How CNN structures influence learning in Flappy Bird

Kelsye Anderson

ABSTRACT

For an agent to be able to learn how to play a video game with nothing but the screen as input, it must be able to correctly identify the important features in the image. Our agent does this through using a Convolutional Neural Network. Once it can identify features in the screen, it can run through thousands of iterations to learn how to react to those features to create the best outcome. There is a question of how we should build our Convolutional Neural Network in a way that allows our agent to correctly identify the features it needs in the image without driving our computational time too high. In this paper we'll explore how changing the structure of our Convolutional Neural Network affects the learning rate of our agent when playing the game Flappy Bird. As we train our different Convolutional Neural Networks on different difficulty levels in Flappy Bird, we'll be able to observe how effectively our agent is recognizing the game's environment and prove that if we don't provide our neural network with enough parameters, it's ability to learn is reduced.

1 INTRODUCTION

A couple of months ago, I was introduced to an agent that learned how to play the atari game Pong. I was really interested in the network that allowed the computer to interpret the game so it could then learn how to win. Because of this, for this paper I wanted to explore those convolutional neural networks and how changing the architecture of them would affect our agent's ability to learn.

2 METHODS

To test how our network architecture affects the agent, we needed an environment and several different convolutional network structures to run it against.

2.1 The Environment

Flappy Bird is a relatively simple game to play. You start with your bird in the middle of the screen. You can either tap the screen to make the bird flap up or do nothing and let the bird fall down. Throughout the game, pipes come towards you and you need to navigate the bird to fly between the holes in the pipes to win points. You get a point for each pipe you successfully make it through.

I chose this game for testing our convolutional network architectures because of how simple the action space was and how heavily it relied on correctly identifying objects in the image. As we run our agents through different difficulty levels of this game, we'll be able to see if our agent actually learned the rules of the game or if it just learned that it had the best probability of earning points if it kept the bird in the middle of the screen.

2.2 The Base Code

After doing a little bit of research online, I was able to find a program that uses a convolutional neural network to train an agent to play the game Flappy Bird. In it, Pičuljan had removed the background and replaced it with a black canvas to help our agent be able to focus on the important features in the image (See Figure 1). From this, I was able to modify the code to begin testing our different convolutional networks.

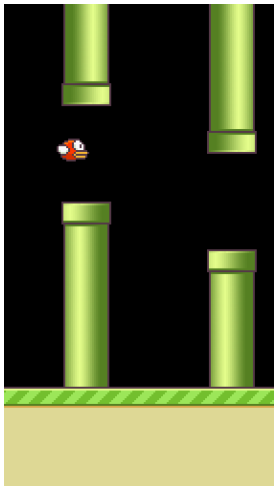

Figure 1: Input Image

2.3 Changing the Environment

The first thing I implemented, was modifying the given environment so we could have 3 levels: easy, normal, and hard. We can alternate between those levels by changing the difficulty level in our environment (easy: -1, normal: 0, hard: 1). Our easy level has twice as much time between pipes to react and less drastic changes in pipe height. It also has a decreased acceleration on the flap which makes our Flappy bird have smaller movements upward. While this would make a human's hand tired faster, it makes the game much easier for our agent. Our normal mode keeps the default settings of the game and can be used as our baseline once we start testing different neural networks. Finally, our difficult level had more drastic height differences between the pipes and the longer the game goes on the faster those pipes come at you. We also increase the player's gravity and decrease the upwards movement as the game progresses, so the agent has more control over our Flappy bird as the environment becomes more difficult to navigate.

Because our agent could theoretically learn how to play forever and never lose, we added a check in our environment that stops our agent from going through more than 20 pipes. If it makes it to the 20th pipe, it receives a reward of 20 points and immediately ends the episode.

2.4 First Neural Network

Now that our different difficulty levels were working, we could start to train our agents. The first Neural network we used had 3 Convolutional layers and 2 Linear Layers (see Figure 6). In total we had 1,685,154 parameters.

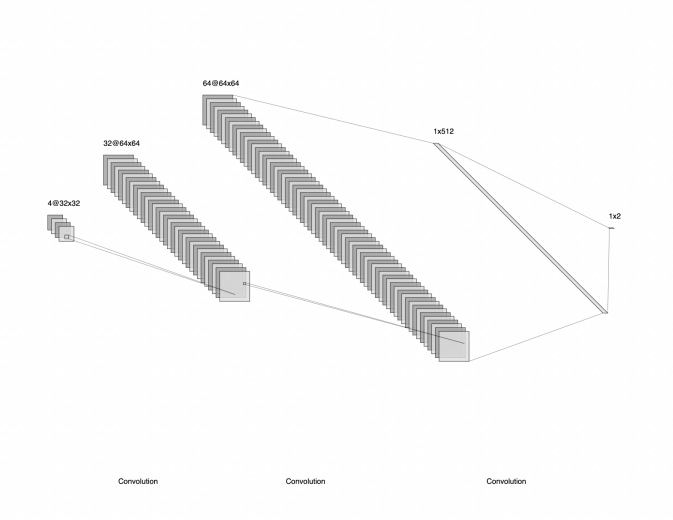

Figure 2: 1st CNN Architecture

Due to the large number of parameters, it took about 72 hours to fully train our our first agent on the normal difficulty level. We can observe it's learning rate by looking at the following reward curve (see Figure 3).

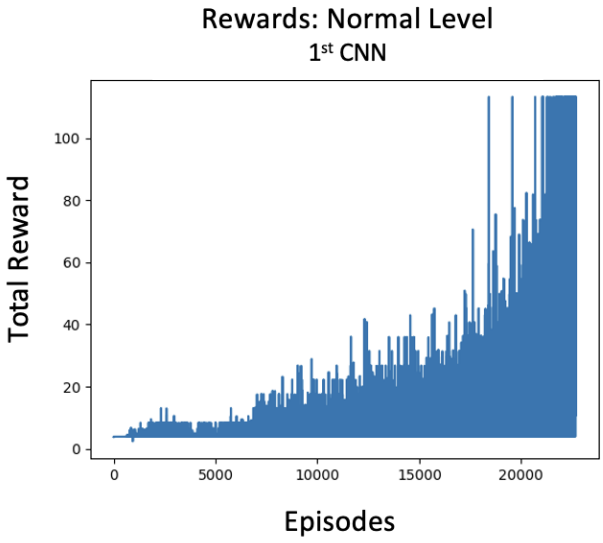

Figure 3: Normal Training Results

Overall, we can see it took about 21,000 episodes for our agent to be able to start consistently winning our game. We can observe a similar pattern when training our agent on the easy level (see Figure 4). Note that the decreased level of difficulty allows this model to be start winning consistently much sooner (around episode 600).

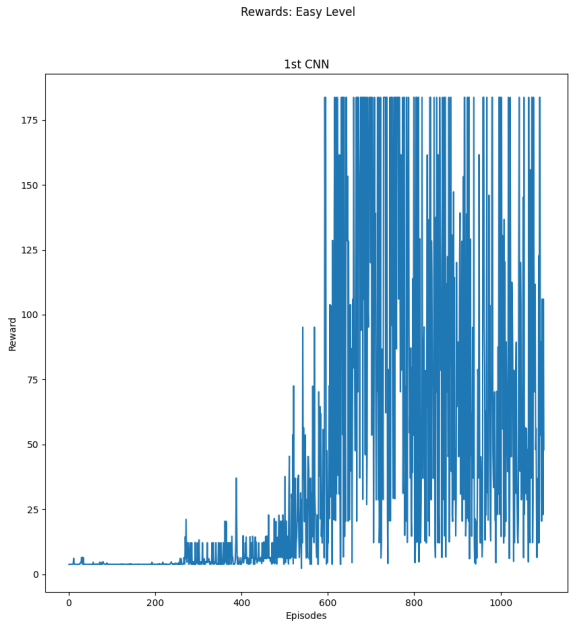

Figure 4: Easy Training Results

2.5 Second Neural Network

Now that we've got some baseline results, we can try shrinking our convolutional network to see if we can get the same or better results with a smaller neural network. The only elements we really care about identifying in our image is our bird and the pipes we need to fly through, so we don't need to worry about having too many features identified in our network. To try and give our network less work, we can crop the section of the image that we are paying attention to (see Figure 5) so it's only focused on the future moves.

Figure 5: Cropped Gray Scale Image

We can also shrink the number of features we are looking for by half in our new network to see if we really needed all those filters in our previous run. After making all these changes, our new

convolutional network has only 60,738 parameters. This cuts our network down to almost a third of the size of our original network (see Figure 7).

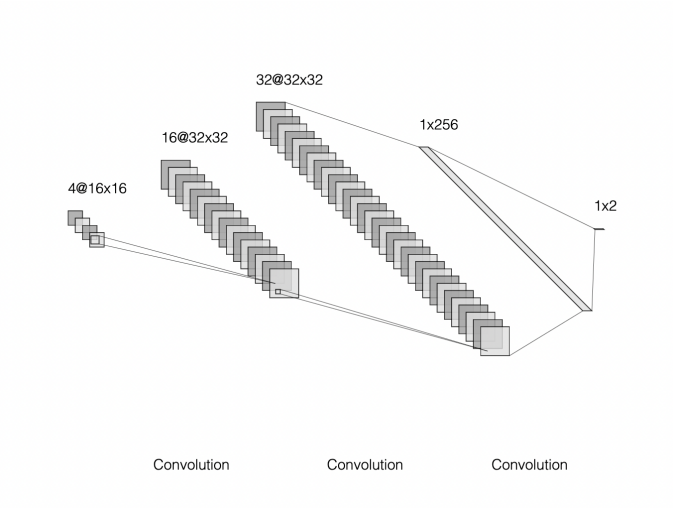

Figure 6: 1st CNN Architecture

After running this next agent for only a day, I was able to notice a significant difference in run time between our first neural network and our second one. While the first CNN took about 3 days to fully train our agent on the normal level, this new one was able to learn in only a day. Despite a significant decrease in CPU time, our new agent appeared to learn far slower then our previous model, only hitting our max reward after close to 60,000 episodes (see Figure 7).

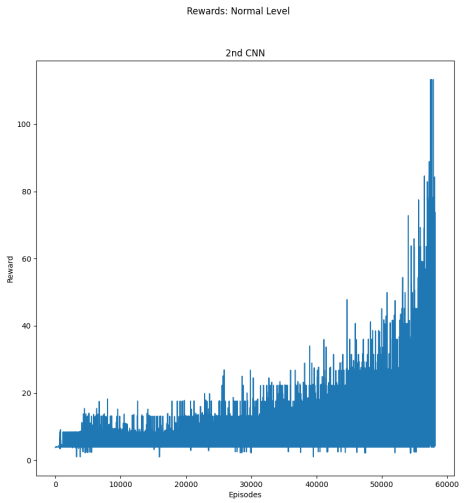

Figure 7: Normal Training Results

After training our network on the easy level, we see a similar pattern. Despite being able to train our agent in almost half the run time of our first CNN, it took our agent more than 2,000 episodes to begin consistently hitting our max score.

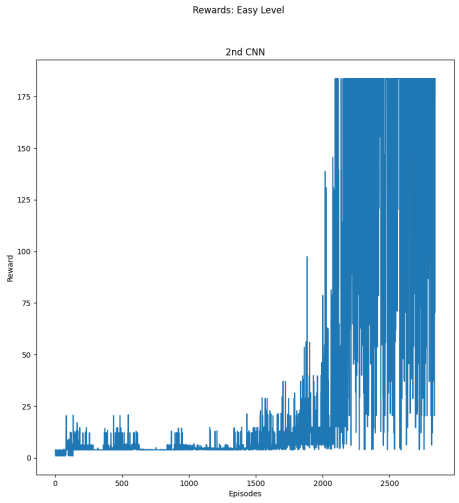

Figure 8: Easy Training Results

2.6 Third Neural Network

For this last neural network, I tried shrinking it down even further to see if it could still learn without one of the layers from our previous network. After removing a convolutional layer and reformatting the linear layers, we were left with a network with only 16,069 parameters. This is once again about a third of the size of our previous network and a little less then 1 percent of the size of our original network.

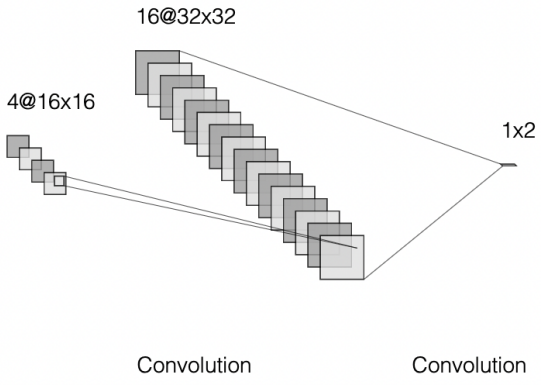

Figure 9: 3rd CNN Architecture

After attempting to train our agent on this network for 2 millions iterations, we can see that it was unable to learn the game at even the easiest level. (See Figure 10).

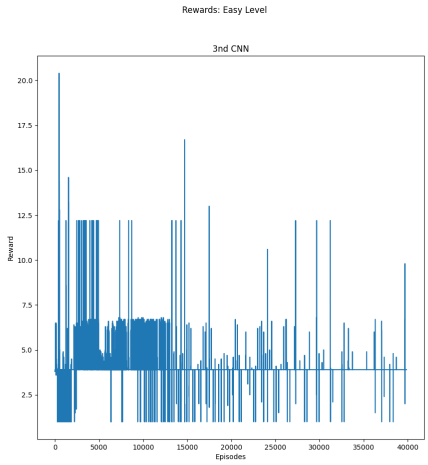

Figure 10: 3rd CNN Normal Training Rewards

This proves that even though the objects in our game are not that detailed, in order for our agent to be able to distinguish a pipe from an open space we need to have enough layers and parameters at each level for it to be able to learn effectively.

3 RESULTS

Now that we've trained all our models, we can begin testing them against more difficult levels to see how well it was able to learn the game. Considering our third CNN was unable to learn it's own level, I didn't include it in the results to avoid trivial charts. To test our other models, we can run each of the models against a harder level for 1000 episodes and create histograms of how well they did.

Starting with our easy models, we can see that none of them were able to perform very well at our normal level (see Figures 11 and 12).

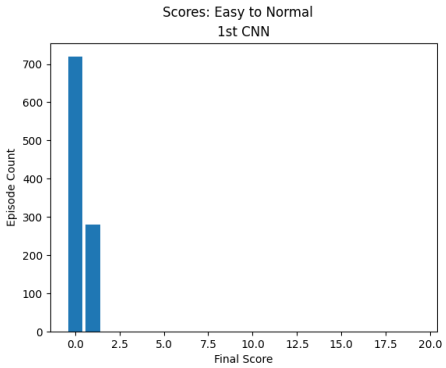

Figure 11: 1st CNN Easy to Normal Test Run

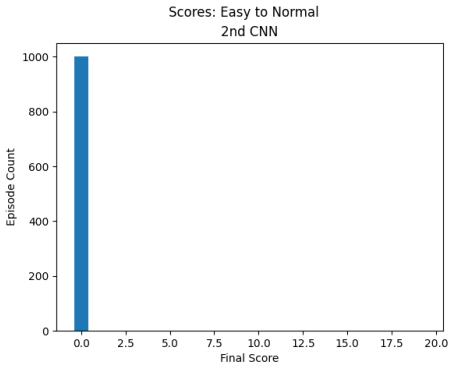

Figure 12: 2nd CNN Easy to Normal Test Run

The first CNN was able to get a score of 1 about 25% of the time, while our second CNN was unable to ever get past the first pipe. This tells us that even though our two models had similar reward curves when training our agents, our first seems to have been able to learn the game better and is more adaptable to changes in the environment. We are able to further observe a this pattern in our normal to difficult level test runs (see Figures 13 and 14).

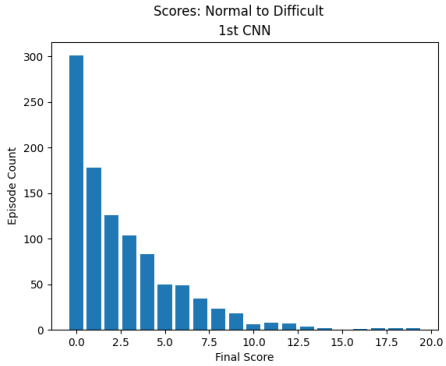

Figure 13: 1st CNN Normal to Difficult Test Run

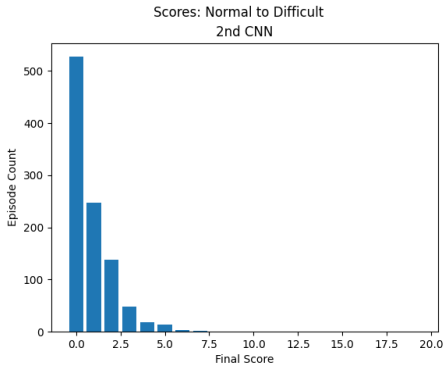

Figure 14: 2nd CNN Normal to Difficult Test Run

The difference is a lot easier to see in this test run. While it's clear that our first CNN model still failed to get through the first pipe the majority of the time, we can see a bell curve where it was able to get pretty far in the game many times. Looking at our second CNN model's histogram, it's obvious that it was not able to do as well, only making it past the first pipe a little less then half of the time.

4 CONCLUSIONS

Despite a significantly shorter training in CPU time as well as similar reward learning curves, our second convolutional network was unable to learn the game as effectively as our original network. This tells us that even though the shapes and images on the screen are fairly simple, in order for our agent to have all the information it needs to play the game consistently well, it needs a larger and more complex neural network then what we provided our second model with.

5 REFERENCES

[1] Piču ljan, N. (2019, January 27). Nevenp/dqn_flappy_bird. GitHub. Retrieved December 2, 2021, from https://github.com/nevenp/dqn_flappy_bird.

[2] Allibhai, E. (2018, November 15). Building a convolutional neural network (CNN) in Keras. Medium. Retrieved December 2, 2021, from https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5.