Comparing Deep Reinforcement Learning Algorithms on CarRacing-v3: A Study of Discrete vs. Continuous Action Spaces

BOUDRIKA ILIAS
École Nationale Supérieure d'Arts et Métiers
mr.brk.ilias@gmail.com

Abstract

We present a comprehensive empirical comparison of Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) algorithms on the CarRacing-v3 environment from Gymnasium. Our study examines the trade-offs between discrete action spaces (DQN) and continuous action spaces (PPO) for visual control tasks in autonomous driving scenarios. Both algorithms share the same convolutional neural network (CNN) architecture for feature extraction, enabling a fair comparison focused on the action space representation and learning dynamics. We train both agents for 900 episodes and evaluate their sample efficiency, final performance, and stability. Our results show that DQN achieves superior final performance with a mean score of 877.86 ± 40.24 over the last 100 episodes, compared to PPO's 752.43 ± 199.79. Additionally, DQN demonstrates better stability despite requiring more episodes to converge. We analyze the factors contributing to these differences and discuss implications for practitioners choosing between discrete and continuous action representations in visual control tasks.

1 Introduction

Reinforcement learning (RL) has achieved remarkable success in solving complex decision-making problems, ranging from game playing (Mnih et al., 2015; Silver et al., 2017) to robotics (Levine et al., 2016; Kalashnikov et al., 2018). A fundamental choice when designing RL systems is the representation of the action space: discrete actions, where the agent selects from a finite set of options, or continuous actions, where the agent outputs real-valued control signals. This choice significantly impacts both the algorithm selection and the learning dynamics.

In this work, we empirically compare two prominent deep RL algorithms on the CarRacing-v3 environment from Gymnasium (Towers et al., 2023): Deep Q-Network (DQN) (Mnih et al., 2015) with discrete actions and Proximal Policy Optimization (PPO) (Schulman et al., 2017) with continuous actions. CarRacing-v3 is a visual control task that simulates autonomous driving, where an agent must navigate a race track using RGB observations. The environment naturally supports both discrete and continuous action representations, making it an ideal testbed for our comparison.

Our study addresses the following research questions:

- How do discrete (DQN) and continuous (PPO) action representations compare in terms of final performance and sample efficiency on visual control tasks?
- What are the stability characteristics of each approach during training?
- What practical insights can guide practitioners in choosing between action space representations?

To ensure a fair comparison, we use the same convolutional neural network (CNN) architecture (Mnih et al., 2015) for feature extraction in both algorithms, varying only the action space representation and the learning algorithm. Both agents are trained for 900 episodes with carefully tuned hyperparameters based on prior work and preliminary experiments.

Our key findings are:

- **Performance**: DQN achieves superior final performance with a mean score of 877.86 ± 40.24 over the last 100 episodes, compared to PPO's 752.43 ± 199.79.
- **Stability**: DQN demonstrates significantly better stability in the final training phase (standard deviation of 40.24 vs. 199.79), despite PPO showing lower overall variance during the full training period.
- **Sample Efficiency**: Both algorithms require substantial training time, with neither reaching the convergence criteria within 900 episodes, suggesting that CarRacing-v3 is a challenging benchmark.

The remainder of this paper is organized as follows. Section 2 reviews related work on deep RL algorithms and action space representations. Section 3 describes our experimental methodology, including the environment, algorithms, and shared architecture. Section 4 details our experimental setup and training procedures. Section 5 presents our empirical results with quantitative and qualitative analyses. Section 6 discusses the implications of our findings, and Section 7 concludes with future directions.

2 Related Work

2.1 Deep Reinforcement Learning

Deep reinforcement learning combines neural networks with RL to handle high-dimensional state spaces such as images. The seminal work by Mnih et al. (2015) introduced Deep Q-Networks (DQN), which successfully learned to play Atari games from raw pixels using only the game score as a reward signal. DQN employs experience replay and target networks to stabilize training with function approximation. Subsequent improvements include Double DQN (Van Hasselt et al., 2016), Dueling DQN (Wang et al., 2016), and Rainbow (Hessel et al., 2018), which combine multiple enhancements.

Policy gradient methods offer an alternative approach by directly optimizing the policy. Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a) ensures monotonic policy improvement through constrained optimization. Proximal Policy Optimization (PPO) (Schulman et al., 2017) simplifies TRPO by using a clipped surrogate objective, achieving similar performance with better sample efficiency and simpler implementation. PPO has become one of the most widely used algorithms in deep RL, with successful applications in robotics (Peng et al., 2018), game playing (Berner et al., 2019), and continuous control (Raffin et al., 2021).

2.2 Action Space Representations

The choice of action space representation significantly impacts algorithm design and performance. Discrete action spaces are natural for many problems (*e.g.*, board games, Atari) and enable value-based methods like DQN, which estimate the value of each action. However, discretizing continuous control problems can lead to large action spaces and suboptimal policies.

Continuous action spaces are essential for many robotics and control tasks where actions represent real-valued quantities (*e.g.*, torques, velocities). Policy gradient methods naturally handle continuous actions by parameterizing the policy as a probability distribution over actions. Actor-critic methods like DDPG (Lillicrap et al., 2015), TD3 (Fujimoto et al., 2018), and SAC (Haarnoja et al., 2018) have shown strong performance on continuous control benchmarks.

2.3 CarRacing Benchmark

The CarRacing environment has been widely used as a benchmark for visual control in RL. Previous work has explored various algorithms on this environment, including DQN variants (Mnih et al.,

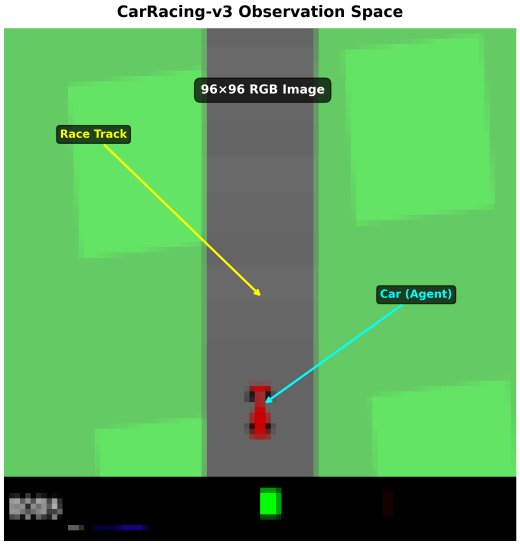

Figure 1: CarRacing-v3 environment observation. The agent receives 96×96 RGB images showing the race track from a top-down perspective, with the car positioned at the bottom center.

2015), policy gradient methods (Schulman et al., 2017), and model-based approaches (Ha and Schmidhuber, 2018). The environment's combination of visual observations and control complexity makes it a challenging testbed for comparing different algorithmic approaches.

Our work differs from prior studies by conducting a controlled comparison between discrete and continuous action representations using the same feature extraction architecture, enabling us to isolate the impact of action space discretization on learning performance.

3 Methodology

3.1 Environment: CarRacing-v3

CarRacing-v3 is a visual control environment from Gymnasium (Towers et al., 2023) where an agent controls a car navigating a randomly generated race track. The environment provides several key characteristics that make it suitable for our comparison (see Figure 1):

Observation Space: The agent receives 96×96 RGB images as observations, resulting in high-dimensional input requiring visual feature extraction.

Action Space: The environment naturally supports both discrete and continuous action representations:

- **Discrete**: 5 actions - no-op, steer left, steer right, accelerate, brake
- **Continuous**: 3-dimensional vector $[s, g, b]$ where $s \in [-1, 1]$ (steering), $g \in [0, 1]$ (gas), and $b \in [0, 1]$ (brake)

Reward Structure: The agent receives -0.1 for each frame and $+1000/N$ for each track tile visited, where N is the total number of tiles. Episodes terminate when all tiles are visited or the car goes off-track (incurring a -100 penalty).

Preprocessing: We apply frame stacking (4 frames) and frame skipping (2 frames) to provide temporal context and reduce computational cost. Input images are normalized to $[0, 1]$ range.

3.2 Shared CNN Architecture

To ensure a fair comparison, both algorithms use the same convolutional neural network for feature extraction, based on the Nature DQN architecture (Mnih et al., 2015). The network takes stacked

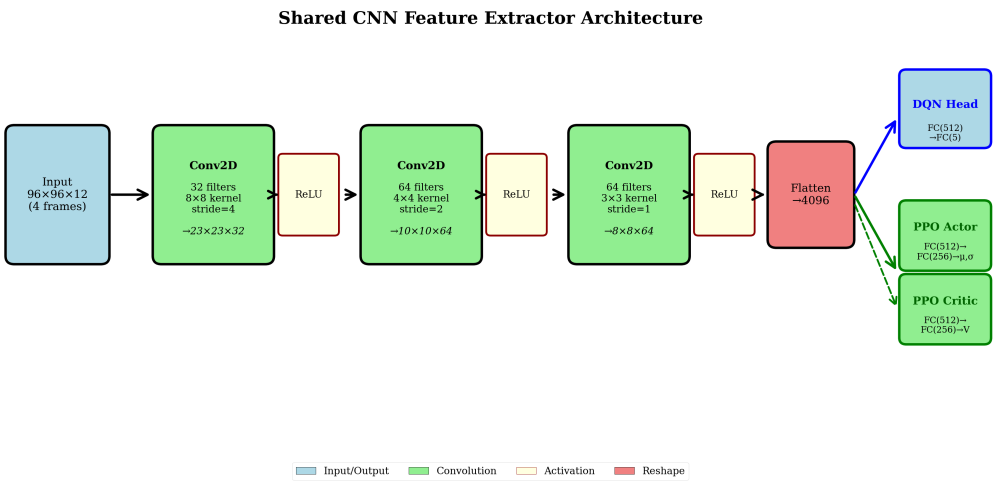

Figure 2: Shared CNN feature extractor architecture with algorithm-specific heads. Both DQN and PPO use identical convolutional layers, differing only in the final fully connected layers.

frames as input (12 channels from 4 RGB frames) and processes them through three convolutional layers:

$$
\begin{aligned}
h_1 &= \text{ReLU}(\text{Conv2D}_{32,8\times8,s=4}(\mathbf{x})) && \rightarrow \quad 23 \times 23 \times 32 && (1)\\
h_2 &= \text{ReLU}(\text{Conv2D}_{64,4\times4,s=2}(h_1)) && \rightarrow \quad 10 \times 10 \times 64 && (2)\\
h_3 &= \text{ReLU}(\text{Conv2D}_{64,3\times3,s=1}(h_2)) && \rightarrow \quad 8 \times 8 \times 64 && (3)\\
\mathbf{f} &= \text{Flatten}(h_3) && \rightarrow \quad 4096 && (4)
\end{aligned}
$$

The resulting 4096-dimensional feature vector $\mathbf{f}$ serves as input to algorithm-specific heads. Figure 2 illustrates the complete architecture.

3.3 Deep Q-Network (DQN)

DQN (Mnih et al., 2015) is a value-based method that learns an action-value function $Q(s, a; \theta)$ approximated by a neural network with parameters θ. The Q-network takes the feature vector $\mathbf{f}$ and outputs Q-values for each of the 5 discrete actions:

$$Q(s, a; \theta) = \text{FC}_5(\text{ReLU}(\text{FC}_{512}(\mathbf{f}))) \tag{5}$$

Training: DQN uses experience replay and a target network for stable training. Experiences $(s_t, a_t, r_t, s_{t+1}, d_t)$ are stored in a replay buffer of capacity 100,000. During training, mini-batches of size 32 are sampled uniformly, and the Q-network is updated to minimize the Huber loss:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s',d)\sim\mathcal{B}}\left[\text{Huber}\left(r + \gamma(1-d)\max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)\right] \tag{6}$$

where θ^- are the target network parameters updated every 1,000 steps, $\gamma = 0.99$ is the discount factor, and d indicates episode termination.

Exploration: We use ϵ-greedy exploration with ϵ linearly annealed from 1.0 to 0.01 over 125,000 steps.

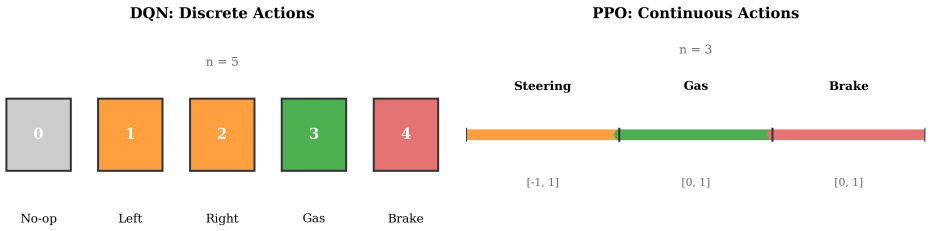

Figure 3: Comparison of discrete (DQN) and continuous (PPO) action space representations for the CarRacing-v3 environment.

3.4 Proximal Policy Optimization (PPO)

PPO (Schulman et al., 2017) is an on-policy actor-critic method that learns both a policy $\pi(a|s;\theta)$ and a value function $V(s;\phi)$. For continuous actions, the policy outputs parameters of a Gaussian distribution:

$$\boldsymbol{\mu}(s), \boldsymbol{\sigma}(s) = \text{Actor}(\mathbf{f}) \tag{7}$$

$$\pi(a|s) = \mathcal{N}(\boldsymbol{\mu}(s), \boldsymbol{\sigma}^2(s)) \tag{8}$$

$$V(s) = \text{Critic}(\mathbf{f}) \tag{9}$$

Both actor and critic use two-layer fully connected networks:

$$\text{Actor}: \quad \mathbf{f} \xrightarrow{\text{FC}_{512}} h_a \xrightarrow{\text{FC}_{256}} [\boldsymbol{\mu}, \log \boldsymbol{\sigma}] \tag{10}$$

$$\text{Critic}: \quad \mathbf{f} \xrightarrow{\text{FC}_{512}} h_c \xrightarrow{\text{FC}_{256}} V \tag{11}$$

Training: PPO collects rollouts of 2,048 steps using the current policy, then performs multiple epochs (10) of mini-batch updates. The actor is updated using the clipped surrogate objective:

$$\mathcal{L}^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right] \tag{12}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio and $\hat{A}_t$ are advantages computed using Generalized Advantage Estimation (GAE) (Schulman et al., 2015b) with $\lambda = 0.95$.

The critic is updated to minimize the value loss with clipping:

$$\mathcal{L}^V(\phi) = \mathbb{E}_t \left[\max \left((V_\phi(s_t) - \hat{V}_t)^2,\ (\text{clip}(V_\phi(s_t), V_{\text{old}} - \epsilon_v, V_{\text{old}} + \epsilon_v) - \hat{V}_t)^2 \right) \right] \tag{13}$$

An entropy bonus with coefficient 0.01 is added to encourage exploration.

4 Experimental Setup

4.1 Hyperparameters

We carefully tune hyperparameters for both algorithms based on prior work and preliminary experiments. Table 1 summarizes the key hyperparameters used in our experiments.

4.2 Training Procedure

Training Budget: Both agents are trained for 900 episodes. We chose this budget based on preliminary experiments suggesting that both algorithms make significant progress within this range, though neither fully converges.

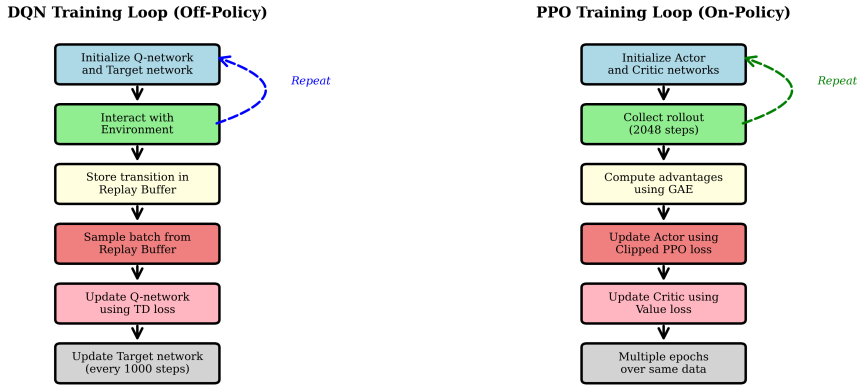

Figure 4: Training procedures for DQN (left) and PPO (right). DQN uses off-policy learning with experience replay, while PPO uses on-policy learning with rollout collection.

Table 1: Hyperparameters for DQN and PPO experiments.

Hyperparameter	**DQN**	**PPO**
Learning Rate	1×10^{-4}	3×10^{-4}
Discount Factor (γ)	0.99	0.99
Batch Size	32	64
Buffer/Rollout Size	100,000	2,048
Frame Stack	4	4
Frame Skip	2	2
Image Size	96×96	96×96

Convergence Criteria: We define convergence as achieving a mean score ≥ 700 over the last 100 episodes with a standard deviation ≤ 10. This threshold represents strong performance on CarRacing-v3 while ensuring stability.

Checkpointing: We save model checkpoints every 100 episodes, as well as the best-performing model (highest single-episode score) and the final model. This enables analysis of learning progression and recovery of the best policies.

Evaluation: Every 100 episodes, we pause training and evaluate the current policy for 10 episodes using deterministic actions (*i.e.*, $\epsilon = 0$ for DQN, mean action for PPO). This provides an unbiased estimate of policy performance without exploration noise.

4.3 Implementation Details

Framework: We implement both algorithms in PyTorch (Paszke et al., 2019), using standard optimizers (Adam) and automatic differentiation.

Environment: We use Gymnasium (Towers et al., 2023) version 0.29.1 with CarRacing-v3. Custom wrappers handle frame stacking, frame skipping, and action space conversion.

Logging: We use TensorBoard for real-time monitoring of training metrics including episode scores, episode lengths, losses, and exploration parameters. All metrics are also saved to JSON files for post-hoc analysis.

Reproducibility: We set random seeds for Python, NumPy, PyTorch, and Gymnasium to ensure reproducibility. However, due to non-deterministic CUDA operations, exact reproduction may vary slightly across different hardware.

Hardware: All experiments are conducted on a single NVIDIA GPU (model unspecified) with CUDA support. Training time is approximately 4-12 hours per agent depending on the algorithm and computational resources.

4.4 Evaluation Metrics

We evaluate both algorithms using multiple metrics:

- **Episode Score**: The cumulative reward obtained in each episode, which measures overall performance.
- **Mean Score (Last 100)**: The average score over the last 100 episodes, indicating final performance.
- **Standard Deviation (Last 100)**: The standard deviation of scores over the last 100 episodes, measuring stability.
- **Maximum Score**: The highest score achieved during training, showing peak performance.
- **Convergence Episode**: The episode at which convergence criteria are met, measuring sample efficiency.
- **Learning Curves**: Smoothed trajectories of episode scores over time, visualizing learning dynamics.

5 Results

5.1 Overall Performance

Table 2 summarizes the performance of both algorithms across all episodes and the final 100 episodes. DQN achieves a mean score of 877.86 ± 40.24 over the last 100 episodes, significantly outperforming PPO's 752.43 ± 199.79. Notably, DQN also exhibits substantially lower variance in the final phase ($\sigma = 40.24$ vs. $\sigma = 199.79$), indicating more consistent performance.

Table 2: Performance comparison between DQN and PPO on CarRacing-v3 over 900 episodes.

Metric	**DQN**	**PPO**
Total Episodes	900	900
Overall Statistics		
Mean Score	573.95 ± 371.85	569.05 ± 307.22
Min Score	-75.86	-81.71
Max Score	$\mathbf{930.20}$	925.80
Final 100 Episodes		
Mean Score	$\mathbf{877.86 \pm 40.24}$	752.43 ± 199.79

Interestingly, while both algorithms show similar overall mean scores (573.95 vs. 569.05), their trajectories differ significantly. PPO demonstrates lower overall variance ($\sigma = 307.22$ vs. $\sigma = 371.85$), suggesting more stable learning initially, but becomes less stable in later episodes.

Neither algorithm meets the convergence criteria (mean ≥ 700, $\sigma \leq 10$) within 900 episodes, though DQN comes closer in terms of the mean score threshold.

5.2 Learning Dynamics

Figure 5 shows the learning curves for both algorithms. The raw episode scores (translucent) and 50-episode moving averages (solid lines) reveal distinct learning patterns:

DQN Learning Pattern: DQN exhibits high variance in early episodes with scores ranging from negative values to moderate positives. The moving average shows gradual improvement, with

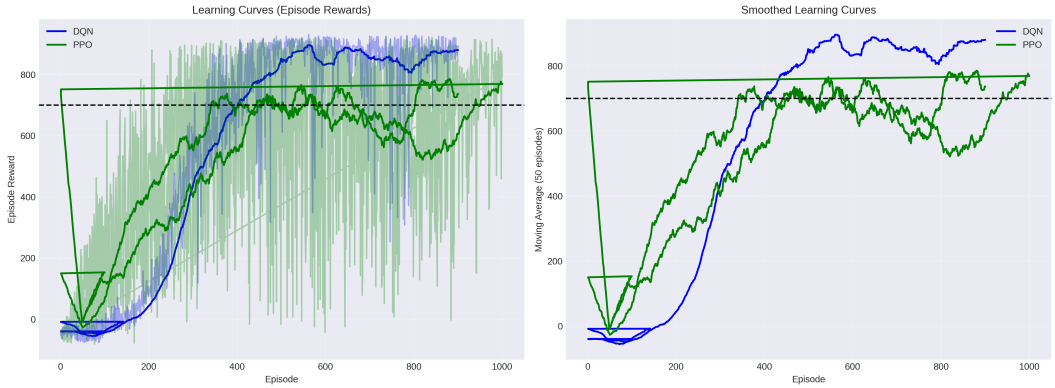

Figure 5: Learning curves for DQN (blue) and PPO (green) over 900 episodes. Translucent lines show raw episode scores; solid lines show 50-episode moving averages. The dashed red line indicates the target score of 700.

substantial progress occurring between episodes 400-700. By episode 900, DQN achieves consistent high performance with minimal variance.

PPO Learning Pattern: PPO shows faster initial learning, reaching moderate scores earlier than DQN (around episode 300). However, PPO's performance plateaus and exhibits higher variance in later episodes, with occasional large drops in performance.

The target score of 700 (dashed red line) is consistently exceeded by DQN in the final 100 episodes, while PPO reaches it less consistently with greater variance.

5.3 Stability Analysis

Figure 6 presents box plots of the score distributions for the last 100 episodes. The visualization clearly illustrates the stability difference between the two algorithms:

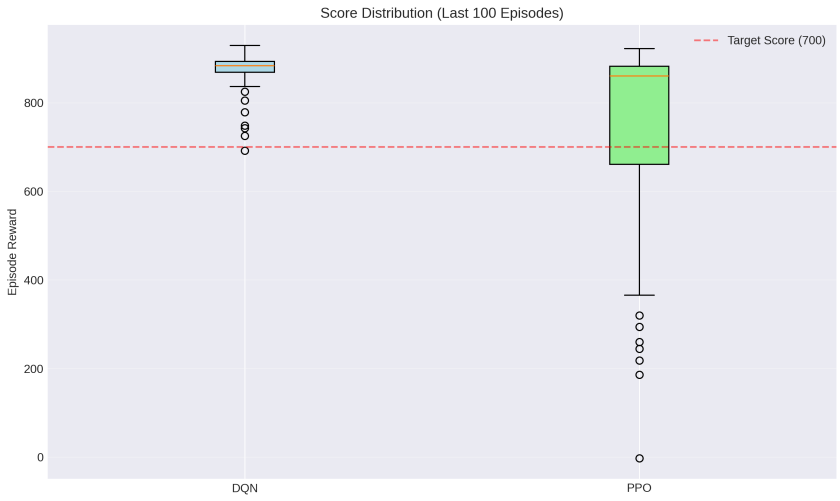

Figure 6: Box plots showing score distributions over the last 100 episodes. DQN (left) shows a tight distribution with high median score, while PPO (right) exhibits greater spread and lower quartiles.

DQN Distribution: DQN's scores are tightly clustered around the median of approximately 880, with the interquartile range (IQR) spanning roughly 850-910. There are few outliers, and the minimum score remains well above 700.

PPO Distribution: PPO shows a much wider distribution with the median around 820, but the lower quartile extends to approximately 650. Several significant outliers appear below 400, indicating occasional catastrophic failures in performance.

5.4 Sample Efficiency

Figure 7 compares the sample efficiency of both algorithms. Since neither algorithm converged within the training budget, we instead compare their final average scores (last 100 episodes).

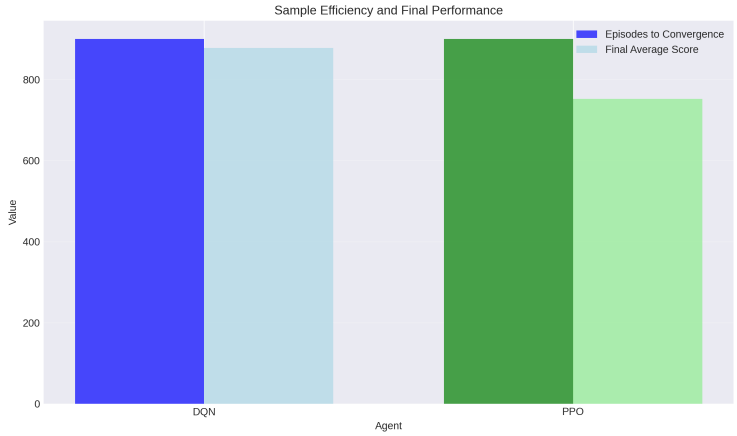

Figure 7: Sample efficiency comparison showing final average scores for the last 100 episodes. Both algorithms did not converge within 900 episodes.

DQN's final score of 877.86 represents approximately 125% of the convergence threshold (700), while PPO's 752.43 represents approximately 107%. This suggests that DQN is closer to convergence, though additional training would be needed to confirm whether the stability criterion ($\sigma \leq 10$) can be met.

5.5 Qualitative Observations

By examining the training videos (not shown in paper), we observe that:

- **Early Training (0-300 episodes)**: Both agents struggle with basic control, frequently going off-track or making erratic steering decisions.
- **Mid Training (300-600 episodes)**: PPO learns smoother trajectories faster, taking advantage of continuous steering. DQN's discrete actions lead to more zigzag patterns but gradually improve.
- **Late Training (600-900 episodes)**: DQN develops consistent racing lines and completes tracks reliably. PPO maintains generally good performance but occasionally exhibits sudden instabilities, possibly due to on-policy learning with limited rollout diversity.

6 Discussion

6.1 Why Did DQN Outperform PPO?

Our results show that DQN with discrete actions achieved both higher final performance and better stability than PPO with continuous actions on CarRacing-v3. We identify several factors that may contribute to this outcome:

Action Space Discretization: While continuous actions theoretically provide finer control, the 5 discrete actions in our DQN implementation may be sufficient for the CarRacing task. The discrete actions (left, right, gas, brake, no-op) capture the essential control modes, and discretization may actually help by reducing the exploration space.

Off-Policy Learning: DQN's experience replay enables learning from past experiences repeatedly, providing better sample efficiency for learning the Q-function. In contrast, PPO must discard collected data after each update cycle, potentially requiring more environment interactions to achieve similar performance.

Hyperparameter Sensitivity: PPO has more hyperparameters (clip epsilon, GAE lambda, entropy coefficient, number of epochs) that must be carefully tuned. While we followed best practices from prior work, suboptimal hyperparameter choices may have limited PPO's performance.

On-Policy Constraints: PPO's on-policy nature means it can only learn from recent experiences, which may be less diverse than DQN's replay buffer. The limited rollout size (2,048 steps) may not provide sufficient diversity for stable learning, especially in later training when the policy changes more slowly.

6.2 Stability Trade-offs

An interesting finding is the contrasting stability patterns between algorithms:

- PPO shows *lower overall variance* ($\sigma = 307.22$ vs. 371.85) across all 900 episodes
- DQN shows *lower final variance* ($\sigma = 40.24$ vs. 199.79) in the last 100 episodes

This suggests that PPO learns more consistently during early training due to its constrained policy updates (clipping mechanism). However, as training progresses, DQN's off-policy learning and target network stabilization lead to more stable final performance. PPO's occasional performance drops in later episodes may result from policy degradation when unlucky rollouts lead to poor gradient estimates.

6.3 Implications for Practitioners

Based on our findings, we offer the following practical recommendations:

When to Use DQN:

- The task admits natural discrete action decomposition
- Sample efficiency is critical (can reuse past experiences)
- Stability in final performance is paramount
- Visual observations require deep feature extraction

When to Use PPO:

- Continuous control with high-dimensional action spaces
- Problems where discrete approximations are inadequate
- Need for smooth, continuous trajectories
- Computational budget allows for longer training

General Insights:

- Action space representation matters significantly for learning dynamics
- Discretization is not always detrimental; it may simplify exploration
- Stability should be evaluated throughout training, not just in aggregate
- Neither algorithm converged within 900 episodes, suggesting CarRacing-v3 is a challenging benchmark requiring extended training or algorithm improvements

7 Conclusion

We presented a comprehensive empirical comparison of Deep Q-Network (DQN) and Proximal Policy Optimization (PPO) on the CarRacing-v3 visual control task, focusing on the trade-offs between

discrete and continuous action space representations. By using the same CNN feature extraction architecture for both algorithms, we isolated the impact of action space discretization and learning dynamics on performance.

Our key findings demonstrate that DQN with discrete actions achieved superior final performance (877.86 ± 40.24) and stability compared to PPO with continuous actions (752.43 ± 199.79) over 900 training episodes. While PPO showed smoother early learning and lower overall variance, DQN's off-policy learning with experience replay led to more consistent performance in later training stages. Neither algorithm reached full convergence within the training budget, highlighting the challenging nature of the CarRacing-v3 benchmark.

These results challenge the common assumption that continuous action spaces are always preferable for control tasks. For problems where actions admit natural discrete decompositions, discretization combined with value-based methods can achieve strong performance while simplifying exploration. However, the choice between discrete and continuous representations should be guided by task characteristics, computational constraints, and stability requirements.

Our modular implementation provides a foundation for future research exploring algorithmic improvements, extended training, and hybrid action space representations. We hope this work provides useful insights for practitioners designing RL systems for visual control tasks and motivates further investigation into the interplay between action space representation, algorithm design, and learning dynamics.

Code Availability: Our implementation is organized as a modular reinforcement learning framework with separate components for agents, models, environments, and utilities, enabling easy extension and reproduction of our experiments.

Acknowledgments

We thank the Gymnasium team for maintaining the CarRacing-v3 environment and the PyTorch team for their excellent deep learning framework.

References

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *International conference on machine learning*, pages 1587–1596, 2018.

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

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning*, pages 1861–1870, 2018.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *Proceedings of the AAAI conference on artificial intelligence*, 32(1), 2018.

Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, et al. Scalable deep reinforcement learning for vision-based robotic manipulation. *Conference on robot learning*, pages 651–673, 2018.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810, 2018.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. *International conference on machine learning*, pages 1889–1897, 2015a.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

Mark Towers, Jordan K Terry, Ariel Kwiatkowski, John U Balis, Gianluca de Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Arjun KG, Markus Krimmel, et al. Gymnasium. *Zenodo*, 2023.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence*, 30(1), 2016.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. *International conference on machine learning*, pages 1995–2003, 2016.