2022 IEEE 16th International Conference on Application of Information and Communication Technologies (AICT) | 978-1-6654-5162-8/22/$31.00 ©2022 IEEE | DOI: 10.1109/AICT55583.2022.10013603

A Memory Efficient Deep Reinforcement Learning Approach For Snake Game Autonomous Agents

Md. Rafat Rahman Tushar[1]
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
rafat.tushar@northsouth.edu

Shahnewaz Siddique[2]
Department of Electrical and Computer Engineering
North South University
Dhaka, Bangladesh
shahnewaz.siddique@northsouth.edu

***Abstract*—To perform well, Deep Reinforcement Learning (DRL) methods require significant memory resources and computational time. Also, sometimes these systems need additional environment information to achieve a good reward. However, it is more important for many applications and devices to reduce memory usage and computational times than to achieve the maximum reward. This paper presents a modified DRL method that performs reasonably well with compressed imagery data without requiring additional environment information and also uses less memory and time. We have designed a lightweight Convolutional Neural Network (CNN) with a variant of the Q-network that efficiently takes preprocessed image data as input and uses less memory. Furthermore, we use a simple reward mechanism and small experience replay memory so as to provide only the minimum necessary information. Our modified DRL method enables our autonomous agent to play Snake, a classical control game. The results show our model can achieve similar performance as other DRL methods.**

***Index Terms*—Deep Reinforcement Learning, Convolutional Neural Network, Deep Q Learning, Hyperparameter Tuning, Replay Size, Image Preprocessing**

I. Introduction

Complex problems can be solved in real-world applications by carefully designing Deep Reinforcement Learning (DRL) models by taking high dimensional input data and producing discrete or continuous outputs. It is challenging to build a agent using sensory data capable of controlling and acting in an environment. The environment is also complex and primarily unknown to the acting agent. The agent needs to learn the underlying distribution of the state and action spaces, and the distribution changes as the agent encounters new data from an environment. Previously reinforcement learning algorithms [1]–[5] were presented with lower constraint problems to demonstrate the algorithms effectiveness. However, these systems were not well generalized for high dimensional inputs; thus, they could not meet the requirements of practical applications.

Recently, DRL has had success in CNN based vision-based problems [6]–[8]. They have successfully implemented DRL methods that learn to control based on image pixel. Although the image-based DRL methods have enjoyed considerable success, they are memory intensive during training as well as deployment. Since they require a massive amount of memory, they are not suitable for implementation in mobile devices or mid-range autonomous robots for training and deployment.

All modern reinforcement learning algorithms use replay buffer for sampling uncorrelated data for online training in mainly off-policy algorithms. Experience replay buffer also improves the data efficiency [9] during data sampling. Since the use of neural networks in various DRL algorithms is increasing, it is necessary to stabilize the neural network with uncorrelated data. That is why the experience replay buffer is a desirable property of various reinforcement learning algorithms. The first successful implementation of DRL in high dimensional observation space, the Deep Q-learning [6], used a replay buffer of 10^6 size. After that, [8], [10]–[12], to name a few, have solved complex high dimensional problems but still use a replay buffer of the same size.

Experience replay buffer suffers from two types of issues. One is to choose the size of the replay buffer, and the second is the method of sampling data from the buffer. [13]–[15] consider the latter problem to best sample from the replay buffer. But the favorable size for the replay buffer remains unknown. Although [15] points out that the learning algorithm is sensitive to the size of the replay buffer, they have not come up with a better conclusion on the size of the buffer.

In this paper, we tackle the memory usage of DRL algorithms by implementing a modified approach for image preprocessing and replay buffer size. Although we want the agent to obtain a decent score, we are more concerned about memory usage. We choose a Deep Q-Network (DQN) [6] for our algorithm with some variations. Our objective is to design a DRL model that can be implemented on mobile devices during training and deployment. To be deployed on mobile devices, memory consumption must be minimized as traditional DRL model with visual inputs sometimes need half a terabyte of memory. We achieve low memory consumption by preprocessing the visual image data and tuning the replay buffer size with other hyperparameters. Then, we evaluate our model in our simulation environment using the classical control game named Snake.* The results show that our model can achieve similar performance as other DRL methods.

[1]Research Assistant.
[2]Assistant Professor, IEEE Member.
*GitHub implementation: https://github.com/rafattushar/rl-snake

II. Related Work

The core idea of reinforcement learning is the sequential decision making process involving some agency that learns from the experience and acts on uncertain environments. After the development of a formal framework of reinforcement learning, many algorithms have been introduced such as, [1]–[5].

Q-learning [1] is a model-free asynchronous dynamic programming algorithm of reinforcement learning. Q-learning proposes that by sampling all the actions in states and iterating the action-value functions repeatedly, convergence can be achieved. The Q-learning works perfectly on limited state and action space while collapsing with high dimensional infinite state space. Then, [6] proposes their Deep Q-network algorithm that demonstrates significant results with image data. Among other variations, they use a convolutional neural network and replay buffer. Double Q-learning [16] is applied with DQN to overcome the overestimation of the action-value function and is named Deep Reinforcement Learning with Double Q-Learning (DDQN) [8]. DDQN proposes another neural network with the same structure as DQN but gets updated less frequently. Refined DQN [17] proposes another DRL method that involves a carefully designed reward mechanism and a dual experience replay structure. Refined DQN evaluate their work by enabling their agent to play the snake game.

The experience replay buffer is a desirable property of modern DRL algorithms. It provides powerful, model-free, off-policy DRL algorithms with correlated data and improves data efficiency [9] during data sampling. DQN [6] shows the power of replay buffer in sampling data. DQN uses the size 10^6 for replay buffer. After that, [8], [10]–[12], [17], among others, have shown their work with the same size and structure as the replay buffer. Schaul et al. propose an efficient sampling strategy in their prioritized experience replay (PER) [13]. PER shows that instead of sampling data uniform-randomly, the latest data gets the most priority; hence the latest data have more probability of being selected, and this selection method seems to improve results. [15] shows that a large experience replay buffer can hurt the performance. They also propose that when sampling data to train DRL algorithms, the most recent data should the appended to the batch.

III. Method

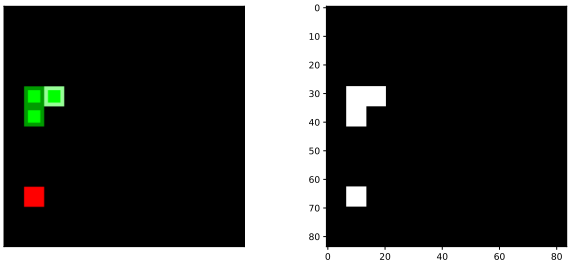
Our objective is to reduce memory usage during training time while achieving the best performance possible. The replay memory takes a considerable amount of memory, as described later. We try to achieve memory efficiency by reducing the massive replay buffer requirement with image preprocessing and the buffer size. The buffer size is carefully chosen so that the agent has the necessary information to train well and achieves a moderate score. We use a slight variation of the deep Q-learning algorithm for this purpose.

TABLE I
Reward Mechanism for Snake Game

Moves	Rewards	Results
Eats an apple	+1	Score Increase
Hits with wall or itself	-1	End of episode
Not eats or hits wall or itself	-0.1	Continue playing games

TABLE II
Memory Requirement for Different Pixel Data

	RGB	Grayscale	Binary
Data Type	float	float	int
Size (kB)	165.375	55.125	6.890
Memory Save % w.r.t. RGB	0%	67%	96%
Memory Save % w.r.t. Grayscale	-	0%	87.5%



(a) Before preprocessing (b) After preprocessing

Fig. 1. Visual image data before and after preprocessing

A. Image Preprocessing

The agent gets the RGB values in the 3-D array format from the games' environments. We convert the RGB array into grayscale because it would not affect the performance [18] and it saves three times of memory. We resize the grayscale data into 84×84 pixels. Finally, for more memory reduction, we convert this resized grayscale data into binary data (values only with 0 and 1). The memory requirement for storing various image data (scaled-down between 0 and 1) is given in Table II. Table II shows that it saves around 67% from converting RGB into grayscale and around 96% from converting RBG into binary. Also, the memory requirement reduces by around 87.5% converting from grayscale into binary. Visual pixel data transformation with preprocessing is given in Fig. 1. The preprocessing method is presented using a flowchart in Fig. 2.

B. Game Selection and Their Environments

The use-case of our target applications is less complex tasks. For this reason, we implemented the classical Snake game [19]

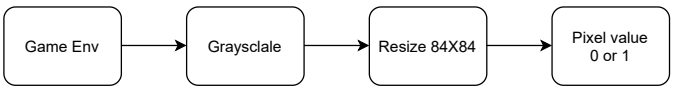


Fig. 2. Diagram of image preprocessing

in the 'pygame' module. The game screen is divided into a 12×12 grid. The resolution for the game is set to 252×252. The initial snake size is 3. The controller has four inputs to navigate. Table I shows the valid actions and respective reward for the snake game environment.

C. Reinforcement Learning Preliminary

Any reinforcement learning or sequential decision-making problem can be formulated with Markov Decision Processes (MDPs). An MDP is a triplet $M = (\mathcal{X}, \mathcal{A}, \mathcal{P}_0)$, where $\mathcal{X}$ is a set of valid states, $\mathcal{A}$ is a set of valid actions, and $\mathcal{P}_0$ is transition probability kernel that maps $\mathcal{X} \times \mathcal{A}$ into next state transition probability. For a deterministic system, the state transition is defined as,

$$s_{t+1} = f(s_t, a_t) \tag{1}$$

The reward is defined as,

$$r_t = R(s_t, a_t) \tag{2}$$

The cumulative reward over a trajectory or episode is called the return, $R(\tau)$. The equation for discounted return is given below,

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t \tag{3}$$

D. Deep Q-Learning

The goal of the RL agent is to maximize the expected return. Following a policy π, the expected return, $J(\pi)$, is defined as,

$$J(\pi) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)] \tag{4}$$

The optimal action-value or q function $Q^*(s, a)$ maximizes the expected return by taking any action at state s and acting optimally in the following states.

$$Q^*(s, a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau) | s_0 = s, a_0 = a] \tag{5}$$

For finding out the optimal actions based on an optimal action-value function at time t, the Q^* must satisfy the Bellman Equation, which is,

$$Q^*(s, a) = \mathop{\mathbb{E}}_{s' \sim \rho}\left[r(s, a) + \gamma \max_{a'} Q^*(s', a')\right] \tag{6}$$

The optimal action-value function gives rise to optimal action $a^*(s)$. The $a^*(s)$ can be described as,

$$a^*(s) = \arg\max_{a} Q^*(s, a) \tag{7}$$

For training an optimal action-value function, sometimes a non-linear function approximator like neural network [6] is used. We used a convolutional neural network.

TABLE III
THE ARCHITECTURE OF NEURAL NETWORK

Layer Name	Filter	Stride	Layer	Acti-vation	Zero Padd	Output
Input						84*84*4
Conv1	8*8	4	32	ReLU	Yes	21*21*32
M. Pool	2*2	2			Yes	11*11*32
Conv2	4*4	2	64	ReLU	Yes	6*6*64
M. Pool	2*2	2			Yes	3*3*64
B. Norm						3*3*64
Conv3	3*3	2	128	ReLU	Yes	2*2*128
M. Pool	2*2	2			Yes	1*1*128
B. Norm						1*1*128
Flatten						128
FC			512	ReLU		512
FC			512	ReLU		512
Output			No. of actions	Linear		No. of actions

M. Pool = Max Pooling, B. Norm = Batch Normalization, FC = Fully Connected

TABLE IV
MEMORY REQUIREMENT EXPERIENCE REPLAY

	RGB	Grayscale	Binary
Memory Usage (GB)	1261.71	420.57	2.628
Memory Save % w.r.t. RGB	0%	67%	99.7%
Memory Save % w.r.t. Grayscale	-	0%	99.4%

E. Neural Network

The action-value function is iteratively updated to achieve the optimal action-value function. The neural network used to approximate the action-value function and update at each iteration is called Q-network. We train the Q-network, parameterized by θ, by minimizing a loss function $L_i(\theta_i)$ at ith iteration.

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{s, a \sim \rho}\left[(y_i - Q(s, a; \theta_i))^2\right] \tag{8}$$

where $y_i = \mathop{\mathbb{E}}_{s' \sim \rho}\left[r(s, a) + \gamma \max_{a'} Q'(s', a'; \theta'_k)\right]$ is the target for that update. Here Q' is another Q-network with the same shape as Q-network but with a frozen parameter called target Q-network for training stability parameterized by θ'_k. We train the Q-network by minimizing this loss function (8) w.r.t. the parameter θ_i. We use Adam [20] optimizer for fast

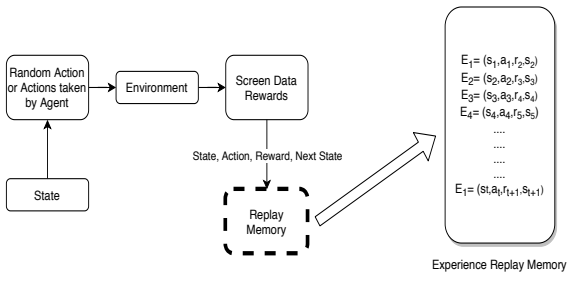


Fig. 3. Structure of experience replay memory and flowchart

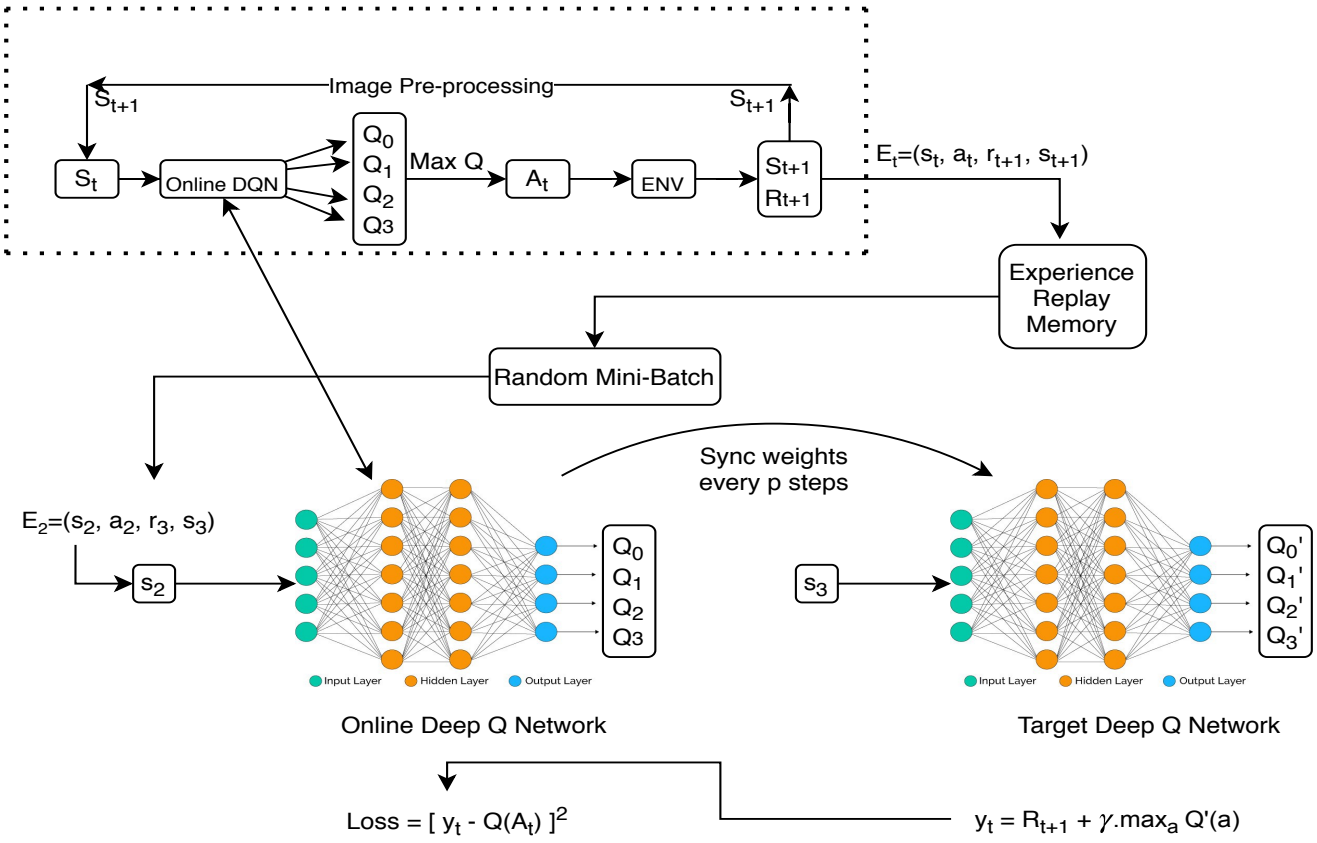


Fig. 4. The deep reinforcement learning design structure of our model

convergence. Our convolutional neural network structure is shown in Table III.

F. Experience Replay Buffer

As our focus is to keep memory requirements as low as possible during training, choosing the size of the replay buffer is one of the critical design decisions. The size of the replay buffer directly alters the requirement of memory necessity. We use a replay buffer of size 50,000, requiring less memory (only 5%) than [6], [8], [17], which use a replay buffer of size 1,000,000. [6], [8], [17] store grayscale data into a replay buffer. Table IV shows that we use 99.4% less memory compared to these works. The replay buffer stores data in FIFO (first in, first out) order so that the buffer contains only the latest data. We present the complete cycle of the experience replay buffer in Fig 3. Fig. 4 illustrates our complete design diagram.

IV. Experiments

A. Training

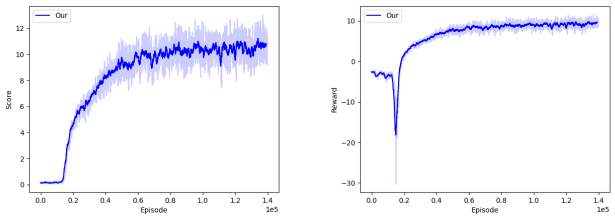
For training our model, we take a random batch of 32 experiences from the replay buffer at each iteration. Our model has two convolutional neural networks (online DQN and target DQN) sharing the same structure but does not sync automatically. The weights of the target network are frozen so that it cannot be trained. The state history from the mini-batch is fed into the Online DQN. The DQN outputs the Q-values, $Q(s_t, a_t)$.

$$Loss = [y_t - Q(s_t, a_t)]^2 \tag{9}$$

The y_t is calculated from the target Q-network. We are passing the next-state value to the target Q-network, and for each next-state in the batch, we get Q-value, respectively. That is our $max_{a'}Q(s', a')$ value in the below equation.

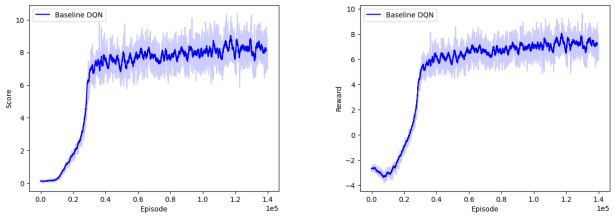
$$y_t = R_{t+1} + \gamma max_{a'}Q(s', a') \tag{10}$$

The γ is the discount factor, which is one of many hyperparameters we are using in our model. Initially, we set γ value to 0.99. The R_{t+1} is the reward in each experience tuple. So, we get the y_t value. The loss function is generated by putting these values in (9). Then, we use this loss function to backpropagate our Online DQN with an 'Adam' optimizer. Adam optimizer is used instead of classical stochastic gradient descent for more speed. The target DQN is synced with online DQN at every



(a) Score vs. episode graph (b) Reward vs. episode graph

Fig. 5. Results of our agent playing Snake game during training



(a) Score vs. episode graph (b) Reward vs. episode graph

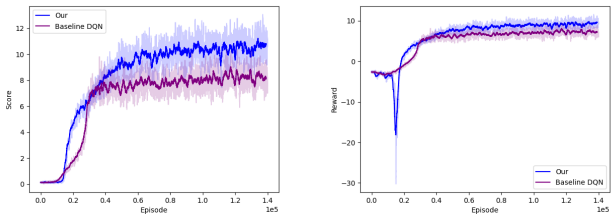
Fig. 6. Results of baseline DQN model playing Snake game during training

10,000 steps. The values of hyperparameters we choose are listed in Table VI.

B. Results and Comparisons

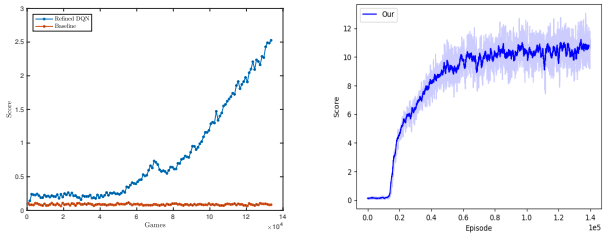
We allow DRL agents to play 140,000 episodes of games to match the training results presented in [17]. We train one agent with our method and another with the DQN method presented in [6], we refer to [6] as the baseline DQN model. Next, we compare our model with the baseline DQN model [6] and the refined DQN model [17]. The results of training the snake game with our model are shown in Fig. 5. Fig. 5(a) shows the game's score with our model during training. Fig. 5(b) shows that even though our reward mechanism is simpler than the refined DQN model, the agent maximizes the cumulative reward optimally.

In section III-F we showed that our model is more memory efficient than the baseline DQN model and the refined DQN model during training. In this section we show that despite low memory usage, our model can achieve similar if not better



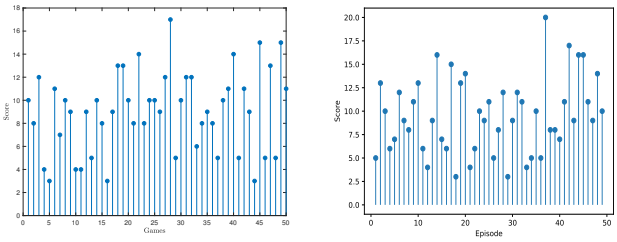
(a) Score comparison (b) Reward comparison

Fig. 7. Comparison between our model and baseline DQN model



(a) Score graph of Refined DQN (graph taken from [17]) (b) Score graph of our model

Fig. 8. Comparison between Refined DQN model and our model



(a) Refined DQN score (Taken from [17]) (b) Our model's score

Fig. 9. Testing evaluation by playing random 50 episodes game

results than the baseline and refined DQN models. Fig. 6 displays the baseline DQN results during training on the snake game. In Fig. 7 we present the score and reward comparison between our model and the baseline DQN model. The blue line in Fig. 7(a) represents our model's score, and the purple line represents the score of the baseline DQN model. During 140,000 numbers of training episodes, our model remains better at episode score though it requires fewer resources. Fig. 7(b) demonstrates that our model is capable of achieving higher cumulative rewards than the baseline DQN model.

We also compare the results between our model and the refined DQN model [17]. Refined DQN follows a dual experience replay memory architecture and a complex reward mechanism. However, our model surpasses their score. Since their game is similar to ours, we compare our results with the results provided in their paper. Fig. 8(a) shows the results presented in [17], and Fig. 8(b) is our model's results during

TABLE V
LIST OF PERFORMANCE COMPARISON OF DIFFERENT AGENTS

Performance	Score
Human Average	1.98 *
Baseline Average	0.26 *
Refined DQN Average	9.04 *
Our Average	**9.53**
Human Best	15 *
Baseline Best	2 *
Refined DQN Best	17 *
Our Best	**20**

* Data taken from [17]

training. By comparing Fig. 8(a) and Fig. 8(b), we can safely say that our model achieves better scores despite having a simple replay buffer, a simple reward mechanism, and less memory consumption.

Fig. 9(a) and Fig. 9(b) show scores of random 50 episodes during testing of refined DQN and our model, respectively. Table V summarizes the scores provided in the refined DQN and our model. We can identify from Table V that their refined DQN average is 9.04, while ours is 9.53, and their refined DQN best score is 17, while ours is 20. So, we can see that our model also performs better in the training and testing phase.

TABLE VI
LIST OF HYPERPARAMETERS

Hyperparameter	Value	Description
Discount Factor	0.99	γ-value in max Q-function
Initial Epsilon	1.0	Exploration epsilon initial value
Final Epsilon	0.01	Exploration final epsilon value
Batch size	32	Mini batch from replay memory
Max step	10,000	Maximum number of steps allowed per episode
Learning Rate	0.0025	Learning rate for Adam optimizer
Clip-Norm	1.0	Clipping value for Adam optimizer
Random Frames	50,000	Number of random initial steps
Epsilon greedy frames	500,000	Number of frames in which initial epsilon will be equal final epsilon
Experience Replay Memory	50,000	Capacity of experience replay memory
Update of DQN	4	The number of steps after each update of DQN takes place
Update Target DQN	10,000	The number of steps after the Target and Online DQN sync

V. CONCLUSION

In this paper, we have shown that better image preprocessing and constructing a better mechanism for replay buffer can reduce memory consumption on DRL algorithms during training. We have also demonstrated that using our method, the performance of the DRL agent on a lower constraint application is entirely similar, if not better. We combined our method with the DQN (with some modification) algorithm to observe the method's effectiveness. Our presented design requires less memory and a simple CNN. We established that our method's result is as good as other DRL approaches for the snake game autonomous agent.

ACKNOWLEDGMENT

This work was supported by North South University research grant CTRG-21-SEPS-18.

The authors would like to gratefully acknowledge that the computing resources used in this work was housed at the National University of Sciences and Technology (NUST), Pakistan. The cooperation was pursued under the South Asia Regional Development Center (RDC) framework of the Belt & Road Aerospace Innovation Alliance (BRAIA).

REFERENCES

[1] C. J. C. H. Watkins and P. Dayan, "Q-learning," in *Machine Learning*, 1992, pp. 279–292.

[2] G. Tesauro, "Temporal difference learning and td-gammon," *Commun. ACM*, vol. 38, no. 3, p. 58–68, Mar. 1995.

[3] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, S. Solla, T. Leen, and K. Müller, Eds., vol. 12. MIT Press, 1999.

[4] J. Peters, S. Vijayakumar, and S. Schaal, "Natural actor-critic," in *Machine Learning: ECML 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 280–291.

[5] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ser. ICML'14. JMLR.org, 2014, p. I–387–I–395.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *Computing Research Repository*, vol. abs/1312.5602, 2013.

[7] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–33, 02 2015.

[8] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, ser. AAAI'16. AAAI Press, 2016, p. 2094–2100.

[9] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Mach. Learn.*, vol. 8, no. 3–4, p. 293–321, may 1992.

[10] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *Computing Research Repository*, 2019.

[11] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust multi-agent reinforcement learning via minimax deep deterministic policy gradient," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, pp. 4213–4220, Jul. 2019.

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor." in *ICML*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 1856–1865.

[13] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015. [Online]. Available: https://arxiv.org/abs/1511.05952

[14] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017.

[15] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *Computing Research Repository*, vol. abs/1712.01275, 2017.

[16] H. Hasselt, "Double q-learning," in *Advances in Neural Information Processing Systems*, J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta, Eds., vol. 23. Curran Associates, Inc., 2010.

[17] Z. Wei, D. Wang, M. Zhang, A.-H. Tan, C. Miao, and Y. Zhou, "Autonomous agents in snake game via deep reinforcement learning," in *2018 IEEE International Conference on Agents (ICA)*, 2018, pp. 20–25.

[18] T. D. Nguyen, K. Mori, and R. Thawonmas, "Image colorization using a deep convolutional neural network," *Computing Research Repository*, vol. abs/1604.07904, 2016.

[19] A. Punyawee, C. Panumate, and H. Iida, "Finding comfortable settings of snake game using game refinement measurement," in *Advances in Computer Science and Ubiquitous Computing*. Singapore: Springer Singapore, 2017, pp. 66–73.

[20] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.