Project Report: PPO for CarRacing-v3

This report details the implementation of a Proximal Policy Optimization (PPO) agent for the `CarRacing-v3` environment. The following sections describe the key components of the project.

1. Preprocessing Pipeline

The raw observations from the environment are 96x96 pixel color images. To make this data suitable for the neural network and to provide temporal context, the following preprocessing pipeline is used:

- **Grayscale Conversion:** The RGB frames are converted to a single-channel grayscale image using `cv2.COLOR_RGB2GRAY`. This reduces the input dimensionality and simplifies the learning task.
- **Frame Resizing:** The frames are resized from 96x96 to a standard 84x84 pixels. This reduces the computational load on the network.
- **Normalization:** The pixel values, originally in the range [0, 255], are normalized to be between [0, 1] by dividing each pixel value by 255. This helps stabilize the training process.
- **Frame Stacking:** To give the agent a sense of motion, velocity, and direction, the last **4** consecutive preprocessed frames are stacked together. This results in an input state with the shape `(4, 84, 84)`.

2. PPO Architecture & Hyperparameters

The agent uses an Actor-Critic architecture where both networks share a common feature extraction backbone. To simplify the learning challenge, the agent operates on a discrete action space.

Discrete Action Space

Instead of choosing continuous values for steering, throttle, and braking, the agent selects one action from a pre-defined menu of 5 options. This "hard discretization" simplifies the decision-making process significantly. The mapping is as follows:

Discrete Action	Continuous Action `[Steer, Gas, Brake]`	Description
0	`[-1.0, 0.0, 0.0]`	Turn Hard Left

1	`[+1.0, 0.0, 0.0]`	Turn Hard Right
2	`[ 0.0, 0.0, 0.8]`	Brake Hard
3	`[ 0.0, 1.0, 0.0]`	Accelerate
4	`[ 0.0, 0.0, 0.0]`	Do-Nothing (Coast)

PPO Architecture

- **Shared CNN Backbone (`cnn_base`):**
  - **Input:** A `(4, 84, 84)` tensor of stacked frames.
  - **Layer 1:** `Conv2d` with 8 filters, kernel size 4x4, stride 2.
  - **Layer 2:** `Conv2d` with 16 filters, kernel size 3x3, stride 2.
  - **Layer 3:** `Conv2d` with 32 filters, kernel size 3x3, stride 2.
  - **Layer 4:** `Conv2d` with 64 filters, kernel size 3x3, stride 2.
  - The output is flattened and passed to a `Linear` layer with 100 units.
- **Actor Head (Policy):**
  - Takes the 100-unit feature vector from the shared layers.
  - Has a single `Linear` output layer that produces logits for the 5 discrete actions. A Categorical distribution is then used to sample an action.
- **Critic Head (Value Function):**
  - Takes the 100-unit feature vector from the shared layers.
  - Has one `Linear` output layer that produces a single scalar value, representing the estimated value of the current state.

Hyperparameters

Hyperparameter	Value	Description
`learning_rate`	`0.0001`	The learning rate for the Adam optimizer for both networks.
`gamma`	`0.99`	The discount factor for future rewards.
`ppo_clip`	`0.2`	The clipping parameter for the PPO objective function.

`batch_size`	`64`	The size of mini-batches used during the update phase.
`ppo_update_time`	`10`	Number of epochs to train on the collected data in each update.
`entropy_coef`	`0.05`	The coefficient for the entropy bonus to encourage exploration.
`update_timestep`	`2048`	The number of experiences to collect before running an update.

3. Training Process

The agent is trained over a series of episodes. The training loop proceeds as follows:

1. **Data Collection:** The agent interacts with the environment, collecting 2048 steps of experience (states, actions, rewards, etc.) into a buffer.
2. **Policy Update:** Once the buffer is full, the `update()` function is called. It calculates the discounted rewards-to-go and advantages, then updates the Actor and Critic networks for 10 epochs using mini-batches of the collected data.
3. **Logging and Saving:** Every 10 episodes, the average reward is calculated and printed. A checkpoint of the model's parameters is also saved to allow for resuming training.