Online Reinforcement Learning with Augmented Data on the Procgen Benchmark

Jacob Bahnsen Schnmidt (s184346)
Kasper Schou Telkamp (s170397)

January 4rd 2022

Abstract

This project explores the world of generalization in Reinforcement Learning using different model architectures and data augmentations. The experiments are performed in the Procgen environment and the policies are trained with the state-of-the-art proximal policy optimization method. This project demonstrates that model architecture is of great importance in Reinforcement Learning, finding that larger models is more sample efficient and significantly improve generalization. Moreover, this project finds that data augmentations can be a valuable tool to further enhance model generalization. Our training code is available on github.com/jaco9012/Deep-Learning-Project.

1 Introduction

Reinforcement Learning (RL) is a field within deep learning that focuses on the interaction between an agent and the environment in terms of states (s), actions (a) and rewards. By interacting with the environment, the agent learns a policy (π_θ) that defines the agent's way of behaving at a given time. The focus of this project is on generalization in RL and for this task the coinrun environment from the Procgen Benchmark [Cobbe et al., 2020] is considered. In coinrun the agent spawns in the far left of the map and is tasked with collecting a coin in the far right. The agent must overcome obstacles, enemies and chasms that lead to death on its mission to collect the coin. We use this benchmark to investigate the agents ability to generalize to unseen levels and backgrounds using different model architectures and data augmentations.

2 Proximal Policy Optimization (PPO)

Supervised learning makes great use of gradient descent. This is not possible in RL as this field uses a range of different reward systems instead of having a cost function. For this reason the field of RL uses PPO, which is a concept used in RL to optimize a policy. PPO uses a clipped likelihood function (Eq. 1) to describe the optimal policy [Schulman et al., 2017].

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

One of the most essential parts of this equation is the *min*-operator. The *min*-operator takes the minimum of $r_t(\theta)\hat{A}_t$, which describes the ratio between the log probabilities of the new policy vs. the old policy multiplied by an estimated advantage function, $\hat{A}$, and a clipped version, $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$. The clipped probability ratio is calculated using a hyperparameter ϵ, which acts as a constraint and ensures that the policy update is not overdone. In Fig. 1 two different scenarios are represented: If the advantage function is greater than zero (Fig. 1a) and $r_t(\theta) \in [0; 1+\epsilon]$ the policy will then be updated according to the probability ratio. If the probability ratio exceeds $1+\epsilon$ then the clipped ratio will be smaller and as visualized in Fig. 1a the function reaches a maximum at $r_t(\theta) = 1+\epsilon$, meaning that the policy will be updated according to the clipped probability ratio. The concept is the same if the advantage function is negative, as seen in Fig. 1b. The reason as to why the clipped probability ratio sometimes is used over the actual probability ratio is to ensure slow updates of the policy. If the policy is updated in large steps we might end up with a policy that contains one single action, that is extremely likely compared to the other actions, this will in turn result in a performance collapse and as the next batch of training data will be generated using a poor policy, it will be very unlikely that the agent can recover.

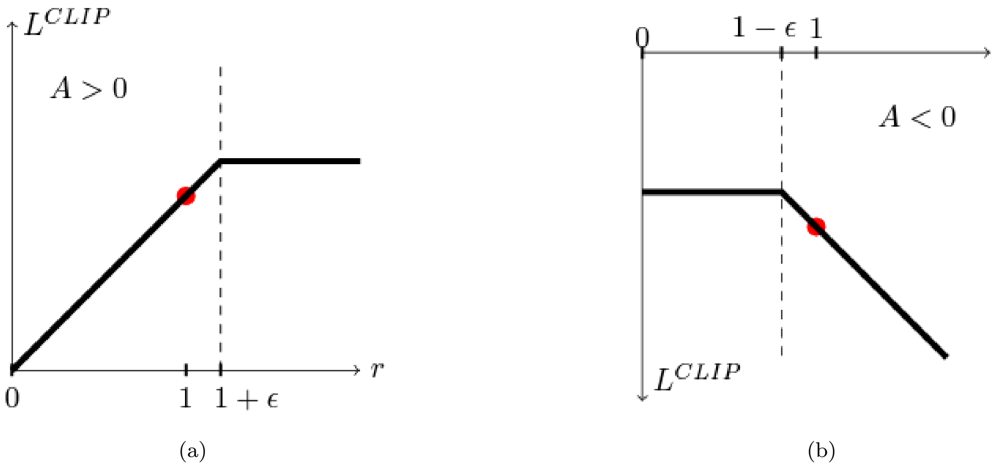


Figure 1: L^{CLIP} is visualized as a function of the probability ratio, for positive advantages (1a) and negative advantages (1b).

The clipped likelihood function (Eq. 1) is combined with the mean square error of the value function (Eq. 3) and an entropy term, $S[\pi_\theta](S_t)$, to yield the final objective function (Eq. 2), which is maximized each iteration.

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{2}$$

Where c_1 and c_2 are hyperparameters used to scale the value function, V^{π_θ}, and the entropy term, $S[\pi_\theta](s_t)$, respectively. The value function is given in Eq. 3:

$$L^{VF}(\theta) = \frac{1}{2} \mathbb{E} \left[max \left((V^{\pi_\theta}(s_t) - R_t)^2, (V^{\pi_\theta}_{CLIP}(s_t) - R_t)^2 \right) \right] \tag{3}$$

Here $V^{\pi_\theta}(s_t)$ describes the value function given the current state and policy, while R_t describes the return from the current policy. A clipped version of the value function is again used to ensure that the value function is updated in small steps.

Lastly the entropy term $S[\pi_\theta](S_t)$ is introduced. This term is a measure of uncertainty, meaning that in early learning this term will be large and introduce more exploration, but as the model learns more about the environment and what actions are good or bad this uncertainty will decrease and the model will start exploiting more.

3 Model

In this project three types of models are considered: a baseline model using the architecture found in [Mnih et al., 2015], which we refer to as the Nature CNN, and two models using the larger architecture found in IMPALA [Espeholt et al., 2018]. A sketch of the model architectures are found in Fig. 2.

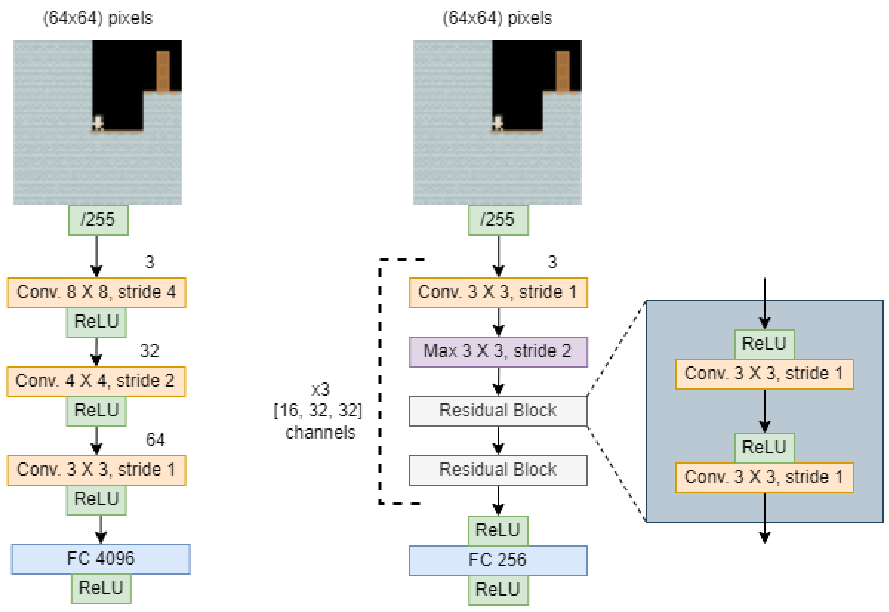
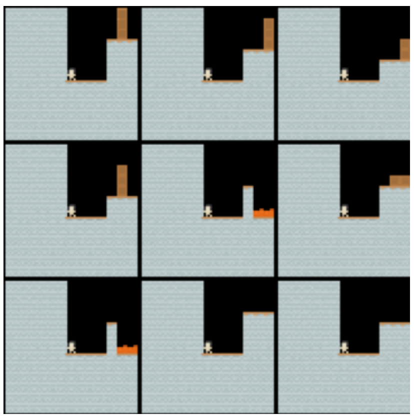


Figure 2: Model architectures. Left: Small architecture inspired by Nature CNN. Right: Large architecture inspired by IMPALA CNN.

Inspired by [Laskin et al., 2020] the two IMPALA CNN models receives different inputs. One of the models receives a normal pixel-based input, whereas the other receives a pixel-based input that have been altered by a random convolution to add noise to the input. This is done to study the effect on generalization when using data augmentations. An image of the pixel-based input and the input altered by a random convolution is visualized in Fig. 3b.


(a)

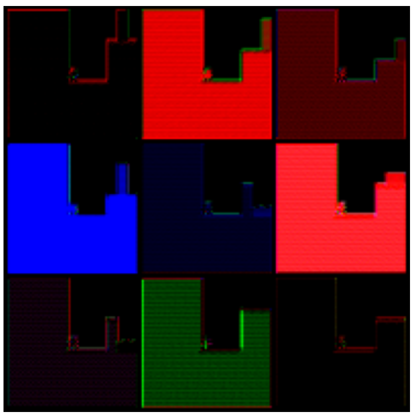

(b)

Figure 3: Pixel based input (3a) and the input altered by a random convolution (3b) for 9 different observation states.

The introduction of a random convolution have also lead to a slight change in hyperparameters. The random convolution inject more noise into the system, which ends up crashing the policy through bad actions. For this reason the PPO clip range (ϵ) have been adjusted from 0.2 to 0.1 as seen in Tbl. 1.

Table 1: PPO hyperparameters used for the Nature CNN and IMPALA CNN architectures.

Hyperparameter	**Nature CNN**	**IMPALA CNN**	**IMPALA CNN + rand. conv.**
γ	0.99	0.99	0.99
λ	0.95	0.95	0.95
Observation rendering	(64,64)	(64,64)	(64,64)
Total timesteps	20M	20M	20M
num levels	200	200	200
# time steps per rollout	256	256	256
Epochs per rollout	3	3	3
batch size	512	512	512
Entropy bonus	0.01	0.01	0.01
PPO clip range	**0.2**	**0.2**	**0.1**
# workers	1	1	1
# environments per worker	64	64	64
Reward normalization?	Yes	Yes	Yes

4 Results

In Fig. 4 the training and validation reward for the models is visualized. All the models are trained using an environment with 200 levels and a black background and evaluated in an environment with unlimited levels and a random background.

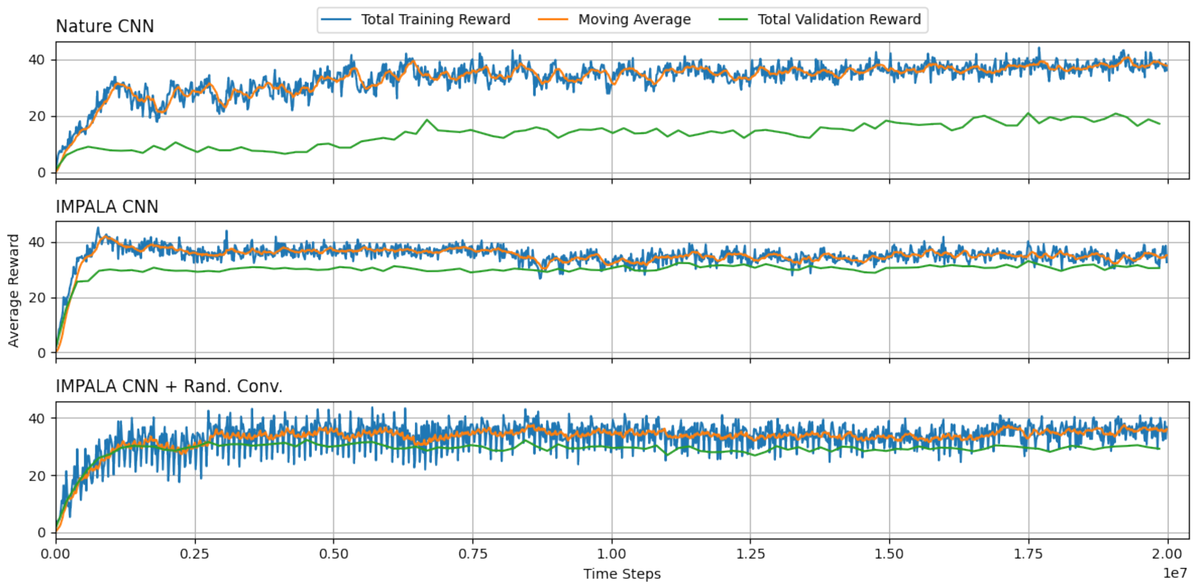


Figure 4: Total training and validation rewards for the Nature CNN, IMPALA CNN and IMPALA CNN with a random convolution.

Fig. 4 shows that the Nature CNN model is able to achieve an optimal performance (reward around 40) on the training data after 20 million time steps, but is unable to generalize to the validation set. Furthermore, it is seen that the IMPALA CNN models are more sample efficient and converges in approximately 1/10 of the time steps required for the Nature CNN model to converge, while still being able to generalize well to the validation set. Altering the pixel based input in the IMPALA CNN model with a random convolution increases the variance in the training rewards, but the moving average reward and the performance on the validation set is very similar.

5 Discussion

Training agents to function in varied environments is to this day a challenging problem in RL and obtaining resilient results is not a guarantee. To address this problem recent research have been focused on improving generalization by applying different model architectures, policy gradient methods and data augmentations.

5.1 Model Architecture

This project demonstrates that model architecture is of great importance, finding that the deep IMPALA CNN architechture is more sample efficient and generalize better to unseen levels and background than the smaller Nature CNN architecture. This comparison might not be completely fair, as the Nature CNN first and foremost could use some tuning of the hyperparameters. It is also seen from Fig. 2 that the Nature CNN model outputs 4096 components from a fully connected layer. These components are then compressed directly into the action space of 15 different actions. The IMPALA structure follows another pattern, in which compression happens at a lower rate, meaning that less information will be lost. Therefore, a more fair comparison of the two model architectures would have seen additional fully connected layers in the baseline model. Given that these compression problems would be solved, it would still be expected that the IMPALA CNN model would be better at generalizing than the Nature CNN model, as it seems unlikely that the compression causes a 50% decrease in validation reward.

5.2 Random convolution

Augmenting the pixel input with a random convolution showed to significantly increase the variance of the training reward, while still obtaining the same high average reward. This is a promising trait for the random convolution, as it means that the model is less likely to converge towards a sub-optimal solution. Furthermore, the models were trained on a black background, meaning that the effect of the random convolution is expected to be smaller, than if the models would have been trained with a random background. The reasoning for this is that the random convolution is supposed to function as a feature extraction, that informs the model with focus points.

5.3 Hyperparameters

During this project we have tuned different hyperparameters. Initially we started with $\gamma = 0.999$ as this type of game only considers hard rewards, in other words there is no time based punishment in the environment. So in order to make the agent play faster we tuned the hyperparameter to $\gamma = 0.99$. This way rewards in the far future is valued less than rewards in the near future, meaning that the agent will try to complete levels faster.

When inspecting Fig. 4 it is seen that the average total reward reaches a maximum around 40. The maximum reward is limited by the number of time steps per rollout and as the agent is playing for 256 time steps, it is equivalent to completing four levels. If the number of time steps per rollout were increased it would likely result in greater rewards, but would also not scale linearly, as the levels generated gets increasingly difficult, as the agent completes them. Additionally, the agent interacts with 64 different environments. This means that when the average reward in Fig. 4 is at 40, the agent is completing 4 levels in each environment. If the number of environments is scaled up it will likely decrease the deviation, as the agent will play more levels and with that also encounter a higher variety of level difficulties. If the number of environments is scaled down it would reduce training time, but as earlier stated this would come at an expense, as the agent would apply the current policy to less levels and therefore have less images to train on.

Lastly we have also experimented with the ϵ-hyperparameter. This hyperparameter describes the clip range and is very important as it works as a constraint as to how much the policy can be updated each time. Updating the policy too much can result in a performance collapse, and eventually mean that the agent never recovers. Ideally the parameter should be as large as possible, as this allows for larger updates and faster convergence, but as seen from Tbl. 1 the PPO clip range (ϵ) is adjusted to be smaller when introducing a random convolution. The reasoning for this is that a noisier input will cause more uncertainty, which in turn means that bad policy updates are more likely to happen. Luckily this trend can be stabilised by adjusting the ϵ-hyperparameter from 0.2 to 0.1, so the policy updates gets smaller.

6 Conclusion

While the Nature CNN model is able to fit reasonably well to the 200 levels it is trained on, the IMPALA CNN models converges in fewer time steps. Additionally, the IMPALA CNN models are also able to generalize to unseen levels and backgrounds, which is not the case for the Nature CNN model. This suggest that the model architecture is very important in generalization. Moreover, a random convolution of the input was tested and while it adds some noise to the input, an increase in the performance of the model was not observed in this project. An explanation for this is, that we did not include background while training our agents.

7 References

[Cobbe et al., 2020] Cobbe, K., Hesse, C., Hilton, J., and Schulman, J. (2020). Leveraging procedural generation to benchmark reinforcement learning. https://arxiv.org/pdf/1912.01588.pdf.

[Espeholt et al., 2018] Espeholt, L., Soyer, H., Munos, R., Simonyan, K., Mnih, V., Ward, T., Doron, Y., Firoiu, V., Harley, T., Dunning, I., Legg, S., and Kavukcuoglu, K. (2018). Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. https://arxiv.org/pdf/1802.01561.pdf.

[Laskin et al., 2020] Laskin, M., Lee, K., Stooke, A., Pinto, L., Abbeel, P., and Srinivas, A. (2020). Reinforcement learning with augmented data. https://arxiv.org/pdf/2004.14990.pdf.

[Mnih et al., 2015] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., 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. https://doi.org/10.1038/nature14236.

[Schulman et al., 2017] Schulman, J., Klimov, O., Wolski, F., Dhariwal, P., and Radford, A. (2017). Proximal policy optimization algorithms. https://arxiv.org/pdf/1707.06347.pdf.