The DDQN Mario Kart Network went through multiple major iterations in both network architecture and reward function, so I felt it right to highlight them here:

The first iteration of the Figure-8 Circuit network was a 2 dimensional DQN convolutional network that took a single frame and predicted the reward solely from that frame alone. As expected, this didn't work, but I used this mainly to test if everything else in my code was working, and if the network would actually converge to something, which it did (just turning the same direction the whole time). The reward was based solely on place, which is something I ended up ditching later on. When I knew everything else was working as intended, I started delving deeper into improving the network and the reward function.

The second iteration of the Figure-8 Circuit network was a Conv2D DDQN convolutional network with it taking a series of 4 frames as input as opposed to a single frame. Again, this didn't work, and at this point I was considering reading the memory of the game to get direction checking to improve the reward function, as it was solely based on place (8 - place), which wasn't giving the network enough data to train properly (I deleted all data when it was stuck in 8th place and didn't do anything, which was something I didn't do the first iteration as well). The first Dense after the Flatten was very small to save on resources (which is something I changed when transitioning into 3D Convolutional Network). As with last time, it converges to just turning one direction for the entire race without any signs of improvement between the beginning and when that happens.

At this point, I knew I would need to include something about detecting whether it was going the right direction or not. I narrowed down my options to reading the memory of the game and making a different convolutional network that would detect whether the bot was going in the right direction or not. I decided to read the memory of the game, as I thought it would be less expensive computationally and it was definitely the easier to implement option.

The third iteration of the Figure-8 Circuit Network was a Conv2D DDQN that took a series of 4 frames with reward based on both place and passing a checkpoint. Again, the big problem I was having with both of these rewards was that they were sparse, with checkpoint rewards happening only when you passed a checkpoint (positive for passing it the right way, negative for passing it the wrong way, thus being an indicator of if the bot is going in the wrong direction or not). Minor changes I made during this iteration were the different things the bot remembered after each race and experimenting with the batch size for resource saving but where it still can "learn". It still converged the same way it did with the last two iterations.

The fourth iteration of the Figure-8 Circuit Network was a Conv3D DDQN that took a series of 6 frames with reward based on place, checkpoint, and speed (rewards taken individually summed to one cumulative reward). I changed the amount of frames to 6 because I discovered that with Conv3D Network I could also use MaxPool3D, which made me able to increase batch size, frame amount, and make the Dense after the Flatten bigger than it was initially (from 4 to 16) because of the free resources I now had with using MaxPool3D. In the iteration I changed the reward function by excluding certain bases for the reward (I had scratch files in my Pycharm project for discluding direction, discluding speed, and discluding place). Discluding place seemed to work the best, which made sense as place was the most confusing reward. However, it still eventually converged to turning one direction the entire time, although there were some signs of slight improvement in between.

The fifth and final iteration (so far) of the Figure-8 Circuit Network is a Conv3D DDQN that took a series of 5 frames and rewarded based on speed and direction. This reward was different from the previous reward function based on speed and direction in that the direction award was dynamic (based on angle relative to the angle that you would pass the checkpoint if you were going straight through it, look at Lua code if you want a better explanation. This is going from the forward direction, and there isn't a big angle change between any two checkpoints, as the checkpoint frequency increases the more curved the road is). Additionally, the direction and speed rewards were combined to where direction would determine the sign of the reward and speed would determine the magnitude of the reward. This worked wonders compared to all the other iterations, producing the scores in the documentation and the training videos in the training video folder. The rest of the changes I made to the code were for efficiency and ability to save weights midway through training, but not to the actual network and reward function.