

Deep Reinforcement Learning for Game-Playing Agents

Project Report

Presented to

Dr. Chris Pollett

Department of Computer Science

San José State University

In Partial Fulfillment

Of the Requirements for the Class

CS 297

By

Ayan Abhiranya Singh

December, 2022

TABLE OF CONTENTS

I. Abstract.....1

II. Introduction.....1

III. Deliverable I: Q-Learning in the Vacuum World2

IV. Deliverable II: Deep Q-Learning in the Vacuum World.....3

V. Deliverable III: Q-Learning Agent for Ms. Pac-Man.....5

VI. Deliverable IV: Deep Q-Network for Ms. Pac-Man.....7

VII. Requirements.....10

VIII. Progression Timeline.....11

IX. Conclusion.....11

References.....13

I. ABSTRACT

Deep learning is a subfield of machine learning that deals with modeling human brains and implementing them as neural networks. Video games are one application of deep learning. Older games, such as for the Atari 2600, provide great environments for testing reinforcement learning algorithms. Through reinforcement learning, an agent learns about its environment through the delivery of periodic rewards. One such algorithm is known as Q-learning. Q-learning is a form of reinforcement learning that allows an agent to act on a quality-function, defined as $Q(s,a)$, that denotes the sum of rewards from state s onwards if an action a is taken. Q-learning has shown great results in training agents to play Atari 2600 games like Space Invaders and Breakout. In this project, we train a neural network that learns control policies for Ms. PacMan and find that it is able to score 3700 points and almost win the first map outright through this process.

Keywords: Deep learning, Q-Learning, Video games, Atari, Reinforcement learning

II. INTRODUCTION

For my project, I aim to learn more about applications of reinforcement learning and how deep learning networks can leverage this to implement agents that can play video games at a near-superhuman level. I will then build agents of my own that are able to play video games of increasing complexity. We now discuss the organization of the rest of this paper. In the next section, we discuss the Q-learning algorithm and how it may be utilized to train an agent to learn the dirt-generation policies of a 5x5 “vacuum-world” game of our own design. The section after that elaborates how the same agent can use a deep Q-network to learn the shortest path among dirt positions across a varying size of grids. These two approaches can encompass the scope of the first two deliverables of this semester.

Over the next two sections (Deliverables 3 and 4), agents that leverage the deep Q-network defined in [1] will be trained and tested on the classic Atari game, “Ms PacMan”. For Deliverable 3, an agent trained with the simple network from Deliverable 2 will be deployed to play the game using the Arcade Learning Environment (ALE) on Python. Deliverable 4 will utilize a deep Q-Network, based on [1], to do the same. The agent for Deliverable 4 will be trained over several thousand episodes and tested to see if it is able to break 3000 points on Ms PacMan. The paper closes with discussion of the technical requirements of the project, the timeline and a conclusion summarizing the results of the semester and what we plan to do with them in the future.

III. DELIVERABLE I: Q-LEARNING IN THE VACUUM WORLD

Deep learning was invented around the 1960s, when Alexey Ivakhnenko devised one of the first “neural networks” [6]. Deep learning utilizes multi-layered neural networks to perform predictions on or analyze data. Neural networks are artificial structures that are composed of interconnected node layers, representing the neurons of the brain. The models used in deep learning are usually trained by huge sets of labeled data.

Deep neural networks (DNNs) can be used to interpret game frames as a set of raw pixels [1], to learn a control policy that achieves the maximum amount of reward. The policy dictates which action an agent should take given a specific state within the game, i.e., a set of pixel values. The networks then output a probability for each of the actions to take. The reinforcement learning feedback loop works such that an agent is presented with an observation of the environment at a given time step. The agent performs an action on the environment based on this state and the environment returns a reward. For the purposes of vacuum world, the reward is 100 if a dirty square is encountered or 0 if the cell is clean.

The popular "vacuum world" is a common introductory problem into the world of artificial intelligence. This is a search problem where our agent is a vacuum that must traverse a world (a grid, in our case) and locate and clean as many dirty cells as possible. Our implementation of vacuum world builds upon this classic problem and attempts to solve for k-many dirty squares utilizing Q-learning.

The agent is trained over 5000 episodes, with each episode consisting of a 100 steps. An exploration probability, ϵ , is assigned to the agent. The value of ϵ is set to 0.5. This value determines the chance that the agent explores in a random direction or uses the q-table to predict which direction returns the maximum potential reward. Additionally, a discount factor, γ , is used to discount (scale down) future rewards. Initially, each square of the grid is initialized to a reward of -1, with only dirty squares having a reward of 100. The lookup table is updated for each choice taken to transition from state s to state s' . The updated q value is computed as the sum reward of the new state s' and the discounted maximum reward of actions for s' . In this manner, the agent can traverse the grid over a number of "random walks" and learn what actions produce maximum reward, given a state observation s . This is of particular importance, as we ultimately want to replace this q-table with a neural net that outputs a tensor of values that do essentially the same thing.

IV. DELIVERABLE II: DEEP Q-LEARNING IN THE VACUUM WORLD

Over the course of Deliverable 2, the deep q-learning algorithm from [1] is implemented through means of a deep neural network. Neural networks, convolutional neural networks (CNN) in particular, have historically been great at image classification tasks [2]. The neural network for this task has 3 layers, each fully connected to the next. The first 2 layers both contain 32 neurons, with the first layer accepting an input of the game screen ($n \times n$ grid). The final fully connected

layer has 4 outputs, corresponding to the actions that an agent can take within the world (up, down, left or right).

To train this network, the state space array (possible states of the game) is initialized to zero. Each visited state is marked as 1. Random walks with a pre-determined set of steps are executed within the vacuum world. Each step is chosen with the same exploration probability from the first deliverable, otherwise, the network predicts a set of actions for the agent to take. The action is executed within the game environment and each state, new state and reward is added to memory. The walk terminates if dirt is found (reward of 100), or if the number of steps for the walk are completed.

Once the memory exceeds a specified number of frames, the frames are replayed with the reward updated in the same manner as deliverable 1. As there is no longer a q-table in this deliverable, the neural network is trained on these sample frames and rewards. Over time, the network is able to determine the path with the highest reward, as demonstrated in Figures 1 and 2.

```

***** EPISODE 0 *****
1/1 [=====] - 0s 27ms/step
Move DOWN
1/1 [=====] - 0s 21ms/step
Move DOWN
1/1 [=====] - 0s 26ms/step
Move RIGHT
1/1 [=====] - 0s 53ms/step
Move RIGHT
1/1 [=====] - 0s 34ms/step
Move RIGHT
1/1 [=====] - 0s 36ms/step
Move DOWN
1.0

```




Figure 1. Deep Q-agent searching for dirt, located at square G, starting from square S. The neural network outputs the shortest path to goal.

```

***** EPISODE 1 *****
1/1 [=====] - 0s 18ms/step
Move DOWN
1/1 [=====] - 0s 18ms/step
Move DOWN
1/1 [=====] - 0s 17ms/step
Move DOWN
1.0

```




Figure 2. Similar to figure 1, the neural network outputs the shortest path to goal.

V. DELIVERABLE III: Q-LEARNING AGENT FOR MS. PAC-MAN

The neural networks outlined in the previous section are derived from the work of Mnih et al. [1]. The goal of their paper was to learn control policies similar to the approach for vacuum world, but for classic Atari 2600 video games instead. The Deep Q-Learning algorithm proposed in their paper was ultimately able to reach superhuman levels of gameplay on Space Invaders, Breakout, Pong, Beam Rider and Seaquest.

They utilize convolutional neural networks to interpret video data, broken down into a series of frames. These frames are provided to the neural network as input in the form of pixel values. The agent's experiences are stored in memory and replayed in batches of random samples. The q-values for these random samples are updated after replay. The samples are chosen at random to boost efficiency and reduce the chance of getting stuck at a local minima due to the correlation between linked consecutive frame samples.

Ultimately, the network is a resounding success at playing the Atari games because of their finite action spaces and relatively simple game environments. For most of these games, like Space Invaders or Breakout, an action taken has an easy observable state change which is measurable by the reward of the new state. The Arcade Learning Environment [4] provides a great package within Python for executing actions within the Stella Atari 2600 emulator and receiving a reward in return. As it turns out, the ALE also returns the new game state after

executing an action and this is of particular importance as it allows the Deep Q-Learning network to be set up. The OpenAI Gym API provides a wide variety of reference environments ALE has native support for, including several hundred classic Atari games. Ms. PacMan, the target game for the scope of the third and fourth deliverable of this semester, is also available as a potential game environment in Gym. PacMan has been studied using deep reinforcement models before. However, a search of the internet and research articles and journals indicates that it has not been the subject of a comprehensive reinforcement learning approach like [1], and as such it is a perfect environment for the purposes of this project.

In Deliverable 3, an agent is trained using the neural network from Deliverable 2, based on a running reward system. In Ms Pacman, the reward is 1 if food was present on the cell and 0 otherwise. The network is initialized with a state space of 500. The running reward is the mean of all rewards across the episodes that the net has been trained. Initially, we expect to see the running reward fluctuate around a low value. Over time, as the agent learns the proper Q-values for the PacMan map, we expect the running reward to rise. Training for a running reward of 110, the agent takes 170 episodes to converge and is able to score a mean of 90 over 10 trials. While this is promising, it does not tell us much about whether the agent is potentially able to solve the game of PacMan or not. However, there are a few observations from this deliverable that will prove to be key in the next section. Pac-Man provides a different, albeit interesting sandbox for testing the deep Q-learning algorithm proposed in [1] for a number of reasons:

- i) The reward signal from consuming food is consistently high at the beginning of the game but becomes sparser as the food around the map gets consumed, akin to Breakout.
- ii) The reward signal can provide a high reward in cases where an alien is consumed

after powering-up, however, the correlation might not necessary reflect in training the network due to random sampling.

- iii) Certain inputs might not always work as the agent is always navigating a narrow space in the maze. Often, only two or three of the total actions available might result in movement, which means there may be timesteps where the agent stays still even though a movement action is executed.
- iv) The ghosts get “better” at playing as the game goes on, i.e., they chase more smartly and become faster.
- v) The state space of PacMan is undefinable due to the randomness of the AI, the size of the map, the variations of movement pattern of the agent across episodes. Therefore, initializing the model with a state space of 500 is not a feasible approach.

Due to these observations, our initial working hypothesis is to build and train a deep learning model that is different from the one in Deliverable 2 and more similar to the one proposed in [1], that is able to score at least 3000 points on Ms. Pac-Man or win a map outright.

VI. DELIVERABLE IV: DEEP Q-NETWORK FOR MS. PAC-MAN

The approach will be to build a deep Q-network that aggregates the learning across the first two deliverables and [1] and then apply that to the game of PacMan. The network implemented is a convolutional neural network as defined in the DeepMind paper and consists of:

1. 3 convolutional layers:
 - i) 32 filters of 8x8, stride of 4, activated by ReLU
 - ii) 64 filters of 4x4, stride of 2, activated by ReLU
 - iii) 64 filters of 3x3, stride of 1, activated by ReLU

2. 2 dense layers:

- i) 512 neurons, activated by ReLU
- ii) 5 neurons (size of the action space for the agent), with a linear activation function

The network takes the raw pixels of the game screen as input. Figure 3 describes the results of agent, in terms of score vs the number of episodes it was trained for.

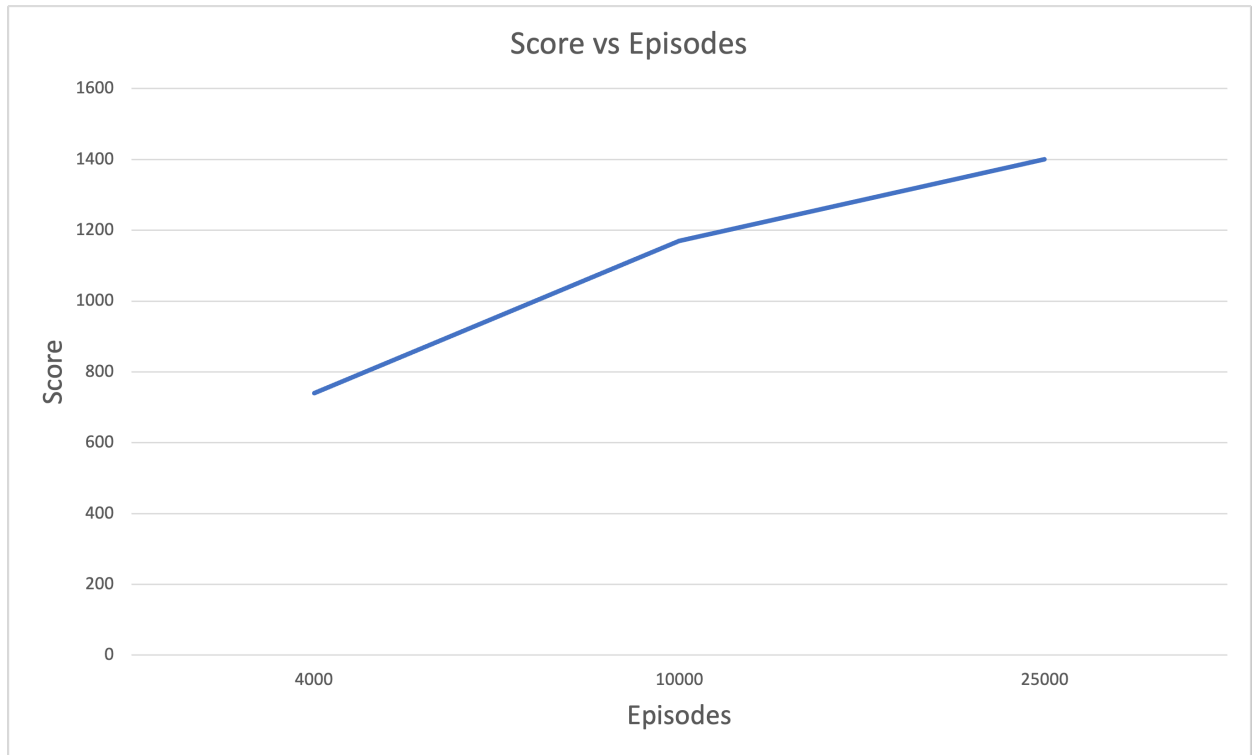


Figure 3. The agent demonstrates increasing performance over training episodes.

The agent is trained on the full move set here, which includes joystick actions that correspond to diagonal input, like UPLEFT, UPRIGHT, DOWNLEFT, DOWNRIGHT. The agent also has a FIRE command, which is not used in the PacMan game. Additionally, the fire command is combined into the directional moves to create moves like UPFIRE, UPLEFTFIRE, etc.

Downsizing the list of valid actions to from the original 18 to just UP, DOWN, LEFT, RIGHT and NOOP might help make training more efficient and as a result, create a smarter agent for our game. Figure 4 illustrates the performance of the agent on the reduced moveset in terms of score, while Figure 5 charts data for training episodes vs time steps survived.

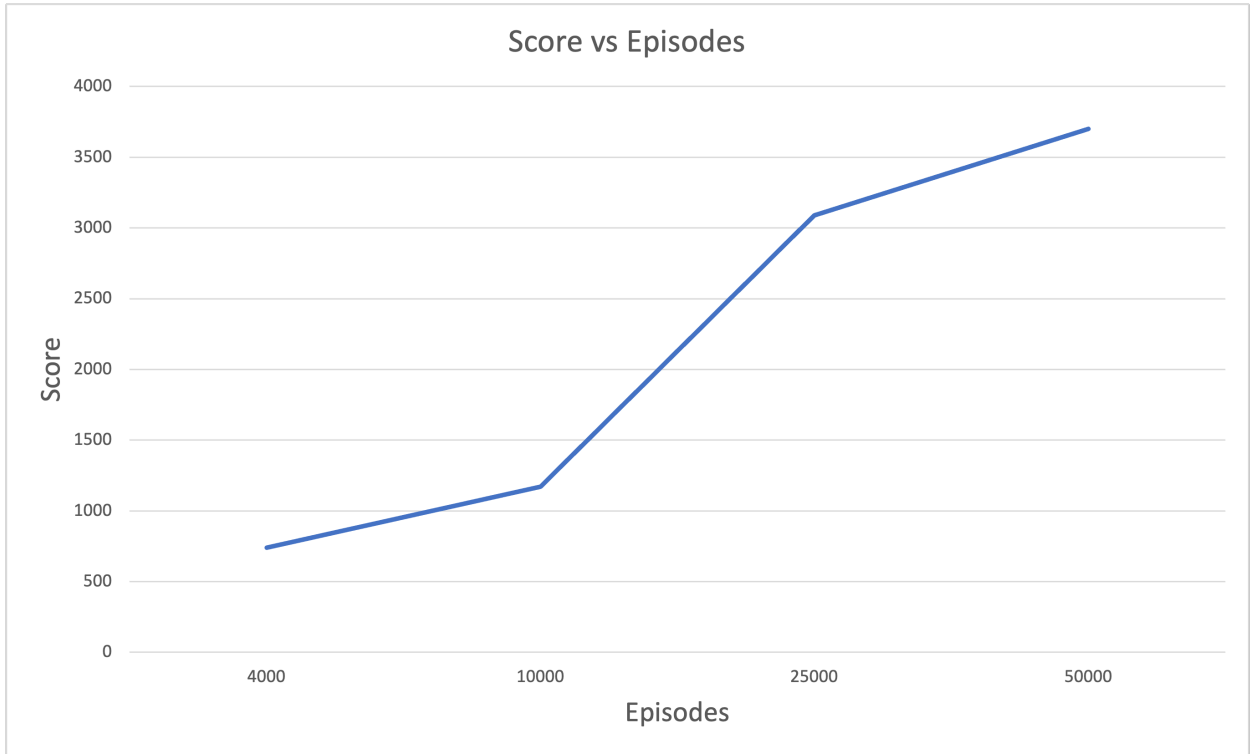


Figure 4. Agent performance with the reduced move set.

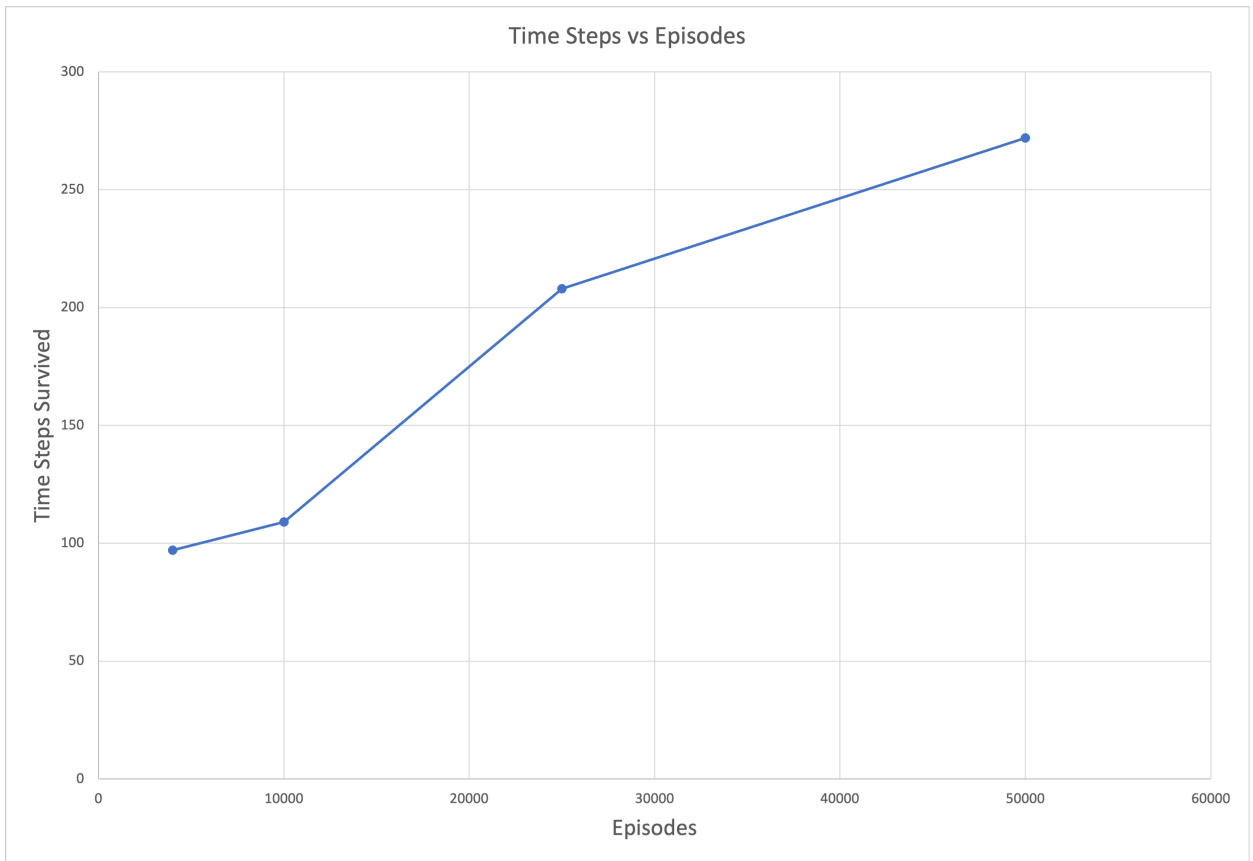


Figure 5. Agent survival in terms of time steps vs episodes trained.

The results over these two deliverables are quite promising. The deep Q-network is able to reach scores of 3000 and, even more impressively, it almost won the game on one instance – with just 6 pieces of food remaining. Further, the agent is observed to be playing to win the game. Due to limited compute power, there is a limit to the amount of episodes the neural network can be trained for. However, we observe the agent's performance is improving as the number of episodes increase and the game is certainly solvable with enough training.

VII. REQUIREMENTS

The requirements to complete the tasks outlined in the proposal are described below:

- The model for this project will be developed on the Python language and programming environment.
- The Open AI Gym Environment, “Frozen Lake”, will be implement the vacuum world environment for Deliverables 1 and 2.
- The initial training dataset will be composed of frames of images from playing Atari games using the ALE.
- The Open AI Gym Environment for Ms. PacMan will be used to train and test the network, across Deliverables 3 and 4.
- The Keras API within TensorFlow will be used to implement the deep neural networks.
- The software and technologies listed above are available for use, free of cost.

VIII. PROGRESSION TIMELINE

Week 1: Aug. 23 – Aug. 30	Finalize project proposal and find relevant research papers
Week 2: Aug. 30 – Sep. 6	Read [5] and find sample datasets for training
Week 3: Sep. 6 – Sep. 13	Read [5] and demo understanding on Q-learning
Week 4: Sep. 13 – Sep. 20	Deliverable #1: Q-learning
Week 5: Sep. 20 – Sep. 27	Research implementing Deliverable #1 on a neural network
Week 6: Sep. 27 – Oct. 4	Read [2]
Week 7: Oct. 4 – Oct. 11	Read [3] and demo progress on Deliverable #2
Week 8: Oct. 11 – Oct. 18	Deliverable #2: Neural network
Week 9: Oct. 18 – Oct. 25	Read [1] and study PacMan game mechanics
Week 10: Oct. 25 – Nov. 1	Complete reading [1] and demo understanding of reinforcement learning within game
Week 11: Nov. 1 – Nov. 8	Present understanding of DeepMind paper and neural net-based learning for video games
Week 12: Nov. 8 – Nov. 15	Read [11] and research enhancing Deliverable #2 to play a simple version of PacMan
Week 13: Nov. 15 – Nov. 22	Deliverable #3: Game-playing agent for PacMan
Week 14: Nov 22 – Nov. 29	Deliverable #4: PacMan with reinforcement learning
Week 15: Nov. 29 – Dec. 6	Deliverable #5: CS 297 Report

IX. CONCLUSION

The results of the deliverables over this semester have been promising. The Q-learning approach proved able to solve the vacuum world game described in this report. The neural network version of the vacuum world agent was able to predict shortest paths to reward squares across the grid. Aggregating the results from the first 2 deliverables and the research of [1], a deep Q-network was implemented to solve Ms. PacMan. The network is able to score 3700 points on Ms. PacMan on the hardest difficulty and is able to nearly win the first stage with only 6 pieces of food remaining. Due to limitations in compute power on the MacBook Air M1, we

are currently unable to train the model past 50000 episodes. However, observing the trend in increasing performance against a higher number of training episodes, it is safe to say the game (at least on the first map) is solvable.

The results also bode well for further research into varying difficulties for games. There is scope to experiment with varying frame rates across the neural networks being used to predict the optimal action to take. Beyond this, we want to experiment with some form of transfer learning to test models across different games and observe results. To evaluate the dynamic difficulty adjustment of the AI, a simplified version of PacMan could be implemented with the ghosts actually being controllable. In this kind of game, two neural networks could be pit against each other to evaluate how the AI gets better and worse over time in reaction to the player or the agent's actions.

REFERENCES

- [1] V. Mnih et al., “Playing atari with deep reinforcement learning,” 2013.
- [2] O. Alsing, “Mobile object detection using tensorflow lite and transfer learning,” 2018
- [3] G. Brockman et al., “Openai gym,” 2016.
- [4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” vol. 47, pp. 253–279, 2013.
- [5] "Part V, Machine Learning, Chapter 22 Reinforcement Learning" in Artificial Intelligence: A Modern Approach. S. Russell and P. Norvig. Fourth Edition, New Jersey: Pearson Education, Inc. 2021, pp. 789-821.
- [6] J. Bird, K. Colburn, L. Petzold, and P. Lubin, “Model Optimization for Deep Space Exploration via Simulators and Deep Learning,” 2020.
- [7] "Part V, Machine Learning, Chapter 22 Reinforcement Learning" in Artificial Intelligence: A Modern Approach. S. Russell and P. Norvig. Fourth Edition, New Jersey: Pearson Education, Inc. 2021, pp. 789-821.
- [8] "Mobile Object Detection using Tensorflow Lite and Transfer Learning.". Alsing, Oscar. 2018
- [9] "Deep Learning for Real-Time Atari Game Play using Offline Monte-Carlo Tree Search Planning." Guo, Xiaoxiao, Satinder Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Advances in Neural Information Processing Systems 27. 2014
- [11] "Transfer Learning for Related Reinforcement Learning Tasks Via Image-to-Image Translation.". Gamrian, Shani and Yoav Goldberg. PMLR, .2019

- [12] "AlphaDDA: Game Artificial Intelligence with Dynamic Difficulty Adjustment using AlphaZero.". Fujita, Kazuhisa. arXiv Preprint arXiv:2111.06266.2021
- [13] "Dynamic Difficulty Adjustment through an Adaptive AI. Silva, Mirna Paula, Victor do Nascimento Silva, and Luiz Chaimowicz.". IEEE. 2015.