

A Keras-Based DQN Agent Achieving 800+ Average Reward in CarRacing-v3 with Domain Randomization

Wei-Chi Aeneas Hsu

College of Semiconductor Research

National Tsing-Hua University

Hsinchu, Taiwan

weichihsu1996@gmail.com

Abstract—This work revisits the capabilities of Deep Q-Networks (DQN) in complex, partially observable environments by constructing a high-capacity, modular DQN agent capable of achieving an average score of 828 over 1000 evaluation episodes in the randomized CarRacing-v3 benchmark. The agent is implemented entirely in TensorFlow/Keras without reliance on policy-gradient methods, distributional value estimation, or NoisyNet-style stochastic exploration. Instead, it leverages architectural overdesign—namely, multiple residual CNN branches, dropout-regularized ensemble Q-heads, and handcrafted visual preprocessing (contrast, Sobel, Gabor)—to achieve performance and behavioral diversity.

Key insights from this work include: (1) ensemble Q-head diversity induces robust, non-greedy behavior even without noise injection; (2) large model size does not necessarily lead to overfitting under domain randomization; and (3) failure cases such as directional confusion reveal architectural limitations (e.g., lack of temporal memory) rather than reward design flaws.

While the approach is not intended to outperform Rainbow or other policy-gradient baselines, it demonstrates that value-based agents—when equipped with sufficient structural diversity—remain competitive and interpretable. The full implementation, training logs, and evaluation visualizations are released to support future reproducibility and architectural ablation studies.

Index Terms—Artificial Intelligence, Bio-inspired Agents, CarRacing-v3, Deep Q-Network, Domain Randomization, Open Source, Reinforcement Learning

I. INTRODUCTION

Solving the Gymnasium CarRacing-v3 environment [1] under domain randomization (the author will use the CRV3-DRT as an abbreviation in this article – CarRacing V3 Domain Randomize = True) has long served as a challenging benchmark in reinforcement learning (RL) [2], primarily due to its high-dimensional visual inputs, sparse reward structure, and complex control dynamics. While policy-based methods such as Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), and Advanced Actor-Critic (A2C) have shown strong performance [3], value-based approaches like Deep Q-Networks (DQN, in 2015, it has been shown that DQN is able to reach human-level control [4], which is a milestone of modern RL), Dueling DQN (DDQN, DeepMind [5]), and Double DQN (another DDQN, DeepMind [6], and if we combine Dueling DQN and Double DQN, we will have DDDQN) are often considered inadequate for CarRacing v3, especially under randomized conditions. To the best of

the author’s knowledge, no existing study has systematically evaluated reinforcement learning agents on CRV3-DRT. Naive DQN in v3 typically collapses (the author tried it once). So, in brief, the mainstream tends to use policy-based approaches, such as PPO and A2C, rather than value-based methods (DQN, DDQN, another DDQN, or DDDQN). Note: value-based methods mean the gradient operates on the Q-function, while the policy-based methods’ gradient operates on the policy π .

Although DeepMind’s RainbowDQN [7] demonstrated that hybrid DQN architectures can approach or surpass human-level performance in various tasks, its internal components remain partially opaque. To date, there has been no accessible, fully open-sourced DQN implementation tailored for CarRacing-v3 with domain_randomize=True.

This work introduces an open-sourced Lite Rainbow-like DQN agent built in Keras that achieves an average score exceeding 800 over 1000 episodes in the randomized CarRacing-v3 environment. The author chose to implement in Keras to minimizing the efforts (time \times energy). Instead of relying on PPO-style actor-critic frameworks or extensive hyperparameter tuning, the agent employs a biologically inspired architecture combining multi-branch visual processing (contrast enhancement, edge detection (Sobel), and Gabor filtering), a shared dense MLP bottleneck, and an ensemble of dueling Q-heads (but the author inserted dropout layers into those Q-heads). The agent is trained solely with default Keras components (Huber loss - check the implementation on [8] - and Adam optimizer), using no task-specific reward shaping aside from minimal penalties for inaction.

Originally developed as an educational side project, this agent’s surprising performance, reproducibility, and architectural transparency suggest its potential as a community baseline for value-based RL in visually rich continuous environments. The author does not claim state-of-the-art (SOTA) performance, but instead offers this work as an accessible and interpretable starting point for further exploration in DQN-based agents under domain randomization. This work does not propose new theoretical components, as the central challenge in CarRacing-v3 lies not in loss function novelty but in achieving stable and interpretable control under extreme visual diversity. The value of this agent lies in its reproducibility and empirical and practical ability for value-based RL.

A. A Brief Review of Deep Q-Networks

We use the following formula to predict the Q-value. The Q-output is a vector, in which the $Q = [Q(s_1, a_1), Q(s_1, a_2), \dots, Q(s_1, a_5)]$.

Q is the expected reward after taking an action. Therefore, suppose we have a Q vector, once the agent knows the $Q = (a_1 = 1.2, a_2 = 2.3, a_3 = 0.3, a_4 = 0.3, a_5 = 3)$, the agent will choose action = argmax(Q). So, the problem is how to have a Q-value prediction machine telling the agent how to select an action from all available actions (called the action space). In this context, the purpose of Q-learning is to build a machine to predict the future reward according to the present state and action. In modern value-based RL, neural networks are used to approximate the real Q-function, even though we may be unlikely to know what the real Q-function looks like.

In the Q-head, we have two sub-networks, called V-value stream, and another is called Advantage stream. The presence of V-value is an estimated reward, so the V-stream outputs a scalar, while the A-stream gives the A-value of actions. To input the Q-value for a given action, the formula is $Q(s, a) = V(s, a) + (A(s, a) - \max A(s, a'))$. Note that, in implementation, the V is a scalar, but the Q is a vector (size = action dimension). This makes the thing a bit tricky. Exactly, the V is reward (or value) in the present. But the $A(s, a')$ is the vector based on the state (the input image) of each action (5 in the CarRacing v3).

For simplicity, we can think of this as $Q = V + (A - A_{\text{mean}})$.

$V(s)$: the scalar state-value, estimating how good the current state is regardless of action.

$A(s, a)$: the advantage of each action, representing the benefit of taking action a in that state compared to the average. Therefore, the formula used in this work is

$$Q(s, a) = V(s) + \left(A(s, a) - \frac{1}{A} \sum_{\{a'\}} A(s, a') \right). \quad (1)$$

$V(s)$ is a single scalar per sample: shape = (batch, 1). $A(s, a)$ is a vector for each sample: shape = (batch, action_dim). The subtraction of the mean of $A(s, a')$ normalizes the advantages to zero-mean, preventing the network from becoming biased in estimating Q.

In a simple DDQN model, the neural network model is used to predict the Q-value vector according to the input state (an image, in implementation, the author used stacked frames, so there will be 12 images (4 frames \times 3 RGB-channels) as the input).

Now, we turn to the Double network structure. In brief, the Double architecture uses two models (but the models' structures are the same). One is to predict the Q-value, which gives the selected action. One is used to estimate. The formula used to calculate the temporal difference error (TD error, $\delta = Y - Q(S_0, A_0)$, where $Y = R + \gamma Q(S_1, A_1)$) is given below (γ is the discount factor, $\gamma \in [0, 1]$)

$$Y_t = R_{t+1} + \gamma Q(S_{t+1}, \arg(\max(Q_{\text{target}}(S_{t+1}, a)))) \quad (2)$$

Recall: $Q_\pi = \mathbb{E}[R_1 + \gamma R_2 + \dots]$ under the input $s = S_0$, $A_0 = a$ and a given policy π . Owing to the Markov Decision

Property, we can set any state to be S_0 and calculate the future expected reward.

Sum up, we then have the loss function \mathbb{L} to be $\mathbb{L} = \text{Huber}(\delta)$, which is a mixed version of MSE (Mean Square Error) and MAE (Mean Absolute Error):

$$\mathbb{L}_\beta(\delta) = \text{case} \begin{cases} \frac{1}{2}\delta & \text{if } |\delta| < \beta \\ \beta(|\delta| - \frac{1}{2}\beta) & \text{else} \end{cases}, \quad (3)$$

where the β is a hyperparameter, which used to be 1. For large δ , the \mathbb{L} becomes MAE, and for small δ , the \mathbb{L} becomes MSE. This is used to prevent gradient explosion.

After we finish the trickiest part, we go over the whole design. We can imagine that the CNN branches are the image feature encoders, and we use them to map the information of images into latent spaces. Here is why the author added a dense layer after the CNN branches. Since we have processed the image-to-vector step, we then use a fully connected MLP bottleneck for non-linear transformation, which plays the role of a high-order information compressor. Finally, we use ensemble dueling Q-heads (10 in this agent) to predict the Q-value.

B. Problem Setting and Motivation

The problem addressed in this work is the development of a DQN-based agent capable of achieving an average score of 800+ over at least 1000 evaluation episodes in the Car-Racing-v3 environment with `domain_randomize=True`. This setup is intentionally chosen to test the viability of DQN approaches. The author aims to reassess the design space of value-based agents, focusing on whether architectural creativity and robustness can compensate for the absence of gradient-based policy optimization.

Instead of following the Rainbow DQN recipe directly, the author deliberately omits several components to simplify implementation and maintain interpretability. In particular, NoisyNet is replaced by a scheduled epsilon-greedy exploration strategy, and Distributional Q-learning is excluded in favor of the standard Huber loss, avoiding complex loss definitions. Nonetheless, key infrastructure elements—such as double Q-networks, dueling architecture, multi-step learning, and prioritized replay—are retained to provide a stable foundation.

This design reflects a practical motivation: to build an open-sourced DQN agent using only default Keras components, without relying on hand-crafted reward shaping (only slightly punishing inaction) or fine-grained hyperparameter tuning. In brief, the author asked: Is it possible to build up a DQN agent that survives in CRV3-DRT?

C. Design Philosophy and Contributions

Given the randomness of the environment, overfitting becomes unlikely; thus, the author opts not to constrain the model scale (the released model is approximately 120 MB), focusing instead on structural diversity. Standard training components such as Huber loss and Adam optimizer are retained to reduce complexity and maintain reproducibility.

The key contributions of this work include:

- An open-sourced, reproducible Keras-based DQN achieving an average score of 800+ over 1000 evaluation episodes on CarRacing-v3 under domain randomization.
- A multi-branch perception encoder and ensemble dueling Q-head architecture that promotes both robustness and interpretability.
- A complete runnable notebook with pretrained weights, visualizations, and analysis tools—designed to serve as both an educational resource and a performance benchmark for DQN-based methods.

While not claiming to reach SOTA performance, the author aims to offer a practical and accessible baseline for future experimentation. For use in other tasks or environments, architectural modifications are expected and encouraged. All training and evaluation steps are fully reproducible via our public Colab notebook/Jupyter Notebook and GitHub Repo. Also, the generated GIF files are available for access.

II. METHODS

In this section, we discuss the model and the technical details. The author skips the theoretical discussion, such as how the dropout ensemble Q-head plays a similar role to NoisyNet and how the author knows that adding more Q-heads leads to behavior diversity and stable control and how the author chooses the hand-crafted visual preprocessing filters and how the author knows the residual in CNN branches can lead to enhanced performance. These decisions are driven by empirical outcomes and can be further explored in future ablation studies (but the author anticipates that the ablation study will encounter a combination explosion because of too many features in the model).

A. Comparison between Rainbow and this work

While RainbowDQN represents an elegant integration of multiple DQN enhancements, the agent proposed in this work adopts alternative mechanisms, such as using dropout-regularized dueling Q-heads instead of NoisyNet, and ensemble Q-value heads instead of distributional value estimation. Notably, the author deliberately opts for a higher-capacity model with redundant structures, reasoning that overfitting is unlikely under a domain-randomized setting, and larger models may promote representational richness and robustness.

The comparison table Table I summarizes the architectural and methodological differences. This comparison is intended purely for clarity, not to imply superiority. Rainbow remains a seminal work in deep reinforcement learning, and the listed characteristics reflect our best understanding of public implementations and documentation. The author expresses sincere respect for the contributions of the original DeepMind team.

B. Model

The total parameter of the model is in Table II. The overall size is about 119 MB. Compared to typical DQN models and PPO models, this size is almost 3X larger or 4X larger. Readers may wonder whether this model size could cause overfitting. The author tends to argue that, considering the nature of `domain_randomize = True`, each scene is randomly generated. This fact may prevent overfitting or mitigate

TABLE I: TECHNIQUES COMPARISON

item	AeneasDQN	RainbowDQN
Frame	TensorFlow/Keras	DeepMind Internal(original Rainbow was not fully open-sourced at the time)/ Pytorch/TF2(most implementation)
Environment	CarRacing-v3 with domain randomize	Atari (mostly fixed-pixel, no domain randomization)
Feature Extraction	Contrast / Edge / Raw / Gabor	Raw Pixel + CNN
Network Structure	Multiple residual CNN branches, latent bottleneck, Ensemble dueling Q-heads	Single CNN, Deuling Q-head
Exploration	epsilon greedy	NoisyNet
Loss Function	Huber	Distributional (C51 loss or quantile regression)
Replay Buffer	Prioritized Multi-Step Replay	Prioritized Replay
Optimization	Adam, learning rate = 0.00025	RMSProp or Adam
Q-value	Multiple Q-heads	Distributional-Q
Dropout	Embedded in Q-head	Not reported in original Rainbow
Model Size	120 MB	10–20 MB (typical RainbowDQN implementations)
CarRacing v3	811 (average over 100 randomized runs)	No reported

TABLE II: TECHNIQUES COMPARISON

Parameter Type	Number	Size
Total params	31,210,293	119.06 MB
Trainable params	31,202,549	119.03 MB
Non-trainable params	7,744	30.25 KB

this issue to some extent. However, it does not mean that the size reduction is infeasible. For the distillation, the author believes the model size can be efficiently reduced.

a) *Overview:* Fig. 1 summarizes the model structure, which is complicated even from the author’s perspective. The released model’s ancestor has only one CNN branch and a simple MLP core with 3 Q-heads. After several generic algorithm iterations, the model scale evolved and became larger. The proposed DQN agent adopts a modular architecture comprising multiple visual preprocessing branches, a shared latent bottleneck, and an ensemble of dueling Q-heads ($5 \text{ types} \times 2 = 10$ heads). Each module is intentionally designed to promote both feature disentanglement and robustness under domain randomization.

b) *Multi-Branch Visual Encoder:* To enhance visual robustness, the agent processes raw observations through multiple parallel feature extractors:

- $\times 0$ Raw input stream (for baseline reference)

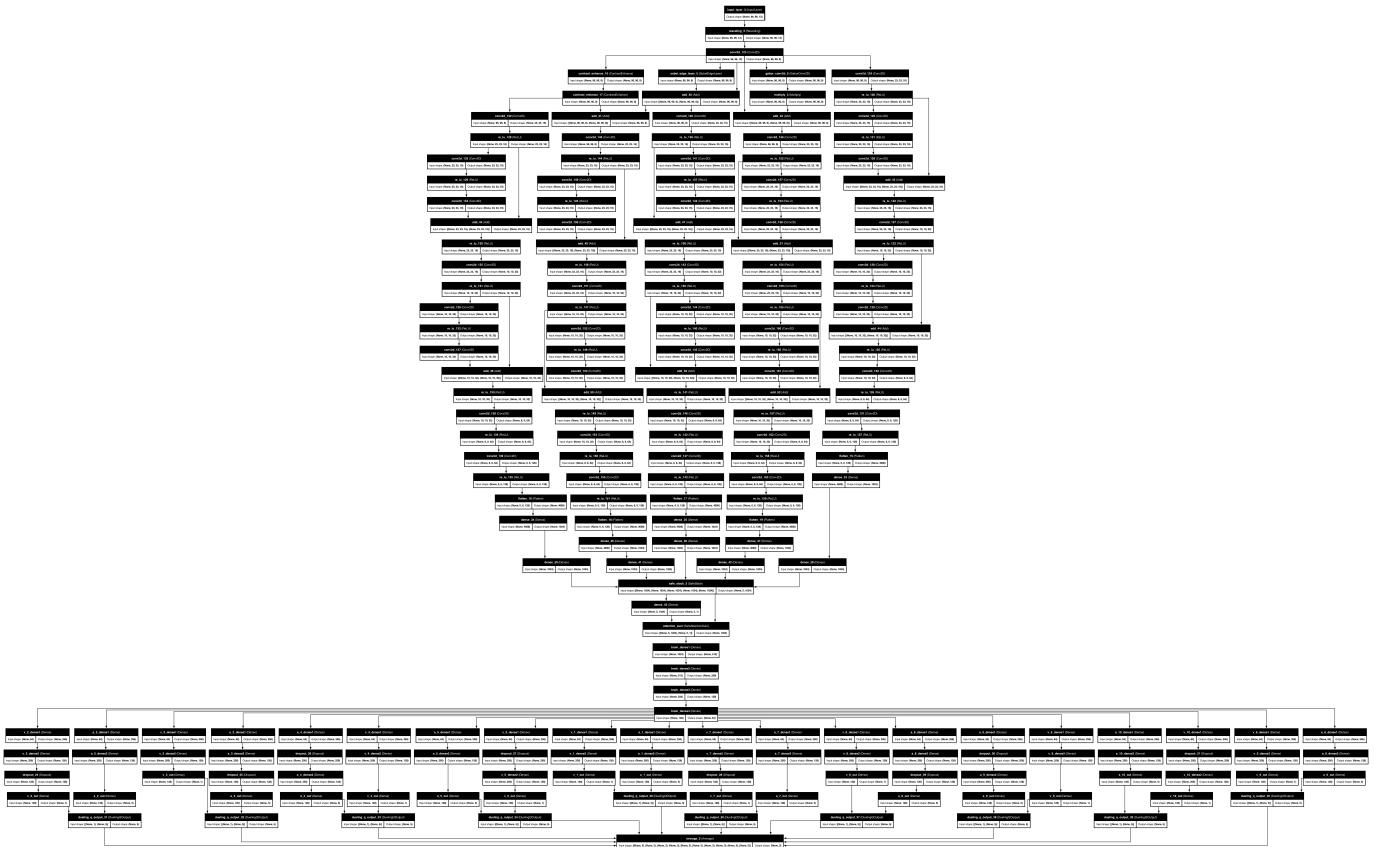


Fig. 1: The model structure summary. Five CNN branches and ten dropout-embedded Q-heads are presented. The author understands that this figure is not reading-friendly. The original version can be found in the GitHub Repo. The model's architecture looks like an octopus. The author designs this architecture by inspiration from octopuses.

- x1 Contrast-enhanced (twice) stream (simple gamma correction)
- x2 Edge stream (Sobel filters) + x0
- x3 Sobel and Contrast-enhanced
- x4 Gabor + x0

Each stream is fed into a separate convolutional block with residual connections. Before the branch merge, a 1024-neuron dense layer was added to each branch to mimic the preliminary visual cortex. All feature maps are concatenated before entering the next stage.

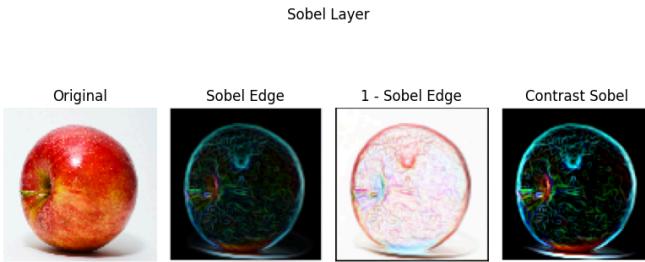


Fig. 2: The Sobel Layer. The apple image was used to demonstrate.

In Fig. 2, the author used an apple image from Wikipedia to show the effects of Sobel layer and the combination of the Sobel with the Contrast enhancement. In Fig. 3, the author demonstrated the processed image for demonstration

the effects of visual preprocessing. In the training process, the input CarRacing image deviates from the Apple image case.

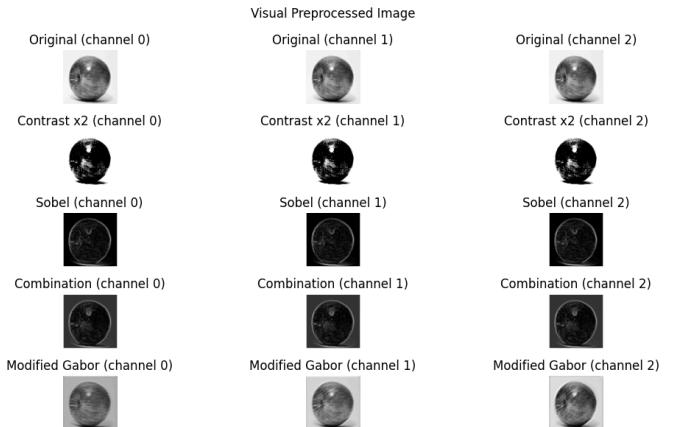


Fig. 3: Visual Processing Stream.

c) *Latent Bottleneck (Central Workspace)*: The concatenated outputs from the encoder branches are passed through a dense multi-layer perceptron (MLP) that acts as a latent “core” or bottleneck. In the model, the bottleneck MLP is [512, 256, 128, 64]. This core serves as a shared representation space from which Q-values are computed.

d) *Ensemble of Dueling Q-heads*: From the latent bottleneck, multiple parallel dueling Q-heads are constructed. Each head has:

- A value stream (V)
- An advantage stream (A)
- Dropout regularization during training
- Slight structural variation (dropout after different layers and in V/A stream)

Final Q-values are averaged across all heads. This ensemble promotes exploration diversity and stabilizes learning under high environment stochasticity.

e) *Loss Function and Optimization*: The model is trained using the standard Huber loss with double DQN target updates. Adam optimizer is used with a learning rate of 0.00025. No task-specific loss functions are introduced to keep the training pipeline reproducible and interpretable.

f) *Replay Buffer and Exploration*: The agent uses a prioritized multi-step replay buffer (7-step TD target) and a scheduled ϵ -greedy exploration policy (based on total steps). In the early training phase, the total step increase was less until the reward curve began to increase. For example, when episode < 5000 , the agent may get terminated in 100 steps; On the other hand, after training 10,000 episodes, the average steps in an episode can be 600 or 800 until the agent gets terminated. The author used exponential decay epsilon with regard to total steps instead of episodes.

The author deliberately avoids NoisyNet and instead leverages dropout and ensemble uncertainty as stochastic exploration aids. For the author, this is an effort-cheap manner of implementation. However, it does not mean that distributional-Q and NoisyNet do not work in CarRacing v3.

g) *Training Loop*: The author used the frame stacking (4 frames for RGB $96 \times 96 \times 3$ image as input). In the first 500 episodes, the author did not use the prioritized replay (dynamically switching memory). This is for collecting experiences for agents. After memory switched, in general, the reward curve increased. The reward-shaping only punishes inaction (reward -0.1). Also, no-op was used.

III. RESULTS

In this section, the author presents the results run on the author's local MacBook Air M2 laptop. The reader can download the original Jupyter Notebook to train and evaluate. The evaluation is based on the released trained model by the author.

A. Evaluation Protocol

The Standard setup of evaluation is under 1000 episodes using stochastic inference (`model_trainable = True`). The environment is CarRacing v3 `domain_randomize = True`, the recorded metrics are average score, standard deviation, max. and min. scores. All GIF files generated by each evaluation are also recorded for failure analysis. The Frame stack (`NUM = 4`) is used. The recorded reward is not shaped.

B. Main Quantitative Result

The 1000-episode evaluation under `domain_randomize = True` is in Table III. It is evident that the agent reached the

average of 828.56. The result is plotted in Fig. 4. So, the goal has been achieved. However, the author found that the std is about 114.05. Could stochastic inference cause this because the author used `model_trainable = True` mode? The author thinks the answer is not really. First, the total reward is not the only indicator of an agent's performance; average reward over 1000 episodes is necessary, but not only. The reward distribution (Fig. 4 bottom) of all trials provides more details: (1) only rare collapse cases in 1000 trials, which means that the agent countered tricky environments instead of the agent not generally surviving in the CRV3-DRT (Fig. 4 middle); (2) if the excluded the extreme collapse trials, the author found the average was about 850 and the score distribution was like a gaussian distribution with high-performance trials (but this just shows that the agent can perform well in some cases, roughly 10% chances that the agent can reach above 900. Compared to the collapse rate, roughly 1.2%, it seems acceptable.) ; (3) The total reward of a single trial and the average and standard deviation of 1000-episode evaluation are not sufficient to comprehensively describing the agent's ability. The author would like to add an additional requirement to the agent: it must drive smoothly rather than a machine-like agent. For instance, in some cases (not the author's work), the author found that some algorithms lead to a shaky agent – the agent tends to use "turn-left" and "turn-right" to maintain equilibrium on the track.

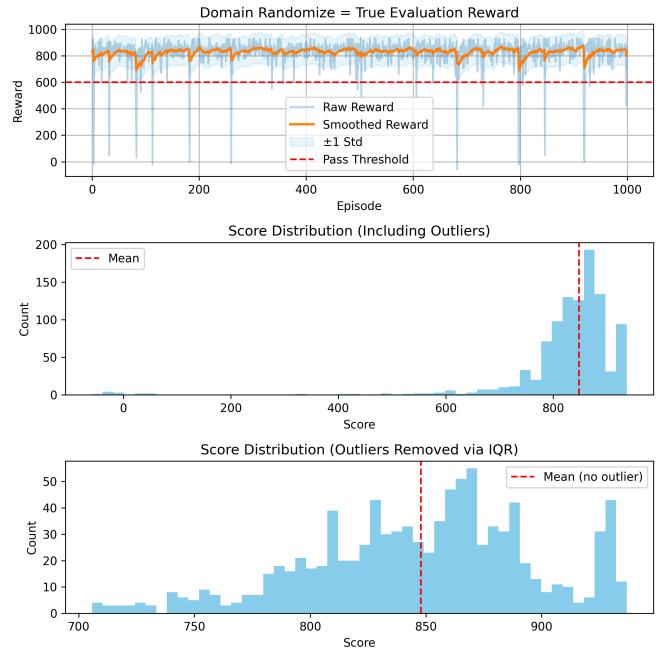


Fig. 4: 1000-episode evaluation under `domain_randomize = True` condition. The `max_step = 1000` was set for each trial. The average score is 828.56 (including extra-low trials) with a standard deviation of 114.05.

The next question is how to handle and explain why the agent failed. Collapse events might stem from the visual failure because the author used visual tricks before CNN branches, such as ContrastEnhancement, SobelLayer, GaborLayer, and merged visual prepro-

TABLE III: THE EVALUATION OVER 1000 EPISODES

Metric	Value
Mean	828.56
Max.	936.80
Min.	-58.33
Std	114.05
25%	809.41
50%	848.80
75%	878.70

cessing layers. For specific cases, such as low contrast (the author will address this issue later). Hand-crafted, intuition-driven visual tricks could be a double-edged sword, even though the author used multiple CNN branches to suppress(or at least mitigate) the occurrence of extreme failure.

In Fig. 4, the evaluation result shows that the agent reaches an average of 800+ with variation. The score distribution indicates that most cases fell on the right side of the average line (red line). From the distribution plot, the author found that those failure cases (score below 500) are worth further study.

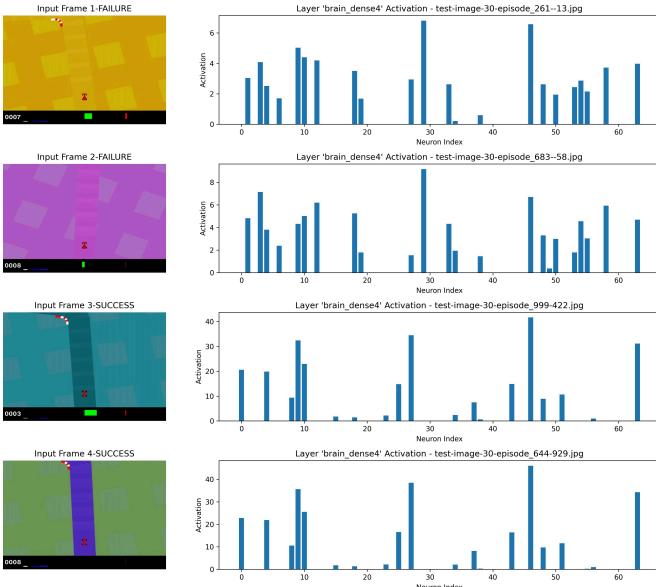


Fig. 5: Failure Analysis (brain_dense4)-64 neurons. In the model’s architecture, this layer is the final fully connected MLP bottleneck connecting to the ensemble Q-heads. The top 2 images were extracted from failure trials (negative rewards, less than 0), and the bottom 2 images were from the success trials (one > 400, one > 800).

In Fig. 5, the author compared two scenarios (2 frames from failure trials, 2 from success trials). Because there are 1000-episode GIF logs, the author decided to select two worst-performing (failure) trials and two success trials as a case study target. By comparing the extreme cases, the author found that one of the critical MLP layers (the last layer of MLP called “brian”, which consists of 64 neurons) was the test target. The author found that the neuron activation plots reveal the distinguishing patterns under two visual environments. In a high-contrast environment, the agent has a similar neuron activation distribution. Similarity in neuron activation

patterns suggests that for some extreme environments, the agent’s visual ability fails, reflected in the neuron activation profile, possibly due to the fuzzy boundary. On the other hand, this also serves as evidence showing that the agent has adapted to the randomized CRV3-DRT. Although the environmental background has a distinct color style, the agent exhibits a similar pattern of neuron activation profile.

While the presented average score over 1000 evaluation episodes provides a common benchmark for performance, it is by no means sufficient to fully characterize agent behavior. In this work, the author highlights the importance of performance distribution and qualitative failure analysis. Notably, inspecting low-scoring episodes reveals distinct failure modes, such as direction confusion (inverse driving) and visual misinterpretation under extreme textures.

However, due to resource constraints, a full-scale taxonomy of failure cases was not conducted. Instead, selected episodes were manually reviewed through GIF logs, revealing behaviors such as V-shaped trajectories for risk avoidance, and strategic shortcuts involving off-road driving in exchange for long-term gains.

The author encourages readers to examine the provided 1000-episode visual logs for deeper insights into agent decisions and potential areas for robustness enhancement.

C. Qualitative Result

The author mentioned that the ensemble Q-heads might contribute to the behavior diversity and make the agent able to survive in the CRV3-DRT environment. To investigate further, the author fed failure and success cases into the model and observed the neuron activation of all ten 10 Q-heads.

In Fig. 6, the author found distinct activation patterns of Q-head A-stream dense-2 (128 neurons) layers. This means that when the agent saw different track shape scenarios in the same background, the Q-heads had distinct responses, which supports the hypothesis that Q-heads contributed to the agent’s ability to survive. However, the author also found that there are 4 dead layers in this environment. Therefore, the author suspected that the dead layers (Q3, Q4, Q8, Q9) may play specific roles in other extreme environments.

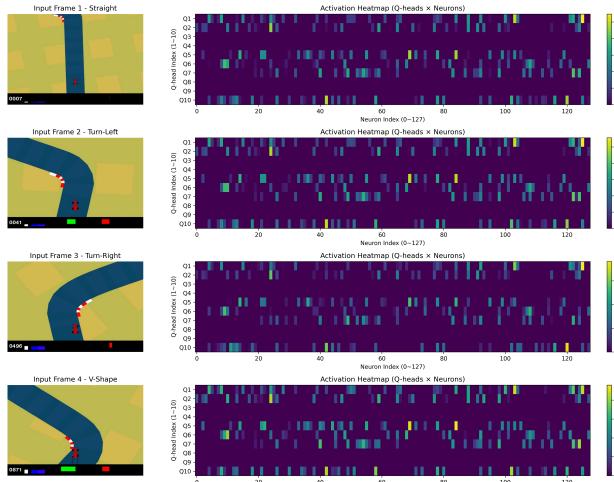


Fig. 6: Q-heads A-stream dense-2 neuron activation of 4 track scenarios of a successful case (episode 933). The author found 4 dead layers in this episode: Q3, Q4, Q8, and Q9.

The author tried to figure out whether the Q3, Q4, Q8, and Q9 are dead layers, so the author used the same frames in Fig. 5 to test those silent layers (Q3, Q4, Q8, Q9). The author confirmed that the silent layers in the selected success trials are not really non-functional because the author captured (in Fig. 7) the activation signals of Q3, Q4, Q8, and Q9 A-stream dense-2 layers. Therefore, the author deduced that the Q3, Q4, Q8, and Q9 heads are specialized for those vague environments. Even though the author did not provide a rigorous theoretical proof of the hypothesis that the ensemble Q-head could increase the ability to survive, by observing the activation signals under distinct environments and scenarios, the author gained insights that suggest that the ensemble Q-head could function, and they could be specialized to distinguish specific environments.

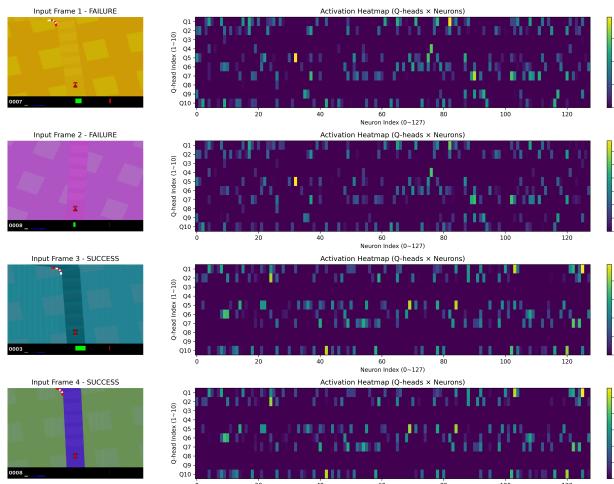


Fig. 7: Q-heads A-stream dense-2 neuron activation of 4 scenarios: 2 are failure trials and 2 are success trials. The author noted that in failure cases (fuzzy boundary, low color contrast), the silent layers (Q3, Q4, Q8, Q9) have response (activation signals).

To demonstrate the agent's ability in CRV3-DRT, the author randomly sampled 10 cases from the 1000 episodes evaluation database. In Fig. 8, it is clear that the agent has the ability to adapt and survive in the CRV3-DRT. And also, we can find that Q3, Q4, Q8, and Q9 are only activated under vague environments (one case in the figure). This could be evidence to prove that this DQN agent is able to handle randomized environments in CRV3-DRT. A crucial observation is that, under distinct color with similar track shapes, the Q-heads activation patterns are similar, which is indirect evidence that Q-heads learnt internal features. Therefore, the ensemble Q-head architecture enhances the agent's ability to survive in CRV3-DRT. Furthermore, it could be possible that a larger model has more capability for the agent to survive, which means redundancy is not a bad thing when the environment variation is sufficiently large to prevent overfitting. In other words, if the environment dynamically changes, the agent requires a larger model capacity to survive. This is an intuition inspired by biology.

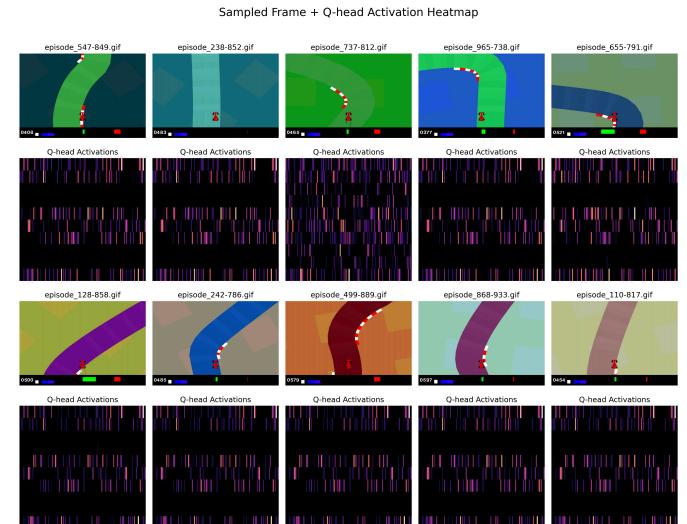


Fig. 8: Sample frames (10 cases) from 1000-episode evaluation GIF log. The filename indicates the score of trials. The 10 Q-heads activation (heatmap) is plotted below each sampled frame. This figure demonstrates that the dead layers are not exactly dead but specialized to some scenarios.

In a small subset of low-scoring episodes, the agent drives in the wrong direction after re-entering the track post-collision. This suggests that long-term direction memory may be limited (commonly happens in DQN scheme), opening a path for memory-augmented or recurrent architectures. This suggests that two possible paths for future work: (1) More precise control, or (2) adding memory-like modules to the agent's architecture. The author tried the second approach twice (LSTM and Attention), but the experiments failed. So, the author gave up this approach and turned to approach 1: adding more Q-heads.

In narrow or visually ambiguous segments, the agent sometimes demonstrates a “V-shape” avoidance pattern, slowing down and avoiding risky acceleration, which appears to be an emergent behavior from the contrast/edge-preprocessing design. The author suspected that, in V-shape trace, the input

image was different from normal cases (single trace). This might trigger a specific part of the network. Importing special cases (i.e., V-shape images) and feeding them into the model , then observing the hot-spot of the network (both dense layers and CNN) could reveal more details of this agent, but further investigation is needed. Due to the effort-time limitation, the author left this work for future work.

Despite no explicit reward shaping, the agent autonomously discovers shortcuts in randomized terrains (e.g., diagonally cutting corners through low-penalty grass). This highlights the exploratory diversity induced by the multi-head ensemble design. Another interesting observation is that the agent seems to dislike using the “brake”. The agent tends to use the rotating speed-down instead of the brake, according to the author’s observation (the author had watched more than 100 GIF logs)

Some failure cases were caused by extreme background or lighting configurations (as seen in GIF logs), where the agent likely fails due to its reliance on visual contrast/edge channels. Note that the author customize the visual preprocessing layers for CarRacing v3. Therefore, this means that if the game is changed, the visual preprocessing needs to be manually modified, even using some intuition. This points to the limitation of handcrafted preprocessing under domain randomization extremes. However, the author trained this agent in `domain_randomize = True` environment (all 20,000 episodes were randomized), which implies that the agent fit the environment just like surviving in a jungle or ocean. So, it is normal to expect that an agent that survived in a jungle could not survive in the ocean. In brief, under this approach, the generalization ability of this architecture to other games is limited.

D. Runtime / Training Efficiency

The agent was trained for 20,000 episodes with `max_step = 1000` on the author’s M2 MacBook Air. Total training time was approximately 4 to 5 days. The model converged without the use of early stopping or manual hyperparameter tuning.

IV. DISCUSSION

This work demonstrates that a deliberately over-parameterized DQN agent—when equipped with modular visual processing and ensemble Q-head structures—can achieve competitive performance in domain-randomized environments without resorting to policy gradients, NoisyNet, or distributional losses.

Several findings emerged through empirical observation:

- 1) **Ensemble Q-heads as a source of behavioral diversity:** Without any injected noise or distributional estimation, structurally diverse Q-heads (with dropout) led to a consistent emergence of robust driving policies. This suggests that architectural variation alone can support policy diversity, a principle potentially extendable to other value-based methods. Furthermore, in this agent, the author found that 4 Q-heads were specialized to specific environments, which was not expected by the author.
- 2) **Overfitting is not a limiting factor under domain randomization:** Despite the model’s large capacity

(~120MB), we did not observe overfitting symptoms (each environment is randomly generated). The author trained the agent in randomized environments only, so the agent must try to survive in the CRV3-DRT. This raises the question: Is domain randomization a viable form of regularization for value-based agents, allowing for overbuilt architectures without risk of memorization?

- 3) **Failure cases highlight architectural—not reward—limitations:** After conducting a failure case analysis, the author found that the color, similar track shapes are not vital factors for the agent’s failure (low score trials), but the visual collapse. This directly leads to an insight for solving CRV3-DRT - modify the visual inputs before the CNN branches, rather than extracting features by using the CNN backbone only. The author mimicked the biological eyes to hand-craft the visual preprocessing layers to ensure the agent has better visual ability for survival.

Looking forward, three key design tensions are identified:

(1) The balance between behavioral diversity and architectural efficiency, specifically comparing ensemble Q-heads with dropout-induced variance versus more traditional stochastic exploration methods such as NoisyNet or internal noise injection.

(2) This work adopts dropout-embedded Q-heads and maintains dropout activation during both training and evaluation. Consequently, the agent’s behavioral patterns are shaped by the diversity of its training experiences, rather than explicit noise injection. While this approach is not mathematically equivalent to NoisyNet, it offers a pragmatic alternative that encourages exploration via interaction history.

(3) The reliance on handcrafted visual preprocessing layers limits the model’s extensibility to other visual domains. Generalizing this architecture to new environments would require significant manual adaptation, particularly in early visual feature design.

While this work does not provide a direct comparison with PPO or Rainbow DQN baselines in the CRV3-DRT setting, this decision is deliberate and grounded in the scope and goals of this study. Specifically:

(1) A fair comparison would require careful re-implementation and tuning of PPO or Rainbow architectures specifically adapted for the randomized CarRacing-v3 domain. However, such implementations are not readily available in the literature or open-source repositories.

(2) Including a weak or naïvely configured baseline would risk being a strawman, while including a strong and carefully tuned PPO would shift the focus away from the main contribution of this paper—namely, demonstrating the viability of a value-based DQN agent in randomized environments.

(3) Moreover, using agents designed for earlier CarRacing versions (v0–v2) without domain randomization would be misleading, as the underlying task complexity differs substantially from CRv3-DRT.

Therefore, this work focuses on thoroughly documenting and analyzing the behavior of a high-performing DQN variant, with transparency in design and reproducibility, rather than

engaging in potentially misleading cross-method benchmarking.

The author does not tend to outperform all existing methods, but to provide an open-source and reproducible DQN agent built purely with Keras.

Future work may explore architectural distillation, memory augmentation, or automated visual layer adaptation as means of scaling this approach to more dynamic domains.

Ultimately, this work does not propose a new SOTA, but rather revisits foundational value-based agents with modern architecture design in mind—highlighting that DQN, when given structural freedom, still has untapped potential.

V. CONCLUSION

This work presents a Keras-based DQN agent capable of achieving an average score exceeding 800 in the randomized CarRacing-v3 environment—a setting traditionally dominated by policy-gradient methods such as PPO. By integrating custom visual preprocessing, an ensemble of dueling Q-heads, and a deliberately overprovisioned architecture, the agent demonstrates strong stability and generalization, without reliance on heavy reward shaping or extensive hyperparameter tuning. Rather than optimizing for elegance or minimalism, the design prioritizes architectural redundancy and diversity. This challenges the common assumption that DQN-based agents cannot effectively handle complex, visually noisy environments with sparse rewards.

The approach demonstrates that robustness can emerge from structural capacity and modular design. Future directions include incorporating memory modules, recurrent networks, or evolutionary search for architecture adaptation. By making both the code and the agent public, this work aims to support researchers and learners interested in pushing the boundaries of DQN-style agents in challenging environments.

At the very least, this work establishes a transparent (the author does not claim this agent is purely interpretable) and reproducible DQN soft baseline—built with Keras—that consistently scores above 800 in domain-randomized CarRacing-v3. While not SOTA, it is functional, understandable-but-uninterpretable, and available for the community to build upon.

VI. OPEN-SOURCED CODE

The SumTree and Prioritized Experience Replay (`DoubleReplay`) and other minor components were implemented with the assistance of AI coding assistants (ChatGPT). The author reviewed and verified their correctness, and modified them to integrate with the custom Keras-based DQN framework.

The full source code, training notebooks, pretrained weights, and evaluation tools are available at: GitHub Repository: <https://github.com/AeneasWeiChiHsu/CarRacing-v3-DQN>.

The model and the evaluation notebook can be downloaded from Kaggle: <https://www.kaggle.com/models/weichihsu1996/dqn-model-on-car-racing-v3-random-environment>

The 1000 episode evaluation GIF log files can be downloaded from Kaggle: <https://www.kaggle.com/datasets/weichihsu1996/dqn-agent-gif-on-randomized-carracing-v3>

The original Jupyter Notebook can be found in the GitHub repository, along with a pretrained model and 100-episode evaluation GIFs accessible via a Google Drive link provided in the README.

This notebook was initially the author’s private working document. After reflection, the author chose to retain personal comments and inline design notes to preserve the thought process behind the architecture. The intent is to demystify the model design and offer a transparent view of the iterative experimentation that led to the final results.

VII. REPRODUCIBILITY CHECKLIST

A. Code and Training Pipeline

- The complete training and evaluation code is available online.
- A Jupyter notebook is provided for one-click reproduction.
- The training code is runnable with minimal configuration or setup.

B. Model Architecture

- The architecture of the DQN agent is fully specified and visualized in the notebook.
- All model hyperparameters are listed and explained.

C. Pretrained Weights

- Pretrained model weights are available for download.
- Evaluation can be performed directly using the pretrained model.

D. Evaluation Procedure

- The environment configuration (CarRacing-v3 with `domain_randomize=True`) is clearly stated.
- Evaluation code is included and outputs visualizations (GIFs, scores).
- Evaluation results over 100 episodes are included with statistical summaries (mean, std, min, max).

E. Experimental Results

- The results reported in the paper can be reproduced using the provided code and weights.
- Visualization outputs (track behavior, Q-values) are available.

ACKNOWLEDGEMENT

The author would like to thank the open-source community—particularly contributors to Keras, TensorFlow, and Gymnasium—for providing accessible and powerful tools that made this project feasible. This work was initiated based on informal feedback and encouragement from peers, leading to the decision to document and release the agent for public use. The author also acknowledges the assistance of AI-based coding tools, including ChatGPT and Gemini, whose sugges-

tions contributed to implementation refinement and debugging throughout the project.

REFERENCES

- [1] M. Towers *et al.*, “Gymnasium: A standard interface for reinforcement learning environments,” *arXiv preprint arXiv:2407.17032*, 2024.
- [2] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep reinforcement learning: A brief survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.
- [3] A. Del Rio, D. Jimenez, and J. Serrano, “Comparative Analysis of A3C and PPO Algorithms in Reinforcement Learning: A Survey on General Environments,” *IEEE Access*, 2024.
- [4] V. Mnih *et al.*, “Human-level control through deep reinforcement learning,” *nature*, vol. 518, no. 7540, pp. 529–533, 2015.
- [5] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, “Dueling network architectures for deep reinforcement learning,” in *International conference on machine learning*, 2016, pp. 1995–2003.
- [6] H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in *Proceedings of the AAAI conference on artificial intelligence*, 2016.
- [7] M. Hessel *et al.*, “Rainbow: Combining improvements in deep reinforcement learning,” vol. 32, no. 1, 2018.
- [8] A. A. Haris, S. Jeevitha, P. Das, P. Munjal, R. K. Bora, and O. K. Kishore, “Revolutionizing NPC Interaction: DQN with Huber Loss for Dynamic Behavior Modeling in Video Games,” pp. 1–6, 2024.