Dokuz Eylul University
Faculty Of Science
Computer Science

AI Algorithms' Performances Analysis & Comparisons in AgarIO Game

İpek SOYDEMİR

Aslı FERİKLİOĞLU

Aslıhan AKBIYIK

Emrecan BALGÜN

Supervisor: Assoc. Prof. Dr. Mete Eminağaoğlu

May, 2022

Izmir

AI Algorithms' Performances Analysis & Comparisons in AgarIO Game is the title of the report written by **İpek SOYDEMİR, Aslıhan AKBIYIK, Aslı FERİKLİOĞLU, Emrecan BALGÜN** and under the supervision of **Assoc. Prof. Dr. Mete Eminağaoğlu's work** was read by us and it was deemed to be of high enough caliber to be used as a graduation project.

Assoc. Prof. Dr.
METE EMİNAĞAOĞLU

First and foremost, Associate Professor Dr. We would like to thank METE EMİNAĞAOĞLU for the values he gave to our lives. We appreciate our loving family' continued financial and emotional support for our initiative.

ABSTRACT

Since its introduction into our daily lives, artificial intelligence, which has a wide range of applications, has also grown popular in the gaming industry. Agar.io, on the other hand, has grown in popularity owing to its use of intuitive game design for strategic decision-making.

The goal of this project is to train CNN, DQN, and SARSA algorithms with 6 different hyperparameters in order to select the training model with the highest average score during training for each algorithm and measure performance analysis using statistics that compare the success of these selected models.

As a consequence of this experiment, it has been discovered that the 'Step' value is the most important parameter impacting the score for all three methods. CNN has the highest average score of 34.26, followed by DQN with a score of 28.32, and SARSA with a score of 25.74. Although CNN gets the best score, there is a huge time difference between Q and SARSA. CNN takes 5 hours to train with equal episode and step values, but the Q and SARSA algorithms take roughly 15 minutes.

The best parameters for CNN were calculated as 50 episodes, 1500 steps, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size for CNN, 50 episodes, 1500 steps, 4 Tau, 0.8 Gamma, 0.001 Learning Rate, 32 Batch Size for DQN and SARSA.

Keywords: Artificial Intelligence, Reinforcement Learning, Deep Learning, Convolutional Neural Network, Deep Q Network, Q Learning, SARSA (State-Action-Reward-State-Action)

TABLE OF CONTENTS

INTRODUCTION 8

GAME 9

Rules 9

AI ALGORITHMS 10

CNN 10

Deep Q Network 11

Q Learning 11

Deep Q Network 12

SARSA 14

TRAININGS OF THE ALGORITHMS 16

CNN 16

Epoch(Episode) Parameter 16

Step Parameter 17

Tau Parameter 18

Gamma Parameter 19

Learning Rate Parameter 19

Batch Size Parameter 20

DEEP Q NETWORK 20

Epoch(Episode) Parameter 20

Step Parameter 21

Gamma Parameter 22

Learning Rate Parameter 22

Batch Size Parameter 23

Epsilon Parameter 23

DEEP SARSA 24

Epoch(Episode) Parameter 24

Step Parameter 25

Gamma Parameter 25

Learning Rate Parameter 25

Batch Size Parameter 26

Epsilon Parameter 27

CONCLUSION 28

REFERENCES 31

LIST OF TABLES

Table 2.1 11

LIST OF FIGURES

Figure 2.1 10

CNN
Figure 3.1.1 12
Figure 3.1.2 12

DQN
Figure 3.2.1.1 12
Figure 3.2.1.2 13
Figure 3.2.2.1 14
Figure 3.2.2.2 14
Figure 3.2.2.3 14

SARSA
Figure 3.3.1 15
Figure 3.3.2 16
Figure 3.3.3 17

INTRODUCTION

The experiential learning theory, which first appeared in the social sciences in the 1870s, is based on the work of Dewey, who emphasizes the importance of personal involvement in the learning process, Lewin, who emphasizes the value of active participation in the learning process, and Piaget, who views intelligence as the result of interactions between individuals and their environments rather than as a solely innate trait (Yoon). , 2000:36; Kolb, 1984:20).

In Reinforcement Learning, a machine learning methodology, our learning machine (agent) senses its interactions with the environment (sensation), provides a goal-oriented response (action), and then waits for a numerical reward signal. This reward point is being maximized by our agent.

When we began working on our project, we first chose the machine learning methods we would employ and researched those algorithms. Then, we came to the conclusion that testing these algorithms in a game with pre-established rules would be more accurate than testing them in a game with rules that we make up. We ultimately decided for Agario because it was more practical for us.

GAME

Rules

The objective of the multiplayer strategy and action game Agar.io is to gather randomly generated pellets (agar, or gelatin made from algae) that increase the mass of a player-controlled cell in a Petri (culture) dish and grow bigger by suckling up smaller cells.

To obtain the largest cell is the goal of the game. The game restarts with a small cell if the bigger players consume every player's cell.

The length of time spent on the leaderboard in the game will increase as a result of hitting green viruses, which will separate them into different segments.

Agent eats other if:

1. it has mass greater by at least CONSUME_MASS_FACTOR, and 2. the agent's circle overlaps with the center of other

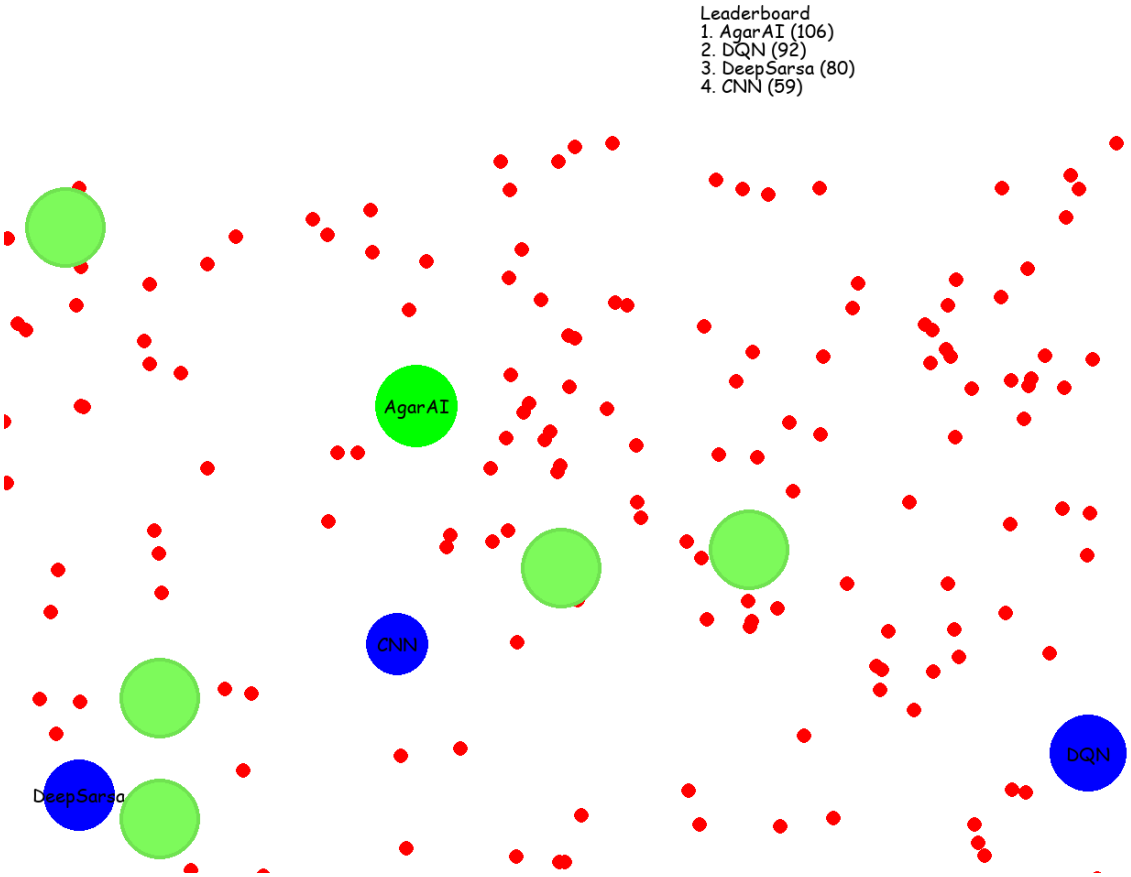


Figure 2.1

AI ALGORITHMS

In our project, we made use of reinforcement learning methods.
A reinforcement learning algorithm, in its most basic form, consists of three parts: an exploration strategy to test various game actions, a reinforcement function to provide feedback on how effective each action is, and a learning rule to link the two.
We employed ReLu, the most frequently used function, in all of our algorithms.

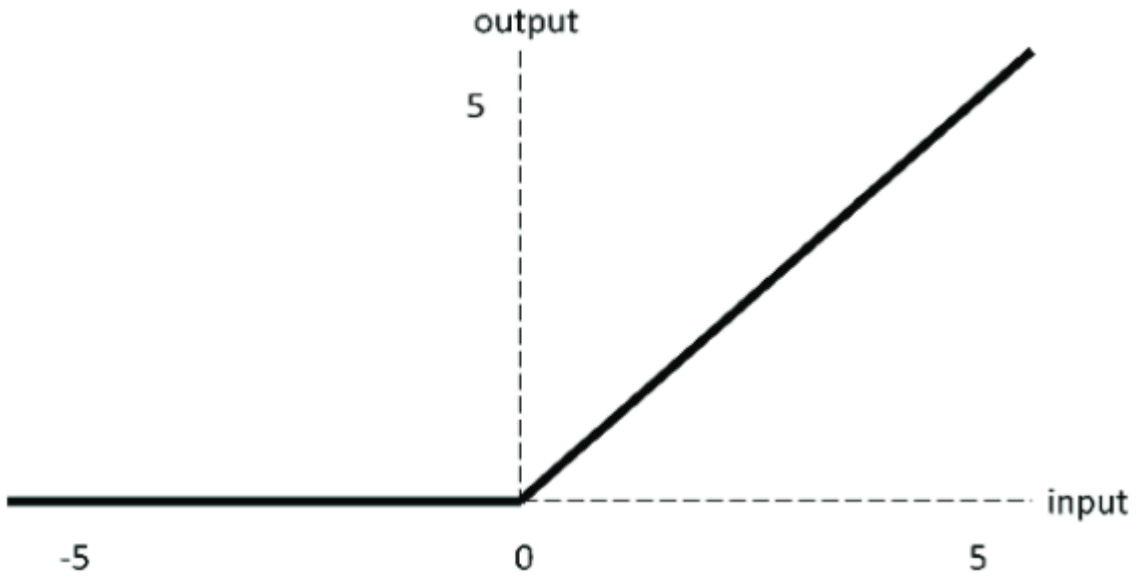


Table 2.1

CNN

CNN's architecture is influenced by the visual perception of living creatures. A single or many blocks of convolution and pooling layers, one or more fully connected (FC) layers, and an output layer make up a typical convolutional neural network. The foundational component of a CNN is the convolution layer. The goal of this layer is to learn the input's feature representations. The learnable convolution kernels or filters that make up the convolution layer are used to create a variety of feature maps. Each unit feature map is connected to a preceding layer receptive field.

```
for ( b = 0; b < B; b++ ) {  // B: number of images in a batch        Outer loops
  for ( h = 0; h < H; h += T_h ) {  // H: height of ofms              (off-chip data access)
    for ( w = 0; w < W; w += T_w ) {  // W: width of ofms
      for ( j = 0; j < J; j += T_j ) {  // J: depth of ofms
        for ( i = 0; i < I; I += T_i ) {  // I: depth of ifms & wghs
          // load ifms, wghs, and ofms
          for ( p = 0; p < P; p++ ) {  // P: height of wghs          Inner loops
            for ( q = 0; q < Q; q++ ) {  // Q: width of wghs         (on-chip processing)
              for ( h_x = h; h_x < min(h_x+T_h, H); h_x++ ) {
                for ( w_x = w; w_x < min(w_x+T_w, W); w_x++ ) {
                  for ( j_x = j; j_x < min(j_x+T_j, J); j_x++ ) {
                    for ( i_x = i; i_x < min(i_x+T_i, I); i_x++ ) {
                      ofms [b][h_x][w_x][j_x] += wghs [p][q][i_x][j_x] *
                                                 ifms [b][str*h_x+p][str*w_x+q][i_x]
          }}}}}}
          // store ofms
}}}}}
```

Figure 3.1.1

```python
def preprocess_state(self, state):
    # convert RGB to grayscale via relative luminance
    gray_state = np.dot(state[...,:3], [0.299, 0.587, 0.114])
    # size down the image to speed up training
    resized_state = transform.resize(gray_state, self.downsample_size, mode='constant')
    return resized_state
```

Figure 3.1.2

Deep Q Network

Deep Q Network consists of a combination of Q-Learning and Neural Network algorithm.

Q Learning

Q-Learning is based on the concept of a Q-function.
Q-Learning is named for the set of quality information (Q-values) it holds about each possible situation and action. The algorithm keeps a value for each state and action it tries. The Q value represents how good it thinks the action was when it was in that state.

$$Q(s, a) = (1 - \alpha)\,Q(s, a) + \alpha\left(r + \gamma \max\left(Q(s', a')\right)\right)$$

Figure 3.2.1.1

Utilizing the learning rate parameter to regulate the linear mixing, the Q-learning rule integrates the two elements. The range of the learning rate parameter used to regulate the mix is [0, 1].

The current Q value for state and action is represented by the first component, Q(s,a). This allows us to preserve some of the current value while ensuring that we never discard knowledge we have already acquired.

A general Q-learning system has the following structure:

```
# Holds the store for Q-values, we use this to make
# decisions based on the learning
store = new QValueStore()

# Updates the store by investigating the problem
def QLearning(problem, iterations, alpha, gamma, rho, nu):

  # Get a starting state
  state = problem.getRandomState()

  # Repeat a number of times
  for i in 0..iterations:

    # Pick a new state every once in a while
    if random() < nu: state = problem.getRandomState()

    # Get the list of available actions
    actions = problem.getAvailableActions(state)

    # Should we use a random action this time?
    if random() < rho:
      action = oneOf(actions)

    # Otherwise pick the best action
    else:
      action = store.getBestAction(state)

    # Carry out the action and retrieve the reward and
    # new state
    reward, newState = problem.takeAction(state, action)

    # Get the current q from the store
    Q = store.getQValue(state, action)

    # Get the q of the best action from the new state
    maxQ = store.getQValue(newState,
                           store.getBestAction(newState))

    # Perform the q learning
    Q = (1 - alpha) * Q + alpha * (reward + gamma * maxQ)

    # Store the new Q-value
    store.storeQValue(state, action, Q)

```

Figure 3.2.1.2

Deep Q Network

Deep Q-Learning is a variant of the classical Q-Learning algorithm with a fundamental contribution.

(1) A deep convolutional neural network architecture for the Q-function approach;

(2) using mini-batches of random training data instead of one-step updates on recent experience;

(3) using older network parameters to estimate the Q-values of the next state.

The pseudo code for DQN is shown in Algorithm 1.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

Figure 3.2.2.1

```
# do Q computation
currQ = self.model(states).gather(
    1, actions.unsqueeze(1))
nextQ = self.target_net(next_states)
max_nextQ = torch.max(nextQ, 1)[0].detach()
mask = 1 - dones
expectedQ = rewards + mask * self.gamma * max_nextQ
```

Figure 3.2.2.2

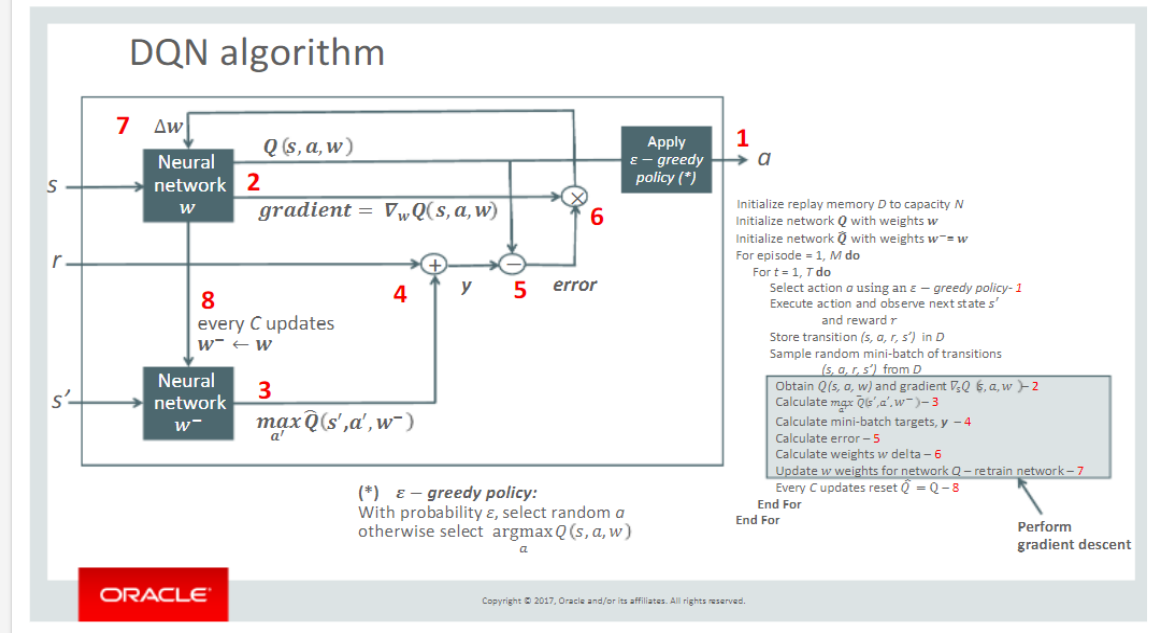


Figure 3.2.2.3

SARSA

SARSA is a policy-based supportive learning method like Q-Learning. With the initials State-Action-Reward-State-Action, "State," which regulates how the agents used for training in the supportive learning process perceive their surroundings, expresses an immediate situation based on how environmental perceptions are processed, "Action," which denotes movements and actions, and "Reward," which symbolizes the reward given to the agent for immediate behaviors.
Calculating the State and Actions that will take place in the following phase based on the selected policy is the major goal of SARSA.

Although Q-Learning and SARSA appear to be comparable algorithms, they have some distinctions. Most significantly, unlike Q-Learning, SARSA does not adjust Q values using a computed maximum reward requirement. In SARSA, a set policy that establishes the core action is used to carry out a new selection of actions and rewards.

Below we can see the mathematically expressed form of the SARSA algorithm:

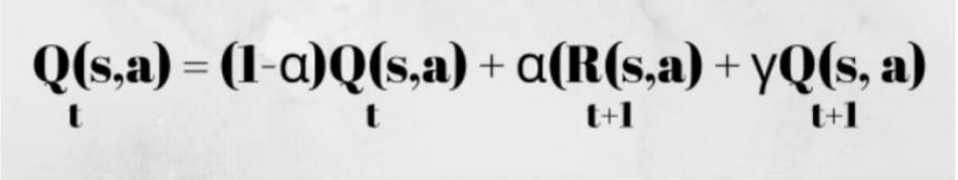


Figure 3.3.1
As can be seen below, SARSA's fundamental mathematical concept is
There are no settings to regulate the prioritizing of the maximal reward, unlike Q-Learning. Because of this, it features use-cases that are distinct from Q-Learning and can perform better than Q-Learning in some circumstances. However, under some other circumstances, Q-Learning might be able to deliver better performance and efficiency than SARSA.

Structure and Mathematical Formulation:
Qt (s, a): It refers to the Q value at time (t).
It is defined as the learning rate, or (1 -), which is the learning rate. It is used to specify how the newly produced state differs from the current state in each stage. The higher disparities between the two states with tracking are referred to as high learning rate, t and t+1.
Rt+1(s, a): The reward amount determined for the subsequent action.
γ: It is referred to as the discount factor and is expressed as such. It accepts a number ranging from 0 to 1. Future steps' significance is indicated by a value between 0 and 1, where 1 means that hypothetical future actions' rewards are equally significant as those at time t.The following state's Q value is represented by the expression Qt+1(s, a).

Q-Learning and SARSA Differences:

● Q-Learning: Check statuses and rewards first, then act out.
● SARSA : Act first. Then check the situation and readjust your views accordingly.
● Q-Learning: Check other states for the next state and make a trend towards maximum reward.
● SARSA: Make a decision to perform an action, update the previous state using the real value.
● Q-Learning: The potential reward at time t+1 is the best possible action.
● SARSA: The reward of the step at time t at time t+1 is the actual reward for the action.
● Q-Learning: It is an Off-Policy reinforcement learning method. Optimal Q values are tried to be developed without meta-cognition on behavior.
● SARSA: On-Policy reinforcement learning method. Q values are developed to ensure appropriate actions.
● Q-Learning: In this methodology, the agent first explores and then decides.

Usage Areas:
If the agent's performance during the learning process matters, SARSA might be a good option. This approach might be considered if the cost of errors is high. However, it can make more sense to choose the Q-Learning algorithm if performance does not play a significant role in the training process.

Below is a form of the pseudocode of the SARSA algorithm:

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Choose a from s using policy derived from Q
         (e.g., ε-greedy)
   Repeat (for each step of episode):
      Take action a, observe r, s'
      Choose a' from s' using policy derived from Q
         (e.g., ε-greedy)
      Q(s, a) <-- Q(s, a) + α[r + γQ(s', a')- Q(s, a)]
      s <-- s'; a <-- a';
   until s is terminal
```

Figure 3.3.2

It should not be forgotten that which supportive learning method to choose is an important decision based on many factors. It is not possible to talk about a single method that is equally successful for every situation, and each method may have certain areas of use in which it will be advantageous. Due to this, multiple models were developed during the project utilizing both SARSA and Q-Learning, and their respective performances were evaluated and contrasted. These comparisons will make it easier to assess how different results SARSA and Q-Learning create when compared to one another in the project's application area.

```
# do Q computation
currQ = self.model(states).gather(
    1, actions.unsqueeze(1))
nextQ = self.target_net(next_states)
mean_nextQ = torch.mean(nextQ, 1)[0].detach()
mask = 1 - dones
expectedQ = rewards + mask * self.gamma * mean_nextQ
```

Figure 3.3.3

TRAININGS OF THE ALGORITHMS

We performed 126 trainings in all, taking into account 6 parameters for each algorithm and 7 distinct values for each parameter.

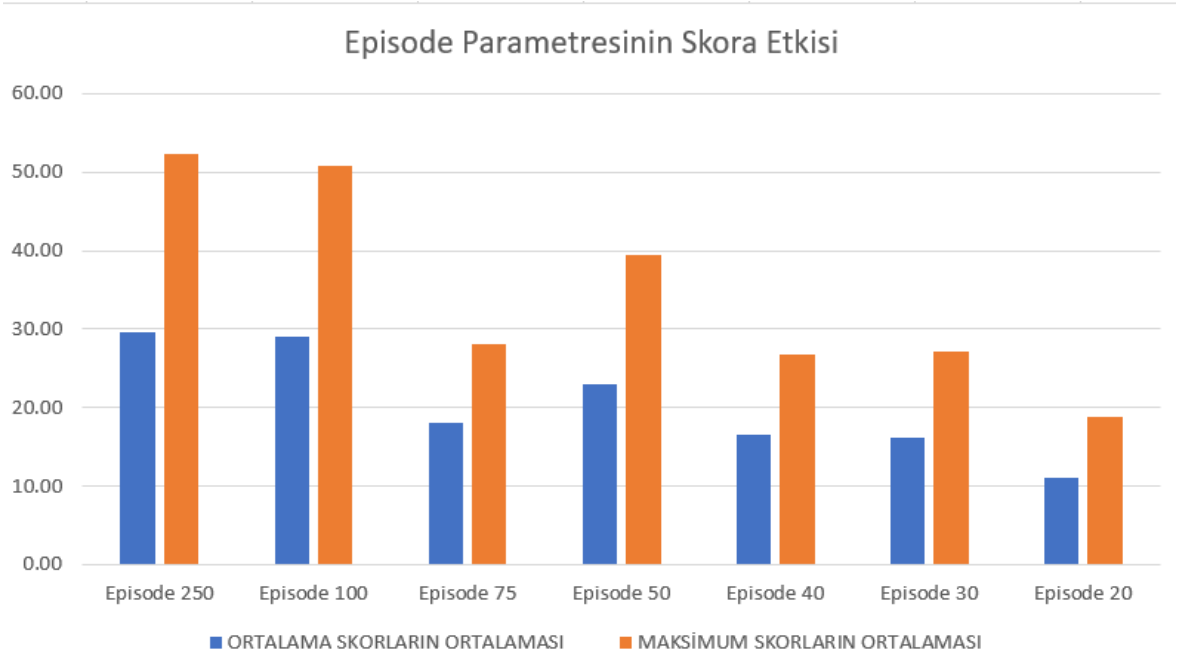
CNN

CNN	**Used Values**						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Tau	1	2	3	4	5	6	7
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35

Epoch(Episode) Parameter

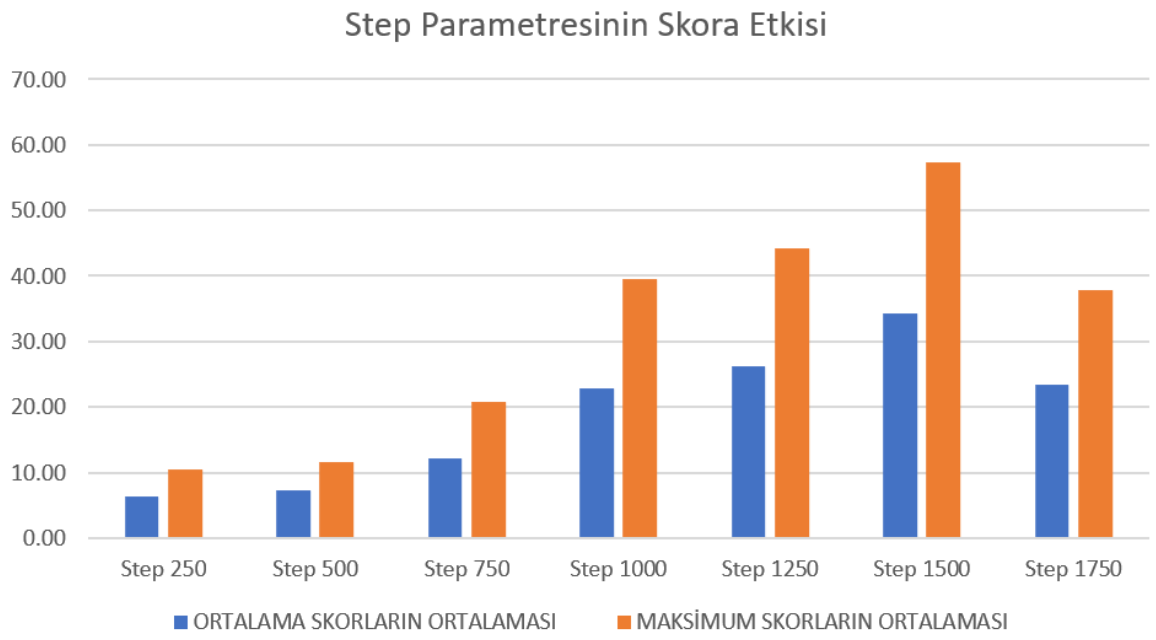
This option indicates how many times the Step parameter that has been chosen will be repeated. The score is reset after each repetition, allowing you to consolidate your prior knowledge. Our stopping condition is also the Epoch (Episode) Parameter (stopping criteria). There is between 50 and 75 overfitting that is seen. We chose 50 Episodes as our completion criteria because a high episode value in terms of time was very wasteful and an increase in the number of epochs did not sufficiently affect the score.

***in graphs, blue = average of average scores orange = average of maximum scores**



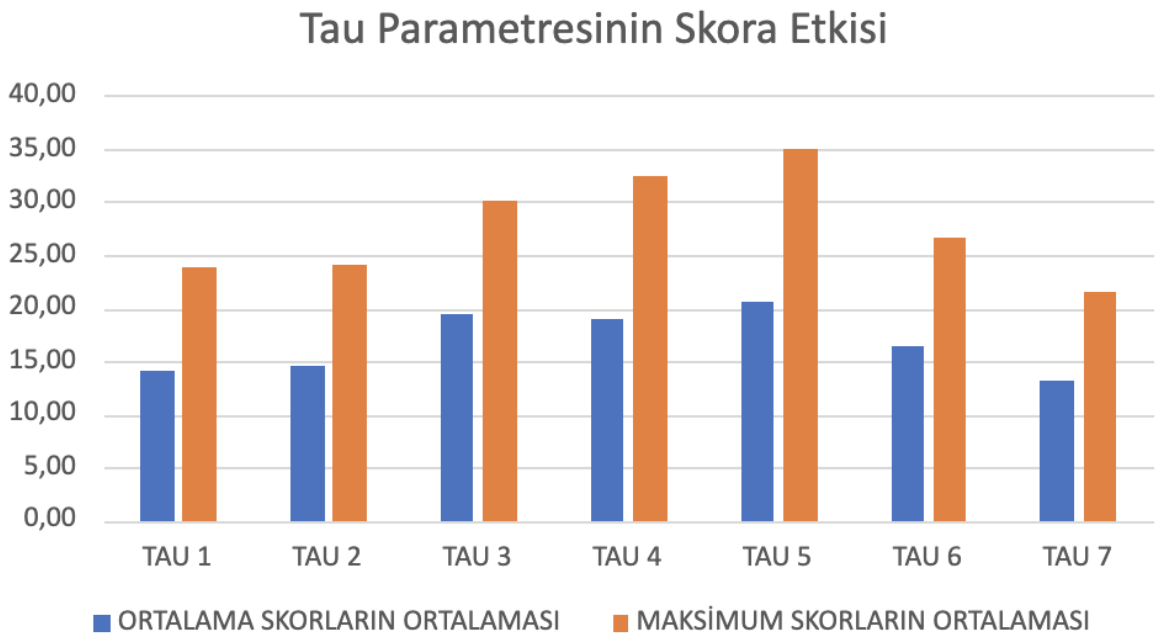
Step Parameter

It contains information on how many repetitions of one epoch the CNN algorithm will execute. It advances incrementally, topping the score with each repetition. Up until step 1500, it was seen that the score rose as the step count rose, then overfitting was noticed at step 1750. The average score with the greatest score is 1500 Steps with 34.26 points, while the average score with the lowest score is 250 Steps with 6.31 points.



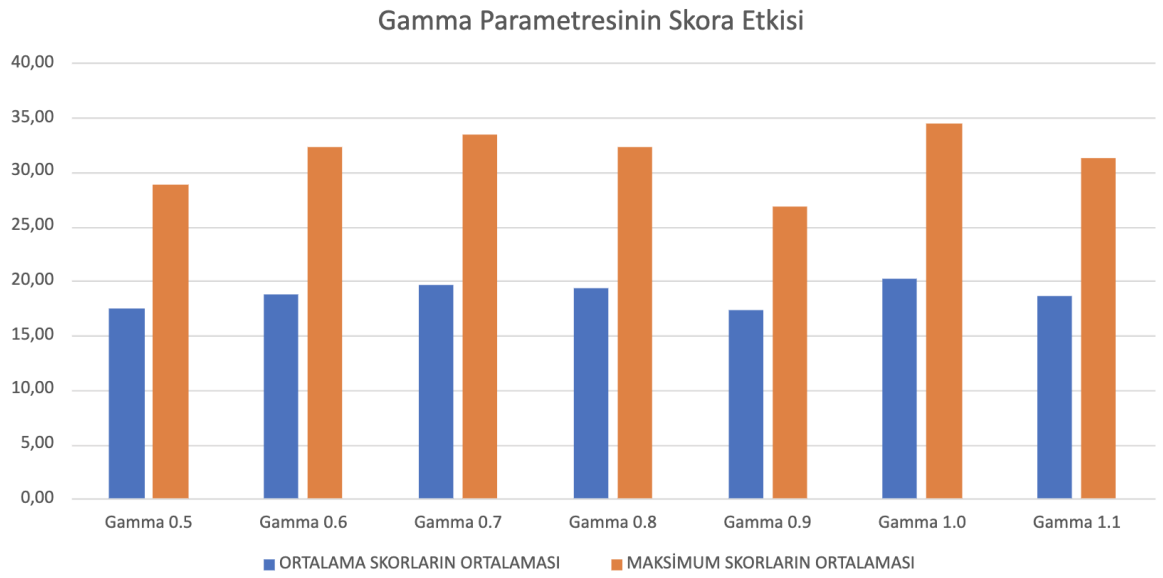
Tau Parameter

In order to determine whether there is a relationship between the variables and how strong it is, correlation coefficients are used. One of these association factors is tau. Up until 5, the tau coefficients were observed to increase steadily, but after that point, overfitting became apparent. The Tau number that receives the highest rating is 5.



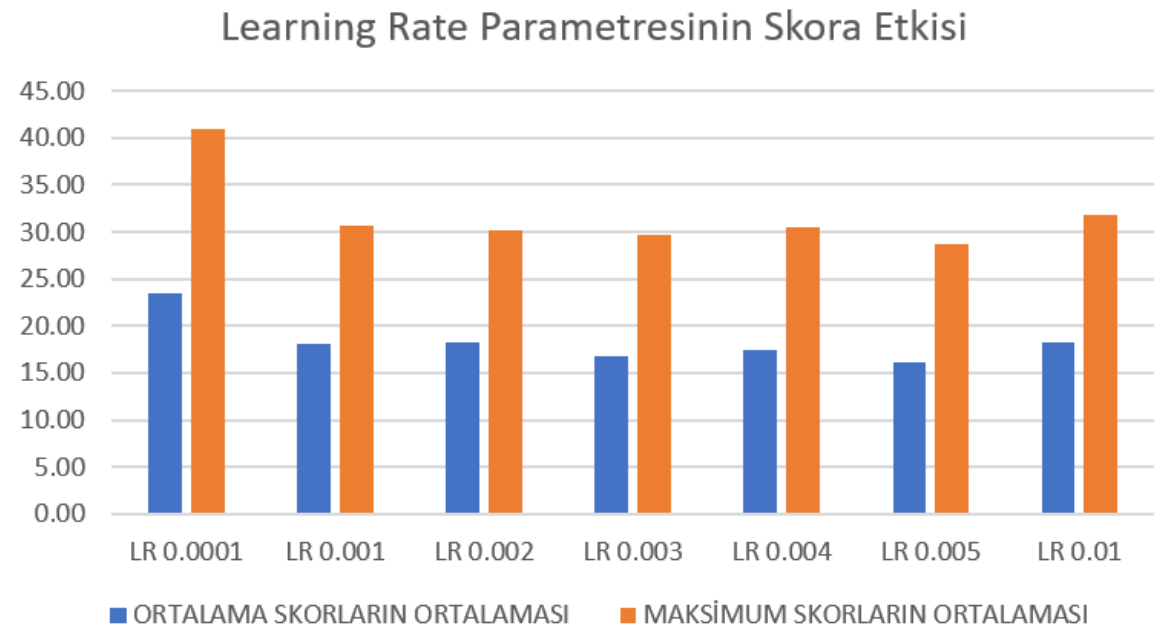
Gamma Parameter

From the starting value of 0.5 to 0.7, an increase was seen steadily, but after this point, drops and increases were seen to be unstable.



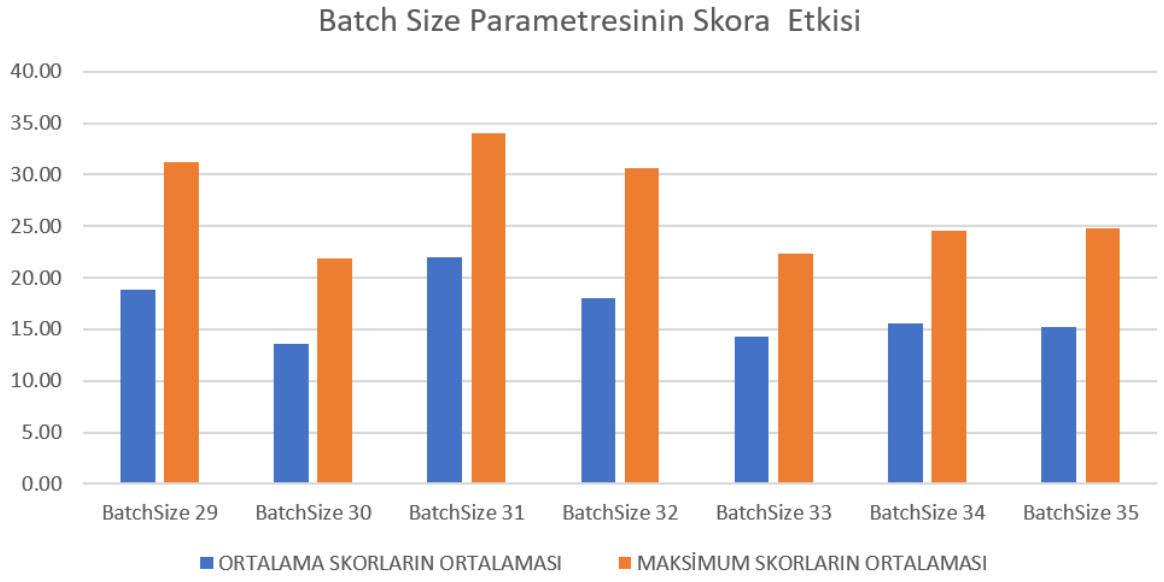
Learning Rate Parameter

A hyperparameter called Learning Rate regulates how much the model adjusts each time the model weights are changed in response to the predicted error. Results were generally similar because the values were near, however when the learning rate was 0.0001, it produced superior results at a lower learning rate. As a result, 0.0001 was our most effective learning rate number.



Batch Size Parameter

The quantity of samples to distribute across the network is determined by the batch size. Generally speaking, during training, the model will finish each epoch more quickly the higher the batch size. This is so that the computer can process numerous instances at once, depending on its computational capabilities. There is overfitting between 29 and 30 values. Our value at the top of the scale is 31.



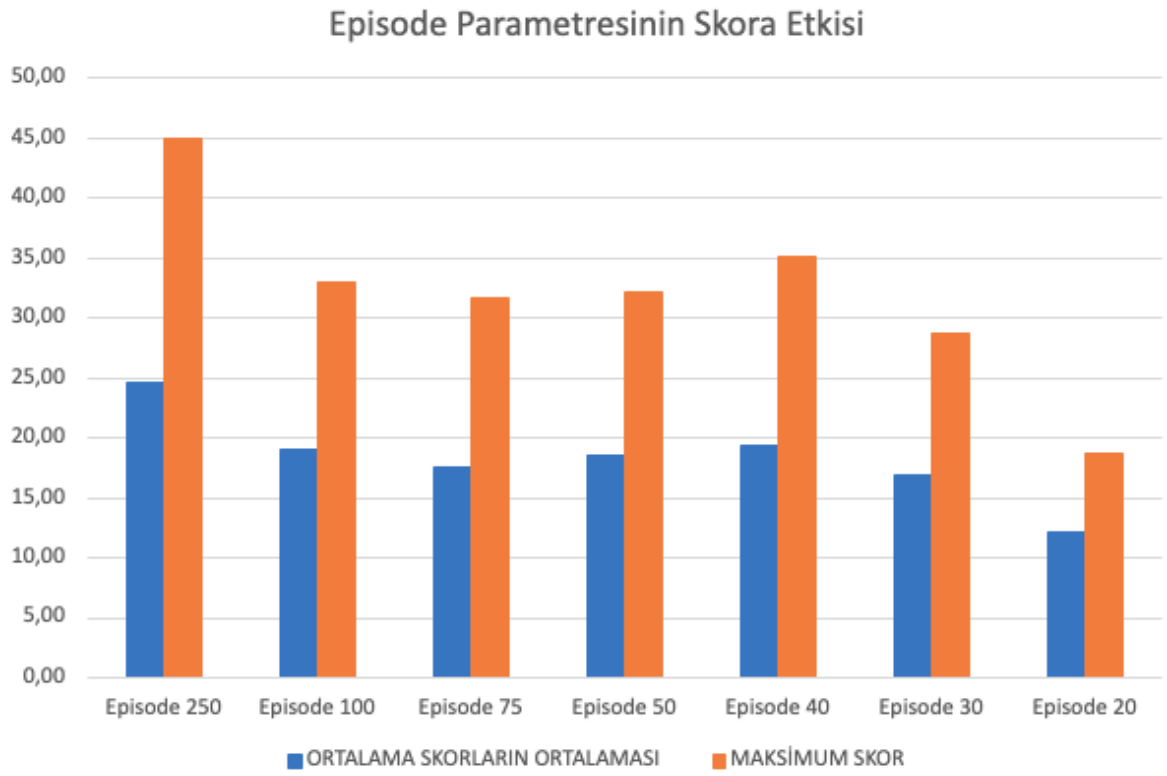
DEEP Q NETWORK

DQN	Used Values						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35
Epsilon	0.25	0.50	0.75	1.00	1.25	1.50	1.75

Epoch(Episode) Parameter

This option indicates how many times the Step parameter that has been chosen will be repeated. The score is reset after each repetition, allowing you to consolidate your prior knowledge. Our terminating condition is also the epoch parameter (stopping criteria). There

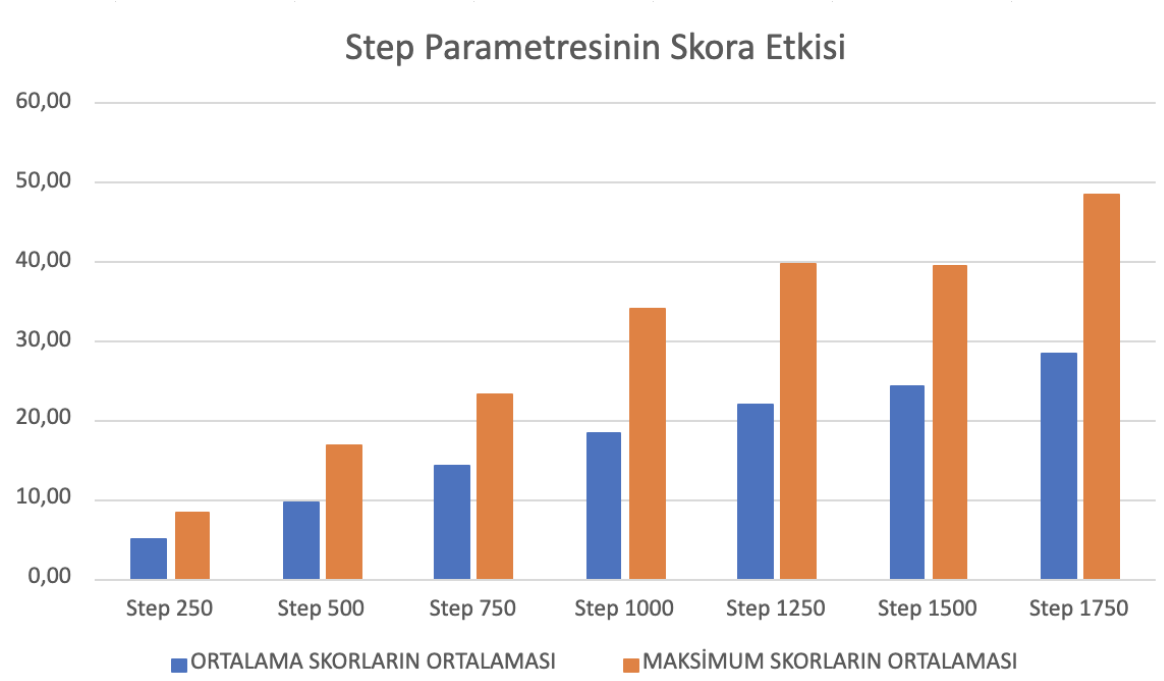
is between 50 and 250 overfitting that is seen. The greatest score was achieved with 250 episodes, despite the fact that the high episode value was exceedingly time-inefficient.



Step Parameter

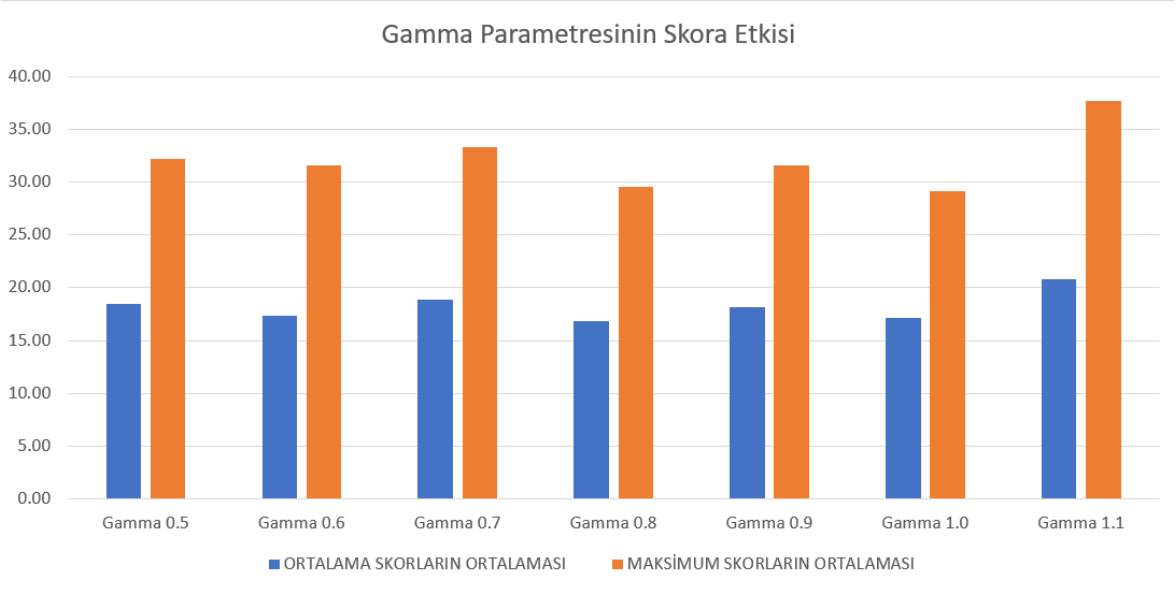
It contains information on how many repetitions of one epoch the CNN algorithm will execute. Each

It gains ground progressively, topping the repetition score. Up until step 1500, it was seen that the score rose as the step count rose, then overfitting was noticed at step 1750.



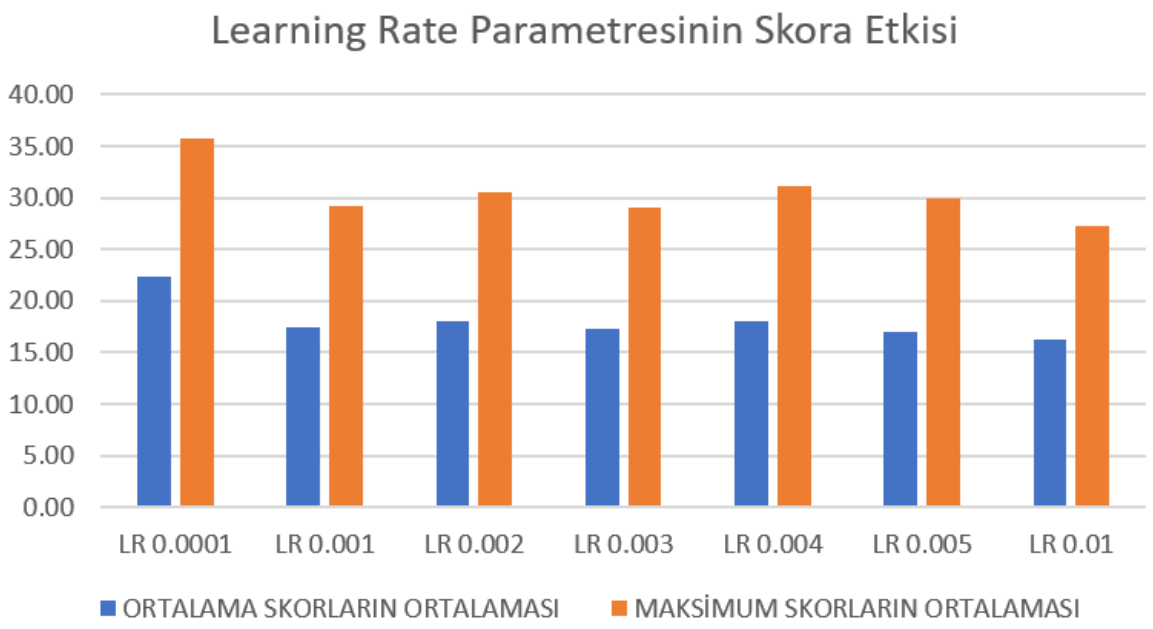
Gamma Parameter

Gamma began with a maximum score of 18.45 at 0.5, dropped to 17.39 at 0.6, then began to rise at 0.7, reaching a high score of 18.91. The structure of this parameter is ascending and descending. Gamma's average and maximum scores are both 1.1.



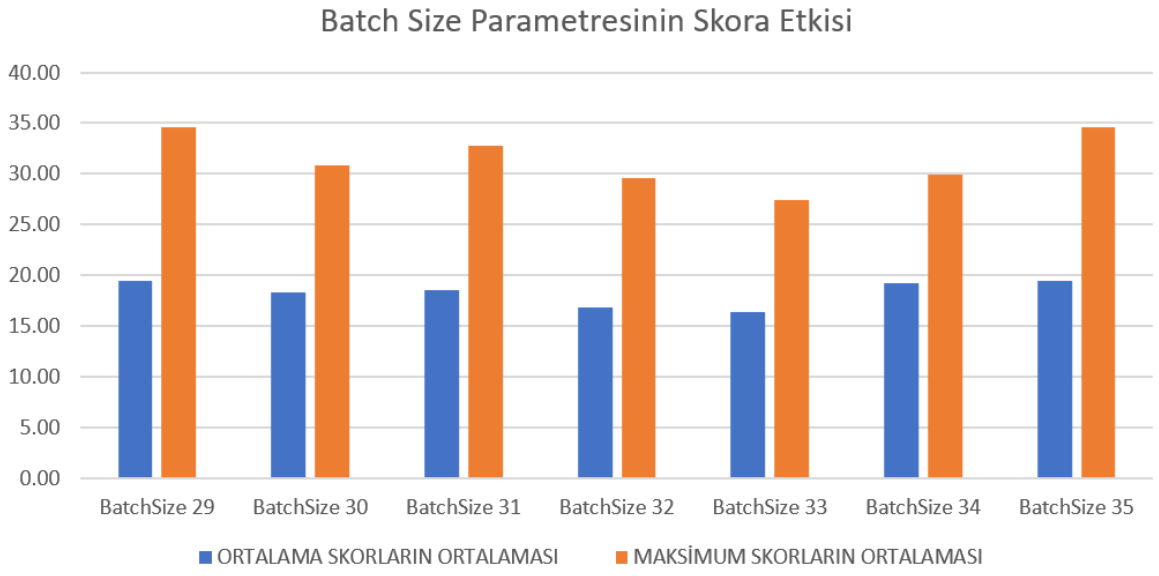
Learning Rate Parameter

The 0.0001 value, which makes a noticeable difference compared to other values, is our smallest value, and has achieved 22.37 points, showing superior success in maximums and averages. The worst success was 16.22 points and our highest value was 0.01. Here we deduce that the Learning Rate value should be small.



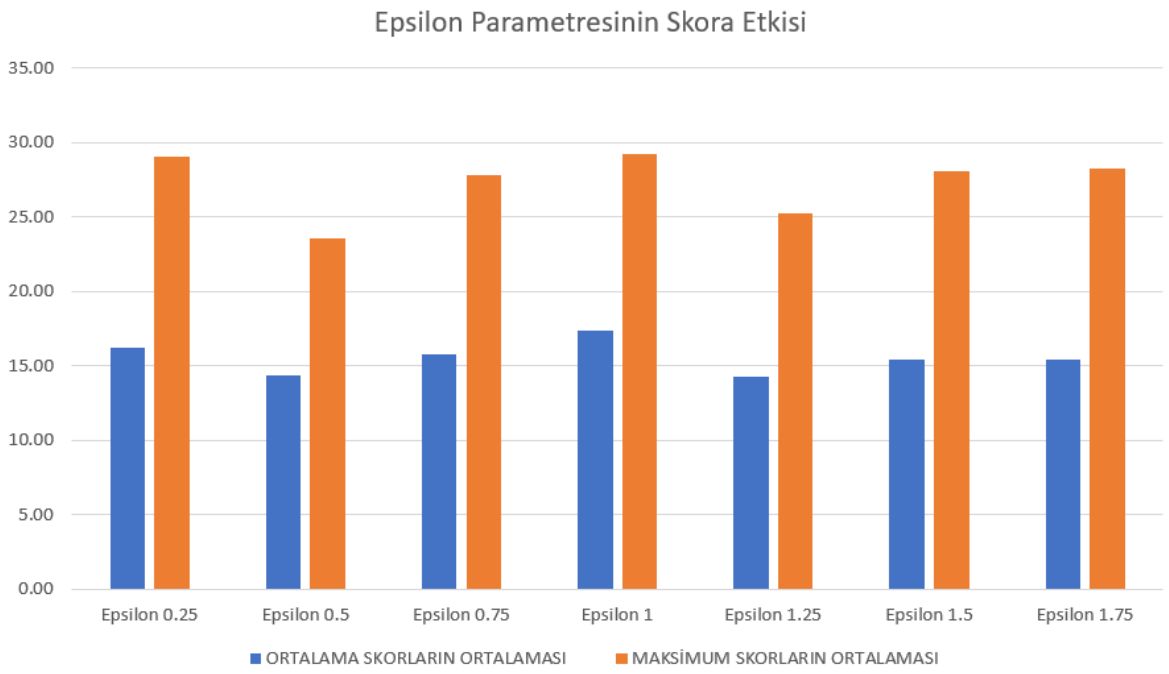
Batch Size Parameter

The batch size has a maximum score of 29, which reduced when it reached 30, but increased again when it reached 31. As a result, it generally has a rising and falling structure, but close values were nevertheless obtained. The maximum score is the same for batch sizes 29 and 35.



Epsilon Parameter

Epsilon parameter is the information of our usage amount of Greedy Exploration Strategy. The value reaching the maximum level on average is 1 with 17.37 points.

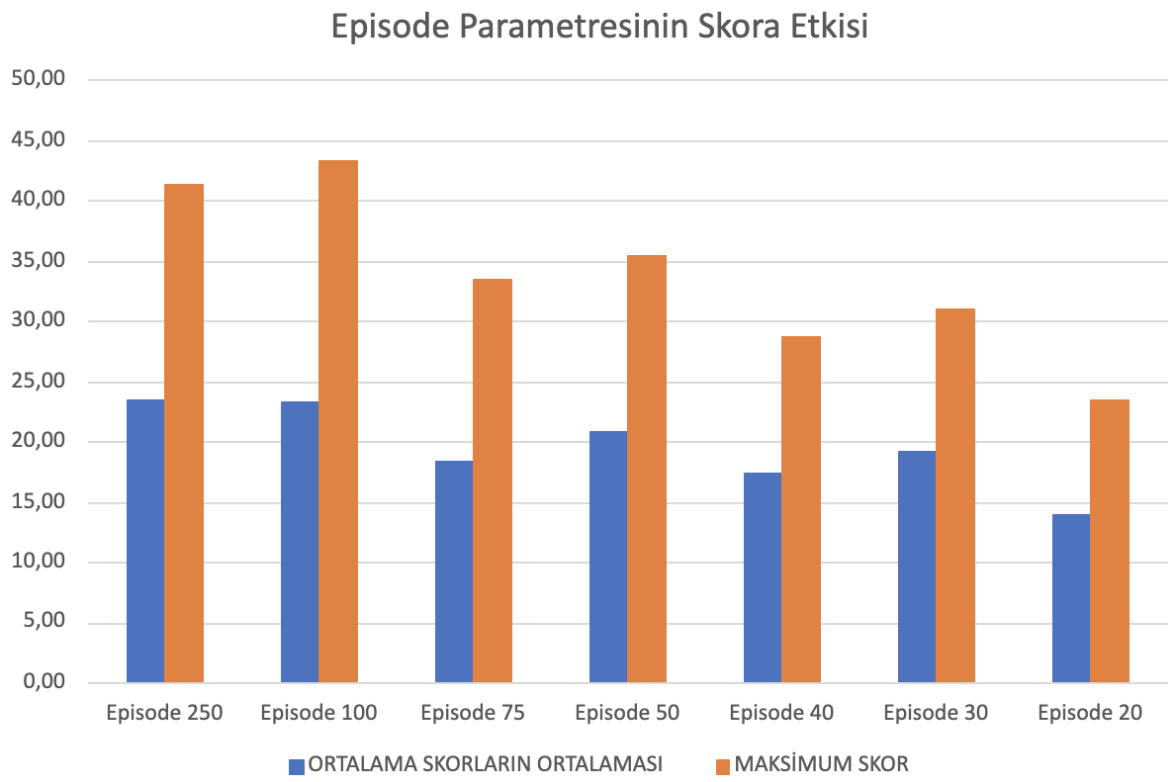


DEEP SARSA

SARSA	Used Values						
Episode	20	30	40	50	75	100	250
Step	250	500	750	1000	1250	1500	1750
Gamma	0.5	0.6	0.7	0.8	0.9	1.0	1.1
Learning Rate	0.0001	0.001	0.002	0.003	0.004	0.005	0.010
Batch Size	29	30	31	32	33	34	35
Epsilon	0.25	0.50	0.75	1.00	1.25	1.50	1.75

Epoch(Episode) Parameter

This parameter indicates how many times the Step parameter that has been chosen will be repeated. Each repetition's score advances by being reset and using what it has previously learned. Our terminating condition is also the epoch parameter (stopping criteria).

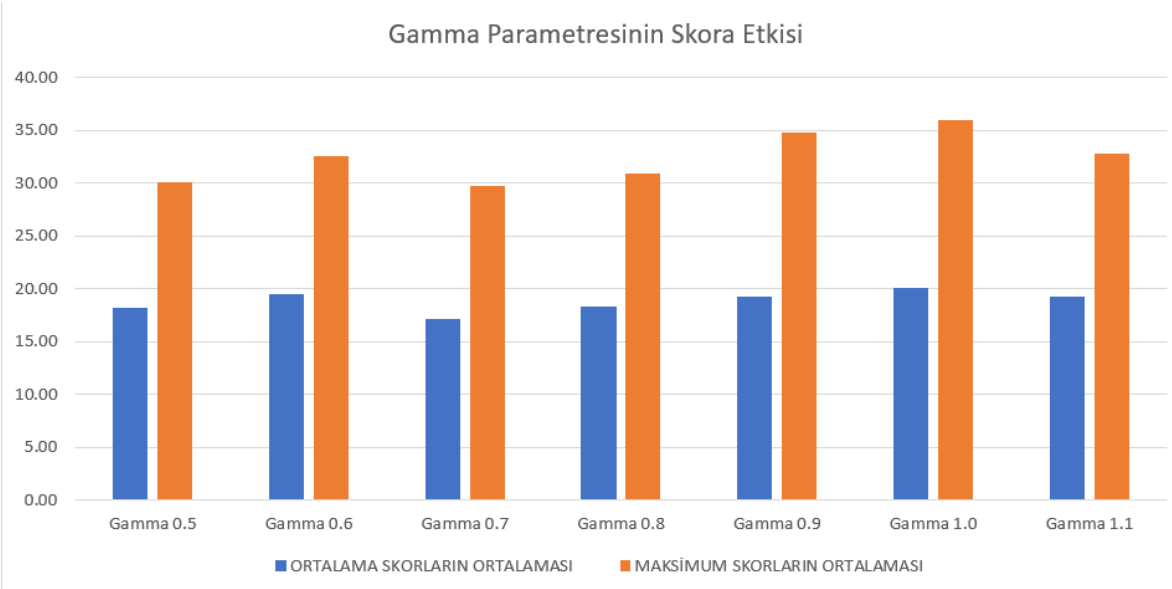


Step Parameter

It contains information on how many repetitions of one epoch the CNN algorithm will execute. It advances incrementally with each step, topping the repetition score. Up until step 1750, it was seen that the score rose as the step count rose.

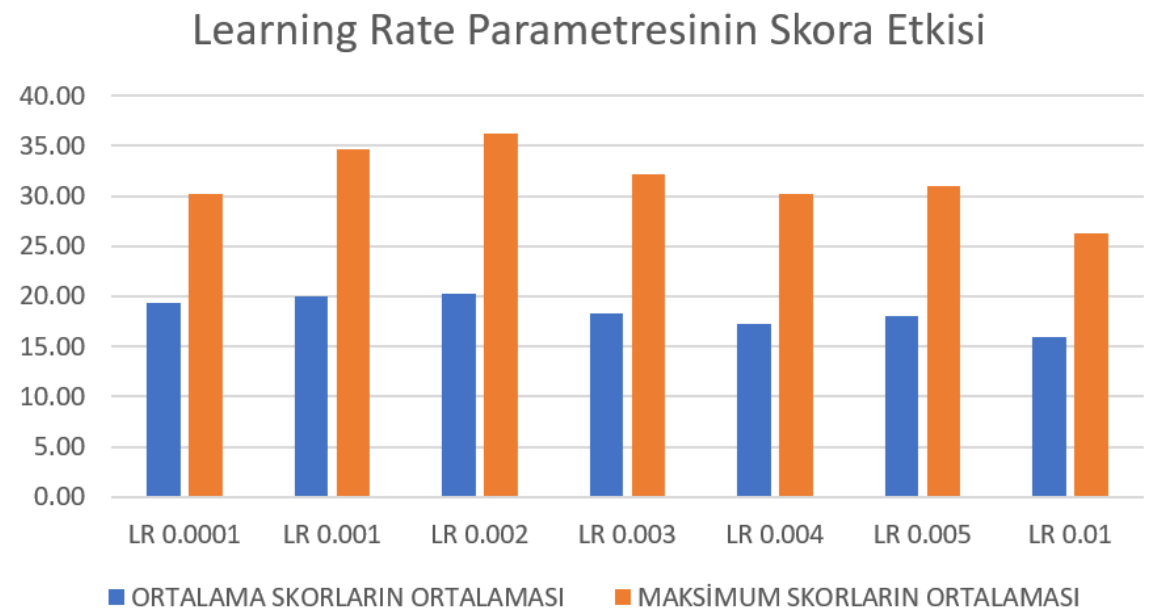
Gamma Parameter

Gamma values of 0.6-0.7 and 1.0-1.1 show overfitting. Although the values of 0.9 and 1.0 appear to be extremely close to one another, the maximum score is 1.0.



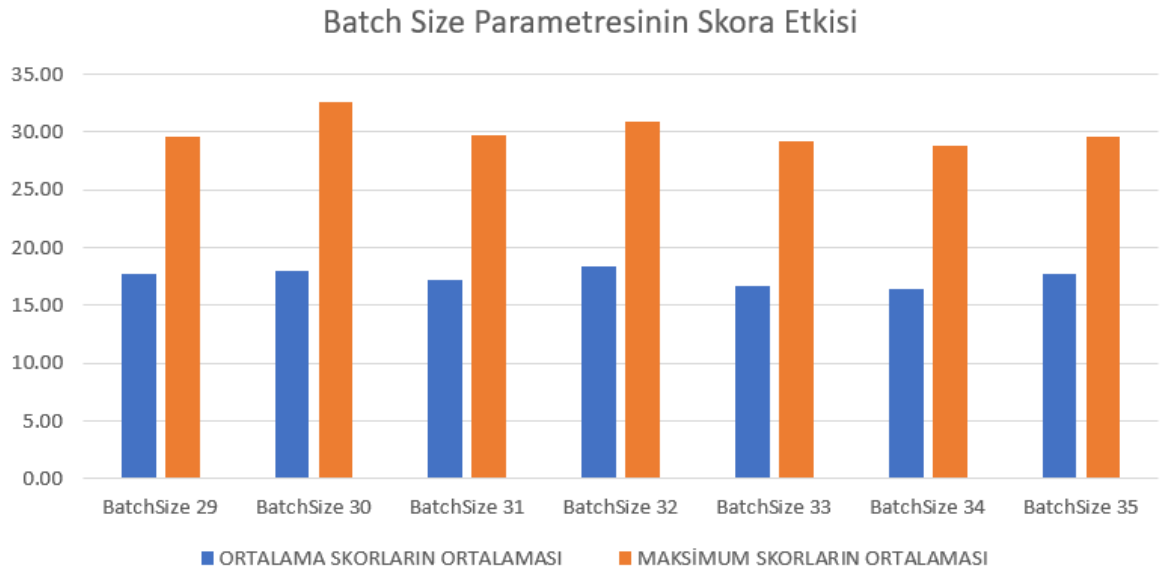
Learning Rate Parameter

The score continuously declined as the learning rate grew, reaching a maximum of 0.002. The 0.002 parameter scored the greatest on average with 20.24 points, while the 0.01 parameter scored the lowest on average with 15.94 points.



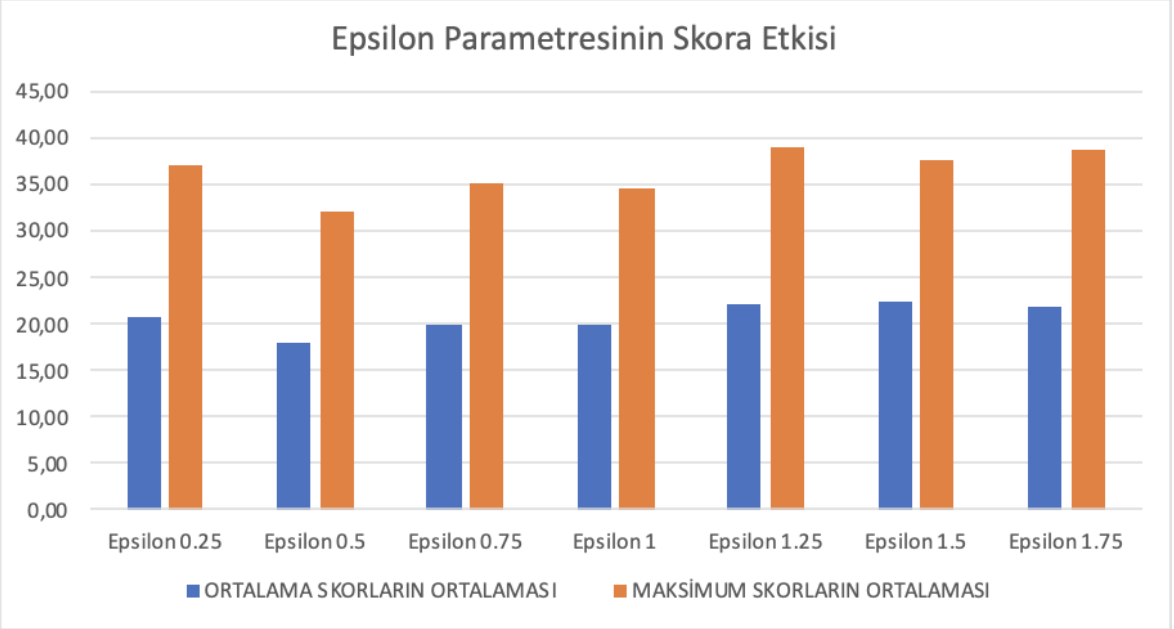
Batch Size Parameter

As can be seen, the maximum score average at values 29-30-31-32-33 has an increasing and decreasing structure. Despite the close proximity of the values, 32 produced the best outcome. 30 provided the best value when we looked at the average of the maximum scores.



Epsilon Parameter

The structure of this parameter is ascending and descending. The Epsilon score is 1.25, which is better than the others with a score of 22.39 on the average of the average scores, even though there isn't a clear winner. The 0.05 parameter had the lowest average score, coming in at 18.05 points.

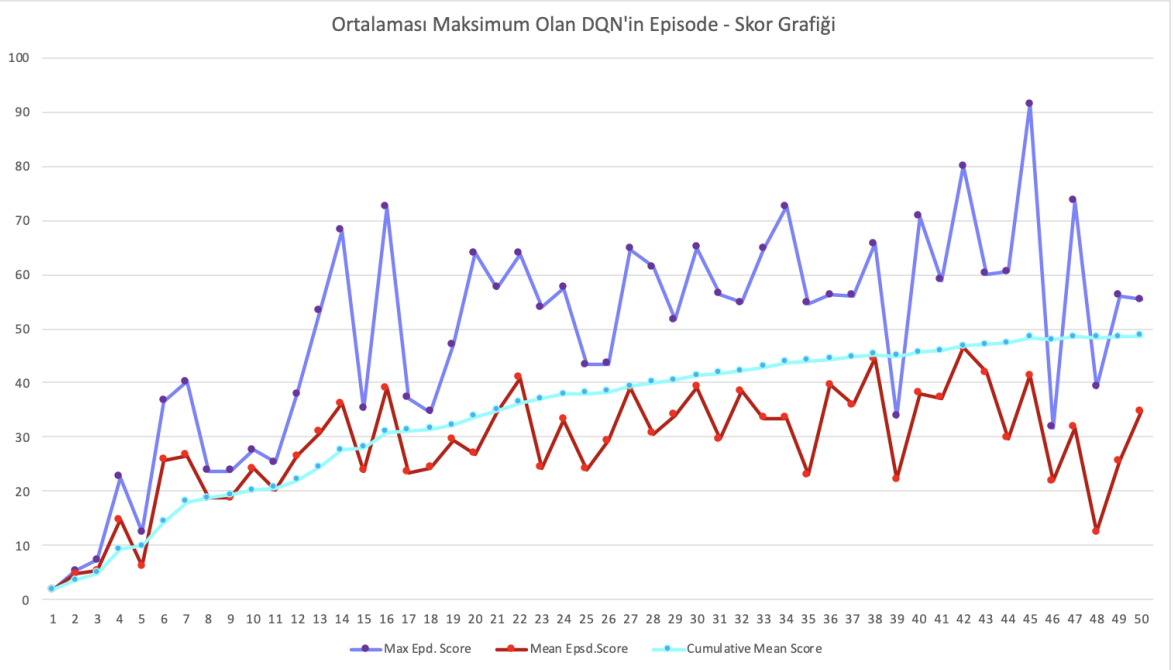


CONCLUSION

To determine the best scoring system for the Agar.io game, we trained an agar model using the each CNN, Deep Q Network, and SARSA algorithms.
in DQN,
The highest value in the average of the averages appeared in Step 1750. The highest value among the maximums appeared in Episode 250.

DQN

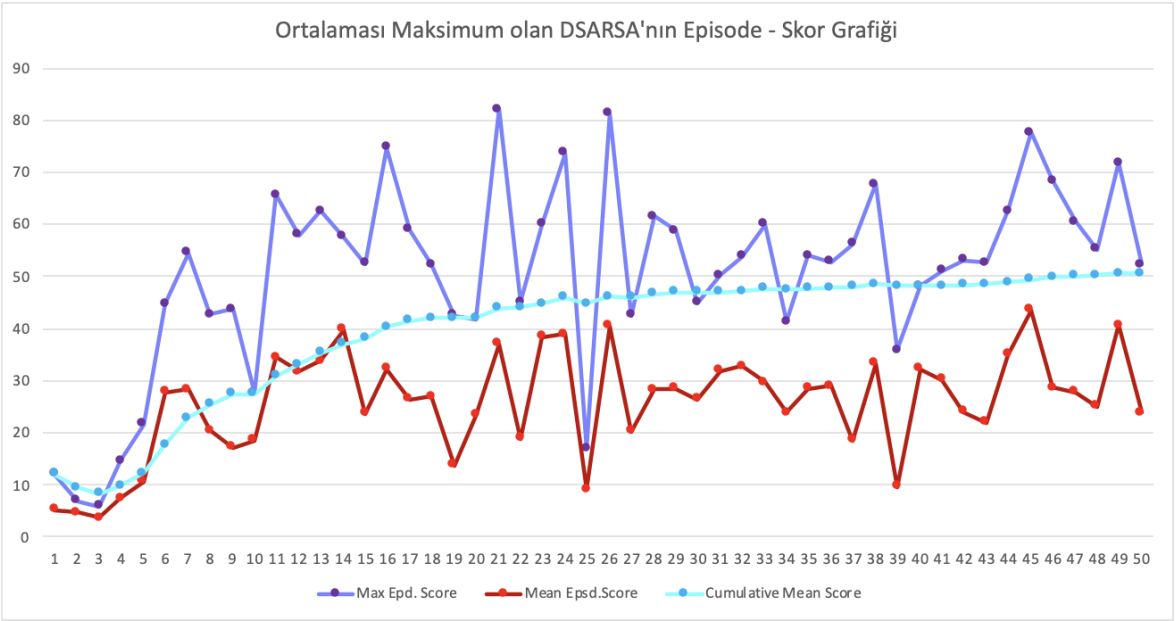
Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	24.72	19.03	17.67	18.65	19.35	16.93	12.14
Max	198.32	64.89	54.01	66.53	61.14	54.55	38.21
Değer	Step 250	Step 500	Step 750	Step 1000	Step 1250	Step 1500	Step 1750
Ort	4.96	9.61	14.28	18.55	21.94	24.42	28.38
Max	21.51	35.97	51.13	62.06	70.09	67.96	91.34
Değer	BatchSize 29	BatchSize 30	BatchSize 31	BatchSize 32	BatchSize 33	BatchSize 34	BatchSize 35
Ort	19.44	18.24	18.49	16.79	16.33	19.24	19.44
Max	55.92	59.87	57.76	57.22	54.48	58.19	66.17
Değer	Gamma 0.5	Gamma 0.6	Gamma 0.7	Gamma 0.8	Gamma 0.9	Gamma 1.0	Gamma 1.1
Ort	18.45	17.39	18.91	16.79	18.16	17.10	20.77
Max	64.3	61.34	57.34	57.22	57.48	62.27	64.25
Değer	LR 0.0001	LR 0.001	LR 0.002	LR 0.003	LR 0.004	LR 0.005	LR 0.01
Ort	22.37	17.37	18.00	17.26	18.02	16.97	16.22
Max	60.18	58.94	64.14	55.22	59.61	57.05	52.9
Değer	Epsilon 0.25	Epsilon 0.5	Epsilon 0.75	Epsilon 1	Epsilon 1.25	Epsilon 1.5	Epsilon 1.75
Ort	16.24	14.34	15.78	17.37	14.31	15.46	15.44
Max	57.19	45.91	51.08	58.99	46.78	49.88	51.06



In SARSA,
In the average of the averages, the highest value appeared in Step 17500. The highest value among the maximums appeared in Episode 100.

DSARSA

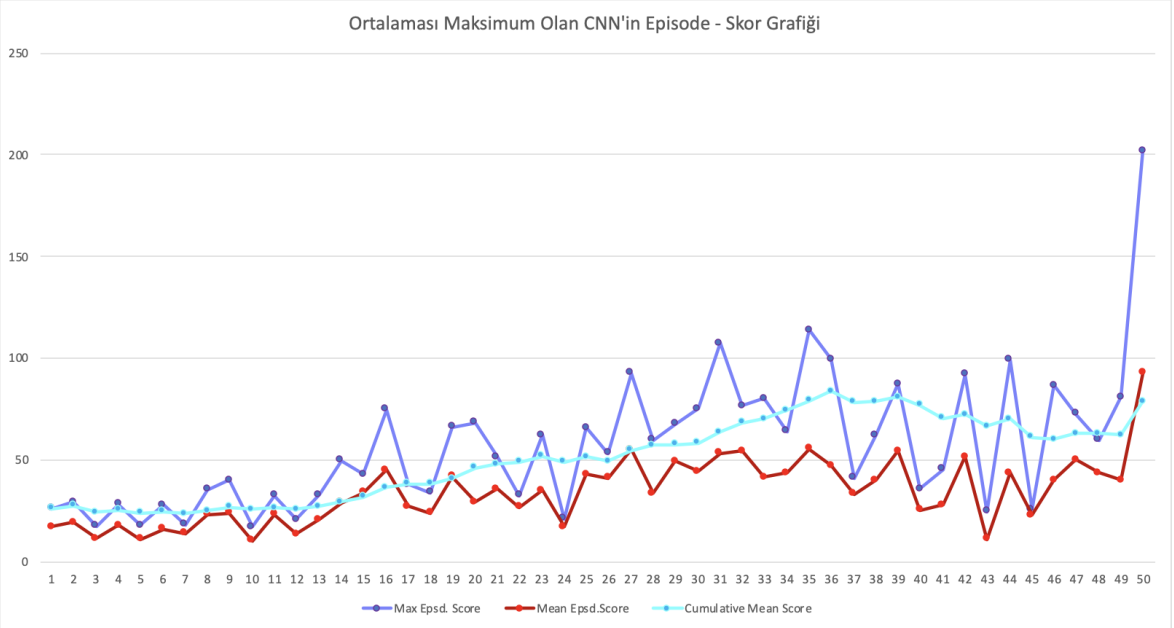
Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	23,49	23,37	18,44	20,87	17,48	19,19	13,99
Max	76,04	79,23	72,25	61,8	55,71	56,54	44,13
Değer	Step 250	Step 500	Step 750	Step 1000	Step 1250	Step 1500	Step 1750
Ort	3,91	11,01	13,58	19,05	22,32	24,47	25,74
Max	17,85	43,55	43,92	63,6	71,44	78,27	82,03
Değer	BatchSize 29	BatchSize 30	BatchSize 31	BatchSize 32	BatchSize 33	BatchSize 34	BatchSize 35
Ort	17,68	18,02	17,16	18,38	16,64	16,43	17,68
Max	55,39	62,07	58,66	58,87	52,37	55,52	73,4
Değer	Gamma 0.5	Gamma 0.6	Gamma 0.7	Gamma 0.8	Gamma 0.9	Gamma 1.0	Gamma 1.1
Ort	18,19	19,51	17,19	18,38	19,29	20,08	19,26
Max	63,24	62,8	56,28	58,87	67,32	62,16	76,08
Değer	LR 0.0001	LR 0.001	LR 0.002	LR 0.003	LR 0.004	LR 0.005	LR 0.01
Ort	19,36	19,96	20,24	18,27	17,25	18,07	15,94
Max	57,56	65,15	71,84	60,74	55,54	54,71	44,63
Değer	Epsilon 0.25	Epsilon 0.5	Epsilon 0.75	Epsilon 1	Epsilon 1.25	Epsilon 1.5	Epsilon 1.75
Ort	20,84	18,05	20,05	19,95	22,17	22,39	21,9
Max	69,95	65,64	65,35	65,15	78,73	63,14	68,78



In CNN,
The highest value in the mean of the averages appeared in Step 1500. The highest value among the maximums appeared in Episode 250.

CNN

Değer	Episode 250	Episode 100	Episode 75	Episode 50	Episode 40	Episode 30	Episode 20
Ort	29,52	29,01	17,97	22,91	16,61	16,18	11,06
Max	222,89	206,08	67,73	83,71	77,36	63,67	31,24
Değer	**Step 250**	**Step 500**	**Step 750**	**Step 1000**	**Step 1250**	**Step 1500**	**Step 1750**
Ort	6,31	7,22	12,24	22,91	26,23	34,26	23,41
Max	21,52	23,84	49,36	83,71	87,65	201,77	198,48
Değer	**BatchSize 29**	**BatchSize 30**	**BatchSize 31**	**BatchSize 32**	**BatchSize 33**	**BatchSize 34**	**BatchSize 35**
Ort	18,89	13,59	21,94	18,07	14,34	15,52	15,20
Max	62,34	62,82	89,63	71,15	54,2	57,84	56,12
Değer	**Gamma 0.5**	**Gamma 0.6**	**Gamma 0.7**	**Gamma 0.8**	**Gamma 0.9**	**Gamma 1.0**	**Gamma 1.1**
Ort	17,48	18,71	19,62	19,36	17,31	20,19	18,67
Max	59,8	59,4	69,72	69,03	54,1	73,28	64,95
Değer	**LR 0.0001**	**LR 0.001**	**LR 0.002**	**LR 0.003**	**LR 0.004**	**LR 0.005**	**LR 0.01**
Ort	23,54	18,07	18,27	16,86	17,49	16,09	18,28
Max	82,62	71,15	61,88	72,29	55,65	62,27	54,48
Değer	**TAU 1**	**TAU 2**	**TAU 3**	**TAU 4**	**TAU 5**	**TAU 6**	**TAU 7**
Ort	13,96	14,43	19,34	18,92	20,49	16,41	13,08
Max	54,97	51,26	59,62	71,7	70,82	81,66	44,3



Even though CNN outperforms Deep Q Network and Deep Sarsa in terms of maximum and average success, it is less effective since it treats each pixel in each scene as a binary in terms of duration. Although Deep Q Network and Deep SARSA offer comparable results when compared to one another, Deep Q Network performs better.

REFERENCES

WEB_1(2021)

WEB_2(2018)

WEB_3(2015)

WEB_4(2020)

WEB_5(2020)

WEB_6(2017)

WEB_7(2020)

WEB_8

Artificial Intelligence For Games (2nd Edition), Ian Millington & John Funge