Contrastive Pre-Training for Reinforcement Learning

David Zhang
davidwzhang@berkeley.edu

Rohith Sajith
rohithsajith@berkeley.edu

1 Introduction

Despite advances in deep reinforcement learning, creating sample-efficient and generalizable models is still difficult. Inspired by recent advances in self-supervised learning, we propose using contrastive learning to pretrain a visual model of the game state, and then train an RL algorithm on top of it. We compare our model to a baseline PPO algorithm, measuring changes in sample-efficiency and generalization. To evaluate our results, we use the FruitBot game from OpenAI's ProcGen environments, which are explicitly designed to evaluate these metrics (Cobbe et al., 2019).

2 Related Work

With the advent of DQNs, deep RL algorithms have proven to be effective at learning policies for game-like environments (Mnih et al., 2015). More recently, Proximal Policy Optimization (PPO) has empirically outperformed vanilla policy gradient and DQN methods (Schulman et al., 2017).

Furthermore, self-supervised learning has made great strides. For example, RotNet suggests that predicting image rotations can produce useful semantic features in early convolutional layers (Gidaris et al., 2018). Such self-supervised methods have also started to become popular for RL. Aytar et al. propose that temporal distance classification between images can enhance a model's knowledge of the game state (Aytar et al., 2018). More recently, CURL demonstrates how adding a contrastive loss to distinguish between augmented images can improve sample efficiency and performance (Srinivas et al., 2020). Naturally, certain augmentations fare better than others; SimCLR demonstrates that contrastive learning using random crops and color distortion produces higher sample efficiency and accuracy on ImageNet than other augmentations (Chen et al., 2020). Our work attempts to synthesize contrastive learning and PPO to produce concrete benchmarks on how various data augmentations impact sample efficiency and generalization to new environments.

3 Background

3.1 Proximal Policy Optimization

PPO attempts to optimize the following objective:

$$L_{PPO} = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)],$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the ratio of current policy to old policy (Schulman et al., 2017).

Intuitively, the new policy θ should not differ too much from the old policy θ_{old} (hence the clip term) but we should obviously try to increase the probability of actions that yield higher advantages. Apart from that, PPO is similar to actor-critic; it uses an actor network to model an agent's policy and critic network to model the values of states. There is also an additional entropy bonus to encourage exploration.

Preprint. Under review.

3.2 Contrastive Learning

As described in CURL, contrastive learning involves a query encoder f_q, key encoder f_k, and a similarity measure W. From two augmented images x_q and x_k (can come from the same base image, or two different images), the encoders produce outputs $z_q = f_q(x_q)$ and $z_k = f_k(x_k)$, which are meant to be meaningful representations. Naturally, if z_q and z_k come from the same image, we would like to maximize the similarity metric $z_q^\top W z_k$ and minimize it otherwise (in the former case, z_k is referred to as a positive and the latter as a negative). This intuition is summed up nicely in the maximization of the InfoNCE loss (van den Oord et al., 2018):

$$L_q = \frac{\exp(z_q^\top W k_+)}{\exp(z_q^\top W k_+) + \sum_{i=0}^{K} \exp(z_q^\top W k_i)},$$

where k_+ denotes a positive and all of the k_i are negatives.

4 Methods and Approach

Initially, we tried implementing a DQN for our policy. However, debugging struggles and low memory for the replay buffer inspired us to look elsewhere. Ultimately, we decided to build a PPO implementation based on an existing open source repository (Lee, 2020). Our PPO model is split into two main parts. First, there is an encoder that produces a feature vector from the game state. Then, there is a single linear layer that maps those features into action probabilities, and a separate linear layer that maps to state values. This is analogous to a shared network design for an actor critic policy.

We started out with a simple VGG-like model for the encoder, which is shown in Figure 1 (Simonyan and Zisserman, 2014). However, this wasn't powerful enough for FruitBot. From Lee's open-source implementation, we modified an IMPALA model that uses multiple residual blocks instead (Lee, 2020; Espeholt et al., 2018). A more detailed diagram for the IMPALA model is shown in Figure 1.

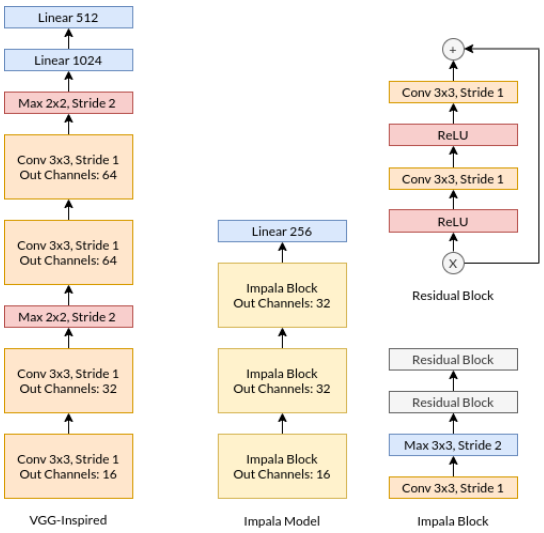

Figure 1: Diagram of both the VGG inspired and IMPALA model.

To train our initial visual representation, we first have the agent take n_{steps} using the untrained policy, storing the images in a buffer. At each training step for contrastive learning, we sample n_{batch} images from the buffer, and copy them to a set of keys and queries. To make the lookup task non-trivial, we pass both the keys and queries through a set of different data augmentations. As proposed by Chen et al., we try rotations, crops, and crops + color distortion (Chen et al., 2020). These augmented keys and queries are then encoded with the same encoder as the RL model, producing feature vectors that can be optimized using the InfoNCE loss (van den Oord et al., 2018).

A few other components were critical to training. In order to minimize human train time, we parallelize rollouts by running multiple environments simultaneously (Cobbe et al., 2019). From Lee's open-source implementation, we also use generalized advantage estimation for more stable training (Lee, 2020; Schulman et al., 2015). While we experimented with linear scheduling for learning rates, these didn't have a significant impact on our results.

5 Results

All our runs are done on the FruitBot environment in ProcGen, using the easy distribution mode (Cobbe et al., 2019). We train on levels 500+, and leave levels 0 to 500 for test purposes.

All our initial visual models use less than 200000 training images, and by extension, game steps. We find that the type of data augmentation heavily affects the amount of training iterations needed for contrastive learning. For example, while rotations converge in under 1500 iterations, crops + color distortion take over 50000. Contrastive training curves for each data augmentation are shown in Figure 2 below. [figure for contrastive loss over training iterations]

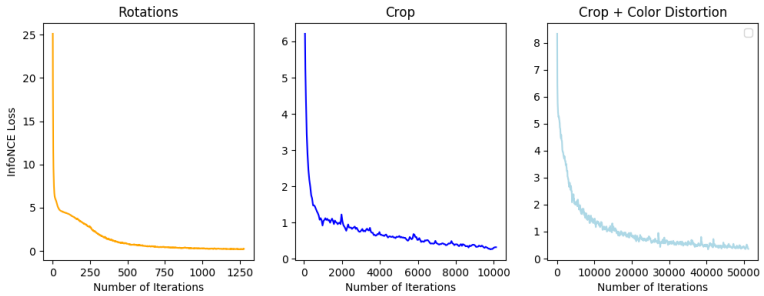

Figure 2: Training curves for contrastive pretraining.

To measure sample efficiency, we consider the number of training steps needed before the average reward per episode converges. While we can consider this with respect to a hard reward threshold, an empirical glance at the reward training graph is more telling. To measure generalization, we measure the average episode reward on test levels not used in training.

We first test the effect of the number of training levels used on generalization, trying 50, 100, 250, and 500 training levels. For simplicity, we only test the baseline PPO model without contrastive pretraining. As expected, we find that using more levels improves the performance on test levels. These results are shown in Table 1.

Table 1: Reward on Test Levels, Baseline PPO

Number of Train Levels	Mean Episode Reward
50	14.33
100	18.40
250	22.60
500	**24.08**

We then test the effect of contrastive pretraining on sample-efficiency and generalization when using 50 training levels. Note that we account for the number of steps used to generate the dataset for our initial visual model, so our RL policy starts at n_{images} instead of 0. We find that the type of data augmentation used also affects sample efficiency. Although rotations have no effect, both crops and crops + color distortion speed up training considerably. To reach 10 training reward, the number of steps required decreases from 1.8M with the baseline PPO to 1.2M with contrastive pretraining. The detailed training curves are included in Figure 3.

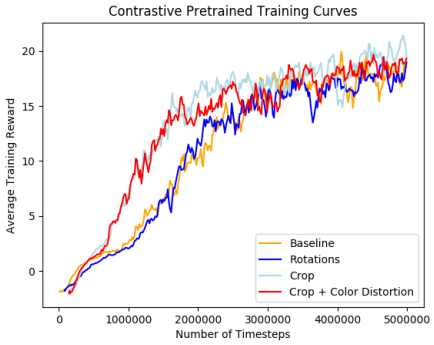

Figure 3: RL training curves after contrastive pretraining.

Note how the baseline PPO and contrastive pretrained model converge to the same average training reward. This makes sense, as while pretraining helps the model form better initial representations, it doesn't provide any extra information beyond what already exists in the environment.

We find that contrastive pretraining also helps generalization. While both the baseline and pretrained model converge to the same training reward, models with contrastive pretraining perform better on the evaluation set. Similar to before, crops and crops + color distortion have the highest performance. The results are shown in Table 2.

Table 2: Reward on Test Levels, 50 Training Levels

Model Type	Mean Episode Reward
PPO	14.33
PPO + Rotations	16.05
PPO + Crop	**19.22**
PPO + Crop/Color Distortion	**19.56**

Specifically, note how the rotations augmentation performs worse than crops and crops + color distortion. We hypothesize that because the rotations task is much easier to learn, the representations are less effective for an RL policy. This is corroborated by results from Chen et al., who find that only using rotations is the worst data augmentation for contrastive learning, while crops + color distortion perform best (Chen et al., 2020).

6 Conclusion

To summarize, we proposed that contrastive pretraining on images learns a meaningful visual representation of game state. By transferring this representation to PPO, we hypothesized the features learned would improve both sample efficiency and generalization to unseen FruitBot environments. After running experiments, we found that images augmented with random crops and color distortion indeed expedite the training timesteps needed, while improving generalization to unseen test levels.

7 Team Contribution

We'd like to emphasize both worked extensively real-time in Zoom to problem solve, debug, and research new ideas. The first number is David's contribution, second is Rohith.

Research: 50/50, RotNet: 50/50, DQN Implementation: 40/60, PPO Implementation: 80/20, Contrastive Learning: 50/50, General Infrastructure: 80/20, Final Report: 50/50, Overall: 60/40

References

Karl Cobbe, Christopher Hesse, Jacob Hilton and John Schulman. Leveraging Procedural Generation to Benchmark Reinforcement Learning, 2019; arXiv:1912.01588.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017; arXiv:1707.06347.

Spyros Gidaris, Praveer Singh and Nikos Komodakis. Unsupervised Representation Learning by Predicting Image Rotations, 2018; arXiv:1803.07728.

Yusuf Aytar, Tobias Pfaff, David Budden, Tom Le Paine, Ziyu Wang and Nando de Freitas. Playing hard exploration games by watching YouTube, 2018; arXiv:1805.11592.

Aravind Srinivas, Michael Laskin and Pieter Abbeel. CURL: Contrastive Unsupervised Representations for Reinforcement Learning, 2020; arXiv:2004.04136.

Ting Chen, Simon Kornblith, Mohammad Norouzi and Geoffrey Hinton. A Simple Framework for Contrastive Learning of Visual Representations, 2020; arXiv:2002.05709.

Aaron van den Oord, Yazhe Li and Oriol Vinyals. Representation Learning with Contrastive Predictive Coding, 2018; arXiv:1807.03748.

Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition, 2014; arXiv:1409.1556.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg and Koray Kavukcuoglu. IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures, 2018; arXiv:1802.01561.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. Playing Atari with Deep Reinforcement Learning, 2013; arXiv:1312.5602.

Hojoon Lee. train-procgen-pytorch. GitHub Repository. github.com/joonleesky/train-procgen-pytorch