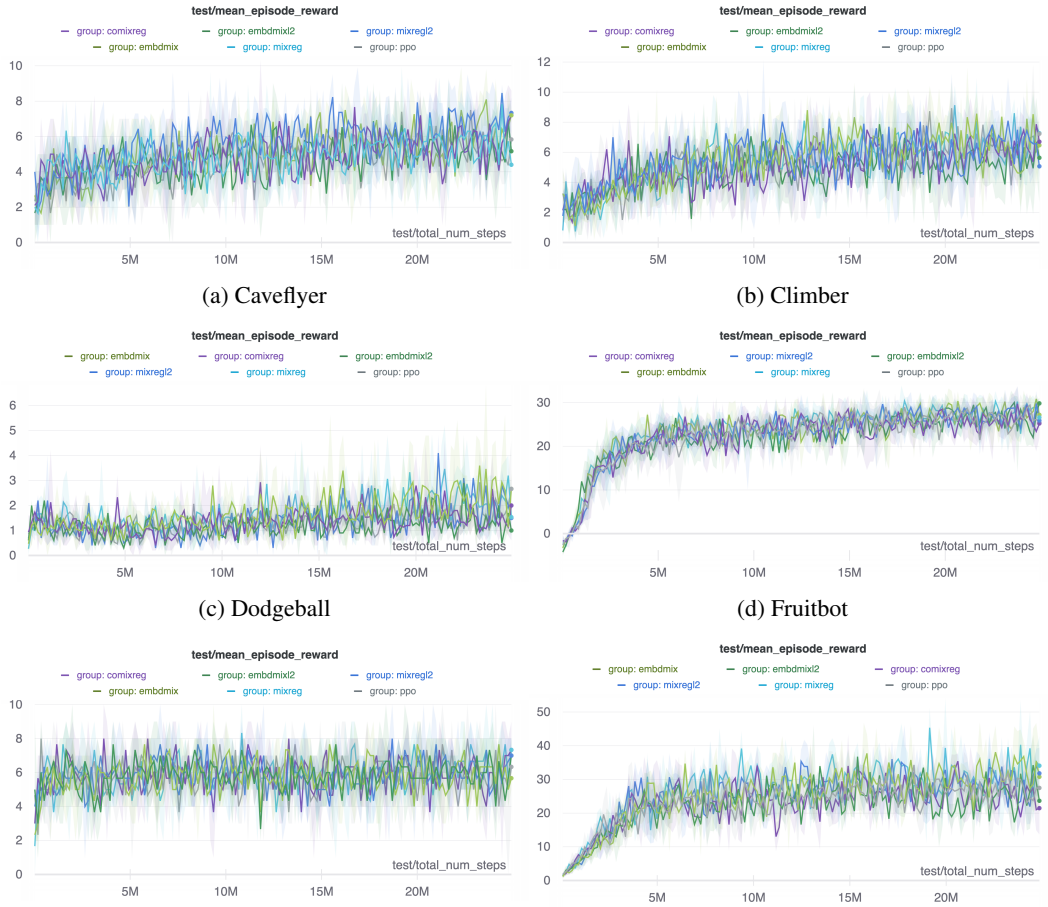
Amy Zhang, Nicolas Ballas, and Joelle Pineau. A dissection of overfitting and generalization in continuous reinforcement learning, 2018a.

Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization, 2018b.

7 Appendix

7.1 ProcGen Test Results

7.1.1 All Models



(a) Caveflyer

(b) Climber

(c) Dodgeball

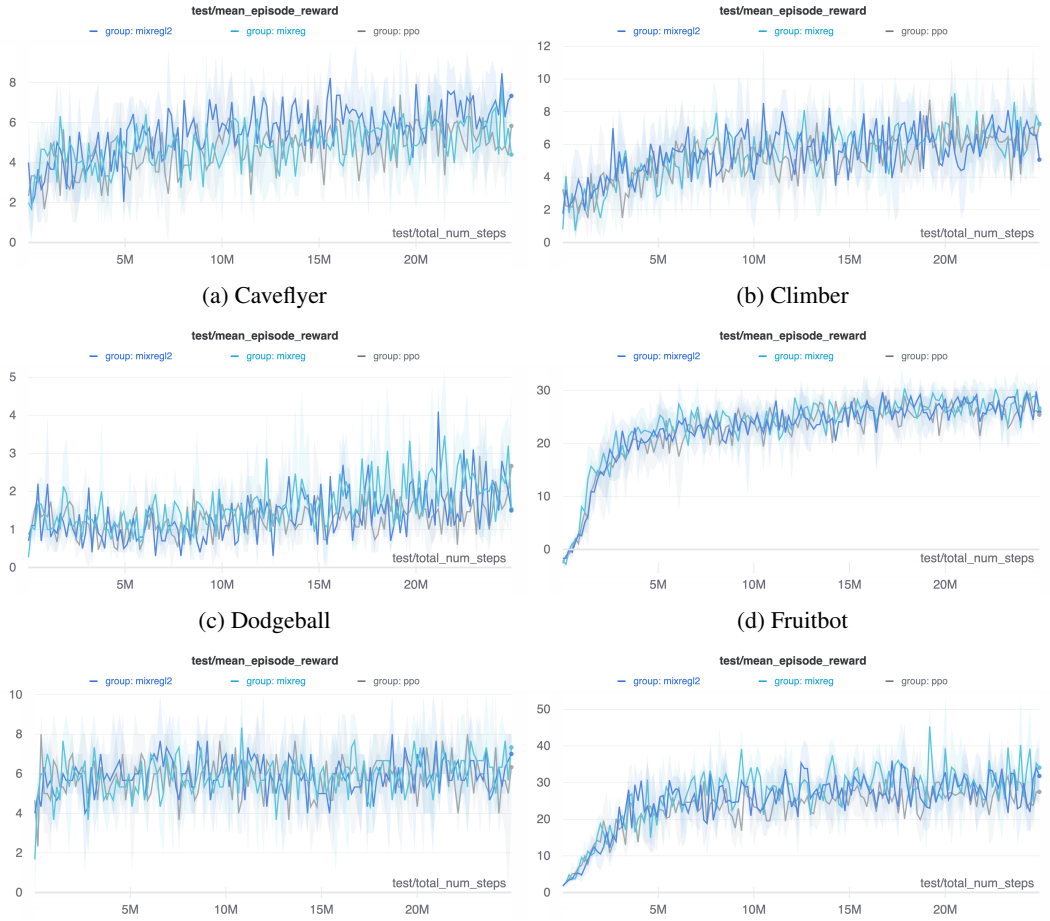
(d) Fruitbot

(e) Jumper

(f) Starpilot

Figure 6: Test results for all models (PPO, Mixreg, Co-Mixreg, Embedding Mixup, Mixreg-L2 and Embedding Mixup-L2) over six ProcGen games. There are 3 runs using different seeds for each model for each game.

7.1.2 Mixture Regularization Models



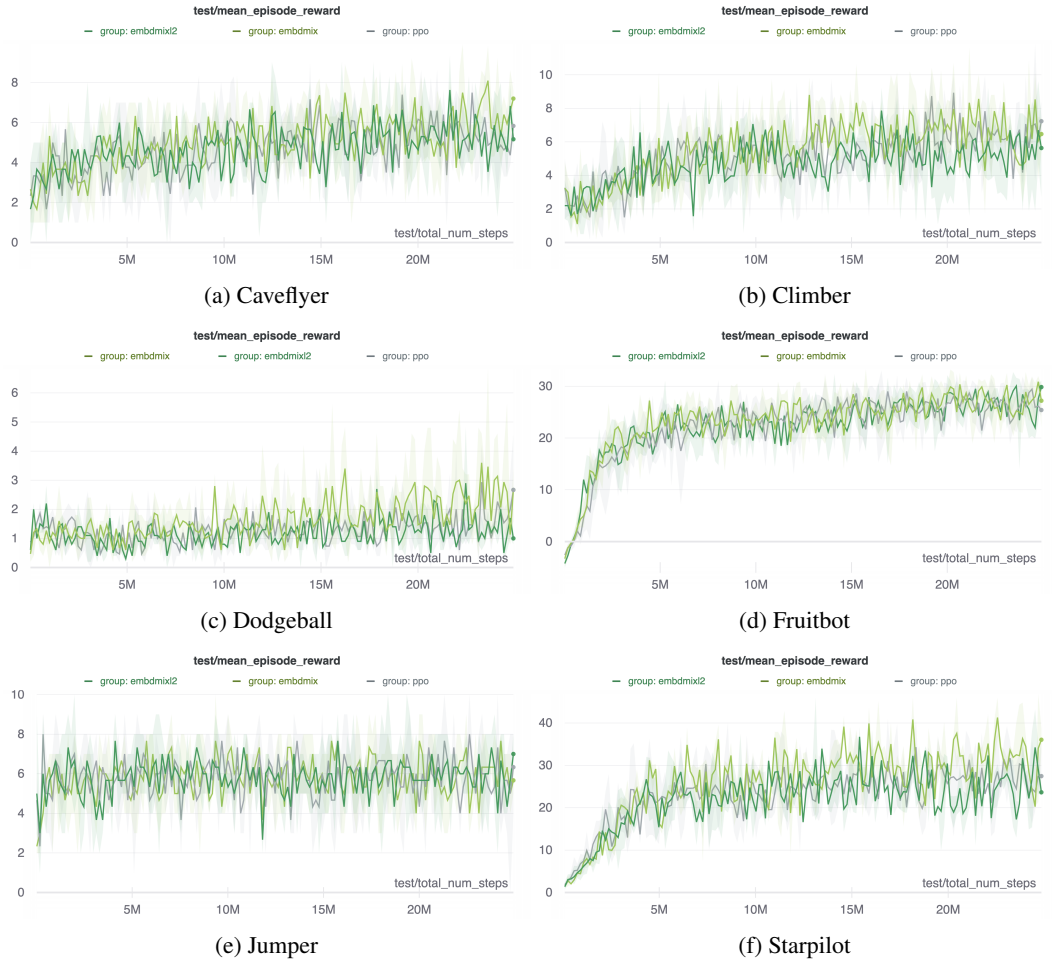
(a) Caveflyer (b) Climber

(c) Dodgeball (d) Fruitbot

(e) Jumper (f) Starpilot

Figure 7: Test results for Mixreg model variations and PPO over six ProcGen games. There are 3 runs using different seeds for each model for each game.

7.1.3 Embedding Mixup Models



(a) Caveflyer

(b) Climber

(c) Dodgeball

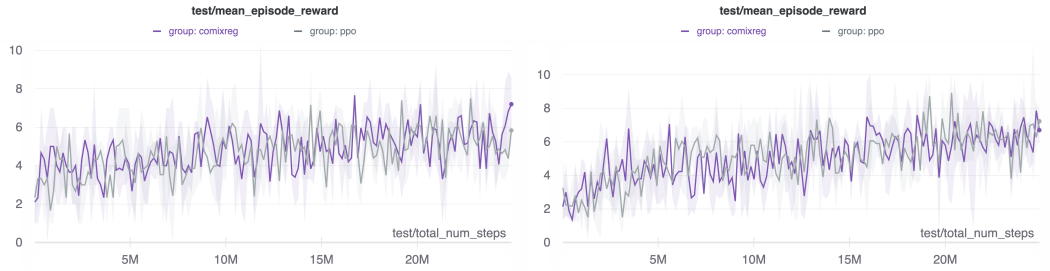
(d) Fruitbot

(e) Jumper

(f) Starpilot

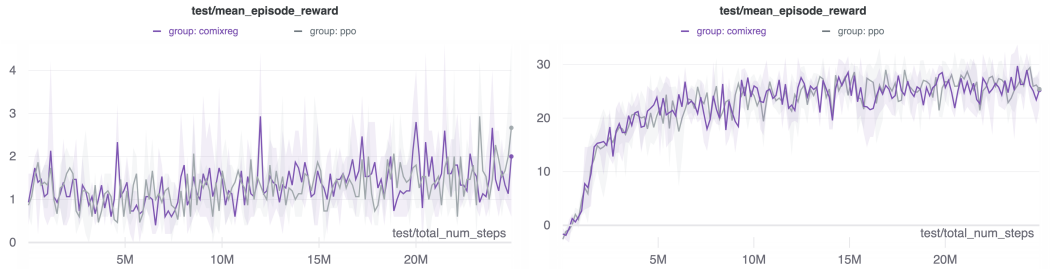
Figure 8: Test results for Embedding Mixup model variations and PPO over six ProcGen games. There are 3 runs using different seeds for each model for each game.

7.1.4 Saliency-based Patch Mixup Models



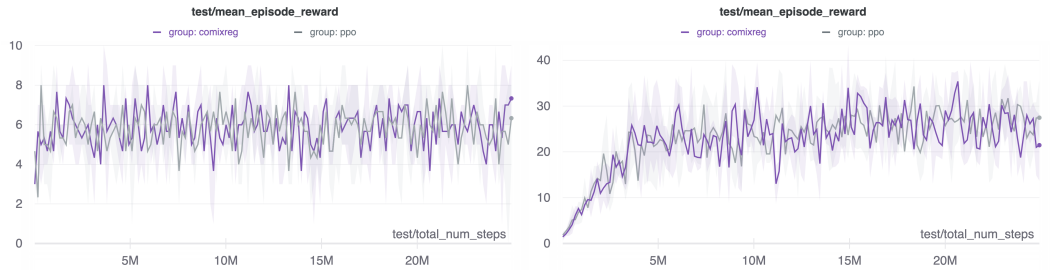
(a) Caveflyer

(b) Climber



(c) Dodgeball

(d) Fruitbot

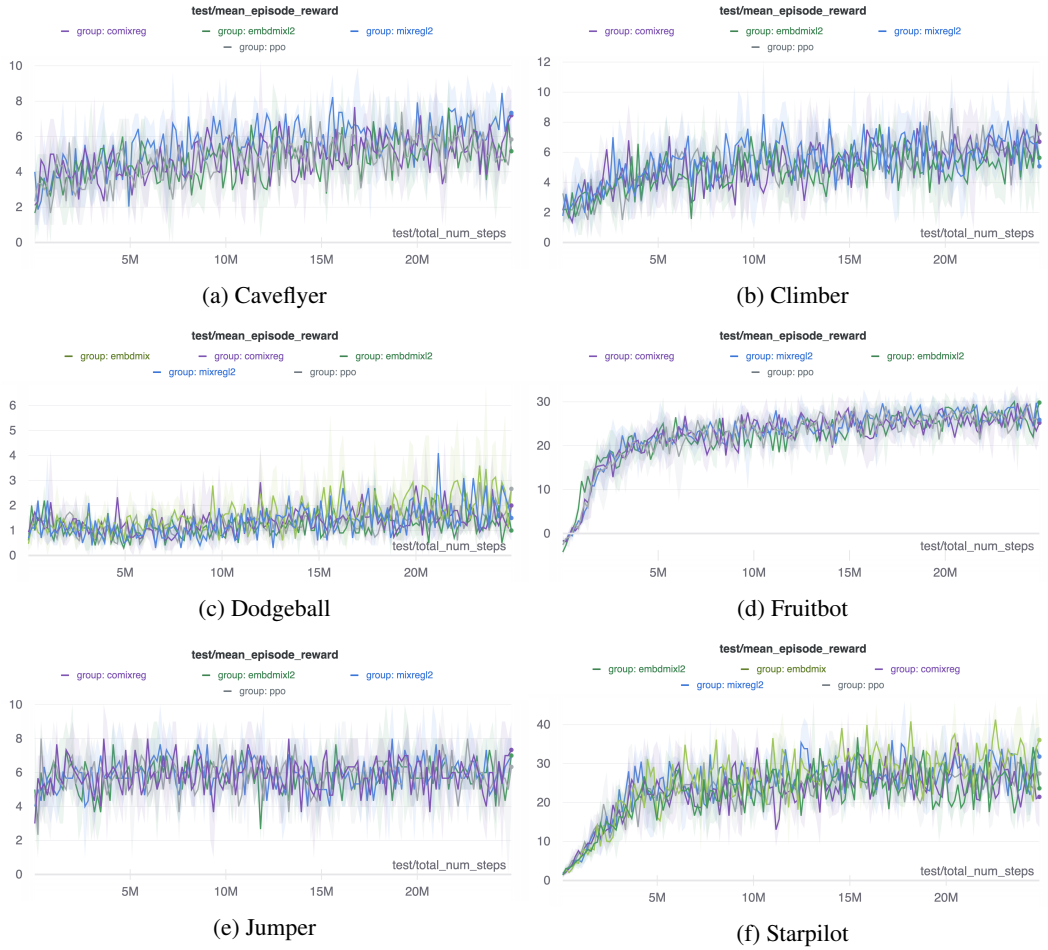


(e) Jumper

(f) Starpilot

Figure 9: Test results for Co-Mixreg and PPO over six ProcGen games. There are 3 runs using different seeds for each model for each game.

7.1.5 Distance-based Mixup Models



(a) Caveflyer

(b) Climber

(c) Dodgeball

(d) Fruitbot

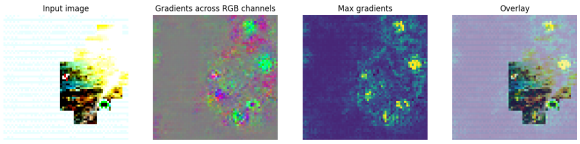
(e) Jumper

(f) Starpilot

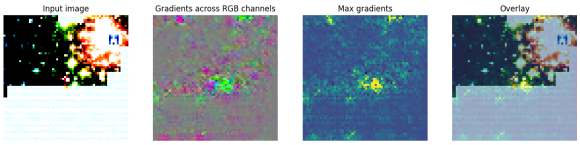
Figure 10: Test results for distance-based Mixup models (Co-Mixreg, Mixreg-L2 and Embedding Mixup-L2) and PPO over six ProcGen games. There are 3 runs using different seeds for each model for each game.

7.2 Saliency Maps

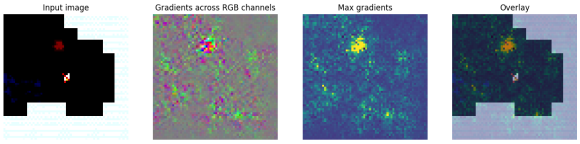
7.2.1 Caveflyer Game Environment



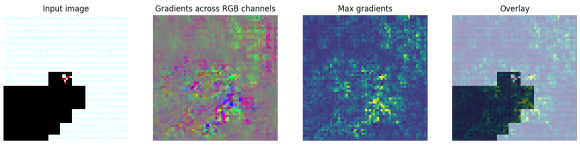
(a) PPO



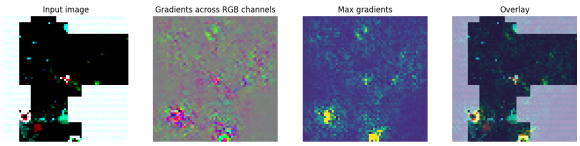
(b) Co-Mixreg



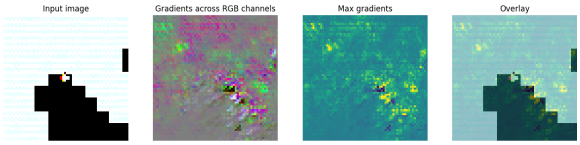
(c) Mixreg



(d) Embedding Mixup



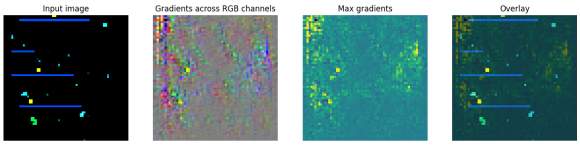
(e) Mixreg-L2



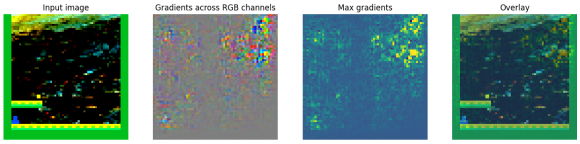
(f) Embedding Mixup-L2

Figure 11: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

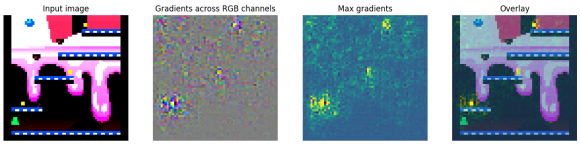
7.2.2 Climber Game Environment



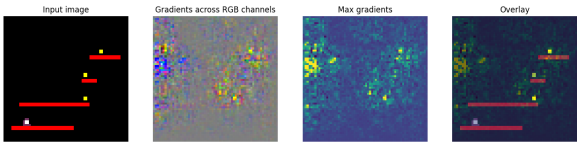
(a) PPO



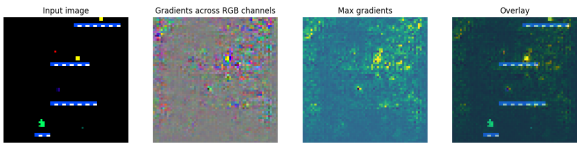
(b) Co-Mixreg



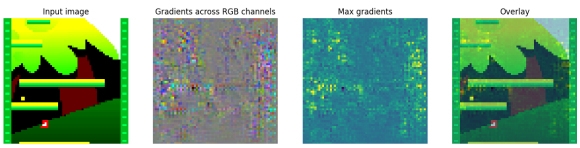
(c) Mixreg



(d) Embedding Mixup



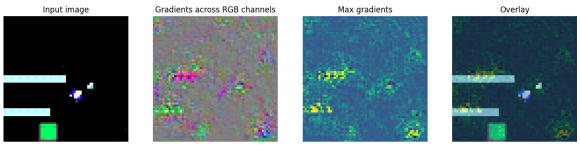
(e) Mixreg-L2



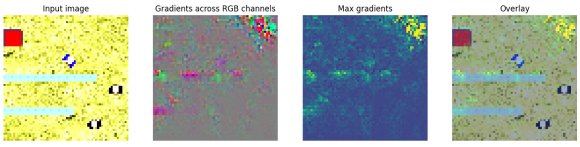
(f) Embedding Mixup-L2

Figure 12: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

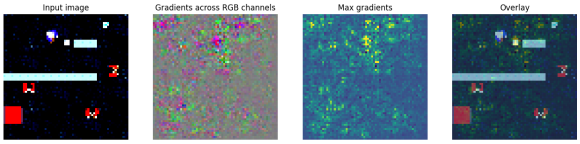
7.2.3 Dodgeball Game Environment



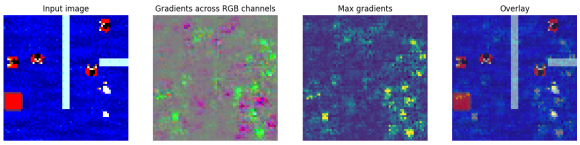
(a) PPO



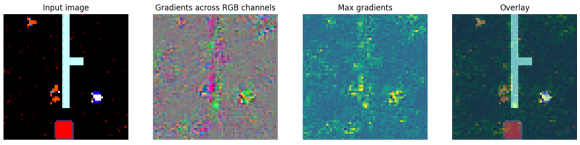
(b) Co-Mixreg



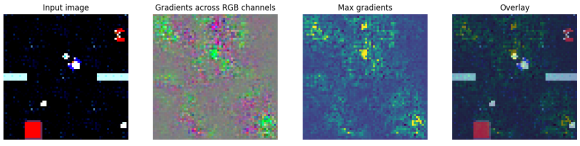
(c) Mixreg



(d) Embedding Mixup



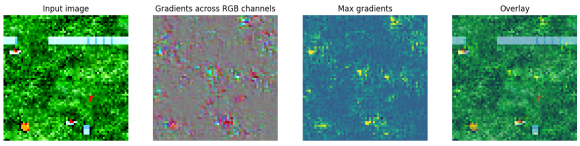
(e) Mixreg-L2



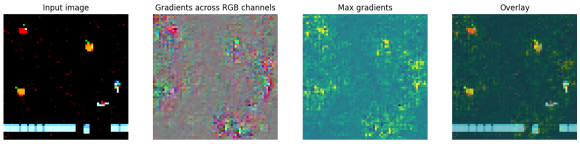
(f) Embedding Mixup-L2

Figure 13: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

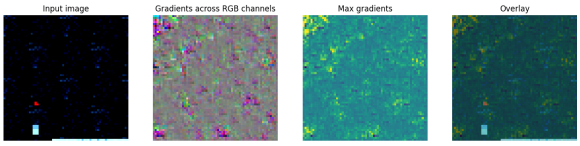
7.2.4 Fruitbot Game Environment



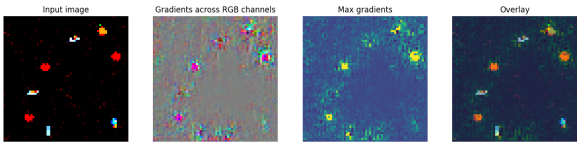
(a) PPO



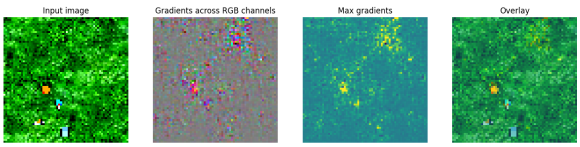
(b) Co-Mixreg



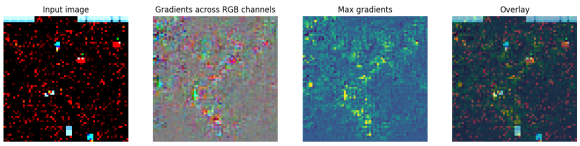
(c) Mixreg



(d) Embedding Mixup



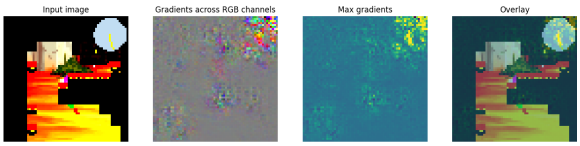
(e) Mixreg-L2



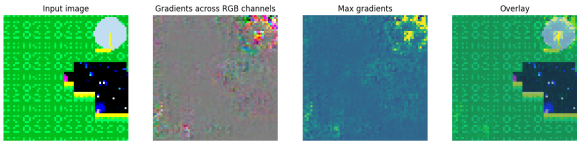
(f) Embedding Mixup-L2

Figure 14: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

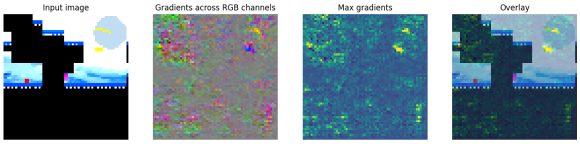
7.2.5 Jumper Game Environment



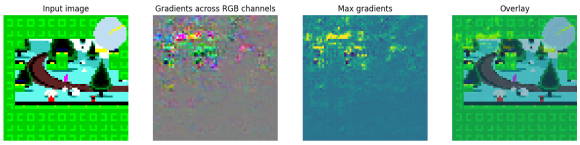
(a) PPO



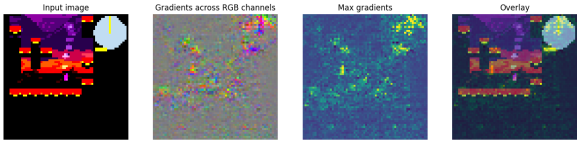
(b) Co-Mixreg



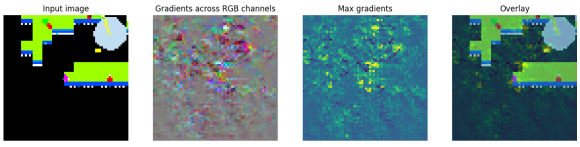
(c) Mixreg



(d) Embedding Mixup



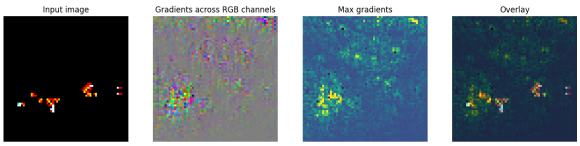
(e) Mixreg-L2



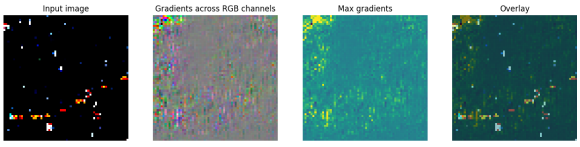
(f) Embedding Mixup-L2

Figure 15: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

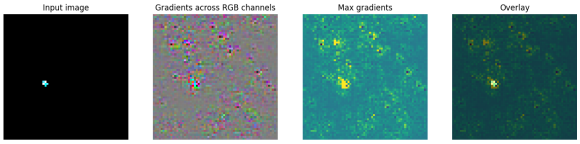
7.2.6 Starpilot Game Environment



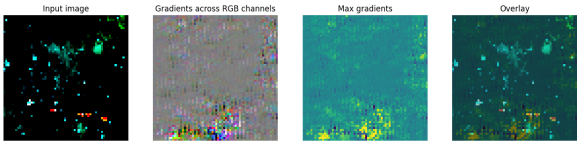
(a) PPO



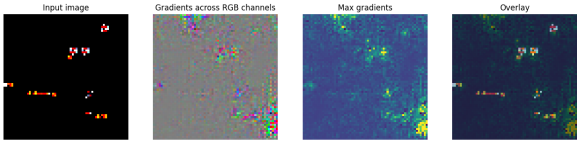
(b) Co-Mixreg



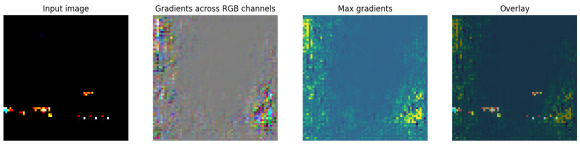
(c) Mixreg



(d) Embedding Mixup



(e) Mixreg-L2



(f) Embedding Mixup-L2

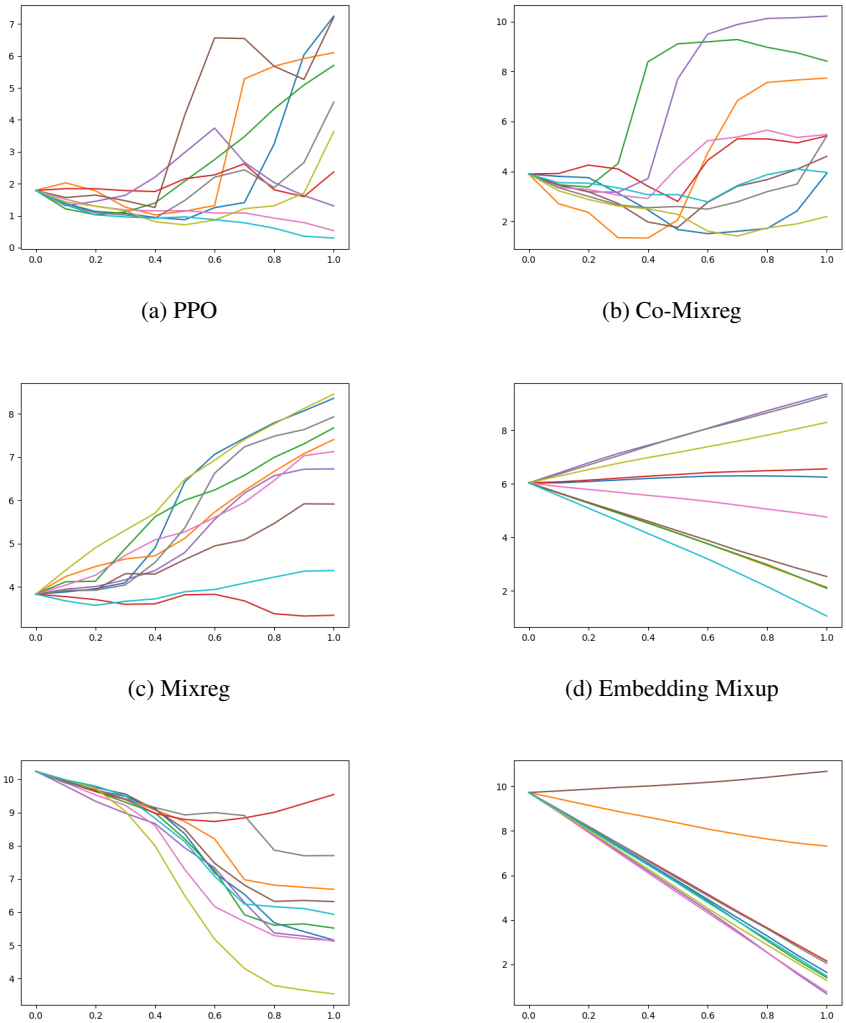
Figure 16: Trained models' saliency. Yellow and blue pixels are high and low gradient values, resp.

7.3 Interpolation of State Value of Mixup Observation Pairs

For the following experiments, we use a trained model and sample pairs of observations from the game environment. Each pair is mixed using mixup and a mixup coefficient between 0 and 1, inclusively. We chose increments of 0.1 for this mixup coefficient. The horizontal x-axis holds the mixup coefficient values. The vertical y-axis reads the learned state-value of the model, given a mixup observation. To sample observation pairs, we sampled 10 observations and compared each to one same sample. The common sample's state-value can be found by reading the graph at $x = 0$, and the 10 samples' state value are at $x = 1$.

Due to the nature of Embedding Mixup, interpolations of observation pairs produce a linear function. This agrees with our finding that Embedding Mixup achieves the best level of generalization among all presented models, since the learned state value function is smooth for unseen observations.

7.3.1 Caveflyer Game Environment



(a) PPO

(b) Co-Mixreg

(c) Mixreg

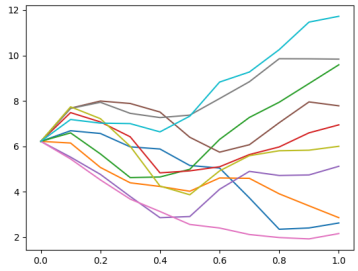
(d) Embedding Mixup

(e) Mixreg-L2

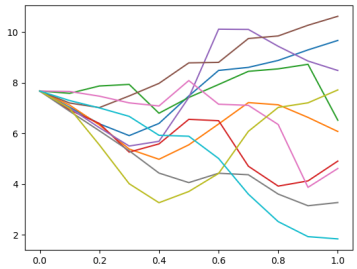
(f) Embedding Mixup-L2

Figure 17

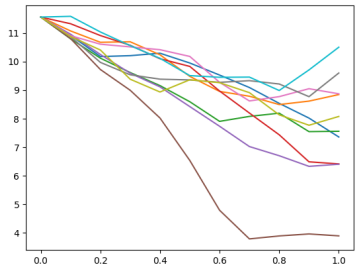
7.3.2 Climber Game Environment



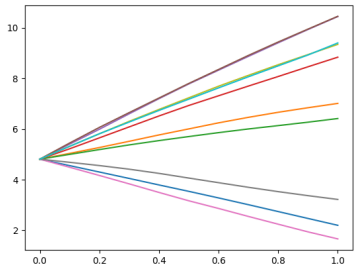
(a) PPO



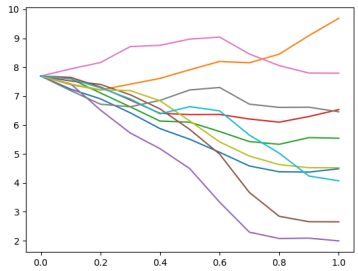
(b) Co-Mixreg



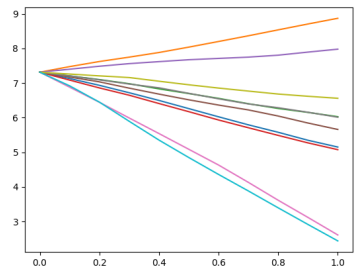
(c) Mixreg



(d) Embedding Mixup



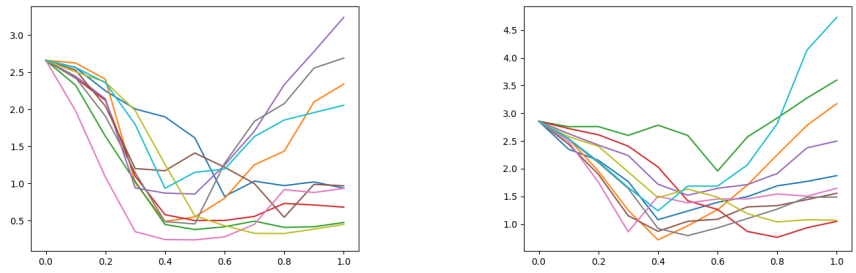
(e) Mixreg-L2



(f) Embedding Mixup-L2

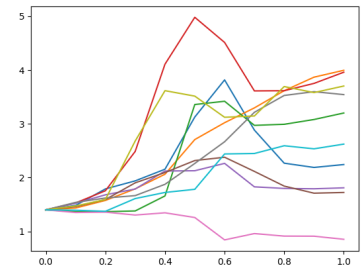
Figure 18

7.3.3 Dodgeball Game Environment

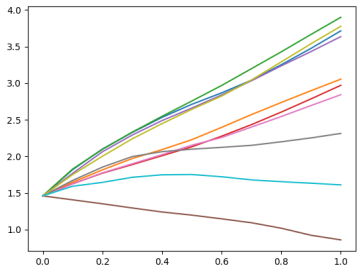


(a) PPO

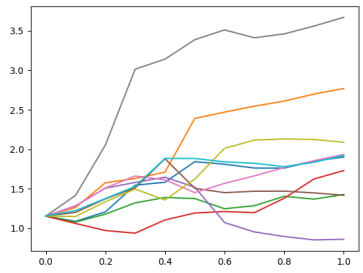
(b) Co-Mixreg



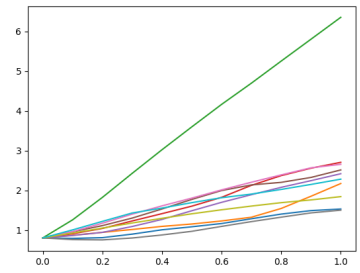
(c) Mixreg



(d) Embedding Mixup



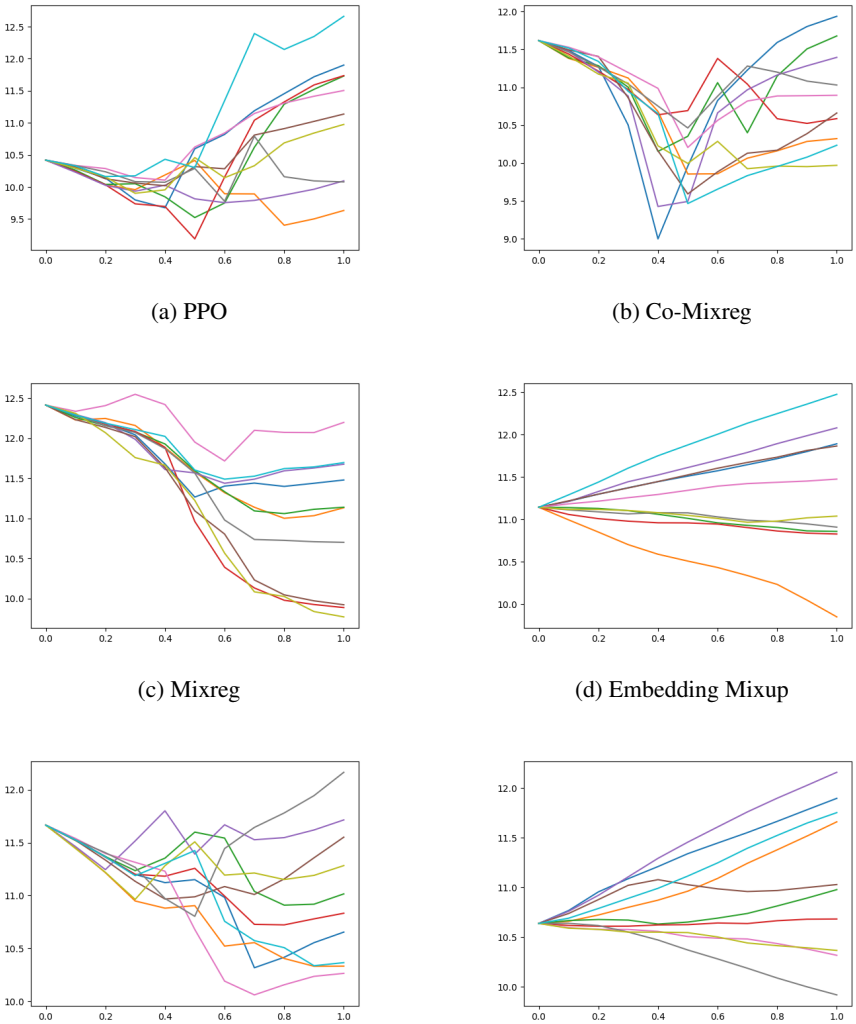
(e) Mixreg-L2



(f) Embedding Mixup-L2

Figure 19

7.3.4 Fruitbot Game Environment



(a) PPO

(b) Co-Mixreg

(c) Mixreg

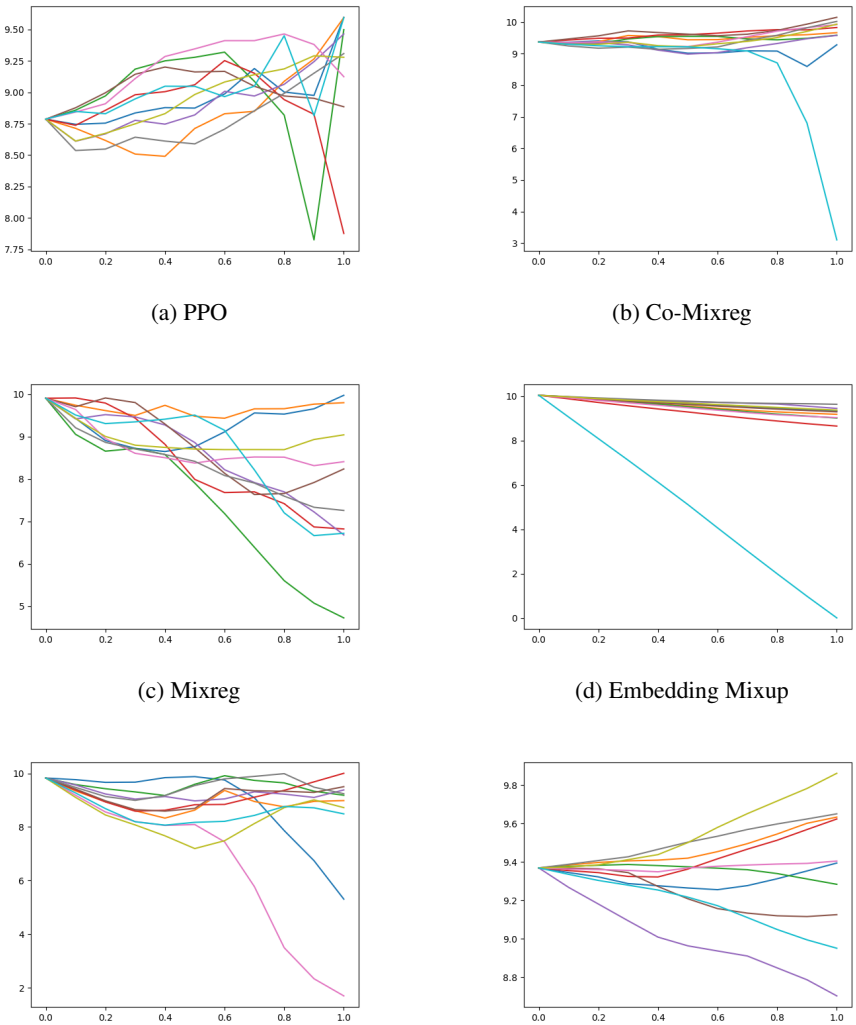
(d) Embedding Mixup

(e) Mixreg-L2

(f) Embedding Mixup-L2

Figure 20

7.3.5 Jumper Game Environment



(a) PPO

(b) Co-Mixreg

(c) Mixreg

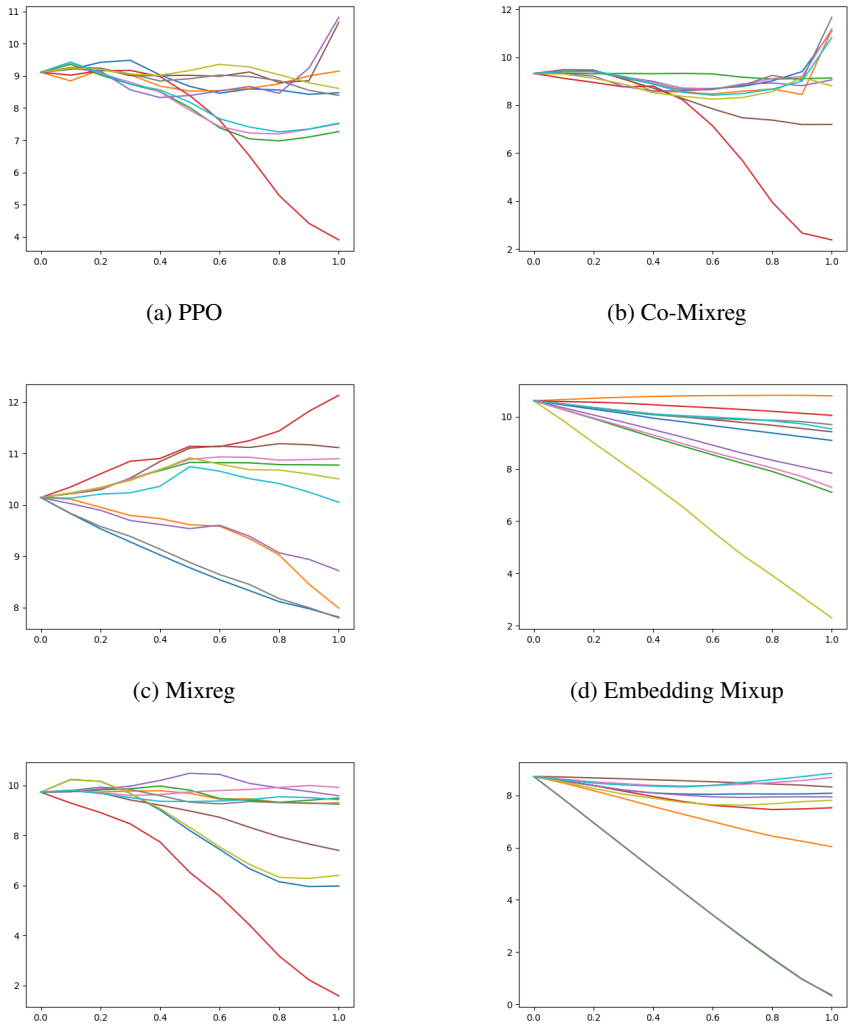
(d) Embedding Mixup

(e) Mixreg-L2

(f) Embedding Mixup-L2

Figure 21

7.3.6 Starpilot Game Environment



(a) PPO

(b) Co-Mixreg

(c) Mixreg

(d) Embedding Mixup

(e) Mixreg-L2

(f) Embedding Mixup-L2

Figure 22