Deep Reinforcement Learning Generalization By Field Of View Reduction

Qiye Chen
Northeastern University
chen.qiy@northeastern.edu

Abstract

We introduce a new data augmentation technique that improves the generalization of deep reinforcement learning agents into unseen but similar environments, when the environment observation is of the right type, which centers the agent at the middle of the screen and has no HUD on the screen margins. The technique is as simple as just cropping a middle portion of an observation and then re-scaling the portion back to the original observation's dimensions, which mimics the behavior of zooming in the screen and hence the field of view is reduced, allowing agents to consider fewer and closer features more. Experiences show that this technique not only improves generalization, but also improves sample efficiency in especially tough environments. Project code can be found at https://github.com/jacky5124/DRL-Generalization-By-FOV-Reduction.

Introduction

Deep reinforcement learning, a combination of deep learning and reinforcement learning, enables AI agents to perceive complex environments and act rationally to achieve goals in real-world like settings. DeepMind is one of the leaders in the frontier of the field, who proposed several important algorithms. A major early milestone is the Deep Q-Network (Mnih et al. 2015), which learns to play various Atari games at or above human levels, with only the raw video frame pixels as state representations and the game scores as rewards, and the valid button presses on Atari console controller as actions. Although not perfect, DQN had successfully let people believe that deep reinforcement learning is one of the most likely approaches to general AI, which led to further improvements later on. Double DQN (van Hasselt, Guez, and Silver 2015) was proposed to make the distribution of learned Q-values more proportional to the distribution of optimal ones, so that the learned policy would be closer to the optimal one. Prioritized Experience Replay (Schaul et al. 2016) was proposed to enable DQN to pick more worthy data to train, measured by quantities proportional to temporal difference errors, so that the agent can learn the most from past experience. Dueling Network Architecture (Wang et al. 2016) was introduced to enable DQN to also estimate state values and action advantages, so the agent can infer more normalized Q-values, whereas in original DQN Q-values are directly estimated. The examples of improvements can continue and are far from exhausting.

Despite major breakthroughs in deep reinforcement learning, it still faces an important problem all the time, which is the generalization in unseen but similar instances of the same kind of environment. It is obvious that the training of AI agents is only meaningful if agents can carry out tasks in such settings, for example we would like a robot to move things to destination in any warehouse, after training it in one warehouse. One of the major cause of being difficult the generalize is that for a long time, the agents were trained and tested in the same environment instance, which is like using the exact same data to train and test neural networks in supervised learning. There are several techniques suggested for improving generalization, such as regularization methods like L2 and dropout (Farebrother, Machado, and Bowling 2020), data augmentation methods like cutout (DeVries and Taylor 2017), as well as advanced methods like training in environments with procedurally generated levels (Justesen et al. 2018). The procedural level generation, in particular, transforms reinforcement learning agents into traditional machine learning like settings, since the instances of environments encountered in training and testing are different.

The aforementioned introduction motivates and inspires this project, which proposes a new way to improve deep reinforcement learning generalization in eligible environments. If we take a look at the training of deep reinforcement learning agents especially in the environments like *Sonic the HedgeHog* suggested by Retro Contest (Nichol et al. 2018), we can notice that the agents take the raw pixels of whole screen as input, regardless of the size of the agent character (the portion of the screen it occupies) and the action range of the agent character (the distance it can go by one step), and the agent character often appears very small on the screen. However, the agent character is almost always centered at the middle of the screen, so that whenever the agent character moves, the environment observation moves along. The portion of the screen where the agent is "out of reach" by just one action can be too noisy to the deep neural network, and removing the noisy part can help improve generalization, because the game details "far from reach" is unlikely to influence the agent's immediate moves, only the details close enough to the agent matter. Since the agent character in those environments is always centered at the middle of the screen, if the the HUD on the screen margins can be ig-

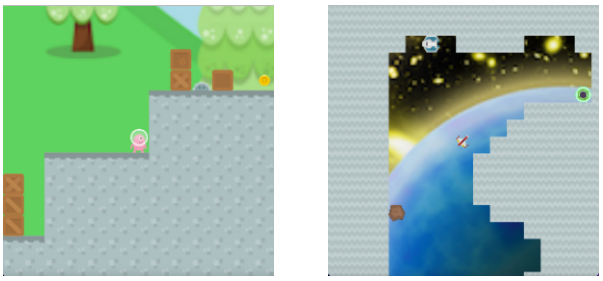


Figure 1: Screenshots of observations of CoinRun (on the left) and CaveFlyer (on the right).

nored, the AI agent can simply "zoom in" the screen to the center, so that the screen will have more details close to the agent character, and that the original outer details will no longer be visible, forcing the agent to only consider close surroundings when deciding which action to take.

Recently, OpenAI developed a platform called ProcGen (Cobbe et al. 2019a) which consists of 16 games that look similar to Atari games as well as Sonic. These games have a common benefit that they run really fast if GUI is disabled, where agents can take thousands of steps per second on a single modern CPU core. In addition, each game is able to generate a unique level given a seed, so that in theory there can be as many unique levels as we want, as long as we have sufficient number of seeds. To put simply, ProcGen offers a great test bed for evaluating deep reinforcement learning generalizations with high speed and unlimited unique but similar environments. Among these 16 games, two of them, CoinRun and CaveFlyer (see Figure 1), have the agent character always positioned at the center of the screen and don't have any HUD on the screen, so that the "zoom in" technique introduced by this project may be applied. In this project, experiments were performed on these two games.

Background

Deep Q-Network

DQN (Mnih et al. 2015) can be viewed as an extension of traditional Q-learning by having a neural network as the Q-function approximator $Q(s, a; \theta)$, where s is a state, a is an action, and Q is parameterized by θ from the neural network. During training, the Q-values predicted by Q itself are used as truth labels, but because each update of θ would certainly change the next truth labels given the same s and a, a target neural network with parameters $\bar{\theta}$, which is initially same as θ but is fixed for a few iterations and gets sync with θ periodically, is used to generate the truth labels instead. In addition, DQN does not directly use the observations just encountered for update, instead it stores each observation (s_t, a_t, r_t, s_{t+1}), where r is the reward, in an experience replay buffer at every exploration step t, and samples a mini-batch from the buffer uniformly for each update. This technique helps remove the correlations among the training data, especially those that are observed within few steps. With these designs, DQN attempts to minimize the following loss L, where γ is the discount factor:

$$L(\theta) = (r_t + \gamma \max_a Q(s_{t+1}, a; \bar{\theta}) - Q(s_t, a_t; \theta))^2 \quad (1)$$

The minimization may be carried out by gradient descent or other optimization algorithms.

Double DQN

DQN has an issue of overestimating some Q-values due to the $\max$ operator in equation (1), which shifts the probability distribution of actions from the optimal policy one. Double DQN (van Hasselt, Guez, and Silver 2015) was proposed to remedy this issue by decoupling action selection and "truth label" selection during training, so that the loss L changes to the following:

$$\begin{aligned} y &= \arg\max_a Q(s_{t+1}, a; \theta) \\ L(\theta) &= (r_t + \gamma Q(s_{t+1}, y; \bar{\theta}) - Q(s_t, a_t; \theta))^2 \end{aligned} \quad (2)$$

The removal of $\max$ operator would most likely reduce the temporal difference error, resulting in a more stable update. In addition, selecting action according to the parameters being trained on would infer the more accurate truth label from the target network, improving the correctness.

Prioritized Experience Replay

During training, DQN samples minibatch of past observations from the experience replay buffer uniformly, regardless of whether they are useful anymore at the time of update. If DQN already predicts good Q-values for the sampled observations, then DQN would improve only by a little, despite at the same time DQN may still predict badly for other observations. It can be a case where those observations having bad predictions are actually more important than those observations having good predictions, if the optimal policy requires. To address this problem, Prioritized Experience Replay (Schaul et al. 2016) was introduced to enable DQN sample "important" observations, whose priority is measured by a quantity p_t proportional to the temporal difference error:

$$p_t \propto |r_t + \gamma \max_a Q(s_{t+1}, a; \bar{\theta}) - Q(s_t, a_t; \theta)|^\alpha \quad (3)$$

where α determines the proportionality, so that it can be linear, quadratic, etc. The temporal difference error need not be exactly the one above, it can be changed to one from Double DQN. With this strategy, new observations ever encountered are most likely having highest priority to be sampled; and when sampled, their priorities would be updated accordingly, so that DQN is able to focus on past observations that have lots of learning potential.

Dueling Network Architecture

Original DQN directly estimates Q-values, ignoring the cases where in some states, no actions are important (ex. the agent just jumped up and is still in the air); but in other states, some actions are vital (ex. the agent must move left or right to dodge bullets). To incorporate such action advantages, Dueling Network Architecture (Wang et al. 2016) was

introduced to alter the portion of any convolutional neural network after the convolutional layers into two streams of hidden fully connected layers with both structures identical to the original hidden fully connected layers, so that one stream has its output layer that estimates state value function V, and the other stream has its output layer that estimates action advantage function A, and finally they are aggregated into a single output layer that estimates the Q-function. Let θ be the parameters in the convolutional layers, α be the parameters in the state value stream, and β be the parameters in the action advantage stream, then the whole network is to compute the following:

$$\begin{aligned} b &= A(s, a; \theta, \beta) - \frac{\sum_{a'} A(s, a'; \theta, \beta)}{n} \\ Q(s, a; \theta, \alpha, \beta) &= V(s; \theta, \alpha) + b \end{aligned} \tag{4}$$

where n is the number of possible actions. Another variant of the equation above replaces the mean calculation by max calculation over all actions, but the author showed that calculating mean is better.

Multi-Step Q-Learning

Traditional Q-learning learns one-step ahead, observing the reward of current step and use the estimated max Q-value of next step to perform parameter update. However, it was shown by (Hessel et al. 2017) that the agent can instead directly move forward n steps according to its current policy, and then use the estimated Q-value at the n-th step to perform update. The loss L on pure DQN would now be:

$$\begin{aligned} R_t^n &= (\sum_{k=0}^{n-1} \gamma^k r_{t+k}) + \gamma^n \max_a Q(s_{t+n}, a; \bar{\theta}) \\ L(\theta) &= (R_t^n - Q(s_{t+n}, a_{t+n}; \theta))^2 \end{aligned} \tag{5}$$

where n is often tuned to be around 3, and it often helps train the agent faster.

Ape-X DQN

Ape-X DQN, or Distributed Prioritized Experience Replay (Horgan et al. 2018), is the algorithm with all combination from the aforementioned components, plus it is designed to highly utilize computing power to asynchronously collect observations and sample minibatches, so that the agent can learn at a dramatically fast rate, thanks to the experience replay buffer being the authority of training data repository throughout all variants of DQN. In aforementioned versions of DQN, the agent itself is just one worker who does all the jobs of exploring the environment, storing the observation to the buffer, and sampling minibatch from the buffer to update parameters. In Ape-X DQN, the agent has multiple workers to split the job: one worker is to only sample minibatch for parameter update with the help from GPU, and all other workers are to only collect observations with the help from multiple CPU cores, while all workers share the same policy model. In fact, each observation collecting worker can even work on multiple environment instances simultaneously, because inferring one action is much slower than stepping in one environment, so environments are "vectorized" in order to utilize CPU SIMD instructions to parallelize computing. In addition to traditional Q-learning or DQN where the agent has just one epsilon greedy policy to balance exploration and exploitation, each observation collecting worker in Ape-X DQN can have different epsilon greedy policies, allowing both exploration and exploitation happen at the same time.

In this project, Ape-X DQN is used as the baseline in the experiment, as this is the only variant of DQN introduced so far that has acceptable performance in the challenging ProcGen games with limited amount of time for training.

Related Work

Regularization

In deep learning era, big models offers surprisingly high accuracy of estimations, but at the same time they are prone to overfit to training data. To overcome the overfitting, L2 regularization (Cortes, Mohri, and Rostamizadeh 2012) was introduced and seen with great success, especially on convolutional neural networks. Later on, a simpler technique called dropout (Srivastava et al. 2014) was proposed, which had become one of the default regularization method by many deep learning practitioners since then. While these methods have different approaches, they ultimately share the idea that the model is made to be effectively lesser than the original size during training by decaying weights or shutting activations, thus the model is easier to estimate on unseen input, at the cost of having lower accuracy on training data.

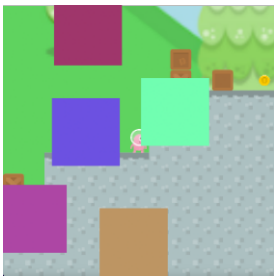
Fortunately, most of the models on deep reinforcement learning agents are similar to those convolutional neural networks used in computer vision, which heavily relies on regularizing models to prevent overfitting, and it was shown by (Farebrother, Machado, and Bowling 2020) that the same regularization methods can be simply applied to improve the agents' ability to generalize in unseen environments.

Regularization methods would have been nice to try in the experiments, but these methods generally require considerable amount of extra time to update the parameters, since they add extra work during the back propagation for each layer of the model. Due to limited amount of training resources available at the time of this project, only generalization methods that are instantly fast are preferred.

Data Augmentation

Regularization techniques often take the approach of effectively altering the model to improve generalization. However, one can take another approach by directly altering the training data. Data augmentation is especially helpful in computer vision, where the same input images may be rotated, and objects to be detected may be just shifted or just zoomed in or out, and it was shown by (Bjerrum 2017) that one can simply duplicate original images and then apply simple transformations such as shifting, rotating, zoomed in/out, flipping, and distorting, so that not only the variety of data is increased, but also the amount of data is multiplied, resulting in improved model performance.

As deep reinforcement learning enjoys a lot from computer vision techniques, data augmentation is of no exception. Since the observations of the environments, like Atari



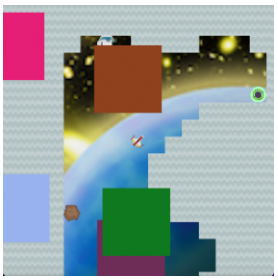


Figure 2: Same observations of CoinRun and CaveFlyer as those on Figure 1 with Cutout applied.

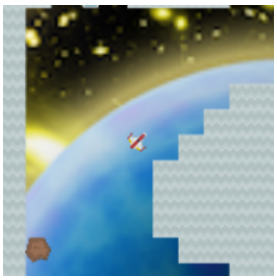


Figure 3: Same observations of CoinRun and CaveFlyer as those on Figure 1 being zoomed in 1.5x, that is, $\beta = 1.5$.

games, are simply frames of images, they can be preprocessed before being fed to the model, if one wants to impose some visual effects on the images. Cutout (DeVries and Taylor 2017) was originally introduced to mimic the dropout regularization on the images themselves by simply masking random square regions through turning off their pixels before sending them to the model. Later on, this was borrowed by (Cobbe et al. 2019b) to run deep reinforcement learning agents on the CoinRun game, showing some success of generalization in unseen game levels being improved.

The idea of reducing the field of view proposed in this project is partially inspired by Cutout. If one can improve generalization by simply removing some features from consideration, then it means those removed features are likely noisy. Since the "zoom in" technique simply removes outer details completely from the screen, one can view it as applying Cutout without randomness just on the margins of the screen. In the experiments, the Cutout technique (Figure 2) is being compared to the "zoom in" technique.

Procedural Level Generation

Traditionally, environments for reinforcement learning were hand-crafted by human, which takes dramatically a lot of time to do so, so that there were few instances of environments for training, and hence the agents were extremely difficult to generalize across even just slightly different instances of the same kind of environment. However, this situation had changed recently.

Procedural Level Generation (Justesen et al. 2018) was proposed to generate environments by simply offering configurations. An environment, such as a level in a video game, consists of many configurable objects, layouts, and even difficulties, and that one combination of the configuration constitutes a unique level. Theoretically speaking, if the number of possible configurations is unlimited, then the number of unique levels is also unlimited. Based on this technique, the authors showed great improvements of generalization by simply training agents on multiple levels long enough.

The ProcGen (Cobbe et al. 2019a) takes Procedural Level Generation to the next level. As mentioned earlier, it consists of 16 Atari like games and each of them is able to produce a unique level by just giving a random seed. With so many seeds available, it can actually generate unlimited amount of levels for each game, which is why this project uses ProcGen as the test bed for generalization experiment.

Field Of View Reduction

A data augmentation technique, field of view reduction is no different than zooming in an image. It is controlled by just a single hyperparameter, let's call it β, that simply tells the scale to zoom in. β may be any real number greater than or equal to 1. Assume an image I which is to be zoomed in has dimensions $(x, y, 3)$, where 3 stands for the RGB channels, then applying field of view reduction to I results in a new image I' that is formed by cropping the middle $(x/\beta, y/\beta, 3)$ portion P of I and then scaling the dimensions of P up to $(x, y, 3)$. The reason of restricting β to be greater or equal to 1 is that if it were less than 1, then $x/\beta > x$ and $y/\beta > y$, making P's dimensions greater than I's and it simply doesn't make sense, as P is supposed to be a crop of I. To illustrate this, consider the environment observations an agent will receive in any ProcGen game, which are of dimensions $(64, 64, 3)$. With $\beta = 2$, the middle $(32, 32, 3)$ portion of the observations will be cropped and then be scaled up to $(64, 64, 3)$. The resulting image will have only the pixels from the middle portion of the original observation, but are enlarged twice. Figure 3 shows a concrete example of applying this technique.

While field of view reduction can help generalization, it can only be applied at particular environments where the agent character is displayed on the center of the screen and no HUD is displayed on the margins. If being applied at any other kinds of environments, the agent character may be off the screen if it is on the margins, which will certainly prevent the agent from seeing himself. In addition, if there is HUD on the margins, the HUD will also be off the screen, preventing the agent from accessing important information that may be necessary to solve the game level.

ProcGen

ProcGen (Cobbe et al. 2019a) is a procedural game level generator that runs on Gym, an environment simulator. It is capable to randomly and uniformly generate game levels across multiple seeds with roughly the same variety distributions on different runs, no matter how big or how small the range of seeds are, to allow generalization evaluation.

In the simulation, when an agent finishes a level, it would automatically enter the next generated level. ProcGen can also be used to generate just one level all the time, allowing level mastery evaluation. Among the 16 supported games, only the following two games have the agent character always centered at the middle of the screen and has no HUD, so the field of view reduction technique may be applied.

CoinRun

As a mini game that is made similar to Sonic, CoinRun has the same objectives that, starting from the far left, the agent must travel to the far right and collect the one and only one coin over there, winning a reward of 10, and the agent must avoid hitting enemies or falling into traps along the travel. In addition, the agent has only 1000 time steps to remain on one level. No matter what action the agent picks, as long as it does not collect the coin at the next step, the agent will always receive a reward of 0.

The seed used to generate levels can dramatically affect the layouts, difficulty, and even all the colors. In general, all CoinRun levels can be categorized into three different difficulties named 1, 2, 3 by (Cobbe et al. 2019a). Difficulty 1 is as easy as toy reinforcement learning environments like cart pole, where the agent only needs to jump one platform to collect the coin, and there is no enemy or trap. Difficulty 2 is moderate, where the agent needs to jump around three platforms containing enemies and/or traps to solve the level. Difficulty 3 is hard, where even human would often feel challenged to play through. In addition to layout difficulties, each level's sky, characters, walls and traps are of different colors level by level, which makes agents who overfit to one level almost impossible to solve other unseen levels.

CaveFlyer

A small game that resembles the Atari game Asteroids, CaveFlyer has the objectives of flying the agent from one end of the level to the other end through a usually narrow passage, while avoiding hitting enemies and traps along the way, and there are rocks that must either be avoided or blasted. In addition, the agent has only 1000 time steps to remain on one level. No matter what action the agent picks except finishing off rocks, as long as it does not get to the other end at the next step, the agent will always receive a reward of 0, unless the agent picks an action that just eliminates a rock at the next step, where the agent will get a reward of 1.

Similar to CoinRun, the seed used to generate levels can dramatically affect the layouts, difficulty, and even all the colors, and each level can be classified as difficulties 1, 2, 3 according to (Cobbe et al. 2019a).

Experiment

Methods

Two kinds of metrics are measured in this experiment. The first metric is sample efficiency, which means how well the agents perform in the same environments used for training. Sample efficiency determines whether agents can learn to adapt environments at all or not, let alone going through unseen but similar environments. This is especially true in settings consist of just one environment, where the agents are expected to master it. The second metric is test efficiency, which directly determines the ability of the agents to generalize in unseen but similar environments after training.

Since this project concerns the ability of a data augmentation technique to improve generalization, we use agents with the same reinforcement learning algorithm and the same neural network architecture but different generalization techniques, and compare their performances during training in the same set of different environments and during testing in the same set of another different environments. Because we also measure sample efficiency, in addition to the generalization evaluation, we also train agents with the same algorithm and architecture and the same set of different generalization techniques in just one environment, see how well they master in it, and do the same for other few environments.

To be specific, we choose the Ape-X DQN algorithm, which has little to no data augmentation and regularization at all, as the baseline. The hyperparameters for updating parameters are similar to what proposed by (Horgan et al. 2018), which consist of discount factor 0.99, learning rate 0.00025, 3-step learning, experience replay buffer size 1000000, prioritized replay alpha 0.6 and beta 0.4 with no annealing, training minibatch size 512, and target network update frequency 50000. In addition, the agent only starts learning after the first 50000 steps to allow sufficient exploration. The hyperparameters for collecting observations are slightly modified to fit with the computing capacity we have, which consist of $n = 8$ observation collecting workers, each with 8 environment instances, and each worker $i \in [0, \ldots, n-1]$ has its own epsilon greedy policy $\epsilon_i = \epsilon^{1+\alpha(i/(n-1))}$ where $\epsilon = 0.4$ and $\alpha = 7$.

For the neural network of the baseline, we choose the same convolutional architecture by (Mnih et al. 2015), which consists of a filter of 32 8×8 kernels with stride 4 as the first layer with a rectifier as activation, a filter of 64 4×4 kernels with stride 2 as the second layer with a rectifier as activation, and a filter of 64 3×3 kernels with stride 1 as the third layer with a rectifier as activation, whose output is flattened. The convolutional architecture is followed by a dueling architecture by (Wang et al. 2016), which consists of two streams of fully connected neural network, where each stream contains only one hidden layer of 512 rectifier units.

And for testing the generalization techniques, we compare the baseline to one configuration of Cutout and four different configurations of field of view reduction. For Cutout, it is configured, as what (Cobbe et al. 2019b) seemed to suggest, to mask 5 square regions with side length equal to 1/4 of the original observation's side length at random locations, where each region has its own random color. For field of view reduction, the factors are $1.25, 1.5, 2, 3$ respectively. We believe that there is a balancing factor can be searched for, as being too low would render the observation roughly same as the original, and being too high would strip too much detail from the original that may be essential for agents to succeed in the environment.

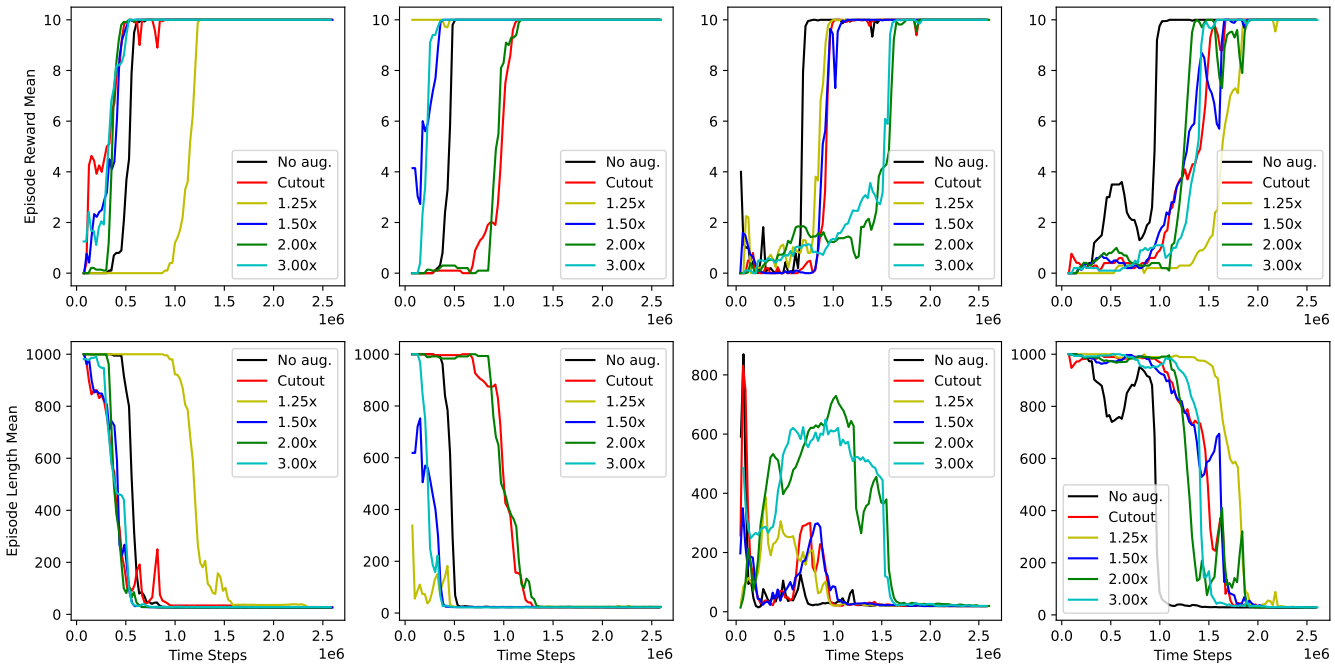


Figure 4: Results of training for 2500000 time steps in a single game level of difficulty 2 with one of the data augmentation techniques. Plots are organized into columns, so that each column corresponds the performance in one game level, where the upper plot shows its episode reward mean and the lower plot shows its episode length mean, aligned by its time steps. From left to right: the first column is the result in CoinRun level 57, the second column is the result in CoinRun level 164, the third column is the result in CaveFlyer level 83, and the fourth column is the result in CaveFlyer level 145.

Finally for the environments to be used for testing, there are different strategies when evaluating sample efficiency, since each level of the same ProcGen game are classified into different difficulties. For difficulty 2, we choose two CoinRun levels 57 and 164 and two CaveFlyer levels 83 and 145, and evaluate the agents in each level for 2500000 time steps. For difficulty 3, we choose two CoinRun levels 107 and 139 and two CaveFlyer levels 1 and 84, and evaluate the agents in each level for 7500000 time steps. However, since ProcGen only has an option to generate levels with a range of consecutive seeds, the generated levels are of mixed difficulties, so there is just one strategy when evaluating generalization. In each of the game CoinRun and CaveFlyer, we train the agents for 10000000 time steps using levels generated by the seeds $[0, 200)$, and we test the agents at the end of every 250000 time step period during training in each of the levels generated by the seeds $[200, 220)$. Although the agents would see the same set of levels, ProcGen generates levels from the seeds randomly, so the agents are likely to encounter each level for a different number of times across different trials of training. Because of this, the generalization evaluation is run 2 times to have an average estimate. ProcGen actually has a setting that configures the mode of the level variety distribution, which is set as "easy" to allow the smoothest distribution across only 200 consecutive seeds.

Mastery In Single Moderate Level

As Figure 4 shows, in the run for single difficulty 2 level, every agent is able to achieve mastery within 2500000 time steps, and they converge at time steps that are within a small range. This result indicates that the field of view reduction not only does not break the agent's ability to learn, but also allows the agent to achieve good performance comparable to the baseline, even though there is no need to generalize.

Across the two games with two levels each, the baseline has a common learning pattern. At the beginning of training, the episode reward mean is almost always 0, but once it bounces up, with the exception of CaveFlyer level 145, it almost immediately goes up to 10, the only possible reward achievable by the game design which indicates a level is solved, and it remains at 10 afterwards. We can combine this with the episode length mean for better analysis. With the exception of CaveFlyer level 83, the episode length mean of the baseline is almost always 1000, the maximum number of time steps an agent can remain in one level, when its episode reward mean is 0; and as the episode reward mean jumps to 10, the episode length mean drops to its lowest value, and it also barely changes anymore while the episode reward mean remains at 10. This indicates that when the baseline is not able to solve the level yet, it is wandering around in the level while trying to find a solution, and fortunately it does not encounter any danger, except in CaveFlyer level 83 the baseline is having bad luck to encounter dangers almost all the time; but once the baseline discovers a path to suc-

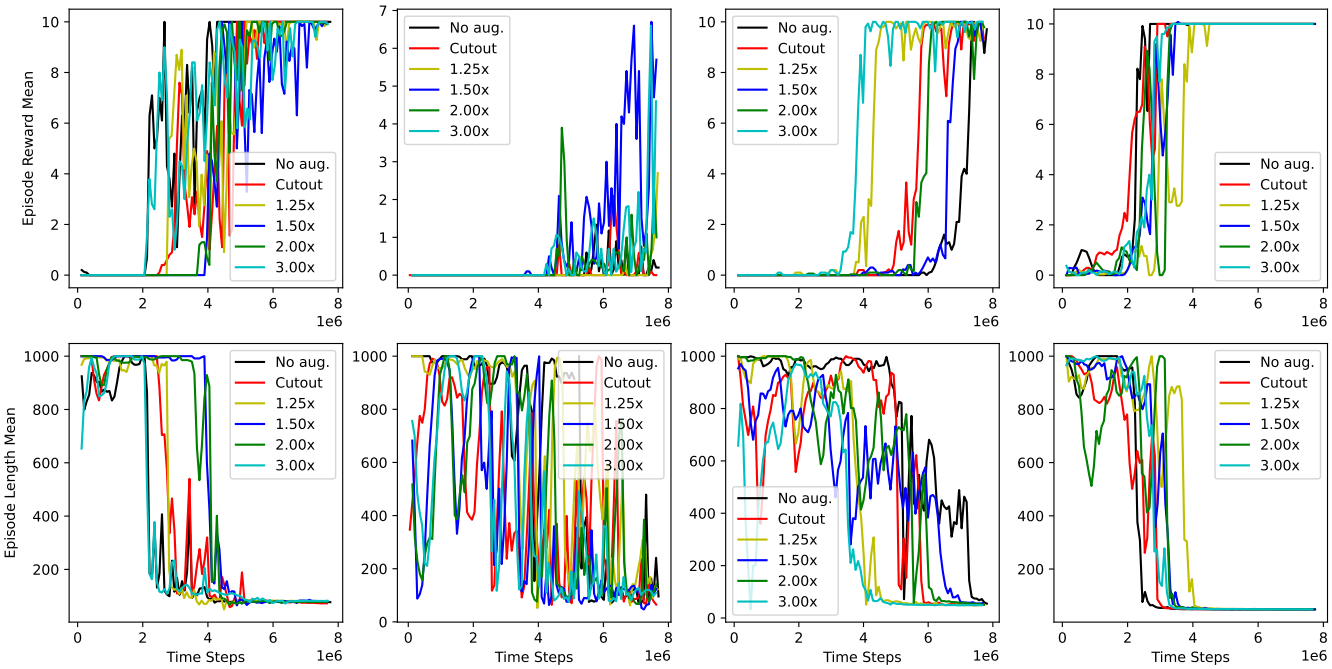


Figure 5: Results of training for 7500000 time steps in a single game level of difficulty 3 with one of the data augmentation techniques. Plots are organized into columns, so that each column corresponds the performance in one game level, where the upper plot shows its episode reward mean and the lower plot shows its episode length mean, aligned by its time steps. From left to right: the first column is the result in CoinRun level 107, the second column is the result in CoinRun level 139, the third column is the result in CaveFlyer level 1, and the fourth column is the result in CaveFlyer level 84.

cess, it would no longer wander around in the level and just follow the successful path in every subsequent run. This fact could imply that the baseline is able to fit to a lot of past experience that helps propogate Q-values that lead to success from the goal back to the starting point almost immediately at most of the time. Looking back to CaveFlyer level 145, we can find that although the episode reward mean does not go straight up to 10 from 0, the episode length mean is changing along with it in a reflective way, so this indicates that when the baseline fails to solve the level, it is wandering around the level; and maybe there are some past experience unrelated to success look too similar those related to success, the baseline are difficult to distinguish them at the moment.

The agents with either Cutout or field of view reduction generally follows the similar learning pattern as the baseline, just with unstable converging times which are generally after the baseline's, and that the learning curves are generally not as steep as the baseline's. Their performances vary a lot across the levels, which make us hard to infer which technique configuration is generally better, but we are certain that they perform worse than the baseline. It is expected that agents with generalization improvement techniques would perform a little bit worse than the same agents without such techniques in the environments for training, because those techniques improve generalization often by either penalizing the model parameters or adding noise to the observations, which make the agents with those techniques less able to consider the features the original observations have, thus they are harder to distinguish observations.

Mastery In Single Hard Level

Similar learning patterns of the agents in single moderate levels also apply to single hard levels, but there are some exceptions, as Figure 5 shows. Because hard levels require much more actions to be carried out from the starting point to reach the goal, in order to master in such levels, the agents are required to fit to much more past experience. Due to this challenge, even the baseline fails to generate a smooth learning curve in all of the hard levels tested, as at times the episode reward mean drops, whereas in the moderate levels this does not happen. And because the levels are much more challenging, even the baseline's converging times become unstable as well, under comparison to the data augmentation techniques. It even fails to converge at CoinRun level 139, receiving almost no reward during the entire training. However, these hard levels are where the data augmentation techniques begin to shine, despite having unstable learning curves that bounce up and down as time steps grow. In CaveFlyer level 1, where the baseline only starts converging at the end of the training trial, all other agents with data augmentation techniques seem to converge way before the baseline's, especially the field of view reduction agents with factors 1.25 and 3. In CoinRun level 139, where the baseline struggle to even receive reward, data augmentation agents start receiving rewards way before the baseline, and field of view reduction agents especially those with factors 1.5 and

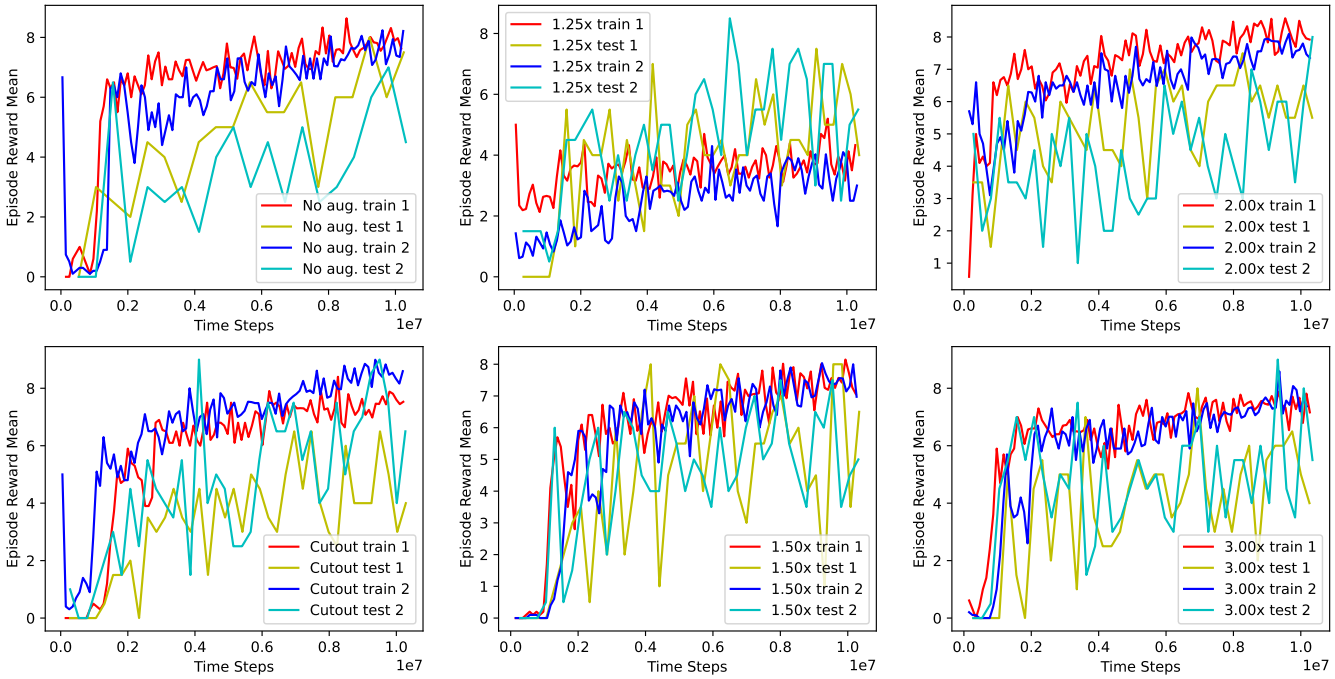


Figure 6: Results of training different agents in CoinRun levels $[0, 200)$ for 10000000 time steps and testing the agents in each of CoinRun levels $[200, 220)$ for evaluating generalization. The variety distribution mode is set as "easy".

3 are performing way better than the others.

The episode length mean could infer what these agents are doing in the hard levels. With no exception, each agent's episode reward mean and episode length mean does not form a reflective relationship, which shows at times when the agent receives almost reward from the episodes, the episodes simply end early because the agent either hits enemies or falls into traps, whereas at the other times the agent just wanders around the level before the 1000 time step limit is up. However, among the episode length mean curves, the data augmentation ones are more likely to be up-and-down before their episode reward mean start going up. This could imply that the behaviors of the data augmentation agents are more "random" comparing to the baseline's, encouraging the agents to explore more than to exploit, which indicates they are likely to have more potential to generalize to unseen observations within the level.

Generalization In Multiple Levels

As Figure 6 shows, all agents except the 1.25x one achieve good sample efficiency in CoinRun levels $[0, 200)$, which contain all 3 difficulties where each difficulty has enough variant of layouts and colors, thanks to the "easy" distribution mode. For each agent, the plot of two runs, which are designed to average the randomly sampled levels by ProcGen, are also agreeing with each other. And as the figure shows, all agents have similar generalization performance in each of CoinRun levels $[200, 220)$, but for some agents especially the field of view reduction ones with factors 1.25 and 1.5, the gaps between training and testing episode reward mean are smaller the other agents, which is again a good indicator that they have the potential to generalize well. For the 1.25x agent, although it has bad sample efficiency, its test efficiency is not only on par with others, but also surpass its own sample efficiency. It seems that field of view reduction with factor 1.5 performs the best, as it has both high sample efficiency and test efficiency comparing to other agents. However the two runs of the test are less likely to agree with each other, because when one run has better result at one time step, the other one often has sub par result, and they alternate the pattern as time steps grow.

Figure 7 shows how well the same agents perform in levels of another game. Perhaps CaveFlyer is harder than CoinRun in general, all agents have low sample efficiency, and their test efficiency is highly variant, having huge drops and ups in consecutive time steps. However, It seems that the field of view reduction with factor 1.5 performs the best at the CaveFlyer levels in general, as it has fewer number of times with its episode reward mean equal to 0, and and the curve grows up to align with the sample efficiency curve, as the time steps grow.

Through the experiments, we find that field of view reduction with factor 1.5 seems to be the best configuration overall, achieving high sample efficiency comparing to all other agents while achieving high and more stable test efficiency. Coincidentally, factor 1.5 is also exactly the one that enables the agent to have acceptable performance during the single CoinRun level 139, while all others are still struggling.

If we combine the experiments results of single level mastery and multiple levels generalization, we can see that in

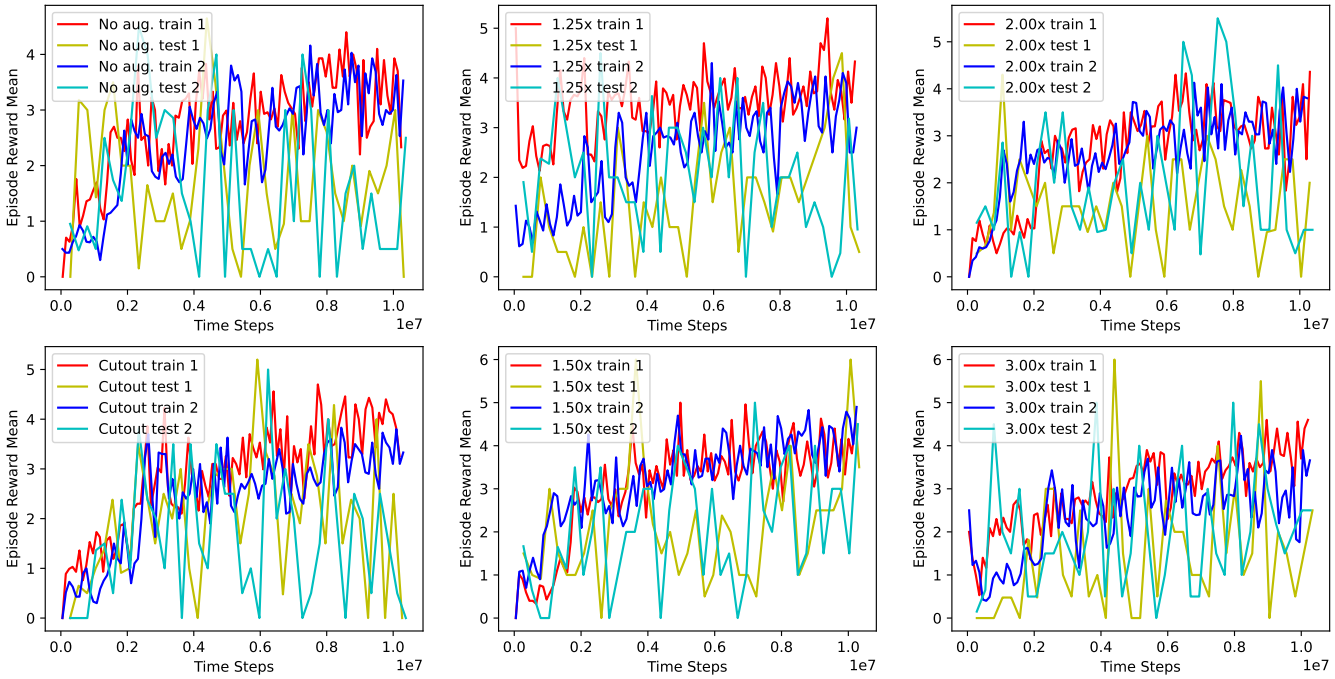


Figure 7: Results of training different agents in CaveFlyer levels $[0, 200)$ for 10000000 time steps and testing the agents in each of CoinRun levels $[200, 220)$ for evaluating generalization. The variety distribution mode is set as "easy".

the multiple levels setting, the agents are probably only performing well in levels of difficulty 1 or 2, because after similar training time steps in one single hard level, the agents are still difficult to perform well in that level. As (Cobbe et al. 2019a) suggested, the agents are simply needed to be trained for way more time steps, like 10 times more than the current, to achieve good performance in terms of both sample efficiency and test efficiency in any level. This strategy works only if computing power is not to be concerned.

Future Work

Due to time limits for this project, comparisons between only different data augmentation techniques and configurations are made, leaving variants of reinforcement learning algorithms as well as variants of neural network architectures untouched. Since this project relates to computer vision the most, it would be likely to see fruitful results if the field of view reduction technique is applied to a different convolutional neural network. However, as the technique has a huge limitation which restricts the applicability of environment observations, we would be more likely to propose different techniques that improve sample efficiency as well as test efficiency in the future.

Conclusion

Like any regularization or other data augmentation technique, field of view reduction is able to improve generalization by reducing the size of input features to be considered, so that possible noise features are removed. However, it may not be applied too aggressively, because that would strip too much important features, making the model unable to fit to any observation. The results of the experiment confirm that.

As the experiment suggests, field of view reduction with factor 1.5 is particularly useful for challenging environments, which allows an agent to achieve good performance within a short amount of training time, while others are still struggling. However, if the environment is moderate enough, then it is better to just use the original algorithm and model without any regularization or data augmentation, especially when generalization is not required.

The field of reduction technique, on the other hand, requires that the environment observations have the agent character centered at the middle of the screen and there is no HUD on the screen, so it is unfortunate that most of the environments are not eligible. Nevertheless, this project could form a foundation for another generalization improvement project, just like how Cutout inspired this project, and how Dropout inspired Cutout.

References

Bjerrum, E. J. 2017. SMILES Enumeration as Data Augmentation for Neural Network Modeling of Molecules.

Cobbe, K.; Hesse, C.; Hilton, J.; and Schulman, J. 2019a. Leveraging Procedural Generation to Benchmark Reinforcement Learning. *arXiv preprint arXiv:1912.01588* .

Cobbe, K.; Klimov, O.; Hesse, C.; Kim, T.; and Schulman, J. 2019b. Quantifying Generalization in Reinforcement Learning.

Cortes, C.; Mohri, M.; and Rostamizadeh, A. 2012. L2 Regularization for Learning Kernels.

DeVries, T.; and Taylor, G. W. 2017. Improved Regularization of Convolutional Neural Networks with Cutout.

Farebrother, J.; Machado, M. C.; and Bowling, M. 2020. Generalization and Regularization in DQN.

Hessel, M.; Modayil, J.; van Hasselt, H.; Schaul, T.; Ostrovski, G.; Dabney, W.; Horgan, D.; Piot, B.; Azar, M.; and Silver, D. 2017. Rainbow: Combining Improvements in Deep Reinforcement Learning.

Horgan, D.; Quan, J.; Budden, D.; Barth-Maron, G.; Hessel, M.; van Hasselt, H.; and Silver, D. 2018. Distributed Prioritized Experience Replay.

Justesen, N.; Torrado, R. R.; Bontrager, P.; Khalifa, A.; Togelius, J.; and Risi, S. 2018. Illuminating Generalization in Deep Reinforcement Learning through Procedural Level Generation.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540): 529–533. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Nichol, A.; Pfau, V.; Hesse, C.; Klimov, O.; and Schulman, J. 2018. Gotta Learn Fast: A New Benchmark for Generalization in RL.

Schaul, T.; Quan, J.; Antonoglou, I.; and Silver, D. 2016. Prioritized Experience Replay.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15(56): 1929–1958. URL http://jmlr.org/papers/v15/srivastava14a.html.

van Hasselt, H.; Guez, A.; and Silver, D. 2015. Deep Reinforcement Learning with Double Q-learning.

Wang, Z.; Schaul, T.; Hessel, M.; van Hasselt, H.; Lanctot, M.; and de Freitas, N. 2016. Dueling Network Architectures for Deep Reinforcement Learning.