MarioKart Bot: A Gamplaying Agent

Aditya Gupta, Harsh Waghela, Jahnavi Vikrama, Vivek Tiwari
University of Southern California

Abstract

The goal of this paper is to summarize our findings observed while training a game playing agent for Super Mario Kart 64 using different Deep Learning techniques. We initially use a search algorithm to capture screenshots of different possible actions and their associated steering angles to construct our dataset. Additionally, we augmented the dataset with random sample states to improve recoverability form error states and avoid the distribution mismatch problem. A Convolutional Neural Network inspired by the Nvidia's Self Driving model architecture is then trained on the dataset as a baseline. We further analyze techniques of tweaking the Convolutional Neural Network, possible use of a Reinforcement Layer to support in game powerups and their optimal use for completing the race.

I. Introduction

Super Mario Kart is a racing game with different characters competing with go-karts on different racetracks. Interesting fact about the game is that it is not just plain and boring racing. Throughout the race the character has access to certain powers that can be used to enhance their own abilities or deter the other racers from winning. Additionally, Mario Kart has some unique challenges such as navigating hazards and jumps.

Our goal is to train a bot to successfully navigate the track by predicting a steering angle by observing the environment from a screenshot. As Mario Kart races can be easily completed without braking, we constrain the bot to constantly hold the accelerator. With this constraint, the bot simply needs to return a steering value for a given screenshot of the game.

Different tracks can have different terrain textures, environments and features (jump platforms, molten lava, etc.). Rather than using conventional techniques we rely on feature extraction capabilities of Convolutional Neural Networks to learn and generalize different tracks and driving scenarios.

Finally we focus on the in-game abilities like using power boosts, banana peels and coins to assist the agent in making decisions and completing the navigation with optimality. This primarily focuses detection of power boosts and identifying scenarios for their valid usage.

II. Background / Related work

A. Super Mario Kart

Since its debut "Super Mario Kart" has been one of the most popular games that allows the user to compete in a race as characters from the Nintendo world. The latest release of the game *Mario Kart 8 Deluxe* selling just under 10 million copies, it is safe to say that this classic game is still much adored by gamers around the globe.

Super Mario Kart is a racing game with different characters competing with go-karts on different racetracks. What is interesting about the game is that it is not just plain and boring racing. Throughout the race the character has access to certain powers that can be used to enhance their own abilities or deter the other racers from winning.

The very first Mario Kart game "Super Mario Kart " was released in 1992 for the Super Nintendo platform, and became a favorite very fast. In fact, it was the fourth most successful game of all time, selling over 850 million copies worldwide. The game had a total of eight characters. You could choose between Mario, Luigi, Princess Peach, Toad, Kroopa Troopa, Yoshi, Bowser, or Donkey Kong. The game has a variety of different racing modes. In single player mode, you can do grand prix, or a time trial. And each of these modes have three difficulty levels: 50cc, 100cc, and if your skills are

sharp enough, 150cc. The abbreviation “cc " means cubic centimeters, referring to the size of the engine, and consequently how fast your go-kart is. The twenty tracks come straight from Super Mario world.

Fig 1: Super Mario Kart

The next game in the series was Mario Kart 64. Better graphics in this version allowed for a better 3D feel to the game, and two more modes were added: versus mode and battle mode. This is the game that is being used in our project since it is easy to run on an emulator and has most of the features that are there in the modern game. Mario Kart 64 contains many tracks and some of which are really difficult to navigate through and would be a good challenge for the AI agent. Compared to the previous version, Mario Kart 64 contains more trace, more playable characters and more special powers as well which add more things to learn for the AI agent.

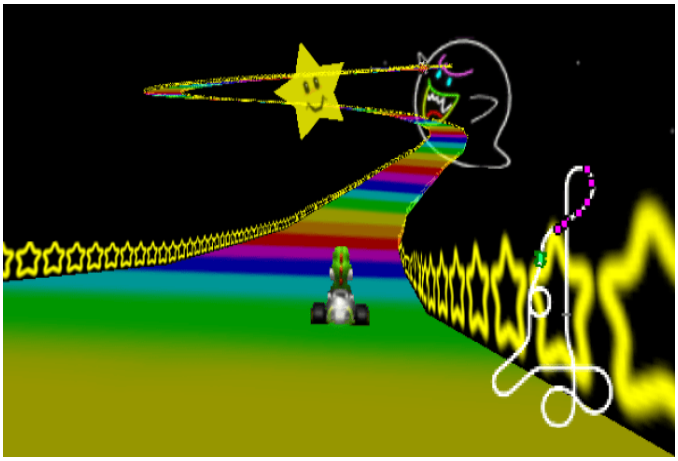

Fig 2: Iconic Rainbow Track in Mario Kart 64

Mario Kart 64 was a pioneering version for the game and after this, a lot more versions were released for different consoles and it went on to gain a lot more popularity and following. The latest game was released in 2017 and runs on Nintendo Switch. There are more than 32 tracks and the ability to compete with other players online.

B. Imitation Learning

Imitation learning is a popular technique for training real-time deep learning controllers. In imitation learning, an expert is recorded performing the task, and observations and resulting actions are recorded at each time-step. A neural network is then trained using these recordings as a dataset, thus learning to “imitate” the expert [1]. In practical terms, experts are often too good, and rarely find themselves in error states from which they must recover. Thus, the controller never learns to correct itself or recover, and small errors in prediction accumulate over time [2]. Ross, et. al introduce the DAGGER algorithm which resolves this issue [3]. After initially training a weak model from human input, they run the controller and sample observations from the resulting trajectories. Then, a human manually labels the sampled observations. The dataset is then augmented with the new human-labeled data.

C. Reinforcement Learning

Unlike imitation learning, reinforcement learning requires no human input at all. Instead, the AI repeatedly tries to execute runs, some of which will be more successful than others. The AI then modifies the network to make successful runs more likely [2]. Much of the deep reinforcement learning literature has been evaluated in the Arcade Learning Environment (ALE), which provides emulated versions of many Atari games [4]. The ALE also provides reward functions for the games, a critical requirement for deep learning. As opposed to imitation learning, which is a form of supervised learning, reinforcement learning is more

complicated and takes longer to converge. However, as the data is generated by the AI itself, Reinforcement Learning is less exposed to distribution mismatch problems (recovering from unseen error states) [2].

D. Autonomous Vehicles & NVIDIA CNN Architecture

Neural networks are a popular choice for training autonomous vehicles. Companies and universities, such as Google and Stanford, have pursued autonomous driving systems [5][6]. These systems often aggregate over a multitude of features to predict optimal trajectories.

In 2016, Researchers at NVIDIA designed a modern end-to-end system for training self-driving cars using CNNs [7]. The CNN is used to map the raw pixels from a front-facing camera to the steering commands for a self-driving car [8]. With minimum training data from humans, the system can learn to steer, with or without lane markings, on both local roads and highways in addition to unpaved roads and parking lots. The system is trained to automatically learn the internal representations of necessary processing steps, such as detecting useful road features, with only the human steering angle as the training signal [8].

NVIDIA's training setup [8] :

Three cameras were mounted behind the windshield of the data-acquisition car, and time stamped video from the cameras was captured simultaneously with the steering angle applied by the human driver (obtained by tapping into the vehicle's Controller Area Network (CAN) bus). Steering commands were obtained as 1/r, where r is the turning radius in meters as 1/r smoothly transitions through zero from left turns (negative values) to right turns (positive values).

Training with data from only the human driver was not sufficient. The network also needed to learn how to recover from any mistakes (distribution mismatch problem). The training data was therefore augmented with additional images that showed the car in different shifts from the center of the lane and rotations from the direction of the road.

Below is a block diagram Nvidia's training system:

- Images are fed into a CNN that then computes a proposed steering command.
- The proposed command is compared to the desired command for that image.
- The weights of the CNN are adjusted to bring the CNN output closer to the desired output.
- The weight adjustment is accomplished using back propagation

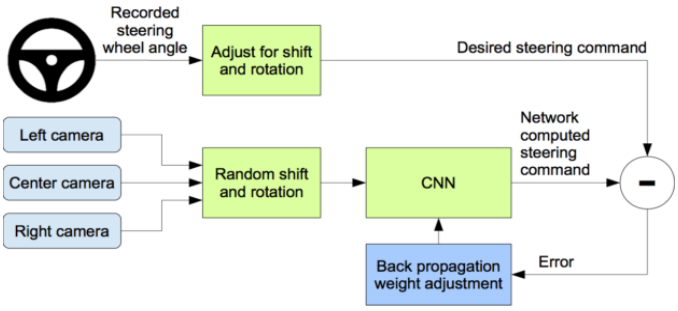

Fig 3: NVIDIA Model

Once trained, the network is able to generate steering commands from the video images of a single center camera.

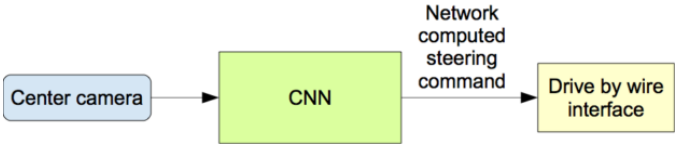

Fig 4: NVIDIA Model Steering

E. Mario Kart 64

CNNs have been applied to Mario Kart 64 before. TensorKart learns to map screenshots to human inputs and is also able to generalize training data over different track scenarios [9]. It uses the model developed by the NVIDIA autopilot paper. However, as a pure imitation learning system, it cannot recover well from error conditions. In

addition, it requires unnatural human play where turns must be performed gradually for TensorKart's CNN to learn properly and turns which fluctuate in steering angles confuse the AI [2]

F. MarIQ [12]:

MarIQ uses a combination of Recurrent Neural Network and Deep Q-Learning to enable the Mario Kart bot to successfully navigate the tracks over a BizHawk Emulator using a TCP server. The track is transformed into a 15x15 grayscale image that acts as an input to the 2-layer LSTM network with 125 and 75 neurons respectively. The network calculates a reward score for each of the 3 directions (left, right, forward) six times per second. The track is divided in multiple checkpoints and receives a +100 reward for each action that leads to the next checkpoint and -100 reward for moving away from the checkpoint. The proposed architecture generalizes well over unseen tracks as well. However, MarIQ suffers from distribution mismatch problem. This is problem is evident on particular tracks that have a jump spot located on it. If the Mario Kart bot misses the jump and gets stuck on a corner, it receives a -1000 reward for traveling back to make the jump. To avoid this negative penalty the algorithm continues causes the Mario Kart bot to remain stuck in the corner.

III. METHODOLOGY

1. BizHawk

We used the BizHawk emulator to play the Mario Kart races in an automated way. It provides an interface to run Lua scripts while playing Mario Kart. This helps us to save/load states, play for any number of frames, access in-game memory locations, and save screenshots. We can also find which buttons are pressed at a given time with a simple program.

2. Search AI

The first component of our model is a search-based AI. It determines the best steering action to take from a given game state. It runs using the BizHawk emulator to simulate different actions. While deciding the net search, this AI saves the current position as the root state. It then tries 11 different steering values from this root state and simulates the results of the gameplay for 30 frames. The angle resulting in the greatest reward is chosen by it. The reward is a weighted sum of the current progress and the current kart speed. Both of these values can be read using in-game memory addresses. Finally, the search AI proceeds, using the selected angle, for 30 frames, and repeats the process using the new state as the next root state.

Our input data to the model is basically the root state game screen and the corresponding angle chosen by the search AI. We have collected 20k training examples across four tracks, with a 10% randomly chosen validation split.

3. CNN

The second component of our model is a convolutional neural network. We used the NVIDIA's autopilot model which is used in the previous work called TensorKart. The actual model has 5 batch normalization-2D convolution-ReLU layers, followed by 5 dense layers that end in a regression. We made certain modifications to the model by incorporating several batch normalization layers, which we found helped with reducing overfitting and smoothing turns taken by CNN. We also used preprocessing techniques like resizing(200x66), data-augmentation and gaussian blur to smoothen the image and to reduce the noise within the image. We trained this model using the data generated by the Search AI and the loss function used is Euclidean loss. It is done in Keras. We used Adam optimizer while training. We trained the model on different tracks separately.

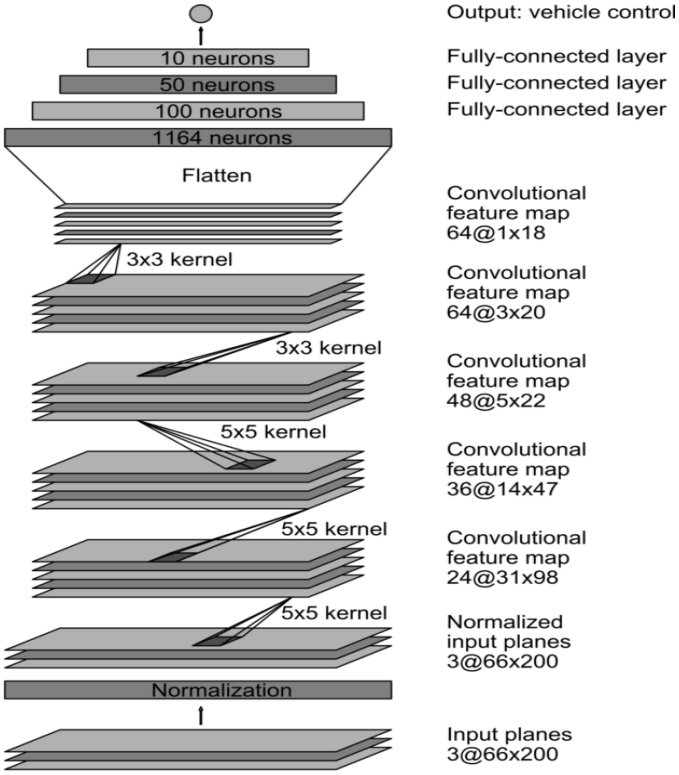

Fig 5: Model Architecture

4. Dataset Aggregation Algorithm

Training the CNN on the search AI alone can yield poor performances because we can never duplicate things exactly. Error accumulates fast in a trajectory and puts us into situations that we never deal with before i.e. the driver slowly might drift off the road and needs to be corrected.

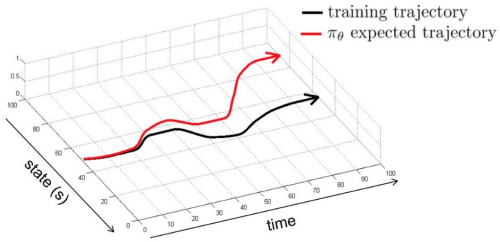

Fig 6: Training

A solution for this issue is the DAGGER algorithm. We first run the search AI by itself on the track; the resulting data is used to initialize the weights of the CNN. Then we use the CNN to play using its predicted steering angles. We then randomly pause the CNN and run the search AI from the current point. We run the search AI for 120 frames and save image-steering angle pairs; the resulting pairs are used to augment the dataset with which we retrain the CNN. Every time we train, we use the previous weights as an initialization. This algorithm helps the AI agent to recover from error states in less very less time.

The algorithm is as follows[]:

1. train $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ from human data $\mathcal{D} = \{\mathbf{o}_1, \mathbf{a}_1, \ldots, \mathbf{o}_N, \mathbf{a}_N\}$
2. run $\pi_\theta(\mathbf{a}_t|\mathbf{o}_t)$ to get dataset $\mathcal{D}_\pi = \{\mathbf{o}_1, \ldots, \mathbf{o}_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions $\mathbf{a}_t$
4. Aggregate: $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_\pi$

Fig 7: Algorithm

5. Playing in Real-time

We start a TCP Python server that loads the Keras model to play the game with the CNN AI. The server has a simple line-oriented protocol where clients can send requests for predictions and receive floating points in return. The Lua script running in BizHawk connects to the server using the Lua Socket library. It takes a screenshot, sends a request to the server, receives the prediction, and sets the joystick value. All of the networking is done asynchronously, meaning that the game doesn't halt while we wait for predictions. Although the CNN is deterministic, the random variations in network timing mean that the trajectory taken by the autopilot is different every single time.

6. Remapping the Input

N64 joysticks return signed bytes, which range from −128 to 127. Potential angles from this range are linearly interpolated. Unfortunately, most of this space is a dead zone, so many of the steering choices resulted in identical trajectories. Furthermore, the horizontal displacement of a turn is not linear with respect to the joystick value. This resulted in some trajectories that were too similar

and others that had noticeable gaps in between. This is undesirable, as the search AI should have a set of distinct trajectories that uniformly cover the track in front of the kart. To solve these issues, we used a mapping function J(s) that maps a "steer" input domain $s \in [-1, 1]$ to joystick values, such that gaps between trajectories are evenly spaced and there are no repeated trajectories. This function is from the NeuralKart paper[2].

$$\alpha(s) = (\text{sgn}(s) \times \text{sqrt}(0.24 \times |s| + 0.01) + 1)/2$$

$$J(s) = [-128(1 - \alpha(s)) + 127\alpha(s)]$$

During search, we apply J(s) before taking any action in the emulator; the value that we save in our recording is the value of s. We train on values of s, and predict values of s, which changes how our loss function responds to steering error. While playing the game, we calculate J(s) before we send predictions to the emulator.

7. Integrating In-game Powerups

We analyzed two techniques of incorporating the use of powerups for the bot.

i) CNN + Q-Learning

We used a Q-Learning network with a modified rewards function structure inspired from MarIQ in conjunction with the existing CNN architecture. The prime focus of the proposed model network is to maximize the use of power boosts to its advantage using reinforcement learning. CNN continues to be the driving factor for navigating the bot with reinforcement layer focusing on powerups and refining maneuverers. Hence once we have a boost available, the reinforcement layer output will be checked at each time step to see if we should take a particular action or use an available power boost. We assign positive rewards for boosts, coins and collecting power boxes and negative rewards for crashing into obstacles and going off track. Reward function can be summarized as below:

Reward	Scenario	Comments
+150	Rank gained	Make it aim for first place
-150	Rank lost	Prevent overtaking
+200	Gain rank using Reward	Maximize powerup usage
+50	Collect coins	Less priority reward
+100	Collect boxes and boosts	Priority rewards
-100	Landing on Banana peel	Causes Kart to spin/halt
-50	Going Off-track	

Table 1: Reward Structure

ii) CNN with Imitation Learning

In the second approach we aimed to leverage imitation learning and using more training data to allow the Mario Kart bot to build a better understanding of capturing and using power ups. We added more training data covering scenarios that show optimal use of powerups and also making alternate navigational decisions in order to capture more powerups. The bot was then trained using the original CNN architecture.

IV. RESULTS & ANALYSIS

A. Training:

Initially the performance of the network was evaluated on the non-augmented dataset and without using the DAGGER algorithm which yielded the below training graphs over 20 epochs.

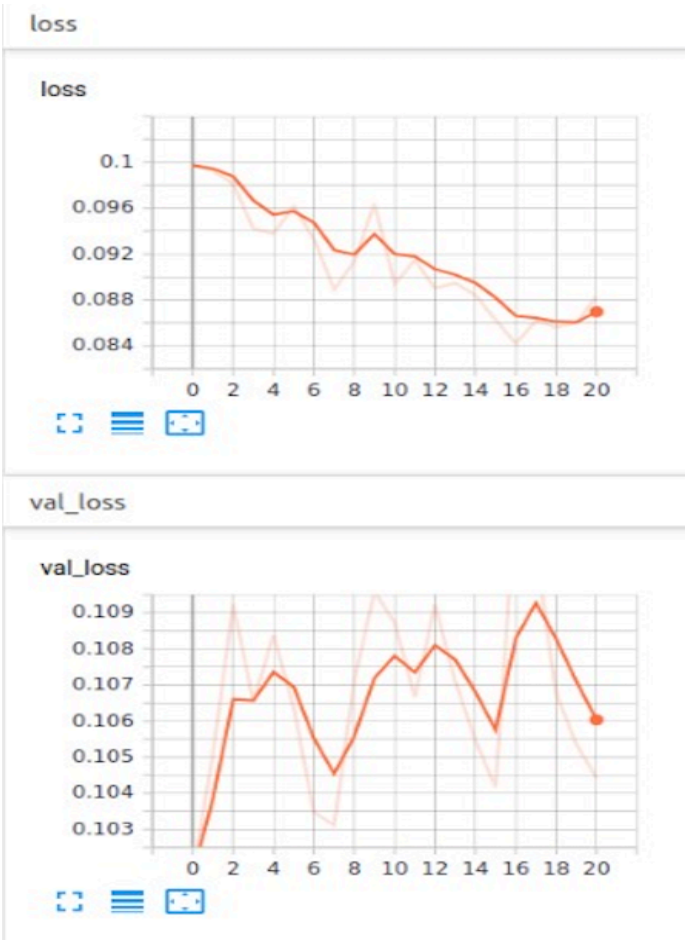

Fig 8: Training without DAGGER and augmented data

We observed that the bot learnt to navigate on the track and perform certain actions. On Luigi's Raceway, we found the bot to often slip off the track and get stuck against the walls of tunnels.

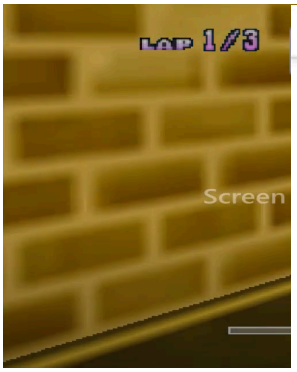

Fig 9: Bot stuck aga
and going off-trac

Later, we used the aug
the Neural Network
following training grap

Fig 10: Augmented dataset training

B. Time-based Analysis

We evaluated our bot based on its completion time in the single-player time trial mode. We ran 5 races on each track in real-time and calculated the mean race time reported by the in-game timer for the successful races. To simplify the testing and draw conclusive results, we limited our choice of tracks to those containing well defined walls around them to

prevent the bot from following into pits or lava. Our model ran on the following four different courses in Mario Kart 64: Luigi's Raceway, Moo Moo Farm, Choco Mountain, and Rainbow Road. The results were then evaluated against the performance of human drivers. Human completion time was the average of the time taken by each of our team members to complete the track. Players performed a test run to get familiarity of the track and limited themselves to actions that can be performed by the bot (no drift / powerups).

Fig 11: Bot on Moo Moo Farm/ Luigi's Raceway

Track	Bot time (s)	Human time (s)
Luigi's Raceway	133.8	126
Moo Moo Farm	102.3	95.1
Choco Mountain	149.3	129.5
Rainbow Road	399.5	365.60

Table 2: Track times for bot and human drivers

The bot performs slightly worse than human players, but still yields competitive performance. The bot performs best on Moo Moo Farm and Luigi's Raceway, both of which have gentle turns. In contrast, Choco Mountain and Rainbow Road have sharper turns, a thinner raceway, and closer walls; navigating these tracks without accidentally bumping into walls and losing speed is a challenge for the bot.

C. Power-ups usage Analysis

1. CNN + Reinforcement layer

Our rewards function structure powered Q-Network working in conjunction with the CNN did not yield qualitative results. Most of the time we observed that the bot would frequently miss picking up the boxes not justifying the need for an added reinforcement network. A second issue we observed was that the bot would start using the powerups almost immediately after gaining it. Our Reinforcement Layer did not seem very efficient in identifying the appropriate time for using powerups. We often saw the bot using power boosts on curved roads leading it to go off the track causing more difficult driving situations for the bot. Additionally, as the bot has limited viewing angles in the current

setup, it is difficult to identify the correct usage time for using banana peels.

We conclude that an integrated version of the CNN with the Reinforcement layer trained over 60 hours may be insufficent and would need a more comprehensive training data to improve performance. Additionally, we observed some integration and compatibility issues between the two networks often leading to conflicting actions by the bot. We found it difficult to apply an appropriate negative penalty on improper use of powerups. Moreover, Reinforcement layer was very expensive to train compared to imitation learning based CNN and it did not yield significant improvement to performance.

2. CNN + Imitation Learning

We found that the re-trained CNN with additional training data covering the use powerups was more effective and leaded to slightly improved performance on some tracks. Learning from human experts, the CNN learnt to effectively avoid use of power boosts on curved roads.It favoured collecting coins while driving. It was also able to avoid banana peels located ahead on the road. However, due to limited viewing angles, the bot could not effectively learn to use offence powerups against other opponents.

Track	Bot time (s)	Human time (s)
Luigi's Raceway	130.8	124.5
Moo Moo Farm	98.3	92.7
Choco Mountain	159.3	129.5
Rainbow Road	407.5	363.2

Table 3: Track times for bots using powerups.

V. CONCLUSION

Our results demonstrate that end to end neural systems can yield good performance as real-time controllers in games like Mario Kart 64. Imitation learning, which is easier to implement and converges faster than reinforcement learning, can be used an efficient technique to train an autonomous gameplaying agent in MarioKart. Imitation learning also proved to be efficient in teaching the agent the use of ingame power ups. Finally, DAGGER algorithm helped improve the recoverability of the bot from unseen error states.

VI. FUTURE WORK

The proposed CNN uses a screenshot at each time step to predict the steering angle. Modifying this approach to view a series of frames for predicting the angle may help improve the bot's overall performance. RNN's can also be considered as an alternative to the proposed architecture owing to their nature of using feedback. For example, the model may wish to swerve more harshly if obstacles are rapidly approaching, or turn more gently otherwise[2].

Secondly, we believe more training data can be used to optimize the performance of the bot to tackle unseen scenarios like roads that do not have well defined walls/boundaries, jumping ramps and tracks where the road and background have similar color and texture. Additional actions like drifting can also be incorporated.

Finally, a more robust Q-learning network with a well-defined reward structure could help improve the overall performance of the bot.

VII. REFERENCES

[1] S. Schaal. Is imitation learning the route to humanoid robots? Trends in cognitive sciences, 3(6):233–242, 1999.

[2] Harrison Ho, Varun Ramesh, Eduardo Torres Montano˜ NeuralKart: A Real-Time Mario Kart 64 AI

[3] S. Ross, G. J. Gordon, and D. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In AISTATS, volume 1, page 6, 2011.

[4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47:253–279, 06 2013.

[5] What we're driving at. https:// googleblog.blogspot.com/2010/10/ what-were-driving-at.html. Accessed: 2017-06- 12.

[6] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, M. Sokolsky, G. Stanek, D. Stavens, A. Teichman, M. Werling, and S. Thrun. Towards fully autonomous driving: systems and algorithms. In Intelligent Vehicles Symposium (IV), 2011 IEEE, 2011.

[7] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars. arXiv preprint arXiv: 1604.07316, 2016.

[8] Mariusz Bojarski, Ben Firner, Beat Flepp, Larry Jackel, Urs Muller, Karol Zieba and Davide Del Testa, https://devblogs.nvidia.com/deep-learning-self-driving-cars/

[9] Tensorkart: self-driving mariokart with tensorflow. http://kevinhughes.ca/blog/tensor-kart. Accessed: 2017-05-01.

[10].https://medium.com/@jonathan_hui/rl-imitation-learning-ac28116c02fc

[11] https://github.com/bzier/gym-mupen64plus

[12] MarIQ: Q-Learning Neural Network