

https://dspace.org/

Analyzing the performance and generalisability of incorporating SimCLR into Proximal Policy Optimization in procedurally generated environments

Gilbert, Nikhil

2024
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/41832

Gilbert, Nikhil. (2024). Analyzing the performance and generalisability of incorporating SimCLR into Proximal Policy Optimization in procedurally generated environments [PhD thesis, University of the Witwatersrand, Johannesburg]. WireDSpace.https://hdl.handle.net/10539/41832

Downloaded from a DSpace repository. https://dspace.org/

An assessment of the performance of contrastive learning methods incorporated into policy gradient algorithms

Analyzing the performance and generalisability of incorporating SimCLR into Proximal Policy Optimization in procedurally generated environments

School of Computer Science & Applied Mathematics
University of the Witwatersrand

Nikhil Gilbert
909092

Supervised by Prof. Benjamin Rosman and Dr Steven James

June 8, 2024



Ethics Clearance Number: Not Applicable

A dissertation submitted to the Faculty of Science, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science

Abstract

Multiple approaches to state representation learning have been shown to improve the performance of reinforcement learning agents substantially. When used in reinforcement learning, a known challenge in state representation learning is enabling an agent to represent environment states with similar characteristics in a manner that would allow said agent to comprehend it as such. We propose a novel algorithm that combines contrastive learning with reinforcement learning so that agents learn to group states by common physical characteristics and action preferences during training. We subsequently generalise these learnings to previously encountered environment obstacles. To enable a reinforcement learning agent to use contrastive learning within its environment interaction loop, we propose a state representation learning model that employs contrastive learning to group states using observations coupled with the action the agent chose within its current state. Our approach uses a combination of two algorithms that we augment to demonstrate the effectiveness of combining contrastive learning with reinforcement learning. The state representation model for contrastive learning is a Simple Framework for Contrastive Learning of Visual Representations (SimCLR) by Chen *et al.* [2020], which we amend to include action values from the chosen reinforcement learning environment. The policy gradient algorithm (PPO) is our chosen reinforcement learning approach for policy learning, which we combine with SimCLR to form our novel algorithm, Action Contrastive Policy Optimization (ACPO). When combining these augmented algorithms for contrastive reinforcement learning, our results show significant improvement in training performance and generalisation to unseen environment obstacles of similar structure (physical layout of interactive objects) and mechanics (the rules of physics and transition probabilities).

Declaration

I, Nikhil Gilbert, hereby declare the contents of this dissertation to be my own work. This proposal is submitted for the degree of Master of Science in Computer Science at the University of the Witwatersrand. This work has not been submitted to any other university, or for any other degree.



Acknowledgements

I want to thank both of my supervisors, Prof. Benjamin Rosman and Dr Steven James. With their guidance and critique, I could take my ideas and thoughts and turn them into a compelling scientific contribution. I want to thank my parents Desi and Sunita Gilbert for all of the financial and infrastructural support they have given me throughout my academic life, without this I would not have been able to make it through the hardest times. I finally give thanks to my brother Rahul Gilbert for the emotional support and advice he has given me through my MSc and COVID-19. It was through his patience and nurturance that I was able to keep forward momentum through this challenging endeavour.

Contents

Preface
Abstract i
Declaration ii
Acknowledgements iii
Table of Contents iv
List of Figures vi
List of Tables viii

1 Introduction **1**

2 Background **5**
2.1 Reinforcement learning and Markov decision processes 5
2.1.1 Function approximation and Deep reinforcement learning 10
2.2 Policy gradient optimization 11
2.2.1 Proximal policy optimization 12
2.3 Convolutional neural networks 14
2.3.1 Residual networks 15
2.4 Contrastive learning 16
2.4.1 SimCLR 18
2.5 Summary 20

3 Related Work **22**

4 Contrastive Policy Optimization **27**
4.1 Problem Outline 28
4.2 Action Contrastive Policy Optimization (ACPO) 29
4.3 Contrastive Reinforcement Learning 33
4.4 Algorithm Specification 34
4.5 Summary 36

5 Experiments and Results **38**
5.1 Research hypothesis and success criteria 38
5.2 Research domain 39
5.3 Experiment design 41
5.3.1 Procgen environments 41
5.3.2 ACPO and baseline comparisons 43

5.4 ACPO Performance 44
5.5 Discussion 47
5.5.1 Comparing ACPO to PPO(OC) 47
5.5.2 Comparing ACPO to PPO baselines 50
5.5.3 ACPO with low-performance trajectory data 54
5.6 Summary 55

6 Conclusion **56**
6.1 Future Work 57

References **61**

A Implementation Details **62**

B Hyperparameter Optimization **63**

List of Figures

1.1 Scenario from the game Sonic the Hedgehog showing a structurally (the physical layout of interactive objects) and semantically (the meaning of these objects to the agent) similar scenario where the agent must employ the same action to overcome the obstacle and collect the reward. [Nichol *et al.* 2018] 2

2.1 The interaction loop between the agent and environment. [Sutton and Barto 2018] 6
2.2 Definitive structure of a CNN. [Balaji 2020] 14
2.3 A residual block denoting a "skip connection". [He *et al.* 2015] 15
2.4 An image matching problem illustrating the basic concept behind contrastive learning. [Chaudhary 2020] 17
2.5 A diagram showing the image transformations forward passes and subsequent latent space representations in the SimCLR framework. [Chen *et al.* 2020] 18

4.1 A contrastive problem requiring an agent to differentiate situationally between each state observation in the game Coinrun. 28
4.2 An expanded SimCLR that incorporates action-values. This serves as ACPO's contrastive state representation model, with our change to the original SimCLR network structure boxed in red. 31
4.3 An observation comparison from the game Bossfight illustrating the difference in situational context for near identical visual representations. The red circle highlights an active shield for the enemy. This makes it disadvantageous for the RL agent to perform an attack action in this scenario. 32
4.4 Our amendment for incorporating contrastive learning into a policy gradient algorithm within the RL interaction loop detailed in Figure 2.1. . . 33

5.1 The Procgen environment featuring multiple games with procedurally generated levels. [Cobbe *et al.* 2019] 40
5.2 Training performance of the ACPO agents in comparison to the baseline PPO agents on the selected Procgen environments. 45
5.3 Structurally similar states with different contexts in the game Ninja. . . . 48
5.4 Performance comparison of ACPO and PPO(OC) agents with increased model parameters. 49

5.5 Performance comparison of an ACPO agent and an ACPO agent with low-quality training trajectories. 54

List of Tables

5.1 Mean-aggregated test rewards achieved by each agent over 1000 evaluation episodes, ± one standard deviation. 46
5.2 Ground truth accuracy of each agent's actor-critic structure across the Bossfight and Ninja environments. 53

B.1 PPO Hyperparameters 64
B.2 SimCLR Hyperparameters 64

Chapter 1

Introduction

Humans can solve phyical problems that have multiple distinct variations using the same set of pre-learned skills and understanding before their attempt, so long as these problems have similar characteristics and physical dynamics. For example, they may use previous driving experience to deal with the various situations encountered when trying to reach a new destination they have not visited before [Ha and Schmidhuber 2018]. Consider the example of turning. When a person takes a left turn, one takes a series of steps while accounting for general obstacles that may impede them. These could include pedestrians crossing the street, oncoming cars, potholes in the road, or a combination of the three. The important thing is that the driver takes these steps (looking in the appropriate directions, turning the wheel, and changing the gear) with the awareness that an instance of these problems may occur and be cautious to avoid an accident. The driver does not need to have practised turning left in every possible scenario with every possible obstacle to succeed, but instead, he or she may apply their general knowledge and ability (from previous experience) in navigating left-turn situations encountered on their journey. For human beings, this adaptability is a fundamental characteristic of applying skills, such as left turns, to new problems in consistently changing environments. We are able to perform this function by summarising the knowledge, intuition and practice we have attained in prior efforts to solve similar problems.

Reinforcement learning (RL) is an algorithmic approach that is characterised by an automated computer agent learning what action to perform in a particular situation to receive the highest possible numerical feedback from a function predefined by a person for a particular problem [Sutton and Barto 2018]. This means that a computational agent can interact with the environment without knowing the rules that govern this feedback initially and learn an optimal mapping of situations (like a left turn) to actions (like turning a steering wheel to the left) to maximise this reward. The learning agent is not told which actions to take for any scenario but must instead discover this through interacting with the environment [Sutton and Barto 2018].

From this, it stands to reason that a generally intelligent computer agent should also learn in a way that will allow it to adapt and reuse old knowledge and skills. This would allow the agent to solve structurally (physical layout of interactive objects) and

semantically (the relationship between objects) similar sub-tasks within environments with the same mechanics (the rules of physics and transition probabilities) [Mazoure *et al.* 2020]. The reason we require the reuse of old knowledge and skills is that even when using more complex RL approaches that employ function approximation and deep learning methods, treating every state encountered by an agent as an entirely unique situation is sub-optimal and computationally expensive [Ivanov and D'yakonov 2019]. Whilst treating every state as unique sometimes solves more predictable problems, like arcade platformers with set, predetermined levels (Atari games of the 1980s and 1990s), environments that have higher state and observation complexities, such as randomly moving objects, will present a problem even for larger, more sophisticated learning models [Cobbe *et al.* 2019]. It also presents a problem when learning in procedurally generated environments (environments that change in object placement and difficulty each time they are attempted), as any observation, though potentially representing a state similar to one previously encountered, is fundamentally unique. This means that the optimal action will have to be learnt indefinitely [Cobbe *et al.* 2019]. For this reason, we require a learned function that can infer the overall physical structure of a situation and apply an appropriately similar strategy for decision-making in similar situations, the same way a person would solve a problem. To do this, one has to extract a particular state's most important features and incorporate them into learning.

To illustrate the problem above, consider the Open AI Retro Gym's Sonic the Hedgehog game. This game requires a player to navigate an agent through a two-dimensional platform world, collecting rings whilst avoiding obstacles and destroying enemies, all within a time constraint, to get the highest possible reward [Nichol *et al.* 2018]. An example from this setting is shown in Figure 1.1 below:

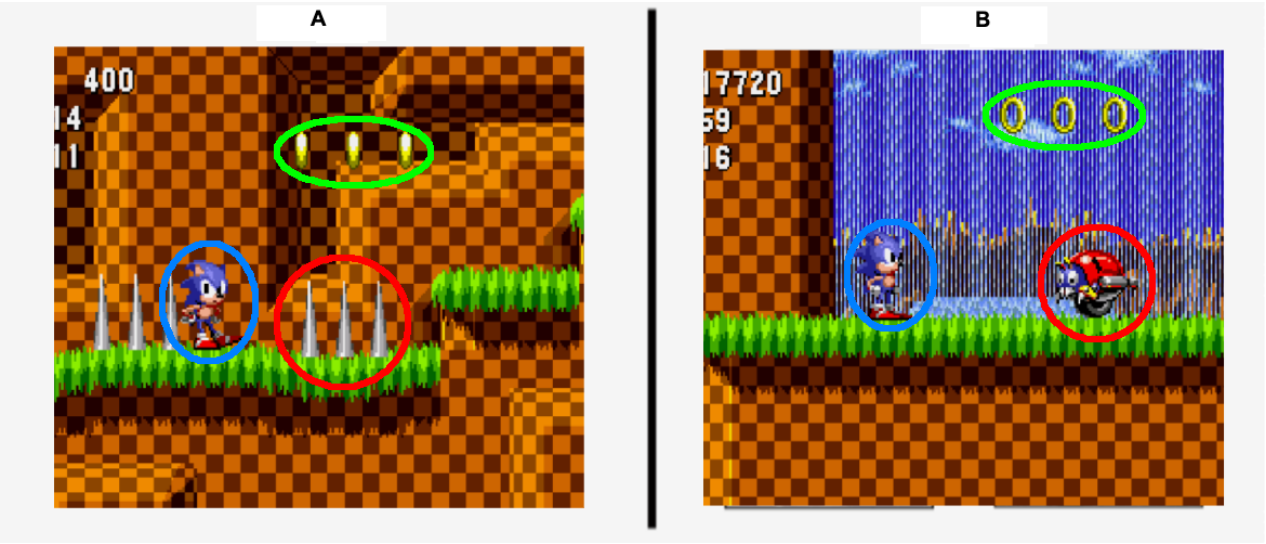


Figure 1.1: Scenario from the game Sonic the Hedgehog showing a structurally (the physical layout of interactive objects) and semantically (the meaning of these objects to the agent) similar scenario where the agent must employ the same action to overcome the obstacle and collect the reward. [Nichol *et al.* 2018]

Figure 1.1 shows two distinct states from the game that are very similar in both physical structure and object semantics. It requires the agent (circled in blue) to take the upward, right jump action to collect the reward (circled in green) of golden rings above whilst avoiding the episode ending factor beneath it. In image A, this is simply a group of spikes (circled in red) that could harm the agent; in image B, this is an enemy agent (also circled in red). The action it must take in both states is the same despite the difference in the identity of its obstacle. Effectively, The agent must be able to infer that it is in a 'jump right' scenario to employ the correct action in all states similar to this.

State representation learning has presented several interesting advances in the field of reinforcement learning (RL) [Raffin *et al.* 2019]. The idea behind this concept is for the representation model to learn the environmental dynamics, observational structure, or transition probabilities and to represent this information compactly for each state to be used for downstream tasks. This can be seen as a more sophisticated form of feature extraction or, more generally, feature learning [Raffin *et al.* 2019]. It is this new state representation that is then used for reinforcement learning. The benefit of these newly learned representations is that they are compact and can contain relevant environmental information if they are trained effectively. They also speed up policy learning, reducing the number of samples needed to reach an overall optimal policy [Raffin *et al.* 2019].

Prior work has demonstrated that state representation learning algorithms can create summaries of state observations and situational dynamics, allowing information to be encoded in such a way that an RL algorithm can infer through enough training when it has encountered a state that is similar in structure and semantics to one that it has encountered before [Eysenbach *et al.* 2023]. This allows this algorithm to take the same action in that state and achieve the optimal outcome. However, the problem with many of these approaches is that they do not include ways to incorporate information about the agent's typical or desired behaviour into the latent space representation. Many of them simply extract the primary spatial or temporal features of an observation and pass this information to an agent in lower-dimensional vector space. We aim to combine observational and behavioural characteristics into a complete contrastive model, allowing for more sophisticated situational inference.

Our core research contribution is the development of a novel algorithm to demonstrate how using contrastive learning as a state representation model can help us learn a contextual summary of a state by combining agent actions and environment observations into our state representation model used for downstream RL. We term this algorithm Action Contrastive Policy Optimization (ACPO). Contrastive learning is a set of self-supervised, differentiation learning algorithms designed to learn discriminatory information about an input [Le-Khac *et al.* 2020]. It does this in images by capturing features within the image and assessing relevant similarities. With this information, the model separates images in latent space, encoding images with like features in closer proximity to one another. This ability to contrast between states and group them on structural and semantic similarity during an RL task is the core contribution of this research. Specifically, we aim to determine if a policy-gradient-based RL agent can use a contrastive state representation model, namely a Simple Framework for Contrastive Learning of

Visual Representations (SimCLR) by Chen *et al.* [2020] to differentiate scenarios similarly to the way human beings do. We choose this framework because it has demonstrated state-of-the-art performance on a complex array of tasks as a self-supervised contrastive method, besting multiple supervised learning methods in a high number of classification tasks [Chen *et al.* 2020]. Effectively, we aim to assess whether an agent augmented with a high-performing contrastive model can use information learned in previously encountered states to improve its performance when encountering a similar state.

Our experiments in Chapter 5 show that policy-gradient-based agents achieve markedly better performance when employing our ACPO approach when the contrastive model is trained using high-quality data. This performance improvement is across both training and test cases, meaning the agent generalises its learned skills well to unseen environments. The agents using contrastive models also experience less performance decay when deployed to unseen test levels, so they generalise better than agents that do not use contrastive models for state representation. These results show that our approach to incorporating contrastive models into policy-gradient methods shows promise for future research.

This dissertation introduces the reinforcement learning and deep learning concepts needed to understand this research in Chapter 2. In this chapter, we explain the specific methods we are using to solve the RL problem. In Chapter 3, we discuss the most recent literature that successfully incorporates state representation learning into RL. In Chapter 4, we detail our contribution by explaining how we augment the RL interaction loop to incorporate a contrastive learning model. This first shows how contrastive learning can work with RL, and secondly, it gives an easily referenced abstraction with which we can explain our core algorithm, Action Contrastive Policy Optimization (ACPO). We present the results in Chapter 5, showing the success of our approach, along with a discussion and analyses to understand why it works. Finally, we conclude our work in Chapter 6, summarising the evidence for our hypotheses and recommending future research initiatives that could expand our idea.

Chapter 2

Background

In this section, we discuss the prerequisite theory for the methods being used in this research. We start by defining the RL problem in section 2.1, relating it to its foundational theory, the Markov Decision Process (MDP). This helps establish a basic overview relevant to any RL problem. In section 2.1.1, we discuss deep reinforcement learning (DRL), a core method of our research, and the general state-of-the-art set of methods used to solve more practical reinforcement learning problems. As a supplement to future sections, we spend section 2.3 discussing convolutional neural networks (CNN), an essential deep learning algorithm used for extracting features from the images in the test environment for our experiments. In section 2.3.1, we discuss the residual network, the CNN of choice for the base encoder of our algorithm. We cover residual networks to assist in understanding our method's contrastive learning algorithm, as this is the CNN of choice in this specific implementation of the contrastive learning framework.

Finally, in section 2.2.1 and 2.4.1, we look at the two methods that we combine in this research for a novel enquiry. The first is Proximal Policy Optimization (PPO), a state-of-the-art policy gradient method, which parameterises and subsequently optimises an RL policy by clipping the trajectory it generates from experience and only updating the policy using this clip. The second is the Simple Framework for Contrastive Learning of Visual Representations (SimCLR), the contrastive learning framework we augment to develop our state representation model. This is a framework for differentiating images using self-supervised image contrasting, allowing our agent to separate the states that it encounters based on these features.

2.1 Reinforcement learning and Markov decision processes

Markov Decision Processes (MDP) are a classical formalisation of the RL problem, which are essentially a series of decisions made in sequence within some environment, with each action influencing the subsequent states and feedback signals an acting agent may receive [Sutton and Barto 2018]. In this section, we formulate the RL problem as an MDP using a series of mathematical expressions. We define returns, discount factors,

value functions and Bellman equations. This forms the basis for all RL theory used in this paper. We then extend this theory to function approximation and policy gradient methods, two fundamental concepts in advanced reinforcement learning.

The RL problem consists of a cyclical interaction loop between two entities, the learner, hence referred to as the agent and an external construct, hence referred to as the environment [Sutton and Barto 2018].

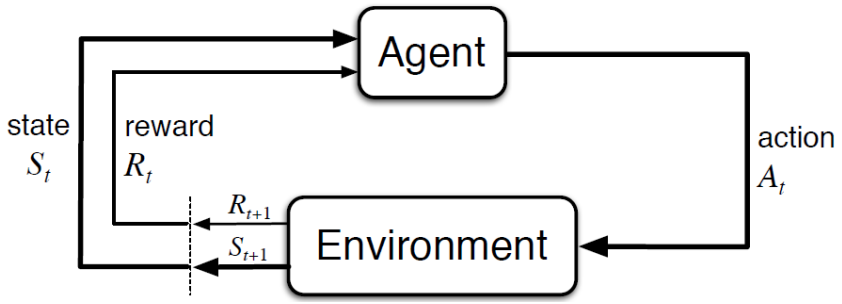


Figure 2.1: The interaction loop between the agent and environment. [Sutton and Barto 2018]

Figure 2.1 describes the interaction loop between these entities as a series of time steps. These steps can be discrete or continuous, but for the purposes of our explanation, we use discrete time steps. At any given step t, the agent will find itself in a circumstance called a state. This variable is customarily defined as s, where $s \in \mathcal{S}$, and where $\mathcal{S}$ is a set of all states reachable within the environment. Within this circumstance, it is allowed to take an action a, where $a \in \mathcal{A}(s)$, where $\mathcal{A}(s)$ is the set of all actions available in state s. In the subsequent timestep $t+1$, the agent will receive a numerical signal called a reward and a new state s_{t+1}. We define this reward as $r \in \mathcal{R}_{t+1} \subset \mathbb{R}$. This essentially means that an agent receives a reward in the next timestep from the set of all possible rewards available at that timestep. The set $\mathcal{R}$ is a subset of all real numbers. This cyclical sequence of state, action and reward forms what is referred to as a trajectory and is the essential mechanism of RL [Sutton and Barto 2018].

For the purposes of this thesis, we limit our explanation of MDPs to finite MDPs. In a finite MDP, all $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}$ have a finite number of elements [Sutton and Barto 2018]. $\mathcal{R}_t$ and $\mathcal{S}_t$ for any timestep t have discrete probability distributions based only on the preceding state and action[Sutton and Barto 2018]. This can be mathematically formulated by the following equation:

$$p(s', r|s, a) \doteq \Pr\{\mathcal{S}_t = s', \mathcal{R}_t = r|\mathcal{S}_{t-1} = s, \mathcal{A}_{t-1} = a\} \tag{2.1}$$

The probability function p defined in equation 2.1 defines the dynamics of the MDP and can be used to compute other information about the environment, such as state-transition probabilities, the expected rewards for particular state-action pairs and state-action next-state triples [Sutton and Barto 2018]. This information is initially unknown and is discovered through a continuous interaction loop with the environment illustrated in figure 2.1.

The goal or purpose of an RL agent is to maximize the numerical feedback signal or reward it is given, through taking actions in various states of the environment at various timesteps. This goal must be maximised cumulatively over time, not just for immediate rewards, and the rewards must be representative of the correct action we want the agent to perform in its environment [Sutton and Barto 2018]. For this, we must understand what we want the agent to receive in a given environment and then attach rewards to this environment so that the agent discovers how to achieve it in the best possible way through the duration of its interactions. This concept is known as the reward hypothesis, or that a goal is the maximization of the expected value of the cumulative sum of the received numerical feedback signal [Sutton and Barto 2018].

We now formally define the agent's goal as the cumulative expected return over a set of timesteps. The set of states S will contain a terminal state τ for a fixed number of timesteps. This is referred to as an episode [Sutton and Barto 2018]. For a continuous task, τ is infinity. We define the cumulative expected return as G_t. The agent performs its interaction loop, receiving a reward r at each successive timestep t. The agent must take action within each state s to optimize G_t by generating the largest possible reward sequence before reaching τ or reach a maximum G_t as fast as possible for a continuous task [Sutton and Barto 2018].

To perform optimally, an agent must consider the value of future rewards compared to its present reward. We add a variable to the reward sequence known as a discount factor γ [Sutton and Barto 2018] to account for this. This gives us the complete equation for the expected reward:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

Adding the parameter γ to the cumulative reward sequence defined in 2.2 gives future rewards, however large or small, a present value. We assign a value to γ such that $0 \leq \gamma \leq 1$. If $\gamma = 0$, then all terms following R_{t+1} will be 0, meaning that the agent is only concerned with immediate rewards, which means it will only choose A_t to maximise R_{t+1}, not being concerned with the invariable 0 values of future rewards, regardless of the action taken. If we adjust γ closer to 1, the agent will act with more regard to future rewards. It is important to consider a form of the expression 2.2 that is unified to express both episodic and continuous problems [Sutton and Barto 2018]. For this, we can use the expression:

$$G_t \doteq \sum_{k=t+1}^{T} \gamma^{k-t+1} R_k \tag{2.3}$$

Equation 2.3 can be used to define episodic and continuous RL problems depending on the value of T, with $T = \infty$ for continuous problems. In the case of a continuous problem, γ must be less than 1 for the expression to remain representative for continuous RL problems, less the cumulative return G_t tend to infinity [Sutton and Barto 2018].

Much of the problem assessment in RL uses a mathematical construct called a *value functions*. Value functions are a numerical measure of how beneficial it is for an agent to be in a particular state or perform a particular action in some state [Sutton and Barto

2018]. This benefit is quantified in the form of the expected return. To understand how value functions work, we introduce the concept of a *policy*. A policy is a mapping from states to the probability of selecting one of the actions available in a state [Sutton and Barto 2018]. This can be expressed as $\pi(a|s)$, where π is a basic probability function that determines the probability of choosing action a, where $a \in \mathcal{A}(s)$, for all $s \in \mathcal{S}$. The goal of an RL agent is then to change π through experience in order to reach an optimal policy, defined as π^* [Sutton and Barto 2018].

We use two value functions for this explanation, the *state-value function* defined as $v_\pi(s)$, and the *action-value function* defined $q_\pi(s, a)$ [Sutton and Barto 2018]. The function $v_\pi(s)$ can be expressed as $v_\pi(s) \doteq \mathbb{E}[G_t|S_t = s]$, or, the expected return, given that the agent is in state s, at any timestep t and follows policy π thereafter. Similarly, $q_\pi(s, a)$ can be expressed as $q_\pi(s) \doteq \mathbb{E}[G_t|S_t = s, A_t = a]$, or, the expected return, given and that agent takes action a in state s, at any time step t, and follows policy π thereafter [Sutton and Barto 2018]. These value functions are determined from the agent's experience interacting with the environment and updating its policy to favour states and actions with higher value functions. This successively leads to higher reward trajectories, eventually converging to an optimum value [Sutton and Barto 2018].

A common set of algorithms used to accomplish this is dynamic programming (DP), a method that can compute an optimal policy if given a complete model of an MDP [Sutton and Barto 2018]. DP is not a functional solution for RL as it requires an unreasonably large amount of computing and time to reach an optimal solution as the environment complexity increases, but rather, it serves as an important theoretical illustration of how an RL problem can be computed [Sutton and Barto 2018]. To obtain a DP solution using value functions, we need to turn the Bellman equations into assignment operations or update rules for evaluating and improving the value functions.

We introduce the concept of the Bellman equation as a consistent abstract expression of how we can compute value functions. A Bellman equation represents a recursive relationship between the value of a state or state-action pair and its successor states. It assesses the probability that the agent will encounter each possible successor state and discounts them accordingly [Sutton and Barto 2018].

$$v_\pi(s) \doteq \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \Big[r + \gamma v_\pi(s') \Big] \tag{2.4}$$

$$q_\pi(s, a) \doteq \sum_{s',r} p(s', r|s, a) \Big[r + \gamma \sum_{a'} \pi(a|s) q_\pi(s', a') \Big] \tag{2.5}$$

Equation 2.4 describes the Bellman equation for state value functions. That is, for each state, it assesses the probability of taking each action a in state s, multiplied by its next state transition probability, and then by its reward and next-state value, and summing this computation for all possible actions [Sutton and Barto 2018]. Alternatively, equation 2.5 describes the Bellman equation for state-action functions. That is, each state-action pair assesses the transition probabilities and corresponding rewards of each possible next state given some action. It then multiplies this by the reward plus the discounted summation of the probability of taking action a in state s under policy π, multiplied by all subsequent actions a' available in the next state s' [Sutton

and Barto 2018]. We can use these equations to begin an iterative process of solving an environment problem using an RL agent.

The first step in the process of improving an agent's performance is the evaluation of some policy π [Sutton and Barto 2018]. This is also known as the prediction problem. We begin with some arbitrarily initialised policy and the complete knowledge of the environment dynamics p. We then perform some number of computational iterations k, to determine the value of each state. We use some threshold, commonly referred to as a delta, to determine when to cease evaluative iterations [Sutton and Barto 2018].

Equations 2.4 and 2.5 express the transition probabilities and the discounted reward calculations of the policy iteration step to determine a value under some policy π. Solving a reinforcement learning task means that an agent must improve its policy to receive larger rewards over a period of time until this is no longer possible for the algorithm [Sutton and Barto 2018]. In brief, the agent must discover an optimal policy. A policy π' is better than a policy π if its expected return is better or equal for all states. This can be mathematically expressed as $\pi' \geq \pi$ if and only if $f_{\pi'}(s) \geq f_{\pi}(s)$ for all $s \in \mathcal{S}$, where the function f is defined as either v or q [Sutton and Barto 2018]. The optimal policy is denoted by π_* and shares the same optimal state-value and action-value functions.

$$v_*(s) \doteq \max_a \sum_{s',r} p(s', r|s, a)\Big[r + \gamma v_*(s')\Big] \tag{2.6}$$

$$q_*(s, a) \doteq \sum_{s',r} p(s', r|s, a)\Big[r + \gamma \max_a q_*(s', a')\Big] \tag{2.7}$$

Equations 2.6 and 2.7 denote the *optimal state-value function* and *optimal action-value function* respectively [Sutton and Barto 2018]. Note that we need not write v_* or q_* with specific reference to any policy to satisfy its consistency condition. The term used to denote the above equations is the Bellman optimality equations and expresses that any state, or state-action pair, in an optimal policy, must be equal to the expected return for the best actions available in that state [Sutton and Barto 2018].

Through exploration of the environment dynamics p, one can solve the system of equations that define v_* and q_* through various mathematical and statistical methods, some of which we discuss in this paper. When v_* and q_* are defined, to find the optimal policy, one need only act greedily for each state or state-action pair [Sutton and Barto 2018]. That is, the agent must simply choose the action that yields the highest evaluation of either function in its immediate state, as the optimality function already reveals to the agent the highest expected return for each state or state-action pair.

The solution discussed at this point is one that exhaustively searches through each state or state-action pair, computing both the probability of transition and expected return. This works in smaller problem spaces where we can accurately learn the environment dynamics, and there are abundant computational resources for the agent to utilise [Sutton and Barto 2018]. Most modern reinforcement learning problems have high space and action complexities that can lead to potentially millions of discrete state-action possibilities or vast-ranging continuous spaces. It is for this reason that reinforcement

learning settles for sophisticated approximation solutions and methods [Sutton and Barto 2018].

2.1.1 Function approximation and Deep reinforcement learning

To alleviate the computational and storage costs of using tabular methods in complex problem spaces, an agent must be capable of generalising current states with previously encountered states to make sensible decisions, as most states an agent will encounter are unique [Sutton and Barto 2018]. It is for this reason that we employ function approximation. Function approximation works by having an agent generate samples from the environment function and then proceed to estimate its dynamics [Sutton and Barto 2018]. For this writing, we restrict our discussion to on-policy function approximation, as our method does not use off-policy algorithms.

In formal terms, a function approximator $\hat{v}$ estimates a state-value function $\hat{v}_\pi$ using a policy π that is known at the current timestep [Sutton and Barto 2018]. The change in approach from tabular methods is that instead of using a table to store our value function, we instead parameterise the function $\hat{v}$ as $\hat{v}(s, \mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^d$. The target to be predicted with this function can be written as $\hat{v}(s, \mathbf{w}) \approx \hat{v}_\pi(s)$ [Sutton and Barto 2018]. We can perform a similar change for the state-action value function by simply adding the action as a parameter, resulting in $\hat{q}(s, a, \mathbf{w}) \approx \hat{q}_\pi(s, a)$ [Sutton and Barto 2018]. The weight values $\mathbf{w}$ in these functions are vectors multiplied over all function variables used to determine the state or state-action values. The function $\hat{v}$ and $\hat{q}$ can be any mathematical function capable of accurately modelling the environment dynamics. For example, linear functions, polynomials, decision trees or artificial neural networks [Sutton and Barto 2018]. In the remainder of the section, we introduce deep reinforcement learning (DRL), a set of state-of-the-art function approximation algorithms used in our method.

DRL methods are state-of-the-art RL techniques that can be split according to two basic approaches. The first is value-based methods, which keep function-approximated state-action values queryable through some function expression to determine an optimal policy. They keep no explicit representation of the policy as it can be inferred directly from the function itself by getting the optimal state-action values. These methods instead try to assess $Q : S \times A \rightarrow R$, or the reward value of each state-action combination [Ivanov and D'yakonov 2019].

In contrast, policy gradient methods keep an explicit, weight-parameterized representation of the policy that is updated from experience gained whilst learning. It uses the trajectories an agent generates to determine the best action a_t when in some state s_t, and instead tries to solve for $\pi : S \rightarrow A$, or the optimal, reward-generating action an agent should take when in some state [Ivanov and D'yakonov 2019]. In our research method, we exclusively use policy gradient algorithms. In the next section 2.2, we discuss proximal policy optimization (PPO), our policy gradient algorithm of choice.

2.2 Policy gradient optimization

The modern implementation of deep reinforcement learning uses deep neural networks to parameterize the state-action values of value-based methods and the policies of policy gradient methods [François-Lavet *et al.* 2018]. For policy gradient algorithms, the policy, defined as the probability of an agent taking action a in an arbitrary state s, is augmented as $\pi_\theta(a|s;\theta)$, where θ denotes the weights of the chosen function approximator. The optimization objective can be mathematically defined as:

$$J(\pi_\theta) = E_{\tau\sim\pi_\theta}[R(\tau)] = \int_\tau P(\tau,\theta)R(\tau) \tag{2.8}$$

$$\theta_{k+1} = \theta_k + \alpha\nabla_\theta J(\pi_\theta)|_{\theta_k} \tag{2.9}$$

Equation 2.8 structures our cost function as the expected rewards equal to the sum of all probabilities P, of a trajectory τ multiplied by its rewards $R(\tau)$. This can be represented as the integral over $P(\tau,\theta)R(\tau)$. The objective of the policy gradient method is then to find a policy π_θ to generate some trajectory τ that maximises the function J. This is performed through the gradient ascent step detailed in equation 2.9. To compute the cost function or expected value in this algorithm, we need to use a form of the probability function that is easily differentiated and summed [François-Lavet *et al.* 2018].

Let us recount that the probability of a trajectory $P(\tau|\theta)$, due to the Markov property is $\rho_0(s_0)\prod_{t=0}^{T} P(s_{t+1}|s_t,a_t)\pi_\theta(a_t,s_t)$ where ρ is the sample distribution from which s_0 is drawn. The optimization uses a technique known as logarithmic differentiation to allow the algorithm to use backpropagation on the parameters θ more simply. T in this function can be defined as some terminal state or horizon. The partial derivative with respects to θ of $P(\tau|\theta)$ can be expressed as $\nabla_\theta P(\tau|\theta) = P(\tau|\theta)\nabla_\theta\, log\, P(\tau|\theta)$. The log representation allows us to take the sum of the probability of a trajectory $P(\tau|\theta)$ instead of its product, allowing for a more seamless gradient calculation. This is defined as $log\, P(\tau|\theta) = \rho_0(s_0) + \sum_{t=0}^{T} log\, P(s_{t+1}|s_t,a_t) + log\, \pi_\theta(a_t,s_t)$. When taking the derivative with respect to θ, the initial state and $log\, P(s_{t+1}|s_t,a_t)$ have no dependency on the θ variable, and hence their values will always be 0. This illustrates why we require the log form of the derivative, as using the normal representation of $P(\tau|\theta)$ would make gradient ascent impossible, as taking the product will always yield 0 [François-Lavet *et al.* 2018]. The final representation of the gradient ascent cost function will be:

$$\nabla_\theta J(\pi_\theta) = E_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta\, log\, \pi_\theta(a_t|s_t)R(\tau)\right] \tag{2.10}$$

This equation uses an expected value that can be estimated by interacting with the environment and generating a mean score overall interactions, allowing us to estimate the best policy using gradient updates for θ [François-Lavet *et al.* 2018]. Examples of policy gradient methods include deterministic policy gradients which output an action given a state, and stochastic policy gradients which output a distribution over all possible actions, giving the highest probability to the action with the highest potential reward [Ivanov and D'yakonov 2019].

2.2.1 Proximal policy optimization

The cost formulation of basic policy gradient methods, as described in 2.2, form the basis of all policy gradient algorithms. However, executing gradient ascent updates (as described in 2.10) entirely without optimisation incurs marked performance drops in more complex RL problems where successful solutions in an environment require more specific optimal target policies. Hence, Schulman *et al.* [2017] proposed Proximal Policy Optimization (PPO), a policy gradient algorithm that attempts to improve policy update stability by restricting the amount of change the algorithm can backpropagate at each training epoch.

PPO is an on-policy actor-critic algorithm that uses a reformulation of the policy update objective defined in 2.10. Schulman *et al.* [2017] term this new function clipped surrogate objective. The clip surrogate objective function essentially constrains a policy update within the bounds of a smaller subset of a prospective trajectory called a clip. We can change the function 2.10 to the following:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big] \tag{2.11}$$

In this new policy update function, we reduce the scope of the term $\nabla_\theta J(\pi_\theta)$, which used a summation of loss for all rollouts from t_0 to a terminal state T, to $L^{CLIP}(\theta)$, which only expresses the loss calculated over a single trajectory.

To understand the updated policy, we need to understand the ratio function, which is denoted by $r_t(\theta)$, and the advantage function denoted by $\hat{A}_t$. The ratio function is defined as the probability of taking an action a_t in the state s_t using the current learned policy, divided by the probability of taking the same action in s_t using the previous iterations policy [Schulman *et al.* 2017]. The resulting formulation allows us to define the first parameter of the **min** function in 2.11 as:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t\Big[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}_t\Big] = \hat{\mathbb{E}}_t\Big[r_t(\theta)\hat{A}_t\Big] \tag{2.12}$$

The ratio function serves as an indicator of how a policy has changed since the last policy update step. If we have the condition of $r_t(\theta) > 1$, we can deduce that the action a_t is more likely with the current policy, if the condition is $r_t(\theta) < 1$ we can assume a_t is less likely. This property allows us to replace the logarithmic function in 2.10 with the ratio function, as it ensures the output value always falls within a non-zero range, which is a prerequisite for functional backpropagation.

The advantage function in these equations is an estimator that attempts to determine how good a particular state-action pair is in comparison to the currently known average return when taking other actions in the same state. Its loss function is defined as:

$$\hat{A}_t = Q_\theta(s_t, a_t) - L^{VF}, \text{ where } L^{VF} = (V_\phi(s_t) - \hat{R}_t)^2 \tag{2.13}$$

In equation 2.13, we take the difference between two independently parameterised functions, Q_θ and L^{VF}, in this instance defined as the mean-squared error between the estimated value of s_t and the actual reward received [Schulman *et al.* 2017]. The

purpose of the advantage function is to ensure that the agent stays informed through some function that can estimate the value of an action, whether or not to increase an action probability or to reduce it. That is, if the advantage is positive for a particular action, the agent will adjust θ such that it increases the probability of taking it, and if the advantage is negative for an action, it updates θ to reduce the probability of taking it. This illustrates the behaviour of the unclipped parameter of the **min** function in 2.11.

The ratio function can potentially lead to very large policy updates if the probability of taking a_t in s_t in the current policy is much higher than the previous policy. This excessively large policy update can lead to problems with convergence to an optimal policy or irrecoverably bad performance. This justifies the clipped parameter in 2.11. The function **clip** serves the purpose of penalising policy updates that lead $r_t(\theta)$ too far away from 1 using the hyperparameter ϵ. Schulman *et al.* [2017] use an ϵ value of 0.2 in their implementation, ensuring the function is constrained as $0.8 \leq r_t(\theta) \leq 1.2$. The **min** function then takes the lower bound of ratio function when comparing clipped and unclipped and returns the product of this value with the advantage function value. This illustrates the behaviour of the clipped parameter of the **min** function. The final algorithm of the PPO-clipped surrogate objective is as follows:

Algorithm 1 PPO-clipped surrogate objective algorithm [François-Lavet *et al.* 2018; Schulman *et al.* 2017].

Input: Initialise policy parameters θ_0 and value function parameters ϕ_0.
for $k = 0, 1, 2$ ... **do**
  Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$, by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates $\hat{A}_t$ (using L^{VF}) based on ϕ_t of $V_{\phi k}$.
  Update the policy by maximising the PPO-Clip surrogate objective defined in 2.11 across all trajectories:
  $\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} L^{CLIP}$ via stochastic gradient ascent.
  Fit value function by regression on mean-squared error defined in 2.13:
  $\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} L^{VF}$ via any gradient descent algorithm.
end for

2.3 Convolutional neural networks

Convolutional neural networks (CNN) have become established as a state-of-the-art image learning function approximator [Alzubaidi *et al.* 2021]. They have shown superior performance in object detection, identification of differing objects, recognition of different objects of the same type, and overall superior performance in object feature detection [Alzubaidi *et al.* 2021]. Deeper CNNs have increased complexity, learning to identify larger portions of the image with greater specificity. The initial layers of the network centre around learning simple visual features such as edge detection, with subsequent layers using the pixel data to identify larger shapes and features until finally being able to form a numerical profile of the image [Goodfellow *et al.* 2016].

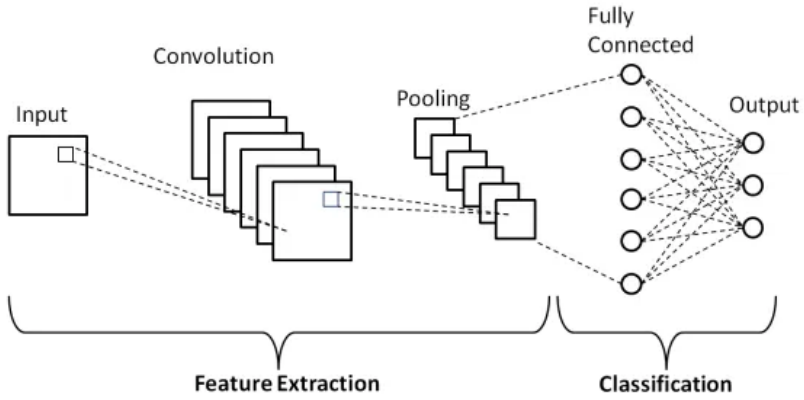


Figure 2.2: Definitive structure of a CNN. [Balaji 2020]

All CNN structures share three definitive layers, and the first is the convolutional layer [Li *et al.* 2021]. We observe in Figure 2.2 that it is the layer of the CNN where the majority of the feature learning occurs within the convolutional model. The purpose of a convolutional layer is to accept a matrix input, which typically includes between one to three channels, one for greyscale images and three for colour images (each channel corresponds to a red, blue or green colour value c, where $0 \leq c \leq 255$, the combination of which gives the pixel a unique colour) [Li *et al.* 2021; Mnih *et al.* 2013].

Within each convolutional layer, there is a construct called a kernel. A kernel is defined as a $n \times n$ matrix that performs a dot product operation on each $n \times m$ input channel using a stride (an incremental horizontal and vertical shift of the kernel over the channel) [Li *et al.* 2021]. The output array is then a collection of dot product results dependent on the stride size, which is dependent on the values of the stride and the size of the kernel. This output array is known as a feature map and is passed to an activation function to apply a non-linear transformation [Li *et al.* 2021].

The pooling layer follows a single or series of convolutional layers as part of the feature extraction step illustrated in 2.2. The purpose of a pooling layer is to perform dimensionality reduction and create a numerical summary of the feature map [Goodfellow *et al.* 2016]. It does this using a kernel and stride similar to the ones described in the convolutional layer. The purpose of this layer is to reduce complexity and limit overfitting. However, the difference between the kernel in the pooling layer is that it uses an

aggregation function to create its output layer instead of a dot product operation. The user can define one of two types, max pooling and average pooling, as the aggregation pooling function. Max pooling returns the maximum value within the kernel, and average pooling returns the mean [Goodfellow *et al.* 2016].

The last definitive component of a CNN, the fully connected layer, is distinct from the feature extraction components that are only partially connected to the final output layer [Li *et al.* 2021]. The fully connected layer is typically represented as a multi-layer perceptron (MLP), that receives input from a pooling layer, and each node of its final layer connects to a node of the output layer which is then used for a downstream task (i.e. image classification) [Li *et al.* 2021].

2.3.1 Residual networks

Since the conception of CNNs in image recognition and classification, researchers have applied multiple variations and augmentations to the LeNet CNN structure described in 2.3 to improve its performance generally or for specific tasks (ref). For example, AlexNet and VGGNet are CNN that use multiple subsequent convolutional layers before using pooling and fully-connected layers. The idea behind this was to increase the depth of the network to improve feature learning with larger training image sets. Li *et al.* [2021] evaluated through comparison that with deeper network structures, a CNN can achieve higher levels of recognition accuracy in general.

Over time, however, it was observed that adding more layers to a CNN model would lead to significant performance degradation due to the vanishing gradient problem. This problem occurs in deeper CNNs due to gradients closer to the fully connected layer rapidly shrinking to zero after multiple applications of the chain rule during back-propagation, making it impossible to learn the correct weights for each image class. To solve this problem, He *et al.* [2015] proposed the Residual Network (ResNet) with the feature residual block that allows for skip connections between convolutional layers.

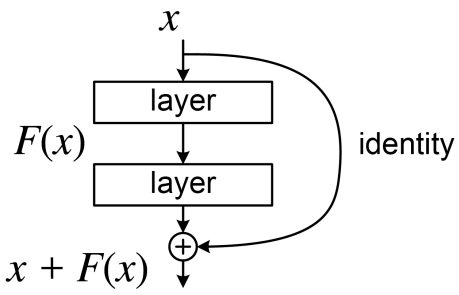


Figure 2.3: A residual block denoting a “skip connection”. [He *et al.* 2015]

Figure 2.3 illustrates the core feature of the ResNet CNN structure, the residual block. To describe the function of a residual block, let x in this figure denote a matrix input layer, and $F(x)$ denote a stack of two convolutional layers, with each downward pointing arrow representing a ReLU activation function. He *et al.* [2015] then define a residual function $H(x)$ as an underlying mapping that the stacked convolutions must

asymptotically approximate. Instead of approximating $H(x)$ directly, however, they instead let it approximate the residual function $F(x) := H(x) - x$, reformulated as $F(x) + x$ where x is an identity mapping of the original input.

The justification behind this reformulation is that if added convolutional layers can be constructed as identity mappings, a deeper CNN should have a training error less than or equal to a shallow CNN. The performance problems encountered by a deeper CNN, however, are due to its inability to learn optimal identity mappings over multiple stacked convolutions [He *et al.* 2015]. He *et al.* [2015] reason that with the residual reformulation, the network only needs to drive the weights of the convolutional layers denoted by $F(x)$ to zero in order to learn the optimal identity mappings. The addition of these identity mappings allows the residual network to pass gradients over deeper networks with only a few layers in between the input and the output layer, allowing for computation with purer gradients.

He *et al.* [2015] formally express the residual block shown in Figure 2.3 as the following general equation that can apply to any number of stacked convolutional layers within a residual function:

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{\mathcal{W}_i\}) + \mathbf{x} \tag{2.14}$$

In the equation 2.14, $\mathbf{x}$ and $\mathbf{y}$ denote the input and output matrices respectively, and the function $\mathcal{F}(\mathbf{x}, \{\mathcal{W}_i\})$ denotes the residual mapping to be learned, with $\mathcal{W}$ denoting the weights of the stacked convolutional layers. We can use this general residual block expression to determine the output of any layer. For example, to obtain the output of the second layer of the residual block in Figure 2.3, we can express $\mathcal{F}$ as $\mathcal{F} = \mathcal{W}_2\sigma(\mathcal{W}_1\mathbf{x})$, where σ denotes a ReLU activation function. The operation $\mathcal{F} + x$ is performed by the skip connection that passes the identity matrix to the function $H(x)$. It performs an element-wise addition to the matrix produced by $\mathcal{F}$. The final non-linear activation function is executed after the addition operation to produce the output layer $\mathbf{y}$.

2.4 Contrastive learning

The term contrastive representation learning, or contrastive learning, encompasses a set of supervised and unsupervised machine learning algorithms [Le-Khac *et al.* 2020], where the goal of the contrastive model is to learn a latent space embedding such that conceptually similar pairs in some set of input data are placed closer together in an n-dimensional vector space [Le-Khac *et al.* 2020].

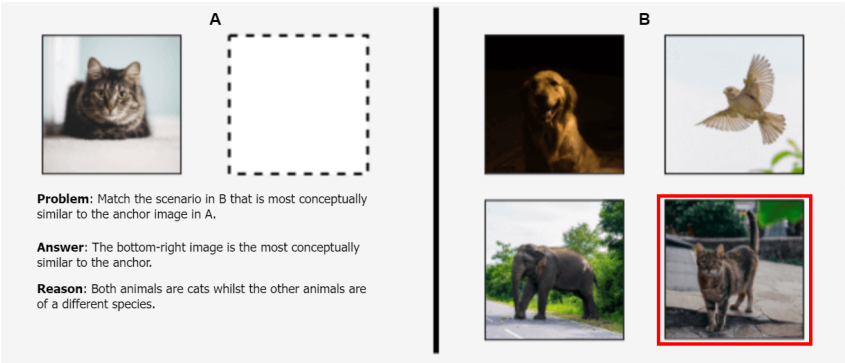


Figure 2.4: An image matching problem illustrating the basic concept behind contrastive learning. [Chaudhary 2020]

To establish an intuition for how this is applied to unsupervised image contrasting problems, consider a simple primary skills learning task taught to humans, the image matching problem. Figure 2.4 illustrates a problem where a human agent must solve a puzzle by matching an anchor image, in this case, a cat, to one of the other four animal images. The correct choice for this anchor is the corresponding image of a standing cat, the common thread being both these images are that of a cat. A human actor is expected to do this by observing the image characteristics or using labels such as animal names. Similarly, contrastive learning aims to learn a representation that keeps members of the same visual properties or labels closer together in latent space.

Contemporary contrastive learning methods typically employ a parameterised function to solve a contrastive task, and as such, they require a learning objective in the form of a loss function. Chopra *et al.* [2005] defined a function simply known as contrastive loss. This is one of the earliest loss functions formulated for Euclidean latent space differentiation. It takes a set of labelled input images and differentiates the latent representation using a parameterized encoding function. To understand how they implement contrastive loss, we can formally define the problem as having a pair of labelled images (I_i, I_j), and some function f_θ that encodes these images into latent space $x = f_\theta(I)$ where θ represents the parameters to optimise. We also include the value Y, which is 0 when pairs have the same label and 1 when they do not. The loss function is then defined as:

$$\mathcal{L}(x_i, x_j, \theta) = (1 - Y)\|f_\theta(x_i) - f_\theta(x_j)\|^2 + Y\mathbf{max}(0, \epsilon - \|f_\theta(x_i) - f_\theta(x_j)\|^2) \quad (2.15)$$

In equation 2.15, we minimize the first term when the images are similar and the second when they are not. The term ϵ is a hyperparameter that sets a lower bound distance between two dissimilar images to regulate backpropagation [Chopra *et al.* 2005].

Contrastive loss used a simple definition of loss that worked for easier image differentiation tasks that used single image pairs. Over time, researchers have developed methods like triplet loss and N-pair loss, a set of supervised contrastive methods that allow for larger image batches to be used in learning. Other algorithms include noise contrastive estimation (NCE) and its class-imbalanced variant InfoNCE, which are ex-

amples of self-supervised contrastive learning algorithms that, by definition, do not require labels.

The more sophisticated algorithms developed after Chopra *et al.* [2005] contrastive loss require specific training methods and data augmentations. For example, for most of the recent contrastive models, it is important to use large batch sizes during training, as this allows the model to sample enough data points that are negative to an anchor for diverse and accurate training. Contemporary contrastive models also require data augmentation to create variance for better generalisability. These ancillary changes are crucial to good performance, especially with self-supervised algorithms. In the next subsection, we discuss a state-of-the-art self-supervised contrastive learning algorithm that forms a cornerstone of our proposed method.

2.4.1 SimCLR

The contrastive learning framework by Chen *et al.* [2020], a Simple Framework for Contrastive Learning of Visual Representations (SimCLR), at the time of this writing, is the current state-of-the-art, self-supervised contrastive learning model for images beating state-of-the-art supervised methods in the ImageNet classification task. The goal of the method is ultimately the same as any self-supervised contrastive model, to bring similar images closer together in latent space using some form of image similarity definition, however, the framework structure allows the implementer to swap out function approximators to fit specific training set characteristics and define their own image transformations when necessary.

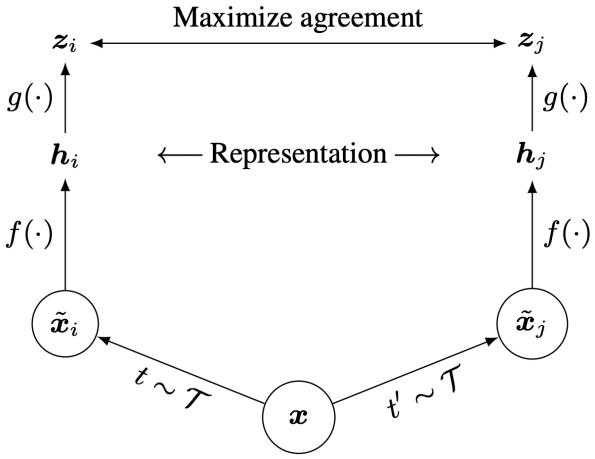


Figure 2.5: A diagram showing the image transformations forward passes and subsequent latent space representations in the SimCLR framework. [Chen *et al.* 2020]

The framework in Figure 2.5 is defined by four major components: a set of data augmentations, a base encoder, a projection head and a contrastive loss function. The data augmentation module is a set of image transformation functions defined as $\mathcal{T}$ that randomly samples two different series of transforms t and t' to be applied to the input image x. This creates two augmented images, $\tilde{\boldsymbol{x}}_i$ and $\tilde{\boldsymbol{x}}_j$ that act as a positive pair. The set $\mathcal{T}$, in Chen *et al.* [2020] SimCLR paper, are random cropping, random resizing, and

using a Gaussian blur with a random blur intensity. They state that through experimentation, random cropping and random resizing are essential to performance [Chen *et al.* 2020].

The base encoder, defined as $f(\cdot)$, is the function approximator that extracts the features of an image and maps them into latent space representations. The paper uses the ResNet50 CNN as the chosen function approximator. The representations $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ are generated using the transformed image pair. The output in this particular instance is the last average pooling layer in the base encoder [Chen *et al.* 2020]. The projection head $g(\cdot)$ accepts the representations $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ and, to obtain $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$, applies the MLP transformation $g(h) = W^{(2)}\sigma(W^{(1)}h)$, where σ is a ReLU function and W^n is the weight matrix at each layer. Whilst applying a projection head is an additional step not mandatory for loss calculation, it has been shown to improve performance substantially and forms a pivotal part of our method discussed in section 4 [Chen *et al.* 2020].

The contrastive loss function used in this paper, known as normalized temperature-scaled cross-entropy loss (NT-Xent), is specifically tailored for self-supervised learning using a temperature hyperparameter that allows the user to control the difference in loss value between similar and dissimilar pairs [Chen *et al.* 2020]. The function is defined as follows:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k\neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{2.16}$$

In the equation 2.16, the authors combine two commonly used functions in contrastive learning. The first is cosine similarity, which in this equation is defined as $\text{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v}/\|\boldsymbol{u}\|\|\boldsymbol{v}\|$, and denotes the Euclidean norm between two vectors. The purpose of cosine similarity is to calculate the angles between two vectors and return a value c where $0 < c \leq 1$ [Chen *et al.* 2020].

The second function is the aforementioned NCE from subsection 2.4. This implementation of the function takes a set of images $\{\tilde{\boldsymbol{x}}_k\}$ as an input where k is the size of the set. The function performs a softmax evaluation with the anchor representation $\boldsymbol{z}_i$, which is the natural exponential of the cosine similarity between $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ over the summed natural exponential of cosine similarities of $\boldsymbol{z}_i$ with every other latent representation $\boldsymbol{z}_k$ derived from the set $\{\tilde{\boldsymbol{x}}_k\}$, with the exception of $\tilde{\boldsymbol{x}}_i$ itself [Chen *et al.* 2020]. A negative logarithmic function is then applied to the result returning $\ell_{i,j}$ where $0 < \ell_{i,j} < 1$. The temperature parameter τ regulates the distance between positive pairs, with lower temperatures resulting in a lower cosine similarity score. This formulation structures the loss function in a way that the weights used by $f(\cdot)$ to generate representations $\boldsymbol{h}_i$ and $\boldsymbol{h}_j$ yield lower loss for the positive image pairs over time when they are compared in latent space, differentiating them from negative pairs [Chen *et al.* 2020].

Algorithm 2 SimCLR's main learning algorithm [Chen *et al.* 2020].

1: **input:** batch size N, constant τ, structure of f, g, $\mathcal{T}$.
2: **for** sampled minibatch $\{\boldsymbol{x}_k\}_{k=1}^{N}$ **do**
3: **for all** $k \in \{1, \ldots, N\}$ **do**
4: draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
5: # the first augmentation
6: $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$
7: $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # representation
8: $\boldsymbol{z}_{2k-1} = g(\boldsymbol{h}_{2k-1})$ # projection
9: # the second augmentation
10: $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$
11: $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # representation
12: $\boldsymbol{z}_{2k} = g(\boldsymbol{h}_{2k})$ # projection
13: **end for**
14: **for all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
15: $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
16: **end for**
17: **define** $\ell(i, j)$ **as** 2.16
18: update networks f and g to minimize $\mathcal{L}$
19: **end for**
20: **return** encoder network $f(\cdot)$, and throw away $g(\cdot)$

To understand the training process of SimCLR, we explain the implementation of the above components using algorithm 2. Line 1 of 2 instantiates our function approximators f and g, the hyperparameters N and τ and transformation function sets $\mathcal{T}$. We then create a batch of samples with size N, which is an important hyperparameter in practice as it determines the number of negative samples we compare against when aggregating loss. Setting this too low can result in poor contrastive comparison, and setting too high can result in overly general latent space representations. The lines 3 to 17 are the execution of the aforementioned SimCLR components. In line 18, $\mathcal{L}$ denotes the aggregated loss overall $\tilde{\boldsymbol{x}}$ pairs in a batch followed by an update to the networks f and g. The algorithm finally returns the network f for downstream tasks.

2.5 Summary

This chapter covers the mandatory DRL, CNN, and contrastive learning theory required to comprehend our proposed method. In summary, DRL methods can be split into value-based methods and policy gradient methods. Policy gradient methods parameterize an RL policy with a function that predicts an optimal action in some state. It then uses trajectories generated through interacting with the environment to update and improve its policy. PPO is our specific policy gradient algorithm of choice and uses an optimization objective that clips a trajectory to avoid volatile policy updates. CNNs have become established as state-of-the-art image-processing neural networks and play a useful role in feature extraction. The residual network is an optimised CNN that uses

a mechanism called a residual block to pass the identity function of a layer's output to layers further down the neural network structure. This consequently allows a deeper implementation of the CNN as the structure will no longer suffer from performance issues such as vanishing gradients.

Contrastive learning is the idea behind differentiating images based on similar features to group like images closer together in latent space and separate images that are dissimilar. These methods have been improved over time with various supplementary transformation techniques. The SimCLR framework is a contrastive learning method that can perform self-supervised image differentiation using a specific model architecture that employs a CNN to extract image features and a temperature-controlled loss function that determines the latent space distance between similar or dissimilar images. In the next chapter, we discuss the body of research that has used a combination of state abstraction modelling with a reinforcement learning algorithm for generalisability and performance improvements.

Chapter 3

Related Work

The idea of state representation learning has been used extensively in past RL research and is generally considered to be a key component for success in RL problems [Eysenbach *et al.* 2023]. Representation learning refers to a set of approaches that can extract the features of a data set and then learn a function to consistently transform each data point from the same distribution as the training set to typically lower-dimensional vectors. These vectors comprise the most important features of a data point, allowing it to be effective in downstream learning tasks. In this section, we discuss recent research that uses representation learning techniques to incorporate information into an encoding for downstream tasks so as to help us understand the current landscape of representation learning in RL. There are two benefits to this: firstly, we give context to the type of problems state representation learning can solve, and secondly, we justify why our particular approach is an important research step to improve this approach in the future.

Representation learning has attempted to incorporate multiple types of information into encodings in the past to better inform an agent of commonly occurring state characteristics. Without doing this, an agent is forced to treat every state it encounters as entirely new and unique. This is both an algorithmic and a computational inefficiency that affects performance and training time, as we are not making full use of mutual information between states [Mazoure *et al.* 2020]. Lesort *et al.* [2018] discuss three significant approaches from their work, which are well represented in contemporary representation learning literature. The approaches are as follows:

1. Observation reconstruction, which entails learning a function that can deconstruct an observation using an encoder and reconstruct it using a decoder.
2. Learning a forward model, which entails learning the next state based on the knowledge of current observations and action choices.
3. An approach that involves using priors to constrain the state space through some form of state evaluation.

In this review, we focus more on the approaches that use priors to constrain the state space and those that train forward models using observations and actions. We focus on

these approaches because they have recently performed well in generalisation tasks in procedurally generated environments, as is evident from the work by Eysenbach *et al.* [2023] and Mazoure *et al.* [2020].

This review discusses two recent papers in which the authors have used the aforementioned approaches. We do this as they are the most recent methods in state representation learning that have shown success in procedurally generated environments. Hence, they are important to consider relative to our own work. Specifically, we assess these approaches relative to our own because they improve the agent's ability to generalise to unseen tasks of similar object layout structures in environments with the same rules of physics and reward dynamics, and this is the problem we aim to solve. The first of these papers is by Eysenbach *et al.* [2023], where the authors set aside the idea of a representation learning component algorithm. Instead, they use contrastive learning algorithms as RL algorithms in their own right, learning from past encounters to favour states closer to the agent's goal state. This paper is thematically similar to the idea of reducing the state space using priors. The second is the research by Mazoure *et al.* [2020], which incorporates spatio-temporal information by using a deep neural network encoder that can estimate the level of mutual information between two observations. This allows an agent to predict what it will encounter in the next state of the environment it is trained in, improving predictive capability and performance. This corresponds to the idea of learning a forward model.

The most common and one of the earlier forms of state representation learning is dimensionality reduction [Lesort *et al.* 2018]. This approach essentially assumes that a state is generated with important, meaningful information coupled with a small degree of noise. From this assumption, one can compress an observation to use only essential state information to positive effect in RL environments. These benefits include reduced learning time and improved accuracy in various environments [Karakovskiy and Togelius 2012]. From this knowledge, more sophisticated methods emerged that employed auto-encoders. These methods emphasised observation reconstruction to improve latent space representation accuracy [Lesort *et al.* 2018]. Examples of applying this to track the object positions from raw pixel inputs are presented by Alvernaz and Togelius [2017] and Finn *et al.* [2016], whereby the authors are able to model the physical characteristics and spacing of a set of objects within an observation. These types of encodings are used for downstream controllers that output actions based on these state representation inputs. For many approaches that use auto-encoders, reducing the reconstruction error proved more vital in learning accurate representations as the tasks became more complex in dynamics and spatial relationships [Lesort *et al.* 2018].

The idea of using priors (previously encountered data points) to constrain a dataset has been widely used in environment modelling for RL [Lesort *et al.* 2018]. Jonschkowski and Brock [2015] shows how this idea can be applied by encoding the object semantics and temporal dynamics of an environment observation, using previously acquired information about world physics, object characteristics, or motions within said environment. Eysenbach *et al.* [2023] employs contrastive learning to perform this state restraint operation by using the contrastive loss functions as an RL objective in and of itself. The authors rework the principal loss objective, as a contrastive loss function

itself can only differentiate between two input samples. The loss objective of the contrastive model outputs states in such a way that they correspond to a value function and, as such, can be used by the agent to determine the quality of a state and allow it to take action accordingly. The reformulated contrastive loss function from their paper is as follows:

$$\mathcal{L}(s, a, s_f^+, s_f^-) \triangleq \log \sigma(c_\theta(s, a, s_f^+)) + \log(1 - \sigma(c_\theta(s, a, s_f^-))) \tag{3.1}$$

Equation 3.1 redefines the contrastive loss function $\mathcal{L}$ to accept state-action pairs s and a (sampled from the replay buffer) instead of just data to be differentiated and couples this with both a negative next-state s_f^- and a positive next-state s_f^+. This is typical of the contrastive learning problem as described in section 2.4. The negative state is taken randomly from the states visited in the past, and the positive pair is taken from a learned state-occupancy function, which improves in accuracy as the agent explores. The critic function, denoted by c_θ, shows us the correlation between the current and future states, where θ is a set of parameters to be learned. In this way, the critic function is analogous to a Q-function, as it determines how close the current state is to a goal state. This ultimately allows the agent to choose actions that will increase the probability of reaching these states and solving the task.

Whilst their approach is effective in helping an agent prioritise reaching a goal state, it does not encode any information that is independent or not directly related to said goal state. This means that the distance to the goal state, using the aforementioned state occupancy function, is the primary measure that allows the contrastive model to differentiate a positive state from a negative state effectively. If deployed in a game that is highly dependant on solving sub-tasks, for example, it may present a difficulty for their approach because the value of a particular state is determined by value and state occupancy functions that are improved through interacting with the environment. This means that the contrastive model's differentiation ability is dependent on the agent improving its state value estimates whilst training online. We address this by developing an approach that trains a contrastive model from pre-generated trajectories that can be collected through any means and can already be effective in contrasting between states when deployed in some environment.

Using forward models for state representation learning can prove to be valuable in enabling an agent to predict changes in its environment temporally [Lesort *et al.* 2018]. Forward models make a projection from an observation to a state to obtain a representation of s_t and then apply a transition function to predict s_{t+1}. The loss is calculated by comparing the estimated next state $\hat{s}_{t+1}$ with the value of the actual next state s_{t+1} [Lesort *et al.* 2018]. The methods used to do this vary from model to model. Karl *et al.* [2016] and Krishnan *et al.* [2015] use KL Divergence to assess the distance between current and next state observations whilst using an autoencoder to minimise the distance. Goroshin *et al.* [2015] emphasises the use of action embeddings in their auto-encodings. When combined with state-action pairing, this approach assists well in mapping to a variable set of next states.

Mazoure *et al.* [2020] use a forward model approach by incorporating an unsupervised representation learning method called Deep InfoMax, by Hjelm *et al.* [2019], into the state-of-the-art Rainbow DQN algorithm. This is one of the most recent examples of success with this approach in procedurally generated RL domains like Procgen. Deep InfoMax is a contrastive learning algorithm designed to maximise information between two images by using dedicated CNNs to learn global and local features and then combine this information into a single representation for use in downstream tasks. Mazoure *et al.* [2020] employ this model to maximize mutual information between state-action pairs and subsequent next-states. As before, the contrastive loss objective needs to be changed to accommodate new input types. The research uses the same base contrastive loss function described in section 2.4.1 to maximise mutual information between two different states, the NCE. Their adaption of the function is as follows:

$$\mathcal{L}_{NCE} := -\mathbb{E}_{p_\pi(s_t,a_t,s_{t+k})}\mathbb{E}_{S^-}\left[\log \frac{\exp(\phi(\Psi(s_t,a_t),\Phi(s_{t+k})))}{\sum_{s'\in S^-\cup\{s_{t+k}\}}\exp(\phi(\Psi(s_t,a_t),\Phi(s')))}\right] \tag{3.2}$$

In the equation 3.2, the value k serves as a temporal offset, meaning it tracks some state that is k timesteps ahead. The positive samples, denoted by s_{t+k}, are drawn from a joint distribution p that is learned using trajectories generated by the agent. This improves as the agent explores the environment. The functions Ψ and Φ are features extracted using a series of CNNs that learn the local and global features of each input. In this application, local features correspond to information like object characteristics, and global features correspond to information like the spatial layout of objects. The variable ϕ is a function that outputs a scalar value from the features generated by Ψ and Φ, enabling compatibility with the exponential function. The states s' are taken from the set S^-, which is created by taking random states encountered by the agent through its exploration. Minimizing this loss function maximises mutual information between states, enabling the generation of a predictive latent space encoding that is spatio-temporally aware. That is, its contrastive model is designed to predict the differences the agent is likely to encounter in future states allowing the agents to take actions that maximise future reward. The authors demonstrate that this improves agent performance over a selection of tasks.

This approach is highly effective in encoding spatial relationships within an environment as it change over time, but this does not make it sensitive to the context in which the agent finds itself. Whilst being able to contrast states using the information on spatial relationships between objects is useful in developing a semantic understanding of the environment, it is important to have some means of contrasting the actual situational value of a state using the actions available within it. This means the relationship must still be explored through continuous environment interaction. This is costly and, to some extent, unnecessary if you use data to train a contrastive model outside of environment interactions.

We address this with ACPO by pre-training our state representation model before exploration begins. We incorporate action values into our contrastive encodings to enable the RL agent to infer typically effective behaviour in specific scenarios. As we see in Chapter 5, this helps it converge to a better performance faster during training and

generalise better when deployed into levels it has not encountered. In this chapter, we have discussed the relevant literature using state representation learning in RL. We have highlighted two primary approaches in this review. The first involves constraining the state space, an approach used by Eysenbach *et al.* [2023], which reconstructs the RL loss function to be a contrastive function in and of itself, favouring states closer in latent space proximity to the goal state. The second approach used by Mazoure *et al.* [2020] entails learning a forward model by employing a component spatiotemporal contrastive model that predicts future states' spatial characteristics based on current ones.

Chapter 4

Contrastive Policy Optimization

This research aims to provide steps towards the development of a method that allows policy gradient-based RL agents to learn by abstracting states using a contrastive learning model that groups states in a context-aware fashion, incorporating a combination of observation image features and the action the agent took within the state that generated this observation. We do this so that a learning agent has context on what behaviours are typically effective in a state and improves on this over the course of training. When coupled with quality training trajectories, this helps the agent infer context whilst training, as the encodings, after training using the aforementioned observation-action trajectories, contain information regarding the attempt of another agent in a similar scenario that allowed it to succeed.

We assess the extent to which using contrastive learning with policy gradient-based methods improves the performance of an agent that would otherwise use raw pixel data and how this results in an improvement to the agent's ability to generalise to unseen environment obstacles that have a similar structure and mechanics to the ones encountered before. This chapter discusses the details of our novel approach, Action Contrastive Policy Optimization (ACPO), and how it performs state representation learning using a contrastive model within the reinforcement learning interaction loop. It explains how we augment the SimCLR algorithm to accommodate action encodings from the environment and apply them within a policy gradient algorithm (PPO) to improve performance and generalisability.

In Section 4.1, we begin by reiterating the problem we are trying to solve: grouping states with common characteristics in latent space to enable the agent to generalise to unseen environment obstacles. This helps us justify our proposal of a new method as well as the specific design choices we make when implementing it. In Section 4.2, we discuss how we construct our novel algorithm (ACPO) to illustrate the conceptual underpinnings of our contribution and further clarify how it is different from what has already been researched. We explain how we incorporate action values into latent space state representations using a contrastive learning model (SimCLR). The final Section 4.3 details the changes we make to the reinforcement learning interaction loop defined in 2.1 to accommodate contrastive learning and explains, through the use of pseudo-code, how we integrate an action-augmented SimCLR into PPO. This section is

the most detailed account of how our algorithm works. It clearly states how the original algorithms must change to be deployed in some environment when implemented. We also motivate why we choose each algorithm for contrastive learning and policy gradients, as this helps understand the algorithm's overall performance relative to its baselines and provides better evidence to support our hypotheses detailed in Section 5.1.

4.1 Problem Outline

We discuss the contrastive problem in Section 2.4 by using Figure 2.4 to illustrate a human agent's rudimentary ability to differentiate between image themes without the use of labels (i.e. a person does not need to know that the object that they are matching is referred to as a cat), but merely through the visual and conceptual likeness of the objects to be matched. We evaluate the effectiveness of such a contrastive learning model as a state representation learning model applied to an RL domain. This specific instance of an RL domain, Procgen (see Section 5.2), is governed by consistent mechanics (the rules of physics and transition probabilities) and can produce infinitely many unique environment episode instances that are situationally and structurally (the layout of interactive objects within the level) similar, but never identical. This helps to avoid performing well through state memorization. Figure 4.1 a domain instance where an agent is required to group a game scenario by the action it is necessary to take:

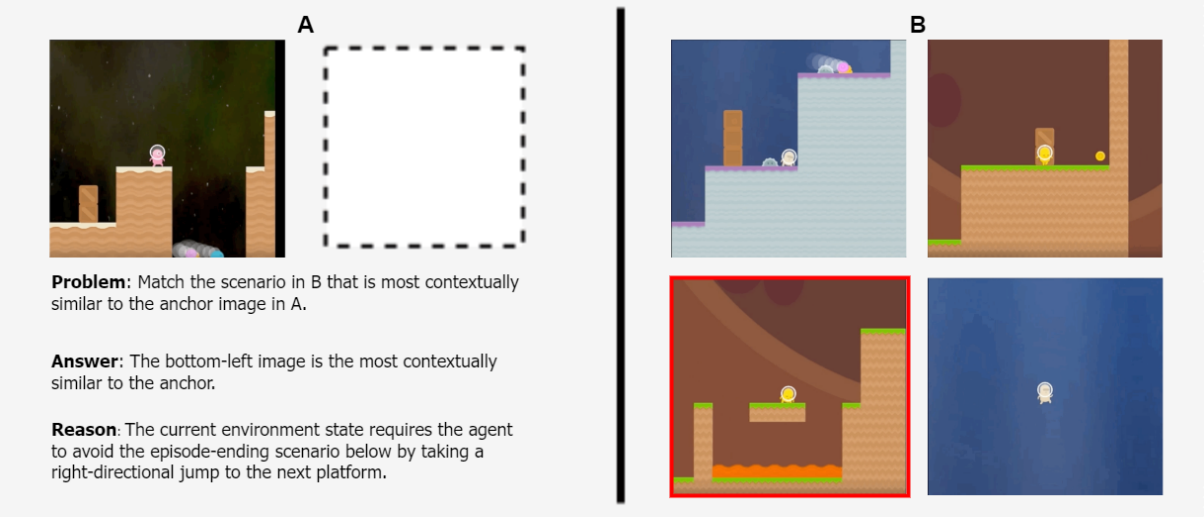


Figure 4.1: A contrastive problem requiring an agent to differentiate situationally between each state observation in the game Coinrun.

We see in Figure 4.1 a collection of four scenarios from the game Coinrun, a platformer arcade game from the Procgen test suite. In summary, the goal of Coinrun is to guide an agent through obstacles and enemy agents to reach the goal state (i.e. the state in which an agent makes contact with a coin), which sits at the far right of any given level. In this case, the anchor image requires an agent to take a right jump and land on a platform

to avoid the episode-ending scenario beneath it and progress in the level. Amongst the four scenario options available, the bottom-left scenario is the circumstance that most closely resembles the anchor image irrespective of the sprites and colour palette the game is using, and so, the goal of our contrastive model is to group these images in latent space, whilst moving the other images apart. The purpose of this is to imbue our agent with the ability to group visually and contextually similar scenarios together and take similar actions in situations that require the same behaviour. We show that this improves learning and enables the agent to reach an optimal performance faster and with fewer training steps.

4.2 Action Contrastive Policy Optimization (ACPO)

To solve the problem of learning using both observation information and agent behaviour, we combine contrastive learning and policy gradient algorithms to create our algorithm, Action Contrastive Policy Optimization (ACPO). ACPO uses pre-generated environment image observations combined with the actions taken by some sub-optimal agent in the respective state to train a contrastive model to abstract the observations into latent space representations. This trained contrastive model is then used during RL training to perform state representation encoding on observations encountered by an agent in a way that differentiates its current state based on the context of the situation.

In assessing the viability of the specific SimCLR framework implementation used by Chen *et al.* [2020] as a contrastive model for ACPO, we examined its contrastive ability in other experiments where it is tasked with differentiating visually similar images with subtle semantic differences that change the meaning of the image (i.e. different types of cat ears can signify a different breed) [Chen *et al.* 2020]. The resulting performance using exclusively image observations from any dataset, whilst having shown promise, indicated that the model struggled to infer subtle visual differences across data points. Instead, it simply groups them by their general visual likeness, resulting in minimal semantic differentiation for more specific types of data.

It is for this reason we rationalise that including action dynamics in an RL, context can help the contrastive state representation model better differentiate states that mean something different. To help the agent infer what we refer to as observational context, we expanded SimCLR to train using both image observations and the action that the agent takes in a particular state, allowing it to differentiate states using contextual information as well as the observation itself. The inclusion of action information into SimCLR shows a marked increase in RL performance when combined with PPO and clearer latent space differentiation when analysed (i.e., the cosine of the angle between their inner product vector space decreases). We demonstrate this in the experiment in Section 5.5.1, showing how similar structural layouts between two image observations can be differentiated by adding the action the agent took to separate them by their cosine similarity score.

To incorporate action values during SimCLR training, we use one-hot encodings of the action taken by the agent. This vector representation of the action is appended to the output of the base encoder for further latent space encoding. Adding extra dimensions

to the latent space output of the CNN encoder separates the observations in latent space to a high enough degree that they are considered sufficiently distinct when evaluated by the cosine similarity function, which is the core loss calculation of the SimCLR instantiation used by Chen *et al.* [2020], detailed in Section 2.4.1. This expanded representation is then passed to an additional dedicated function approximator to provide a non-linear transformation of the context-expanded base encoder representation for better differentiation during loss evaluation.

The resulting representation pairs are successfully distanced from observations similar to each other in visual likeness but different in terms of the required agent behaviour to reach its goal. These observations, which were formerly grouped in latent space using the original SimCLR approach, are instead given distant inner product latent space encodings by the base CNN encoder. These encodings are used for downstream reinforcement learning with PPO and improve the agent's performance through the now context-aware SimCLR contrastive state representation model.

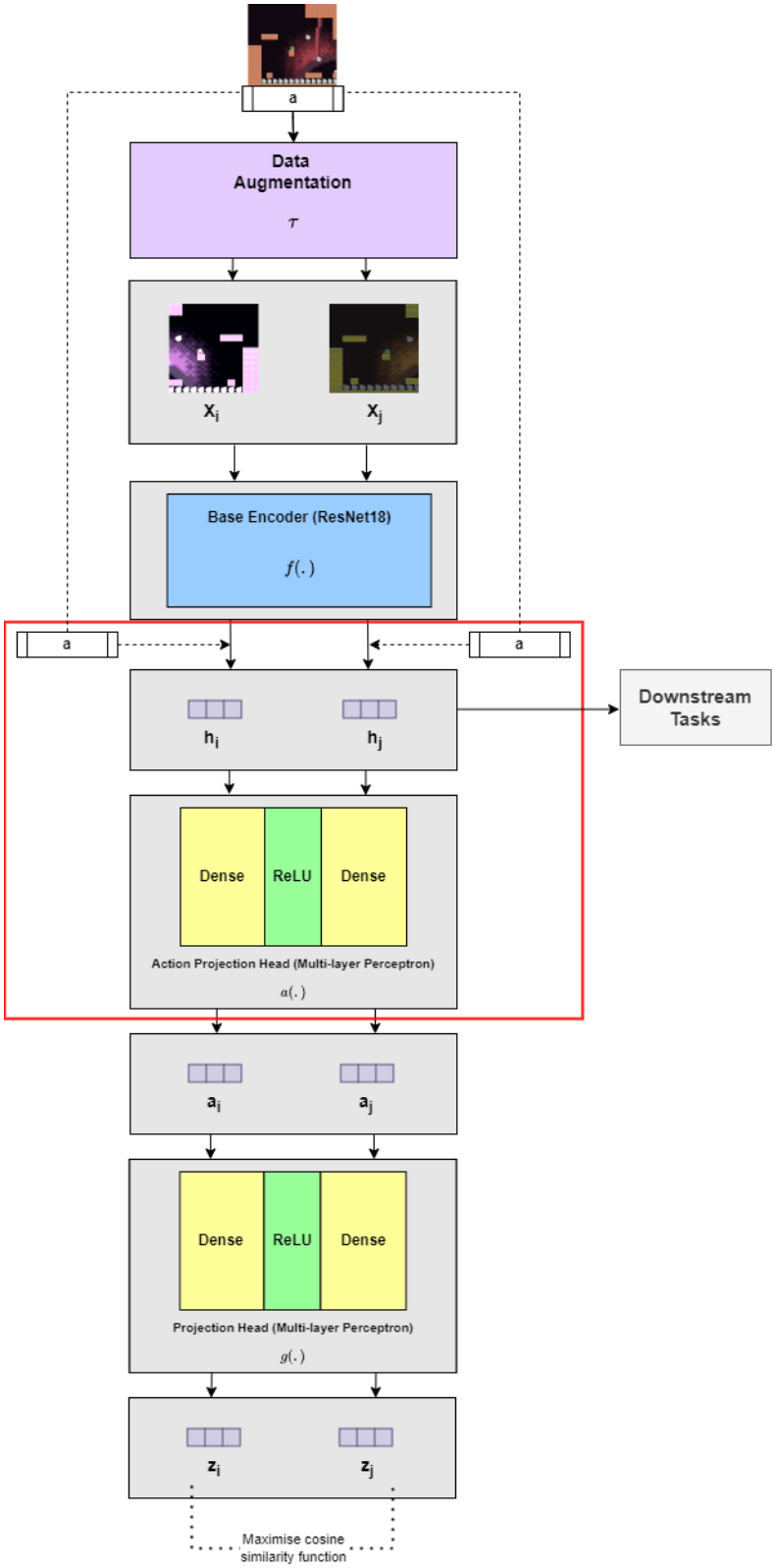


Figure 4.2: An expanded SimCLR that incorporates action-values. This serves as ACPO's contrastive state representation model, with our change to the original SimCLR network structure boxed in red.

Figure 4.2 illustrates an expanded SimCLR that allows us to incorporate the action a that an agent took in some state s into its latent space representations, using the observation for that state. It transforms the action into a one-hot encoding and then concatenates that to the vector representation produced by the base encoder $f(\cdot)$. The function $f(\cdot)$ receives an observation with two sets of random transformations taken from a transform superset τ. The expanded vector representation outputs h'_i and h'_j are then passed into a dedicated projection head function, referred to as the action projection head, and denoted by $a(\cdot)$. This produces the latent representations a_i and a_j using a non-linear transformation, the values of which will be different depending on the action taken for each transformed, context-augmented observation that passes through the model. This step is annotated with a red box and forms the core of our contribution. It is essentially how we incorporate the action embeddings into the contrastive learning model. We finally pass this to the next projection head $g(\cdot)$ to include non-linear transformation for similarity differentiation, ensuring more fine-tuned learning during backpropagation. This produces the vectors z_i and z_j for maximization using the cosine similarity function.

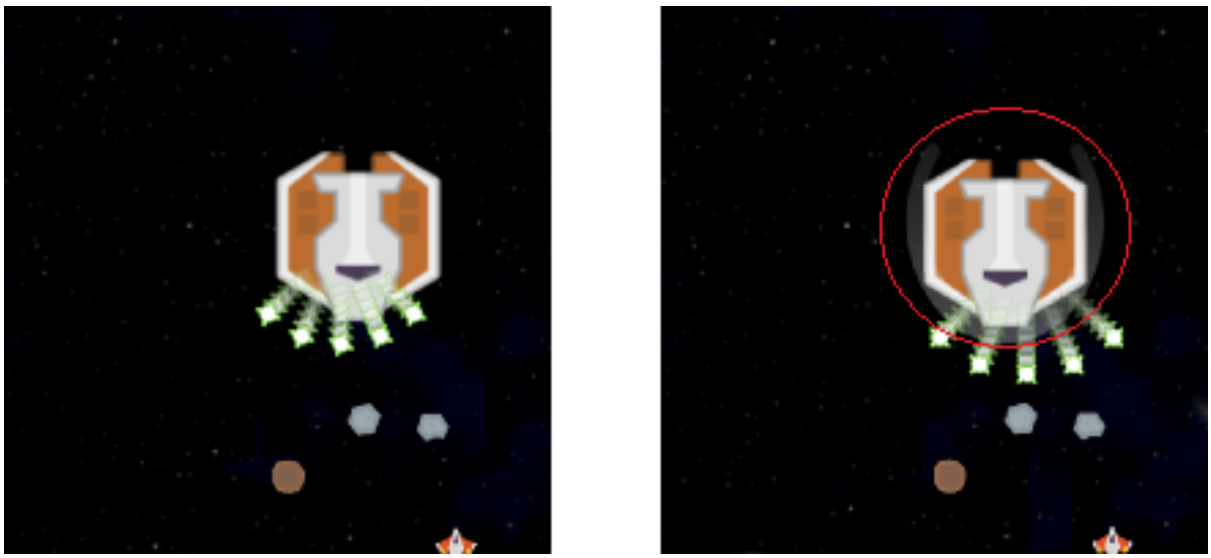


Figure 4.3: An observation comparison from the game Bossfight illustrating the difference in situational context for near identical visual representations. The red circle highlights an active shield for the enemy. This makes it disadvantageous for the RL agent to perform an attack action in this scenario.

To illustrate the benefit of a contrastive model that is aware of situational context, we use Figure 4.3 as an example of an observation that is nearly visually and structurally identical in terms of the game mechanics of the Procgen game Bossfight. The objective in the game is for the agent (in the bottom right) to destroy the larger ship (in the upper right) using projectiles whilst avoiding the incoming projectiles by using cover and manoeuvrability. The critical difference between these two images is the subtly visible blur around the larger ship (the boss) in the second image, highlighted by the red circle. This simple mechanic, despite being so visually discreet (yet still detectable by the CNN encoder), changes the entire dynamic of the game state, as the agent can no

longer damage the enemy boss to achieve victory, thereby obtaining its reward. In this situation, it must instead opt for defensive manoeuvres to avoid the enemy's projectiles, whereas, in the left situation, it can risk offensive actions due to being able to damage the boss. In this way, a subtle visual difference implies significantly different optimal action selection.

The augmented SimCLR contrastive model of ACPO assists the policy gradient method in solving this problem through context-assisted contrastive learning. Without this, the images would be grouped very closely in latent space during state abstraction despite their distinct semantics. During training, the agent instead receives a distinct encoding for each state as the contrastive model training has already learned the difference in situational context using the action embedding.

4.3 Contrastive Reinforcement Learning

In this section, we begin by illustrating and discussing the interaction loop that forms the basis of our approach's learning cycle. We then justify the selection of two important concepts in our research: first, we discuss why contrastive learning is useful for state abstraction in an RL setting, and second, we discuss why we chose policy gradient algorithms instead of value-based methods as our RL agent's learning method.

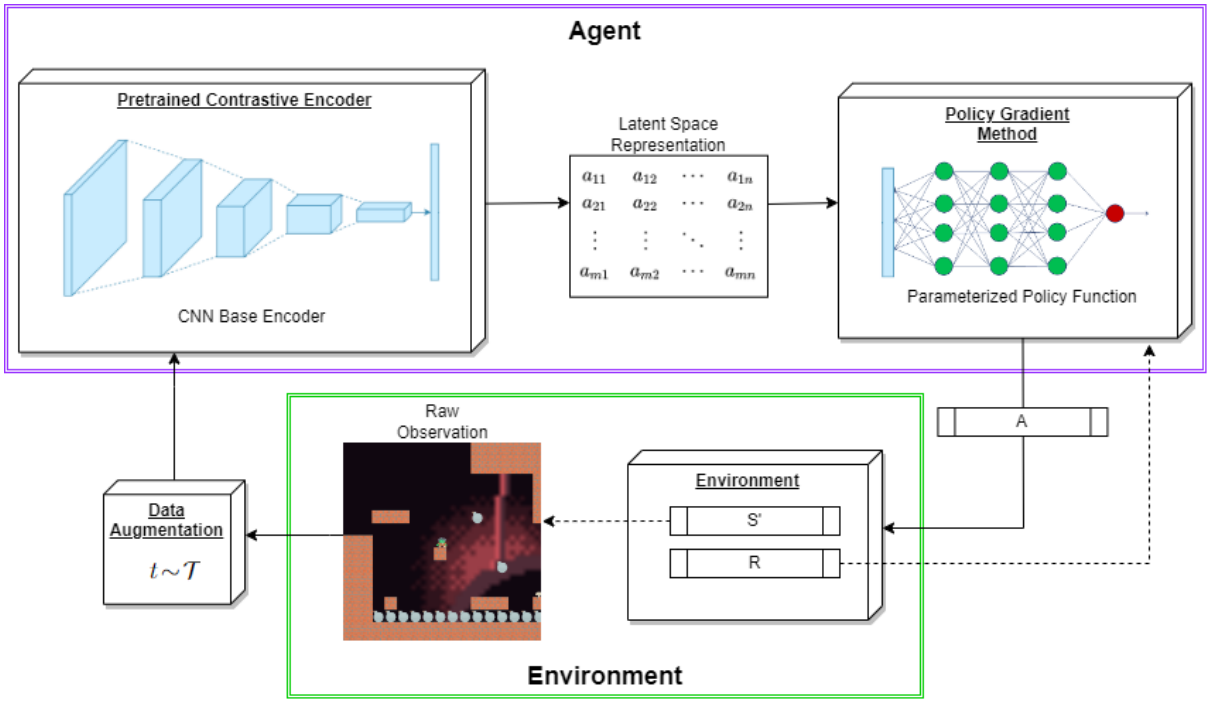


Figure 4.4: Our amendment for incorporating contrastive learning into a policy gradient algorithm within the RL interaction loop detailed in Figure 2.1.

Figure 4.4 illustrates the change we make to the RL iterative learning sequence illustrated in Figure 2.1. This framework is not exclusive to performing with the algorithms used in our research but instead can be used with any contrastive learning model and

policy gradient-based algorithm. The agent's training loop begins with it using a parameterised policy function to take action in the environment. This successively generates a reward signal and a new environment state from which we extract an image observation. This observation is then encoded by a contrastive encode, defined as function f in Figure 4.2, that has been pre-trained to map a given environment's image observation to a latent space to produce an abstracted, summarised representation of the state observation to be used in training the agent. To perform the policy update, we pass the abstracted state's latent vector representation as part of the state-reward tuple the agent uses as per the update step in the interaction loop in Figure 2.1.

We have discussed in Section 2.1, that an RL agent requires a means to evaluate its policy when interacting with the environment it is deployed into. To understand our choice to use a policy gradient learning algorithm as our approach, we need to compare the benefits of policy gradient methods relative to value-based methods. Computational efficiency in policy gradient methods is substantial when used with state abstraction as it only requires the user to encode a single state per time step [Bennett *et al.* 2021]. That is, the input to a policy function in policy gradient methods is simply the abstracted state. In contrast, the procedure of value-based methods with state abstraction requires encoding each possible state-action pair relative to the next state encountered to determine the state-action value. This creates a computation overhead that increases in complexity the larger the action space becomes [Bennett *et al.* 2021].

Value-based approaches also create difficulties with explainability when analysing latent space encodings of states relative to the action taken for that particular state. The contrastive problem we have described requires us to group states based on the semantics of the game dynamics and subsequently encourage the agent to take the same action in the same semantic context. If we opt for value-based methods, the agent will assign a value to a state representation based on its value over an action and next-state distribution. This creates difficulty in assessing whether the agent favours a particular action because it recognises that it is in a similar semantic context or because it perceives the value of the state as higher due to the experience it received.

The option to use PPO as the agent's specific policy gradient algorithm was a purely evaluative justification. Of all the policy gradient algorithms we tested in our ablation studies, PPO was the best-performing algorithm overall for this particular domain. This is further supported by Cobbe *et al.* [2019] in their evaluation of Procgen as a suitable testing environment for RL. They employ PPO as their choice of policy gradient algorithm to evaluate the behaviour of policy gradient methods in the Procgen domain.

4.4 Algorithm Specification

We detail the algorithmic changes to the original SimCLR model instantiation, along with the addition of the observation encoding function to PPO's training steps in the following two algorithm definitions. This section serves to detail the implementation details of our method by demonstrating how the algorithm is trained following the obtainment of data. Algorithm 3, detailed below, shows the exact steps to be taken when incorporating action embeddings into the SimCLR training iterations, as well as

how we prepare the data before training. Algorithm 4 shows how the context-aware contrastive model trained in Algorithm 3 is integrated into PPO. This combination of the context-aware SimCLR model with PPO gives us the complete novel algorithm ACPO.

Algorithm 3 Action augmented SimCLR for context-aware contrastive learning.

Train SimCLR using pre-generated trajectories:

1: **Input:** batch size N, constant τ, structures for function f, g, a (as action projection head), $\mathcal{T}$
2: **for** sampled minibatch $\{\boldsymbol{x}_k, \boldsymbol{a}_k\}_{k=1}^{N}$ **do**
3: **for all** $k \in \{1, \ldots, N\}$ **do**
4: draw two augmentation functions $t \sim \mathcal{T}$, $t' \sim \mathcal{T}$
5: # the first augmentation
6: $\tilde{\boldsymbol{x}}_{2k-1} = t(\boldsymbol{x}_k)$ # transform observation
7: $\boldsymbol{h}_{2k-1} = f(\tilde{\boldsymbol{x}}_{2k-1})$ # Resnet18 latent representation
8: $\boldsymbol{a}_{2k-1} = a(\boldsymbol{h}_{2k-1} + \boldsymbol{a}_k)$ # action projection
9: $\boldsymbol{z}_{2k-1} = g(\boldsymbol{a}_{2k-1})$ # similarity projection
10: # the second augmentation
11: $\tilde{\boldsymbol{x}}_{2k} = t'(\boldsymbol{x}_k)$ # transform observation
12: $\boldsymbol{h}_{2k} = f(\tilde{\boldsymbol{x}}_{2k})$ # Resnet18 latent representation
13: $\boldsymbol{a}_{2k} = a(\boldsymbol{h}_{2k} + \boldsymbol{a}_k)$ # action projection
14: $\boldsymbol{z}_{2k} = g(\boldsymbol{a}_{2k})$ # similarity projection
15: **end for**
16: **for all** $i \in \{1, \ldots, 2N\}$ and $j \in \{1, \ldots, 2N\}$ **do**
17: $s_{i,j} = \boldsymbol{z}_i^\top \boldsymbol{z}_j / (\|\boldsymbol{z}_i\| \|\boldsymbol{z}_j\|)$ # pairwise similarity
18: **end for**
19: **define** $\ell(i, j)$ **as** equation 2.16
20: update networks f, g and a to minimize $\mathcal{L}$
21: **end for**
22: **return** encoder network f

In Algorithm 3, related to Figure 4.2, there are three significant changes to the original SimCLR algorithm by Chen *et al.* [2020]. First, in line *1* (highlighted in pink), we begin by adding the additional function approximator a to the initialisation step. The function is a multi-layer perceptron and serves as the aforementioned action projection head. The second change, in line *2* (highlighted in orange), is the addition of a one-hot encoded action value vector a_k to the mini-batch tuples used for each training sample k. This allows the model to embed context into an environment observation x_k when training the base encoder f, which is eventually used for state abstraction in PPO. Finally, in line *8* and *13* (highlighted in green), we append the one-hot encoded vector a_k for some training sample k to the latent space output of the Resnet18 CNN function f, denoted by h_{2k-1} and h_{2k}. The representations a_{2k-1} and a_{2k} are used in the similarity projection function g in place of h_{2k-1} and h_{2k} to ensure that the pairwise similarity step adequately differentiates the positive pairs from the negative pairs within the mini-batch. The algorithm ends after a predefined number of epochs and returns the encoding function f.

Algorithm 4 Action contrastive policy optimization (ACPO) algorithm using PPO.

Train PPO agent using observation encoder f:

1: **Input:** Initialise policy parameters θ_0, value function parameters ϕ_0, and load contrastive encoder f from Algorithm 3
2: **for** $k = 0, 1, 2$ ... **do**
3: Collect and encode a set of trajectories $\mathcal{D}_k = \{f(\tau_i)\}$, by running policy $\pi_k = \pi(\theta_k)$ in the environment.
4: Compute rewards-to-go $\hat{R}_t$.
5: Compute advantage estimates $\hat{A}_t$ (using L^{VF}) based on ϕ_t of $V_{\phi k}$
6: Update the policy by maximising the PPO-Clip surrogate objective defined in equation 2.11 across all trajectories:
$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} L^{CLIP}$ via stochastic gradient ascent
7: Fit value function by regression on mean-squared error defined in function 2.13:
$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} L^{VF}$ via any gradient descent algorithm
8: **end for**

Algorithm 4 defines a complete ACPO, related to Figure 4.4. There are two significant changes to how we train PPO following the successful training of the action-augmented SimCLR model to achieve simple contrastive policy optimization. In line *1*, highlighted in purple, we load the contrastive encoder f, which has been trained using Algorithm 3. This serves as the state abstraction function used by the agent to produce encodings for training the policy π. The second change in line *3* is highlighted in blue. This is the step that collects an environment observation set τ_i and uses the function f to produce a latent space representation of that set, denoted as $\mathcal{D}_k$. These encoded trajectories are now used to train the agent.

4.5 Summary

In this section, we have discussed the core contribution of our research. We begin by reiterating the problem we aim to solve in Section 4.1. In this section, we discuss the contrastive problem we are trying to solve, including an example scenario in Figure 4.1, showing that we are trying to group scenarios by semantic and contextual similarity. This is critical in understanding how our proposed method is justified in its conception. We then introduce the specific instance of our algorithm, Action Contrastive Policy Optimization (ACPO). This algorithm incorporates an action-embedding into the SimCLR contrastive learning framework using our own selection of neural networks to build a structure similar to the one used by Chen *et al.* [2020]. The details of this are outlined in Figure 4.2. The resulting model accepts environment observations along with the agent's corresponding choice of action and encodes them into latent space using a series of neural network structures.

We follow this by showing the way in which we change the RL interaction loop to accommodate state abstraction using contrastive learning in Section 4.3. This is pivotal to learning as it must integrate into the agent-environment interaction loop to be a viable form of RL. We also justify our choices of contrastive and policy gradient algorithms as this combination allows for a unique inquiry into RL that to our knowledge at the time of this writing, has not been researched in any capacity. Finally, in Section 4.4, we detail the pseudo-code for the implementation of ACPO, showing the means through which we embed actions into a SimCLR model structure in Algorithm 3, and the means through which include the action embedded SimCLR model into the PPO in Algorithm 4 to form the complete ACPO. In Chapter 5, we present the experience that forms the body of evidence for the success of our approach.

Chapter 5

Experiments and Results

This chapter details the key outcomes of the research and effectively forms the body of evidence that we need to determine the success of our approach. We start by formally stating our research hypothesis in Section 5.1, where we detail our inquiry about ACPO and its performance and generalisation as a contrastive state representation RL algorithm. We also specifically state what counts as a success in this regard, as this helps us determine our algorithm's success or failure in the following experiments. We follow on to detail our research domain in Section 5.2 and experiment design in Section 5.3. This details how and in what environment we set up our experiments and evaluate our agents. This is followed by reporting the performance of our agent in Section 5.4. The performance section serves as a presentation of the overall success of our method. Here, we show that our method is a superior approach to using standard RL methods in training and test environments. Finally, in Section 5.5, we delve into a discussion of the performance of ACPO. For this section, we detail a series of ancillary experiments performed to determine why agents perform well in the environments that they do and what factors influence an agent's ability to learn good contrastive representations.

5.1 Research hypothesis and success criteria

The goal of the approach discussed in Section 4.2 is to assess whether using contrastive learning with policy gradient algorithms improves the performance of an agent that would otherwise use raw data (i.e. data with no latent space contrastive encoding) and whether this augmentation results in an improvement to the agent's ability to generalise to unseen environment level structures that need to be solved with the same mechanics (the rules of physics and transition probabilities). To conduct a formal experiment, we state our research question, the accompanying hypotheses, and the success criteria used for evaluation. The research questions and accompanying hypotheses are as follows:

Research Question: Does ACPO improve the generalisability of a policy gradient-based reinforcement learning agent that would otherwise use raw pixel data for similar tasks without a drop in performance?

- Question 1: *Does ACPO match or improve on the training performance of policy gradient algorithms that would otherwise use raw data?*
  - H_0: Using ACPO does not improve the training performance of the agent.
  - H_1: Using ACPO improves the training performance of the agent.
- Question 2: *Does ACPO match or improve on the generalisability of policy gradient algorithms that would otherwise use raw data?*
  - H_0: Using ACPO does not improve the general performance of the agent.
  - H_1: Using ACPO improves the general performance of the agent.

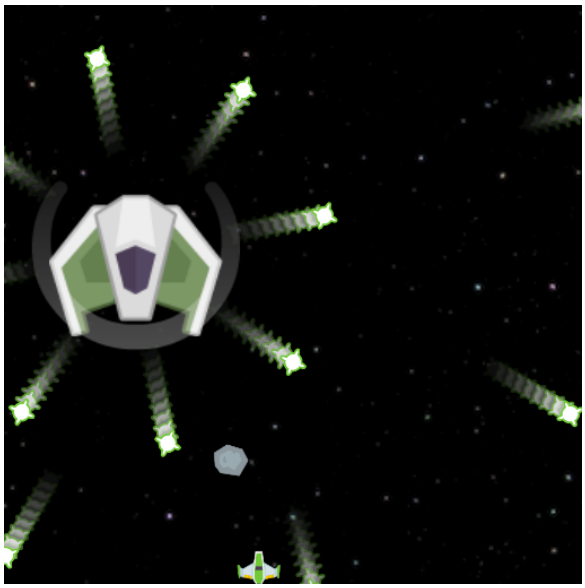
The hypothesis describes what we are investigating regarding the performance of ACPO. To measure this behaviour specifically and evaluate our approach, we need clear definitions of success criteria for each research question to determine which of the hypotheses we accept for each. The criteria are as follows:

- The PPO agent using ACPO must match or improve on the converged reward value in comparison to the standard PPO benchmark agents that do not use contrastive learning. The results must be aggregated over multiple levels and multiple seeds. This is a measure of performance.
- The PPO agent using ACPO must match or improve on the total mean-aggregated reward obtained by the agent in levels it has not encountered before in a particular environment. The details of these environments are discussed in Section 5.2. This is a measure of generalisability.

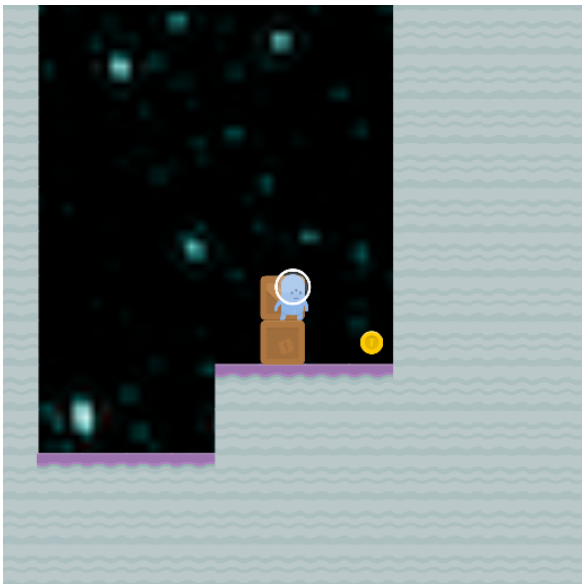
5.2 Research domain

We test our algorithm in the Procgen virtual environment. Virtual environments are simulated domains which can test an agent with a complex array of problems with varying degrees of difficulty [Yen *et al.* 2002]. They include high levels of stochasticity and typically require non-linear approaches for optimal outcomes [Machado *et al.* 2018]. Our justification for choosing this domain is that virtual environments are complex, low-cost, low-risk testing frameworks comparable to the complexity levels of real-world problems [Pan *et al.* 2017]. These abstract similarities can justify confidence in an algorithm for future exposure to real-world scenarios, assuming the virtual rendition has the same essential characteristics or at least similar complexities [Pan *et al.* 2017].

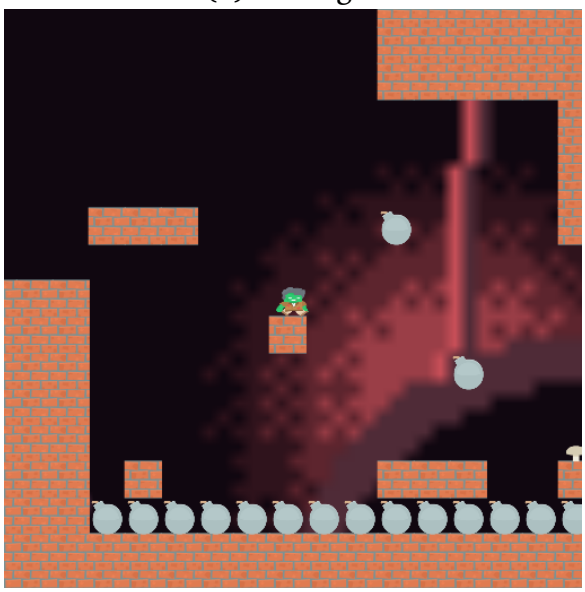
The Procgen benchmark developed by Cobbe *et al.* [2019] is a virtual environment selected for testing and evaluating the performance of the algorithm developed in this research. Procgen is a research platform designed to evaluate an agent's ability to generalise knowledge it gains in one level of the environment to another level with similar concepts and mechanics but a different physical structure [Cobbe *et al.* 2019]. It accomplishes this by generating new-level structures, obstacles and antagonists with a pre-selected seed. The games are designed so that the levels are always different and almost always solvable.



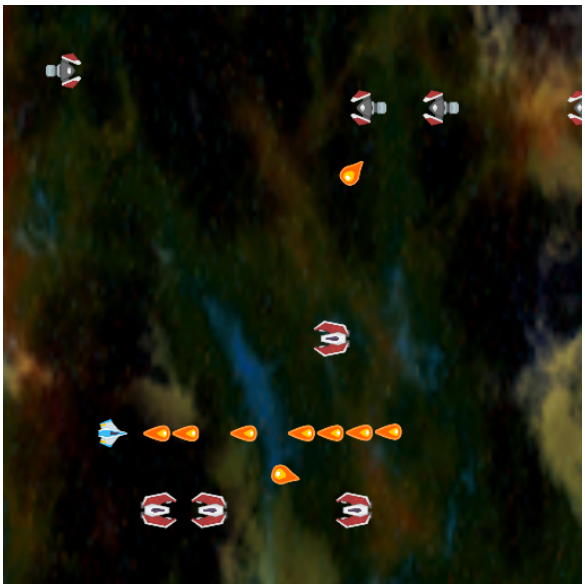
(a) Bossfight



(b) Coinrun



(c) Ninja



(d) Starpilot

Figure 5.1: The Procgen environment featuring multiple games with procedurally generated levels. [Cobbe *et al.* 2019]

The Procgen game suite shown in Figure 5.1 was selected due to the characteristics of the game mechanics and its suitability as a test bed for the problem we aim to solve. The environments involve changing and complex challenges through dynamic level design and require the agent to perform well-timed actions with a high degree of precision to complete a level with the optimal score [Cobbe *et al.* 2019]. This allows us to construct reward functions and ensure the agent has clear goals to achieve. The environments also have an acceptable level of stochasticity that presents difficulties for hard-coded approaches, a sufficiently high level of dimensional complexity that standalone RL algorithms cannot solve for, and an endpoint for each level that can serve as a suitable terminal state [Cobbe *et al.* 2019]. All of these games were designed for human agents who can clearly understand relationships between objects, perform acts of skill to traverse obstacles and plan medium to long-term strategies to obtain the highest possible scores in each level. The description of the specific set of games used in this research is described in Section 5.3.1.

5.3 Experiment design

To investigate our hypotheses from Section 5.1, we conduct a series of evaluations to test the training performance and generalisability of ACPO. This section outlines the details of the experimental procedure used to evaluate ACPO. We begin by describing the task characteristics and goals for each of our chosen games in Section 5.3.1, so as to be clear in describing what the agent must try to achieve. We continue on to describe how we implement ACPO and what baselines we use as a performance comparison in Section 5.3.2, by making specific references to neural network structures and data transformations. We conclude this section by detailing and justifying our experiment specifications by discussing the number of game levels we use for each training run, how we assess the ability of the agent to generalise to new levels thereafter, and the number of timesteps and episodes used for each of those evaluations respectively.

5.3.1 Procgen environments

After evaluating the concepts, mechanics, and goals of all sixteen Procgen games, we assess that many can be grouped into task characteristics that are similar to those of other games within the suite. We choose two games from two distinct game types for our evaluation creating a total of four evaluation environments. The first type is a platformer, which requires the agents to navigate a series of platforms and accompanying obstacles to reach an object at the end of the level. They do this by using a discrete number of actions and must obey the physics within the respective games. The second type of game is a shooter, which requires an agent to launch projectiles at enemy agents to score points. These also employ a discrete action space and are not bound to any principles of gravity, hence the agents move entirely on their own.

To properly understand the problem the agents are attempting to solve in each game environment, we describe the objective of the game, a description of the challenge the agents must solve, what they receive rewards for and what elements in the game

are controlled with procedural generation. The description of the games and their objectives, taken from Cobbe *et al.* [2019] are as follows:

- **Bossfight** Figure 5.1(a): The player controls a starship and must defeat a shielded enemy boss starship, which randomly selects a projectile attack from its arsenal when attempting to destroy the player. The agent must evade the incoming projectiles or be destroyed. The agent can make use of randomly placed rocks for cover. After a set number of seconds, the boss's starship shield is disabled, making it vulnerable to damage from player projectiles. A reward will be given once the agent damages the boss enough times, and the boss's shield will be reactivated. After damaging the boss enough times, it will be destroyed, and the episode will end. A random seed controls the boss's health points as well as the number and placement of rocks in the level. It also controls the attack pattern the boss starship uses against the agent.

- **Coinrun** Figure 5.1(b): This is a platformer game where the goal is to collect a coin placed at the far right of a level. The agent begins on the left and must evade stationary obstacles, enemies that move left and right, and drops that lead to the end of the episode. Procedural generation in this game controls the number of platform areas along with the type of platform. It also controls the position and types of crates and obstacles, as well as the positions and movement of enemies.

- **Ninja** Figure 5.1(c): This is a platformer game where the agent must jump across ledges whilst navigating around explosive objects, at which the agent can launch projectiles from various angles to clear. The agent can also charge a jump over several timesteps to jump higher. The goal is to collect a mushroom at the far right of the level, terminating the episode and rewarding the agent. The agent always begins on the far left. Procedural generation determines the level's platform layout and the positions of the explosives.

- **Starpilot** Figure 5.1(d): Starpilot is a side-scrolling shooter game requiring an agent to consistently evade attacks while attacking enemies. The agent must move from left to right to reach a terminal state at the far right of the level. Obstacles in the game include enemy agents that move at variable speeds, turrets targeting the agent with projectiles, there are clouds obscuring agent observations, and impassable rocks the agent must learn to evade. The agent is rewarded for destroying enemy agents and turrets and reaching the end of the level. Procedural generation controls the spawning of all enemies and obstacles.

Coinrun and Ninja have similar game mechanics except for the projectile actions and stationary obstacles (bombs) in Ninja. Bossfight and Starpilot have similar game mechanics, with the differences being the types of enemy agents they face, the size of the map, and the orientation of the game, with Bossfight being set in an arena and Starpilot being a side-scroller. Starpilot is also the only game where the agent is forced to move forward at all times from left to right. All games are played by the agent in easy mode. We opted for this over hard mode due to the higher computing requirements of training agents to perform at higher difficulty levels.

5.3.2 ACPO and baseline comparisons

As discussed in Section 4.2, our implementation of ACPO uses a combination of the augmented SimCLR and PPO algorithms, defined in Algorithm 3 and Algorithm 4, respectively. The function f in both algorithms is implemented as a Resnet18 CNN, and the functions a and g are represented as multi-layer perceptrons with equivalent structures. We choose Resnet18 to maintain consistency with the original CNN choice used in the SimCLR implementation by Chen *et al.* [2020], except we opt for the smaller version of Resnet18 instead of Resnet50 as our images are considerably less intricate than those used by Chen *et al.* [2020]. We use multi-layer perceptrons as our policy function for PPO as there is no need to use another CNN when combined with a contrastive model that already encodes observation using a specially trained contrastive model that is composed with a CNN (base encoder f) in its structure. The data augmentation set $\mathcal{T}$ contains a horizontal flip function, a random crop function, a colour-jitter function, and a grayscale function. The benefits of these transformations are discussed in Section 2.4.1. All observations collected by the agent are transformed into grayscale images before use and after the random application of all other functions in $\mathcal{T}$. Our implementation's hyperparameters and activation functions are described in Appendix B. Our ACPO implementation uses an MLP policy for both the policy and value functions.

To construct a meaningful experimental baseline, we also implement a second contrastive RL agent that does not use action embeddings but instead uses just the image-based observation from the state in which the agent finds itself. This experiment serves to assess the extent to which embedding action values into latent space during contrastive learning influences the agent's performance. The structure of this agent is the same ACPO except for the function a, as there were no action values to justify a second projection function. This algorithm is hence referred to as observation contrastive PPO or PPO(OC). We show in Figure 5.4 in Section 5.5 that increasing the parameters of the contrastive learning SimCLR model of PPO(OC) makes no substantial difference in its performance. PPO(OC) serves as the contrastive baseline experiment to ACPO.

To ensure that we have fair comparisons for the ACPO algorithm when evaluating its performance in general, we established baseline experiments that test the constituent algorithms that make up our ACPO implementation. Isolating each algorithm to create comparison performance metrics helps us understand the behaviour of standard PPO implementations in Procgen relative to the contrastive approach of ACPO along with its contrastive baseline PPO(OC). The baseline experiments are as follows:

1. PPO using an MLP policy network and value function identical in structure to the PPO algorithm used in ACPO and PPO(OC), except for using raw observations and a larger input layer. This is hence referred to as PPO(MLP).

    - This experiment informs whether we were obtaining value from using encoded observations from a contrastive model.

2. PPO using a Resnet18 CNN actor-critic network structure with raw, grayscale images used as the input during training. This is hence referred to as PPO(Resnet18).

- This experiment serves to determine whether using Resnet18 CNN as the actor-critic network structure with raw image observations would perform better than using Resnet18 as a contrastive encoder.
- We want to determine how well Resnet18 would respond to images from the Procgen environments to see if this would be a good choice of an encoder for the SimCLR model

3. PPO using a Resnet18 CNN actor-critic network structure with randomly augmented grayscale images as the input during training. This is hence referred to as PPO(Resnet18 RAD).
    - This experiment served to determine to what degree augmented observations influence performance
4. PPO using a Nature CNN detailed in the paper by Mnih *et al.* [2013], as an actor-critic network structure with grayscale image input. This is hence referred to as PPO(Nature).
    - This experiment served primarily to test for any changes in performance when using an alternative CNN.
    - We want to assess the performance of CNNs in general when used with PPO

For ACPO and all 5 baselines, we train an agent across 500 levels for 10^6 timesteps on each level. This is evaluated across five seeds, with each set of 500 levels within a seed being unique to another seed. Following the training phase, to assess test performance, we evaluate each agent on 500 levels that have not been trained previously. The episode reward is aggregated across 1000 episodes, with each episode beginning in a successive level. We train the agent through each of the 500 levels sequentially so that it receives a sufficiently broad exposure to different obstacle types. The agent's performance determines episode length, and it runs through as many episodes as possible within 10^6 timesteps.

5.4 ACPO Performance

This section evaluates the performance of ACPO and PPO(OC) along with the aforementioned baselines in Section 5.3.2 for both training and test environments. We begin by reporting the training results for each agent in each environment, which measures how well each agent learns to solve the levels it trains on. This result corresponds to research question 1 in Section 5.1. We then follow up by reporting test performance for each agent, which measures how well each agent can generalise on tasks it has not yet encountered. This result corresponds to research question 2 in Section 5.1.

For the contrastive RL agents, we perform pre-training on the SimCLR model, which is used to encode observations before we deploy it into its environment. As explained in Section 4.2, we have two types of contrastive RL agents in this research, one which is trained purely with observation trajectories and the other which is trained using observations and the action the agent took in the corresponding state. Each type of

contrastive RL agent receives its own SimCLR contrastive model for each Procgen environment on which it's tested. This means we train four SimCLR models for PPO(OC), and four for ACPO, with each of the four models for each algorithm corresponding to each of the games outlines in Figure 5.1. It is valuable to note that ACPO always generates lower contrastive loss in training. As stated in Section 4.2, these SimCLR models are subsequently used to encode observations encountered by the agents, which are then passed to its policy function for learning.

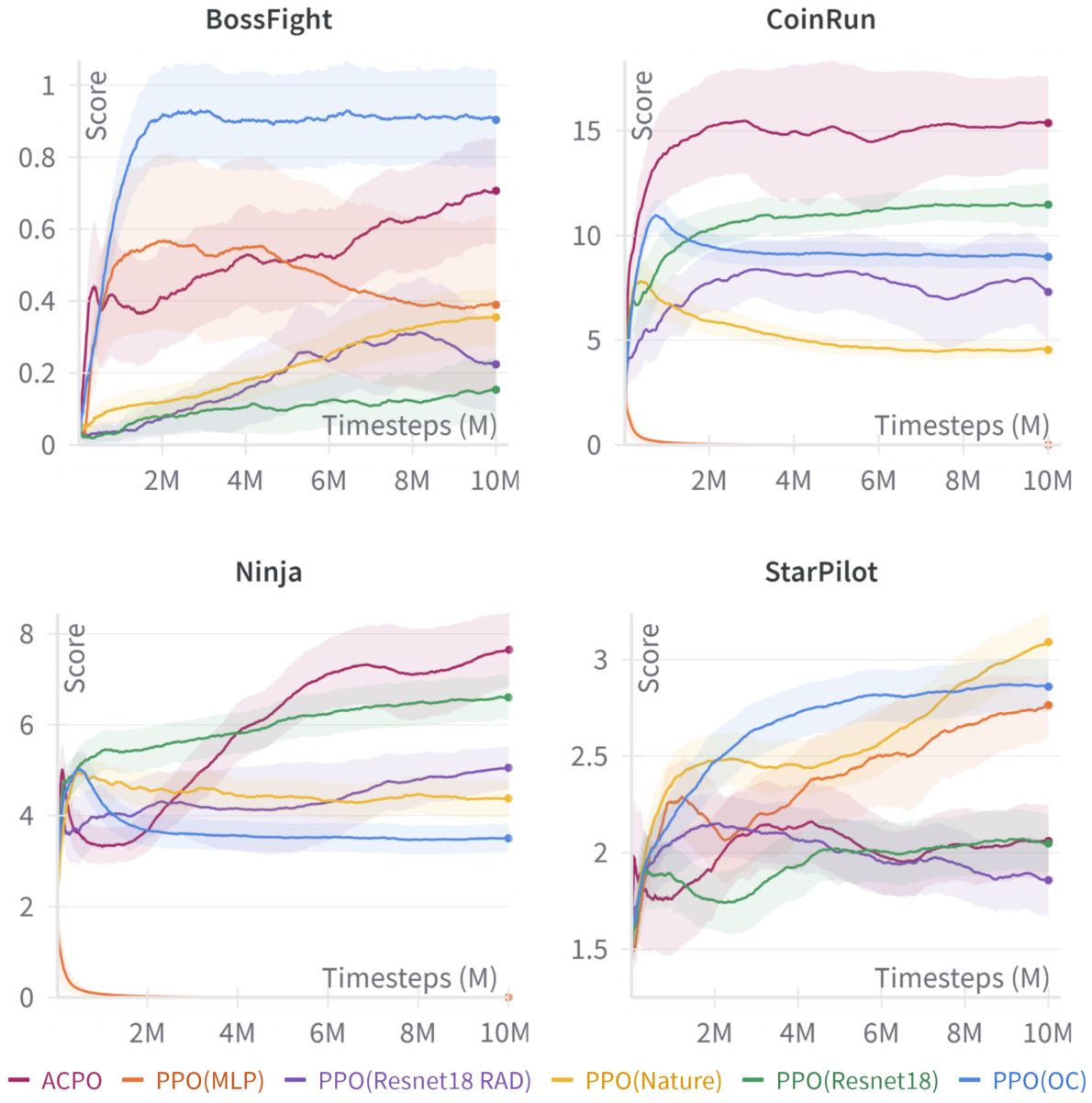


Figure 5.2: Training performance of the ACPO agents in comparison to the baseline PPO agents on the selected Procgen environments.

We observe in Figure 5.2, the performance of each agent in the four selected Procgen environments. The x-axis of each graph shows the number of timesteps each agent experienced training over, and the y-axis shows the episode-reward-mean received by the

agent at that timestep. The episode reward mean is calculated using an aggregation of the rewards received over every episode of a Stable Baselines training iteration consisting of 2048 timesteps. This shows us how an agent improves or worsens performance as it trains through the 500 levels of its seed for each algorithm.

ACPO manages to outperform all baselines on two of the selected environments, Coinrun and Ninja. It is only outperformed by the PPO(OC) contrastive learning baseline in Bossfight, with both agents surpassing all non-contrastive PPO baselines. In Coinrun and Ninja, the only baseline that manages to compete with ACPO and solve the environment to a partial degree is PPO(Resnet18). The baseline agents in Starpilot generally outperform our contrastive methods, most notably PPO(Nature) and PPO(MLP). Whilst PPO(OC) performs well enough to surpass the PPO(MLP) for Starpilot, it does not surpass the PPO(Nature).

Table 5.1: Mean-aggregated test rewards achieved by each agent over 1000 evaluation episodes, ± one standard deviation.

Environment	ACPO	PPO(OC)	PPO(MLP)	PPO(Resnet18)	PPO(Nature)	PPO(Resnet18 RAD)
Bossfight	0.56±0.16	**0.9±0.09**	0.33±0.41	0.22±0.44	0.46±0.05	0.24±0.34
Coinrun	**15.46±5.37**	10.4±1.03	0.0±0.0	4.17±2.08	0.72±0.13	7.52±9.75
Ninja	**7.25±1.52**	1.34±0.32	0.0±0.0	6.39±1.01	0.23±0.11	5.68±0.89
Starpilot	1.22±0.56	1.42±0.12	1.46±0.14	1.87±0.37	**2.84±0.22**	1.78±0.58
Norm.score	2.5	1.98	0.31	1.55	1.44	1.65

Table 5.1 shows the performance of each contrastive and baseline agent when being evaluated on unseen levels for the environment it was trained on. As stated in Section 5.1, this measures how well the agent generalises its learning to solve environments it has not yet encountered. Each value in the table is the mean-aggregated reward achieved over 1000 episodes across 500 levels the agent has never encountered in the past. This evaluation is run over five seeds, over which we take the mean of the aggregated rewards returned for each seed.

The best-performing algorithm for the Coinrun and Ninja environments is ACPO, which is consistent with the training results. For Bossfight and Starpilot, PPO(OC) and PPO (Nature) are the highest-performing agents, respectively, which is once again consistent with the training results shown in Figure 5.2. It is important to note that the ACPO agents have little to no performance attrition from training to test levels. This means that they generalise well to problems encountered at unseen levels. Other agents often experience a substantial performance drop, as is the case with PPO(Resnet18) in Coinrun and PPO(OC) in Ninja and Starpilot. ACPO also scores highest on the overall normalised score summed for all returns gained by the agents in all environments, followed by PPO(OC). This metric measures which agents perform best in general when using a normalised scale of rewards for all games.

To evaluate the training performance of each algorithm so as to determine success or failure as defined in Section 5.1, it is important to compare the performance of each algorithm relative to the optimum rewards achievable for each game as reported by Cobbe *et al.* [2019] in their evaluation of the reward optimums for their games. For Coinrun and Ninja, our contrastive agents reach values comparable to the upper-bound

reward for each game. In the case of Bossfight and Starpilot, we achieve considerably lower than the proposed optimums for all agents. Our best-performing agent in Bossfight achieves just below 1 for an optimum of 11, and the best agent in Starpilot achieves just under 3 for an optimum of 64 [Cobbe *et al.* 2019]. We can conclude from this that some of our agents succeed in solving the environment in the cases of Coinrun and Ninja, and all agents generally fail to solve Bossfight and Starpilot. These results are similar to the results achieved by Mazoure *et al.* [2020] in their research, which uses the C51 Rainbow DQN as a baseline algorithm in addition to their novel approach.

In the following section, we seek to explain the success or failure of each agent. We discuss the performance of ACPO and the respective contrastive and non-contrastive baselines in the following Section 5.5, using auxiliary experiments conducted in retrospect to our primary evaluations.

5.5 Discussion

This section discusses the results presented in Section 5.4. We begin by demonstrating that it is the embedding of action encodings into our contrastive learning process and not the increase in the model parameters of ACPO that improves the performance of the contrastive agents. We then discuss the non-contrastive baseline experiments outlined in Section 5.3.2, discussing the notable differences in performance to help us understand why the contrastive agents performed the way they did. Finally, we show that the use of low-quality, pre-generated data trajectories is not good for contrastive learning, further justifying why our agents struggle on Bossfight and Starpilot.

5.5.1 Comparing ACPO to PPO(OC)

In Figure 5.2, we see clearly that the PPO(OC) agent in Coinrun and Ninja fails to adequately solve the environments, flat-lining after approximately 2 million training steps at a sub-optimal return. ACPO, on the other hand, is the best-performing algorithm, showing consistent improvement over time. This section compares the PPO(OC) contrastive approach with the action-embedded contrastive method ACPO. We state in Section 4.2 that our preliminary evaluation of SimCLR as a contrastive encoding function showed that it did not achieve satisfactory results due to the model being incapable of inferring context across visually similar images, instead simply grouping them into one class. Through state comparisons in the game Ninja, we discuss how the latent space differentiation suffers without action embeddings, helping the agent learn the context of a state and how adding action embeddings resolves this issue, enabling the agent to perform better than it otherwise would.

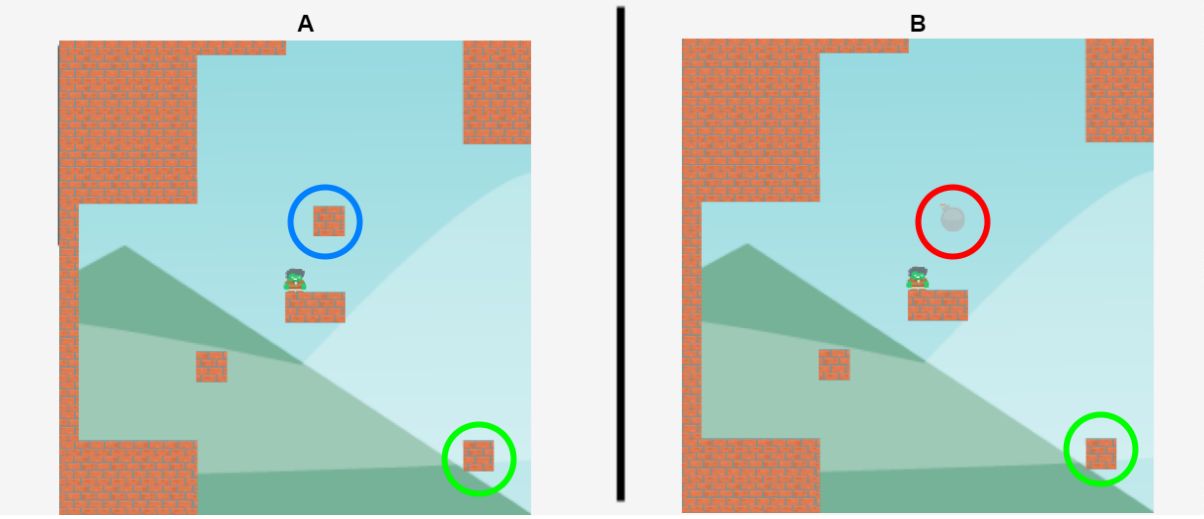


Figure 5.3: Structurally similar states with different contexts in the game Ninja.

In Figure 5.3, we observe two typical states from the Procgen game environment Ninja that are structurally and visually similar, with the exception being the placement of a bomb as opposed to a platform in scenario B. In scenario A, the agent is faced with a brick platform (circled in blue), requiring it to take the optimal right jump action to place it in a better position to reach the lower platform (circled in green). Using the platform, it can progress further in the level to reach its goal state. In scenario B, the agent is faced with a bomb (circled in red) that serves as an obstacle to the platform (circled in green). In this situation, taking a right jump action would lead to a terminating state as the agent will make contact with the bomb, hence it must employ a strategy that uses an upward, right diagonal projectile to destroy the bomb first. As stated in section 4.2, to do this, it must be able to differentiate these states in latent space after training. We discuss the experiment that shows that ACPO does this successfully, whilst PPO(OC) does not in the paragraph below.

As stated in Section 2.4.1, the SimCLR framework is a self-supervised contrastive learning algorithm that employs the cosine similarity function to determine the distance between two image observations in latent space to determine whether they should be grouped. We replicate this process in isolation from training to demonstrate the effectiveness of distancing latent space representations with action embedding. Using the pre-trained Resnet18 contrastive encoders f_{ppo-oc} for PPO(OC) and f_{acpo} for ACPO, we encode and compare a series of three different sets of image pairs, including the pair shown in Figure 5.3, to assess the cosine similarity of their latent representations when encoded by these functions. All three image pairs are structurally similar yet contextually different to ensure the encoder is context-sensitive. The output of each encoder is what the respective contrastive agents (PPO(OC) and ACPO) use to encode observations encountered in the environment during RL training. We performed this using the same set of augmentations and image preparation techniques used when training the agent in practice. The results show that f_{ppo-oc} groups the images as 0.967 similarity whilst f_{acpo} achieves a score of 0.891. This shows that the latent space differentiation

achieved by f_{acpo} is better, as it further separates the contextually distinct observations when compared to f_{ppo-oc}.

To further explore the performance benefit of the inclusion of action embeddings into contrastive learning, we experiment to show that it is the inclusion of action embeddings and not the expansion of model parameters that leads to the improvement of the agent. We do this by expanding the PPO(OC) model to include a second projection head that is identical in structure to the action projection head outline in 4.2. The difference in the expanded PPO(OC) contrastive model is that we do not include a one-hot encoded action value to the latent output of the Resnet18 CNN encoder. Instead, we simply train the model the way we trained the original PPO(OC) model that uses only one projection head function as outlined in 4.2. The resulting performance when tested on Ninja is as follows:

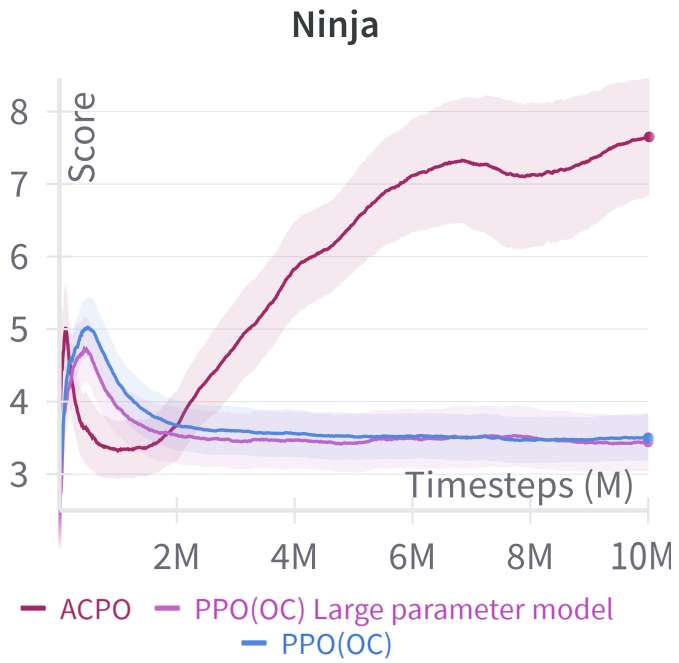


Figure 5.4: Performance comparison of ACPO and PPO(OC) agents with increased model parameters.

Figure 5.4 shows the performance difference between the two identical model structures, the only difference being that ACPO includes action embeddings. The PPO(OC) model uses a larger parameter space than the standard PPO(OC), which is without action embeddings, and shows an overall performance drop of 0.5 when compared to ACPO. This is despite having the same number of parameters. This experiment successfully establishes that the addition of action embeddings and not the expansion of the model parameters are responsible for the performance improvement observed in ACPO. It must be noted that all agents had their contrastive models trained with high-quality data for this experiment. The experiment was conducted over five seeds.

5.5.2 Comparing ACPO to PPO baselines

In Section 5.3.2, we outline five baseline experiments we use as an evaluation against ACPO agents in our environments to determine the degree to which contrastive learning affects performance. We evaluate four of the non-contrastive PPO baseline experiments against ACPO to determine the extent to which each algorithm design choice impacted our agents' performance, as these baselines are constituents of the ACPO algorithm. We only use the Coinrun and Ninja environments as part of our discussion because these are the only environments where agents were meaningfully successful compared to other state-of-the-art algorithms. The following list discusses each non-contrastive baseline experiment from Section 5.3.2:

1. PPO using an MLP policy network and value function identical in structure to the PPO algorithm used in both contrastive RL agents, except for using raw observations and a larger input layer due to the absence of contrastive encoding.
    - This experiment successfully shows that we are deriving value from encoding observations with a contrastive model.
    - We observe in Figure 5.2, that all PPO(MLP) seeds fail to solve the environment on their own, converging to 0 much faster than all other algorithms for Coinrun and Ninja.
2. PPO using a Resnet18 CNN actor-critic network structure with raw, grayscale images used as the input during training.
    - In Figure 5.2, we observe two interesting observations in this experiment for both Coinrun and Ninja. The first is that PPO(Resnet18) outperforms all other baseline experiments, getting a performance closer to ACPO than any other agent. The second is that it manages to outperform PPO(OC) in both environments.
    - We can assess from this that the algorithm works very well with raw images, likely due to the characteristically vast network structure of Resnet CNNs [He *et al.* 2015]. This allows the network to solve the problems associated with deep convolutional networks described in Section 2.3.1 so that it can accurately identify and represent ground truth information from a large set of observations with distinct visual information.
3. PPO using a Resnet18 CNN actor-critic network structure with randomly augmented grayscale images as the input during training.
    - We can observe from the performance shown in Figure 5.2 that simply using the same augmentations (random crop, random horizontal flip, and colour-jitter) used to augment data for training the contrastive models of the contrastive RL agents, does not provide a performance benefit.
    - We further show through the use of ground truth prediction experiments that this is largely due to the agent's inability to distinguish important observation characteristics due to the extent to which visual information in the observation is varied when augmented. It is important to note that the agent does

not have a specialised model designed to contrast and encode augmented states of this kind through pre-training, hence it cannot perform the same way. We can conclude from this that it is not augmented data alone that improves a contrastive agent's performance but the contrastive model itself.

4. PPO using a Nature CNN actor-critic network structure using grayscale images as the input during training.

    - We observe a marked drop in performance when using the Nature CNN as opposed to Resnet18 for the actor-critic network structure in both Coinrun and Ninja. We can conclude from these experiments that Resnet18 is a superior CNN for this domain and is likely the better option for contrastive learning; hence, it is the selection for our contrastive encoder.

To further evaluate the baseline performance, we conduct a series of tests that evaluate how well each algorithm manages to predict ground truth information in the environment it trains on. This helps us understand the general behaviour of each algorithm and also shows why the agent failed for the Bossfight environment. Note that we could not create a ground truth experiment for Starpilot as it was not possible to extract information from the environment using the Procgen framework's current software implementation. We also opt for Ninja over Coinrun as preliminary experiments show that the results are very similar for both environments. For each of the six algorithms evaluated, we reconstruct its actor-critic network as an isolated network to create a supervised learning task that allows us to train said network using a small set of image observations as training data. The purpose of this is to evaluate the ability of the actor-critic network to detect ground truths in the game.

We return the following ground truths for each state in each environment when generating our training data for the reconstructed supervised model training. For Bossfight, we return information regarding whether the enemy boss has its shield activated, an important mechanic in the game. For Ninja, we return whether the agent is on a platform or in mid-air, which is also an important mechanic as the game is classified as platformers, meaning the agent is required to land successfully on said platforms to navigate to its goal state. All of the values returned are booleans (they evaluate as either true or false) and are attached to the vector representation of the observation of the state they were taken from.

We now describe the supervised learning experiments for each algorithm to determine the ability of the network structure to learn ground truths. Note that for all experiments, we use the same dataset generated from each environment for each algorithm evaluation. The datasets are composed of 10000 image observations taken from the explorations of an untrained RL agent (untrained so as to encounter variable states through increased exploration) using PPO with a Nature CNN actor-critic structure. We choose this number as it gives the agent a fair chance to encounter a wide distribution of states when exploring the environment. We use 8000 observations as our training set and 2000 observations as our evaluation set. The purpose is to see if the network's ability to predict ground truths would correlate to its performance during RL.

The experiments for each algorithm are as follows:

1. PPO using an MLP policy network and value function identical in structure to the PPO algorithm used in the contrastive RL agents, except for using raw observations and a larger input layer.
    - For this experiment, the primary actor-critic structure is the MLP network used by the agent to evaluate the ground truths within an observation.
    - For the ground truth accuracy experiment, we train an MLP using the same structure and hyperparameters as PPO(MLP), with the same input, with the change in objective being the network structure being required to predict the ground truth boolean of the state representation (i.e. the state of the enemy agent's shield) as the target.
2. PPO using a Resnet18 CNN actor-critic network structure with raw, grayscale images used as the input during training.
    - For this experiment, the primary actor-critic structure is the Resnet18 CNN network used by the agent to evaluate the ground truths within an observation.
    - For the ground truth accuracy experiment, we train a Resnet18 CNN using the same structure and hyperparameters as PPO(Resnet18) with a greyscale observational input.
3. PPO using a Resnet18 CNN actor-critic network structure with randomly augmented grayscale images as the input during training.
    - For this experiment, the primary actor-critic structure is the Resnet18 CNN network used by the agent to evaluate the ground truths within an observation.
    - For the ground truth accuracy experiment, we train a Resnet18 CNN using the same structure and hyperparameters as PPO(Resnet18 RAD) with randomly augmented grayscale observational inputs.
4. PPO using a Nature CNN actor-critic network structure using grayscale images
    - For this experiment, the primary actor-critic structure is the Nature CNN network used by the agent to evaluate the ground truths within an observation.
    - For the ground truth accuracy experiment, we train a Nature CNN using the same structure and hyperparameters as PPO(Nature) with randomly augmented grayscale observational inputs.
5. PPO(OC) using an MLP actor-critic network structure using encoded images from a non-action embedded contrastive model as an input.
    - For this experiment, the primary actor-critic structure is the MLP network used by the agent to evaluate the ground truths within an observation.
    - In this ground truth evaluation, we encode all observations in our dataset using the SimCLR contrastive model that is trained without action embedding.

The encoded observations are used to train the MLP network to be used to assess ground truth prediction capability.

6. ACPO using an MLP actor-critic network structure with encoded images from a context-aware, action-embedded contrastive model as an input.

    - For this experiment, the primary actor-critic structure is the MLP network used by the agent to evaluate the ground truths within an observation.

    - In this ground truth evaluation, we encode all observations in our dataset using the SimCLR contrastive model that is trained with action embeddings. These encoded observations are then used to train the MLP network to be used to assess ground truth prediction capability.

Table 5.2: Ground truth accuracy of each agent's actor-critic structure across the Bossfight and Ninja environments.

Environment	ACPO	PPO(OC)	PPO(MLP)	PPO(Resnet18)	PPO(Nature)	PPO(Resnet18 RAD)
Bossfight	0.846±0.012	0.853±0.009	0.766±0.061	0.65±0.032	0.723±0.084	0.75±0.024
Ninja	0.85±0.026	0.846±0.017	0.736±0.068	0.956±0.031	0.873±0.012	0.85±0.06

Table 5.2 shows the accuracy at which each agent's actor-critic structure can predict the aforementioned ground truth booleans for Bossfight and Ninja in a supervised learning setting. The scores are aggregated over three seeds. We observe a pattern of prediction accuracy that partially relates to the training performance of the Bossfight and Ninja agents. In Figure 5.4, all non-contrastive baseline agents in the Bossfight environment converge to failing values, and only the contrastive agents manage to show convergence to any meaningful score. In Table 5.2, we observe the pattern of both contrastive actor-critic structures scoring above 0.8 in accuracy and all non-contrastive baseline actor-critic structures scoring below 0.8 in accuracy. This inability to meet a ground truth perception threshold is detrimental to the agent's ability to solve the environment as it often fails in interpreting the game's core mechanic. In the case of Bossfight, this mechanic is centred around attacking when the enemy agent's shield is not active.

In the Ninja environment, all agents except PPO(MLP) manage to score a ground truth accuracy above 0.8 and so manage to converge to some sub-optimal score and partially solve the environment. The agent that scores below 0.8, PPO(MLP) converges to 0 as it once again struggles to learn the game's core mechanic, which is to detect whether an agent is currently standing on a platform. It must be noted, however, that although some agents score even higher than others in accuracy (i.e. PPO(Resnet18) scores higher than ACPO), they do not necessarily perform better; rather, they simply solve the environment to some degree. This is because detecting ground truths is only one aspect of solving an RL problem, as general environment mechanics and semantics must also be learned. This helps us explain why ACPO is the best-performing agent for Ninja, as it is trained in a way that incorporates more information than just ground truths.

5.5.3 ACPO with low-performance trajectory data

In this Section, we aim to address the issues in performance with the Bossfight and Starpilot game environments detailed in Figure 5.4. Although the augmentation to the ACPO contrastive model generally improves the agent's performance, it is important to note that the effectiveness of the encoding produced by the model in informing the agent's action is highly dependent on the quality of trajectories it receives. That is, creating an informative encoding requires at least a moderately well-performing agent to generate environment trajectories as training data. In a prestige setting, one would ideally use expert knowledge from a human agent trained to play the environment well. We demonstrate that when an agent that generates training data cannot perform at least moderately well in solving the environment, the performance of a contrastive agent will drop significantly due to the poor performance data. This will assist in explaining the sub-optimal performance in both the aforementioned games for contrastive agents that use the training trajectories generated by the poorly performing non-contrastive baselines in their contrastive model training.

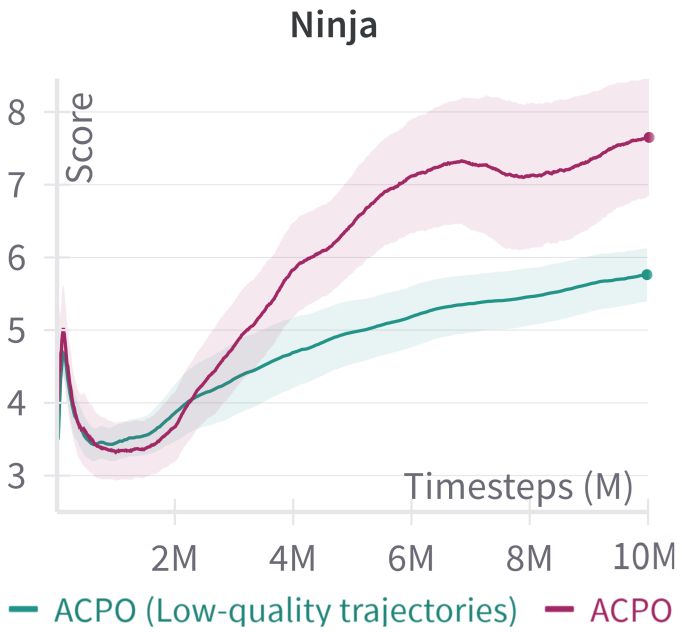


Figure 5.5: Performance comparison of an ACPO agent and an ACPO agent with low-quality training trajectories.

Figure 5.5 compares two agents' performance over five seeds. ACPO (Low-quality trajectories) is trained with roll-out trajectories generated by a PPO(Nature) agent trained for 100000 timesteps instead of a PPO(Nature) agent trained for 2 million timesteps as per the original experiment. The roll-outs produced by this agent suffer an approximately 60% drop in performance when compared to the PPO(Nature) agent that receives more pre-training. As a result, the contrastive model does not separate states in a semantically meaningful way, resulting in a marked drop in performance. In Fig-

ure 5.4, we note that all non-contrastive agents perform poorly for both Bossfight and Starpilot, leading to a small fraction of the possible return for each environment. We reason that if these agents perform poorly, they cannot generate trajectories that are of high-performance quality. The result is an overall poorly performing contrastive agent.

5.6 Summary

In this section, we have discussed the experiments we performed to evaluate the performance of ACPO. We started by declaring our hypotheses and the subsequent success criteria in Section 5.1. This formed the basis for determining whether or not the approach can be deemed successful. We then discuss the research environment in which we deploy our algorithm for evaluation in Section 5.2. This details the specific tasks the agent must learn to solve. Section 5.3 discusses the experiment procedure we use to evaluate and analyse the performance of our algorithm. In this section, we establish baselines used to compare the performance of our agent relative to its constituents and discuss the choice of games we use to evaluate our agents and baselines.

The performance of ACPO is discussed in Section 5.4 beginning with a presentation of the training result in Figure 5.4. It shows a clear performance improvement compared to both non-contrastive baselines and the observational contrastive baseline. This is followed by a presentation of the improved test performance in Table 5.1, where ACPO demonstrates good generalisation ability to an unseen level, showing almost no performance deterioration when compared to its training performance. We end the chapter by analyzing ACPO relative to PPO(OC). This experiment is important in understanding the extent to which ACPO benefits from including action information in its contrastive state representation model training. We then analyse ACPO in comparison to its non-contrastive baselines, and this helps us understand what components of ACPO benefit its performance. Lastly, we analyse the effect of poor-quality performance trajectories on the ability of the contrastive state representation model to encode state observations. The marked drop in performance demonstrates that we require at least a moderately well-performing agent to succeed in enabling our contrastive state representation model to encode states meaningfully.

Chapter 6

Conclusion

In this research, we have introduced a new approach to state representation learning using contrastive learning. We show that the inclusion of action values into a contrastive learning model, with quality data gathered from a prospective RL environment, significantly improves an agent's ability to differentiate or group states by their contextual characteristics, separating them or bringing them closer in inner product latent space. To create a state representation learning model that can employ contrastive differentiation for state observations, we use the Simple Framework for Contrastive Learning of Visual Representations (SimCLR) by Chen *et al.* [2020] to differentiate states by using both the state observation and the action an agent took for that state. The resulting contrastive model is a self-supervised learning neural network that can differentiate states after training with a sample trajectory set from the environment in question. We couple this contrastive model with PPO to create a contrastive RL agent that improves both its training performance and generalisability to unseen levels with similar structure and identical mechanics (the rules of physics and transition probabilities). We call this algorithm Action Contrastive Policy Optimization (ACPO).

We use the Procgen test suite to provide a suitable environment for deploying and training our contrastive RL agent. This specialised test suite employs procedural generation to ensure that no level for a particular game is generated more than once. The platform's infinite level generation capabilities give our agents ample testing room to ensure that they can generalise to a large enough number of unseen environments to ensure that our approach is well-tested.

We conclude from Figure 5.2 that for research question 1 in Section 5.1, we can reject the null hypothesis H_0 and accept our alternative hypothesis H_1, meaning that ACPO improves the training performance of an agent that otherwise uses raw pixel data. We see a clear improvement in the converged reward value in comparison to standard PPO benchmarks for Coinrun and Ninja, the two environments we manage to solve. We can conclude from Table 5.1 that for question 2 in Section 5.1, we can reject the null hypothesis H_0 and accept our alternative hypothesis H_1, meaning that ACPO improves the test performance of an agent that otherwise uses raw pixel data. We see a clear improvement in the total mean-aggregated reward obtained by the agent in levels it

has not encountered before when deployed in environments where we were able to generate decent trajectories.

Through examining each of the constituent parts of ACPO, we determine that the performance of the algorithm is due to the inclusion of action values in Section 5.5.1, the constituent design choices of the algorithm as determined in Section 5.5.2, and finally the quality of trajectories generated prior to training the contrastive state representation model in Section 5.5.3. With these discussions and analysis, we can conclude that it is not the baseline components themselves, such as the specific choice of neural network or the contrastive learning framework SimCLR, that provide the improvement of performance and generalisability seen in our results, but our specific changes to these algorithms to work in a new way.

6.1 Future Work

The information that can incorporated into contrastive learning models has been shown to be quite extensive [Chen *et al.* 2020]. Future work could employ the use of rewards generated by the agent when collecting training trajectories. Doing this would require the expansion of the contrastive model or some value function that can map the contrastive groupings created by the encoder to rewards. This could allow an agent to understand the value of a set of states during the training of the contrastive model as opposed to learning this for each individual state during training. This could allow an agent to quickly converge to an optimal return. If this information were available, one could also incorporate physics properties into the contrastive model. Lutter *et al.* [2021] show how this is particularly useful in environments like Mujoco as they enable the agent to learn physical environment dynamics through small data samples. This makes it better able to deal with nuances in physical structure when navigating its environment.

We emphasise that the success of our algorithm is strongly hinged on the quality of data generated by the agent before contrastive model pre-training. The resulting agent, should it have received good enough data, will show a marked improvement in performance. A follow-up experiment could generate new data for contrastive learning every time the agent reaches a performance threshold. Moerland *et al.* [2023] discuss multiple successful approaches that use a similar cycle of training and updating an environment model, with new, improved data leading to enhanced long-term performance in multiple reinforcement learning settings. Casper *et al.* [2023] show that the incorporation of human expertise in RL has promising results. In the case of contrastive RL, we could test the performance of the ACPO approach using human-generated trajectories to examine the extent to which the agent can improve upon them.

References

[Akiba *et al.* 2019] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

[Alvernaz and Togelius 2017] Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8. IEEE, 2017.

[Alzubaidi *et al.* 2021] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, José Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8:1–74, 2021.

[Balaji 2020] Sai Balaji. *Binary Image classifier CNN using TensorFlow*, 2020.

[Bennett *et al.* 2021] Daniel Bennett, Yael Niv, and Angela J Langdon. Value-free reinforcement learning: policy optimization as a minimal model of operant behavior. *Current Opinion in Behavioral Sciences*, 41:114–121, 2021.

[Casper *et al.* 2023] Stephen Casper, Xander Davies, Claudia Shi, Thomas Krendl Gilbert, Jérémy Scheurer, Javier Rando, Rachel Freedman, Tomasz Korbak, David Lindner, Pedro Freire, et al. Open problems and fundamental limitations of reinforcement learning from human feedback. *arXiv preprint arXiv:2307.15217*, 2023.

[Chaudhary 2020] Amit Chaudhary. *The Illustrated SimCLR Framework*, 2020. https://amitness.com/2020/03/illustrated-simclr.

[Chen *et al.* 2020] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey E. Hinton. A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709, 2020.

[Chopra *et al.* 2005] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verification. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 539–546. IEEE, 2005.

[Cobbe *et al.* 2019] Karl Cobbe, Christopher Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. *CoRR*, abs/1912.01588, 2019.

[Eysenbach *et al.* 2023] Benjamin Eysenbach, Tianjun Zhang, Ruslan Salakhutdinov, and Sergey Levine. *Contrastive Learning as Goal-Conditioned Reinforcement Learning*, 2023.

[Finn *et al.* 2016] Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 512–519. IEEE, 2016.

[François-Lavet *et al.* 2018] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *CoRR*, abs/1811.12560, 2018.

[Goodfellow *et al.* 2016] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[Goroshin *et al.* 2015] Ross Goroshin, Michael F Mathieu, and Yann LeCun. Learning to linearize under uncertainty. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015.

[Ha and Schmidhuber 2018] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[He *et al.* 2015] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[Hjelm *et al.* 2019] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. *Learning deep representations by mutual information estimation and maximization*, 2019.

[Ivanov and D'yakonov 2019] Sergey Ivanov and Alexander D'yakonov. Modern deep reinforcement learning algorithms. *arXiv preprint arXiv:1906.10025*, 2019.

[Janner *et al.* 2019] Michael Janner, Justin Fu, Marvin Zhang, and Sergey Levine. When to trust your model: Model-based policy optimization. In *Advances in Neural Information Processing Systems*, pages 12498–12509, 2019.

[Jonschkowski and Brock 2015] Rico Jonschkowski and Oliver Brock. Learning state representations with robotic priors. *Autonomous Robots*, 39:407–428, 2015.

[Karakovskiy and Togelius 2012] Sergey Karakovskiy and Julian Togelius. The mario ai benchmark and competitions. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):55–67, 2012.

[Karl *et al.* 2016] Maximilian Karl, Maximilian Soelch, Justin Bayer, and Patrick Van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv preprint arXiv:1605.06432*, 2016.

[Klein and Celik 2017] R. Klein and T. Celik. The Wits Intelligent Teaching System: Detecting student engagement during lectures using Convolutional Neural Networks. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 2856–2860, Sep. 2017.

[Krishnan *et al.* 2015] Rahul G Krishnan, Uri Shalit, and David Sontag. Deep kalman filters. *arXiv preprint arXiv:1511.05121*, 2015.

[Le-Khac *et al.* 2020] Phuc H Le-Khac, Graham Healy, and Alan F Smeaton. Contrastive representation learning: A framework and review. *Ieee Access*, 8:193907–193934, 2020.

[Lesort *et al.* 2018] Timothee Lesort, Natalia Diaz-Rodriguez, Jean-FranoQis Goudou, and David Filliat. State representation learning for control: An overview. *Neural Networks*, 108:379–392, 2018.

[Li *et al.* 2021] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: analysis, applications, and prospects. *IEEE transactions on neural networks and learning systems*, 2021.

[Lutter *et al.* 2021] Michael Lutter, Johannes Silberbauer, Joe Watson, and Jan Peters. Differentiable physics models for real-world offline model-based reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4163–4170. IEEE, 2021.

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

[Mazoure *et al.* 2020] Bogdan Mazoure, Remi Tachet des Combes, Thang Doan, Philip Bachman, and R Devon Hjelm. *Deep Reinforcement and InfoMax Learning*, 2020.

[Mnih *et al.* 2013] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*, 2013.

[Moerland *et al.* 2023] Thomas M Moerland, Joost Broekens, Aske Plaat, Catholijn M Jonker, et al. Model-based reinforcement learning: A survey. *Foundations and Trends® in Machine Learning*, 16(1):1–118, 2023.

[Nichol *et al.* 2018] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018.

[Pan *et al.* 2017] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving. *arXiv preprint arXiv:1704.03952*, 2017.

[Raffin *et al.* 2019] Antonin Raffin, Ashley Hill, René Traoré, Timothée Lesort, Natalia Díaz-Rodríguez, and David Filliat. *Decoupling feature extraction from policy learning: assessing benefits of state representation learning in goal based robotics*, 2019.

[Rolon-Mérette *et al.* 2016] Damien Rolon-Mérette, Matt Ross, Thaddé Rolon-Mérette, and Kinsey Church. Introduction to anaconda and python: Installation and setup. *Quant. Methods Psychol*, 16(5):S3–S11, 2016.

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

[Sutton and Barto 2018] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 2018.

[Yen *et al.* 2002] Gary Yen, Fengming Yang, and Travis Hickey. Coordination of exploration and exploitation in a dynamic environment. *International Journal of Smart Engineering System Design*, 4(3):177–182, 2002.

Appendix A

Implementation Details

All algorithms are implemented in Python 3 within an Anaconda virtual machine. We opt to use Python as our programming language of implementation as it can be used to implement an extensive number of deep learning and reinforcement learning frameworks currently available [Rolon-Mérette *et al.* 2016]. Many of these frameworks contain working, widely referenced policy-gradient algorithm implementations and general neural network libraries to develop a research prototype. We chose Anaconda as it is a popular data science platform that allows users to easily isolate a development environment on multiple different systems [Rolon-Mérette *et al.* 2016]. This enables us to deploy multiple experiments requiring different framework versions, granting us the versatility we need to successfully conduct all necessary experiments in our research.

PyTorch is the base deep learning framework used to implement all deep neural network models. PyTorch is a deep learning framework implemented in C++ with a Python interface. We opted to use a CUDA-enabled version of PyTorch 11 as this enabled us to use both a preexisting implementation of SimCLR as well as the highly compatible, widely referenced StableBaselines3 framework, which contain multiple pre-implemented actor-critic algorithms that allow for both ease of deployment and ease of augmentation within its code base. StableBaselines3 has been used in a plethora of other RL research in the past and is one of the standard implementations of PPO in contemporary RL research.

Appendix B

Hyperparameter Optimization

Hyperparameter optimization is conducted using the hyperparameter optimization framework Optuna. Optuna is a software package designed by Akiba *et al.* [2019] to create a framework that evaluates hyperparameters using what they define as a *define-by-run* principle. This means that the framework allows users to dynamically construct the search space to find the optimal parameters. The framework performs by picking the best strategy based on the results of each subsequent hyperparameter evaluation episode and changes the range or set of values used for each subsequent episode based on the results of the previous episode. This allows it to reach its optimal value faster.

To elaborate on this process, we discuss how Optuna formulates the optimization process as a task to minimize or maximize some objective function that takes a set of hyperparameters as an input and returns its score over a series of trials. Firstly, it allows the user to pass in categorical or continuous parameter values depending on the need or constraints of each hyperparameter in each model. Following this, the user must instantiate an environment within which they wish to test their model. As mentioned, the project then runs a series of statistical evaluations to determine the best values based on each parameter's previous set of values. The number of evaluations it runs is determined by the performance of the best current run and the number of combinations it can form using the predefined parameter ranges.

The following tables show the hyperparameters for each algorithm and each sub-variant in the case of PPO. We have explicitly shown the hyperparameters for PPO(MLP) and PPO(Nature). We do not do this for PPO(Resnet18) or PPO(RAD) as the parameters used in these experiments are identical to the Resnet18 parameters used in SimCLR to keep consistency in the baseline evaluation. All activation functions used in this research are Rectified Linear Units (ReLU).

Table B.1: PPO Hyperparameters

γ	.9
λ	.95
# TIMESTEPS PER ROLLOUT	512
# BATCH SIZE	64
ENTROPY COEF (k_H)	0
VALUE FUNCTION COEF	0.1
MAX GRADIENT NORM	0.1
PPO CLIP RANGE	.5
LEARNING RATE	3E-5
# WORKERS	1
# ENVIRONMENTS PER WORKER	4
TOTAL TIMESTEPS	10M
MLP	
ACTOR-NETWORK (NINJA, STARPILOT)	32
VALUE FUNCTION NETWORK (NINJA, STARPILOT)	32
ACTOR-NETWORK (COINRUN, BOSSFIGHT)	64
VALUE FUNCTION NETWORK (COINRUN, BOSSFIGHT)	64
NATURE	
PROJECTION DIMENSIONS	512
KERNEL SIZE	8
STRIDE	4
PADDING	0

Table B.2: SimCLR Hyperparameters

# EPOCHS	140
# BATCH SIZE	64
PROJECTION DIMENSIONS	512
WEIGHT DECAY	1E-6
TEMPERATURE	0.1
WORLD SIZE	1
RESNET18: CONV2D AND MAXPOOL LAYERS	
KERNEL SIZE	3
STRIDE	1
PADDING	1