Playing Flappybird using Deep Q-Networks

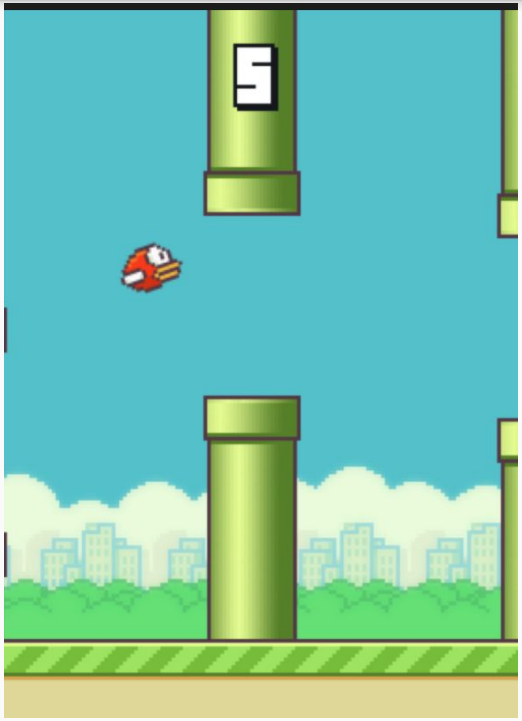
Masashi Omori

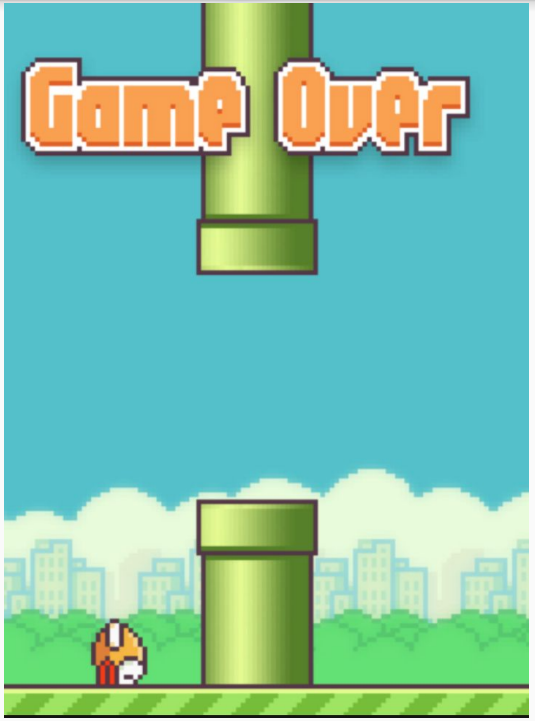
Goal:

To use reinforcement learning to play Flappybird

What is Flappybird?

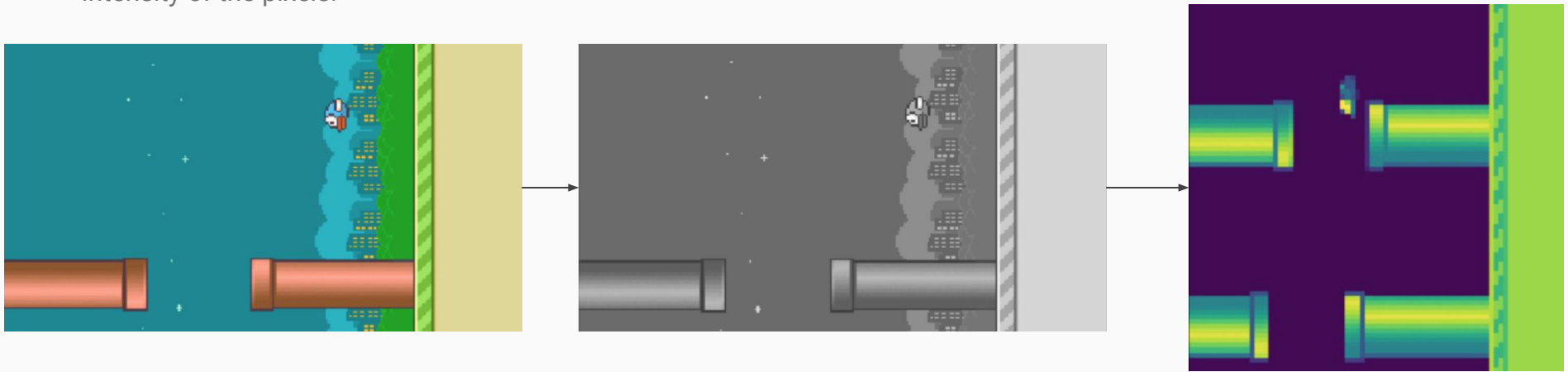
- Automatic side-scrolling game where the user controls a bird and maneuvers it through the moving environment.
- Random pipes are generated and must be maneuvered between them.
- By default the bird moves downwards, and user can make the bird ‘flap,’ causing it to move upwards.
- Game reaches a terminal state if the bird hits the ground, roof, or the pipe.





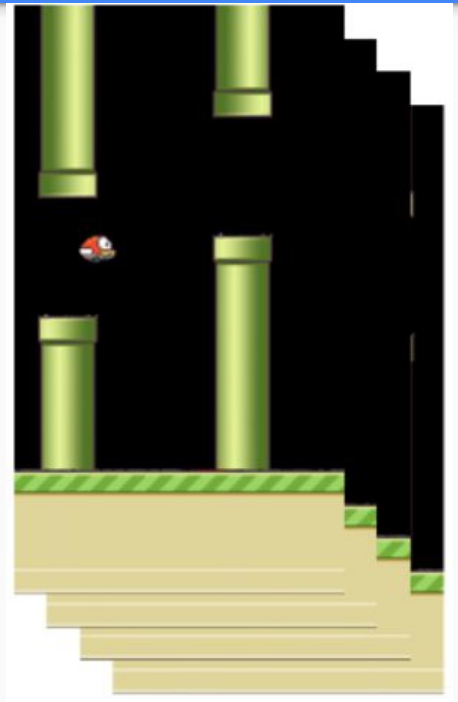
Data Wrangling

- Game state is represented in a (288, 512, 3) matrix, where the third dimension is the RGB colors.
- We reduce the dimensionality by converting it to grayscale, resizing the image and altering the color intensity of the pixels.
- This will allow the network to make computations quicker and hopefully stabilize learning.



Data Wrangling Cont.

- One thing to keep in mind at this point was how a model can tell if the bird is moving upwards or downwards. Given the current state, we needed a way to tell the velocity (or trajectory) of the bird, which is why we chose to squash four consecutive frames and define that as our ‘state.’ This squashed frame has dimensions 80 by 80 by 4.



Deep Q Networks

We use the q-learning function to create an optimal action selection policy for the agent. The q-learning function is defined as follows:

$$Q(s, a) = r + \gamma * max(Q(s', a'))$$

Where:

s - current state of the game

a - action taken

$Q(s, a)$ - total reward point for taking action a at state s.

r - current reward value

γ - discount factor to determine the importance of future rewards

s' - next state after taking action a in state s

a' - all possible actions in s'

Deep Q Networks Cont.

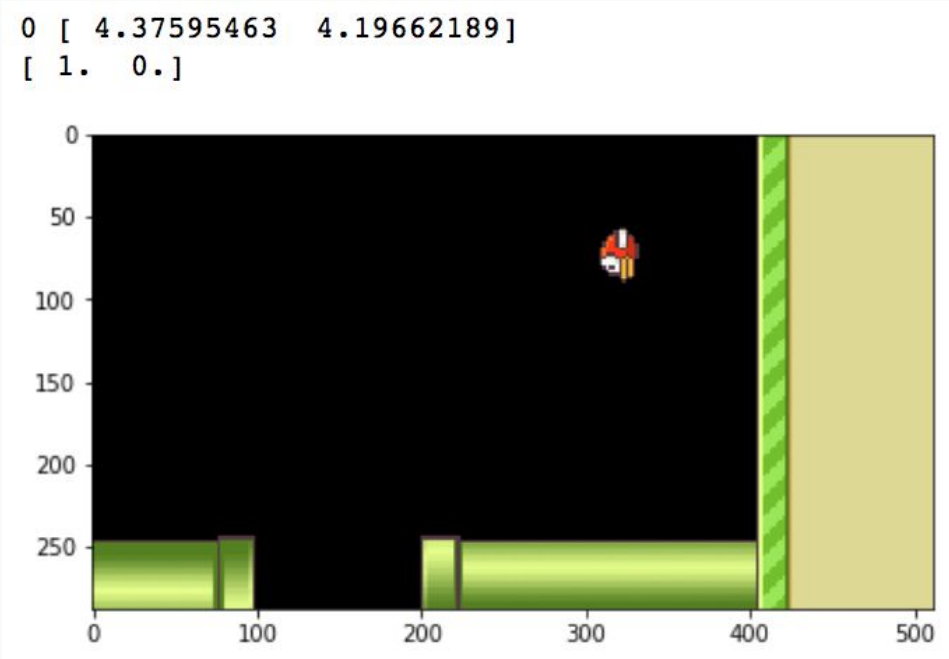
- We model the q-learning function from previous slide by representing it in a deep convolutional neural network, and find parameters (weights/biases) such that we maximize the q-value for each state in the game.
- Exploration vs Exploitation and epsilon annealing
  - Exploration is the concept of choosing a random action to see what new states the agent can be exposed to, where as exploitation is the act of letting the agent choose the action to take. If we rely 100% on exploitation, then the agent will never learn the rewards for certain uncharted scenarios, hence the ratio of exploration and exploitation is a key factor in reinforcement learning.
  - We utilize a method called *epsilon annealing* where we decrease epsilon gradually as the training time goes on. The epsilon annealing methodology recommends that we start off our epsilon at 1.0, which means we explore 100% of the time, not relying on agent prediction. Then as we observe more scenarios and train the agent, we decrease epsilon until a very small number (in our case, we used 0.0001, which means we explore 0.01% of the time). Intuitively, as the model becomes more trained, we trust its decisions more.
  -

Deep Q Networks Cont.

- Batch Training
  - Generally training a neural network contains the below steps:
    - Feed forward
      - Input a data sample and process it all the way until the output neuron
    - Backpropagation
      - Calculate the gradient of the error function with respect to the weights/biases
    - Stochastic gradient descent
      - Process of updating weights/biases based on the gradient found from backpropagation.
  - If it takes the above three steps per training sample, it can get very computationally expensive. Batch training differs in that stochastic gradient descent is not executed until all the samples in a batch have been passed forward. What this means is that it can drastically decrease the number of updates required to train the neural network and in general networks train faster with batches.

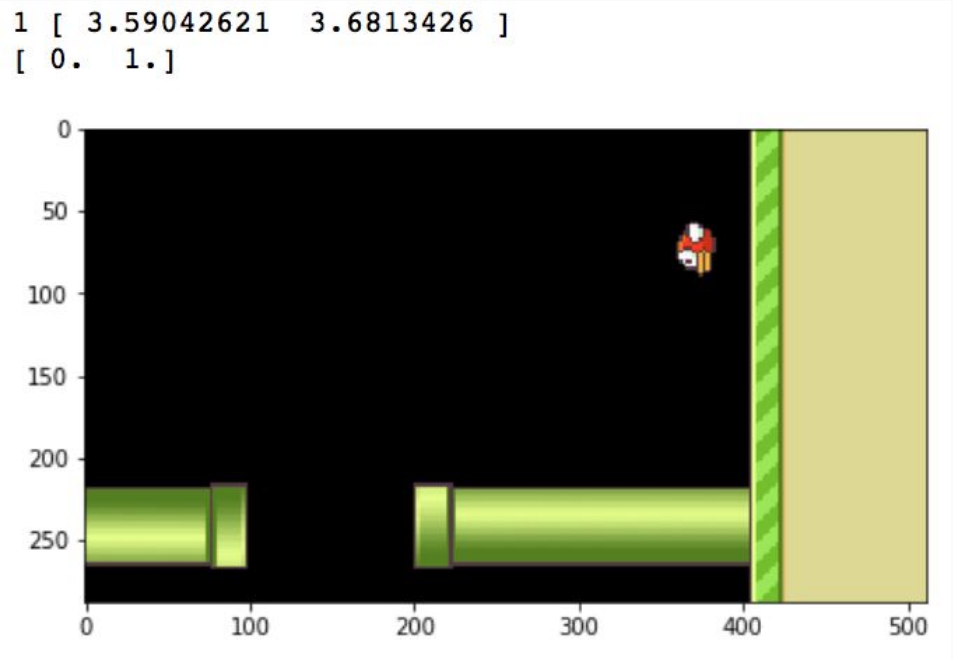
Results

- Using the methodology described in previous sections, we explored 50k frames of the game and used batch training size of 32. In total, this took roughly 4 hours on my local laptop and 2 hours on AWS p2.xlarge instance. With this, we noticed very interesting and important results. The following slides talk about some of the features the agent learned to detect.
- The images will look like the right, with these information present:
  - Prediction output: 0 means do nothing, and 1 means to move up.
  - Reward array: expected reward value of taking a certain action
  - Action array: output of the agent([1,0] for no action or [0,1] for moving up.



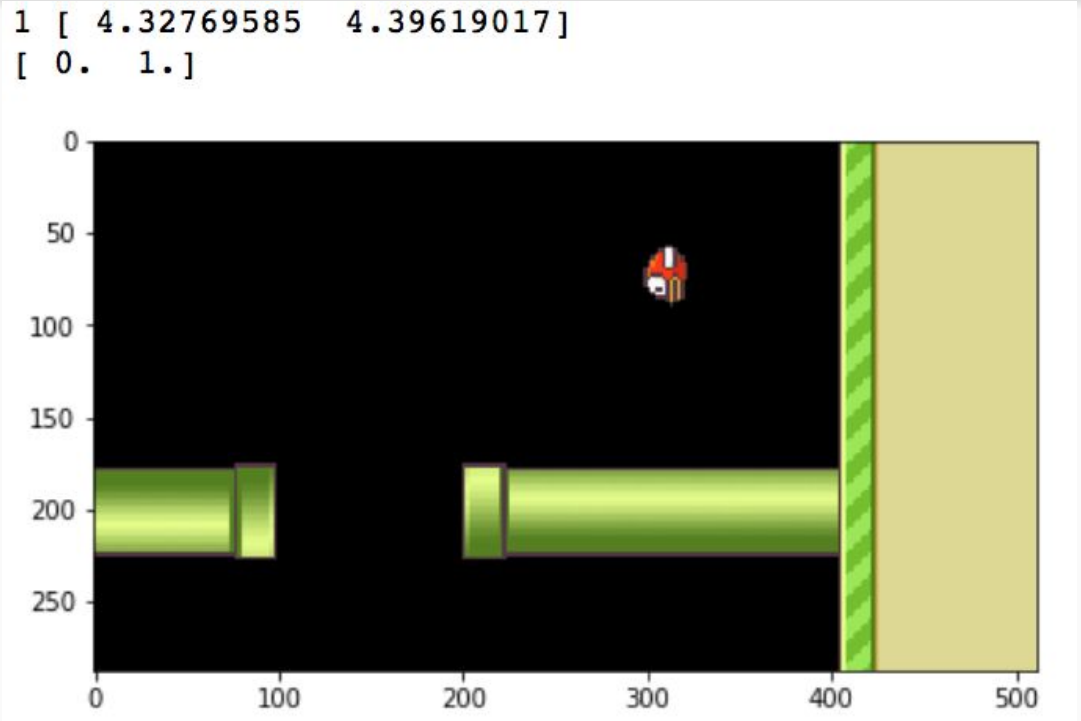
Results Cont

- In the initial stages of the game where the pipes are not close, it does nothing and keeps going down, but learns that hitting the ground results in a terminal state.
- Once it nears the ground, the value in the second index exceeds the first index, and makes the bird move up.



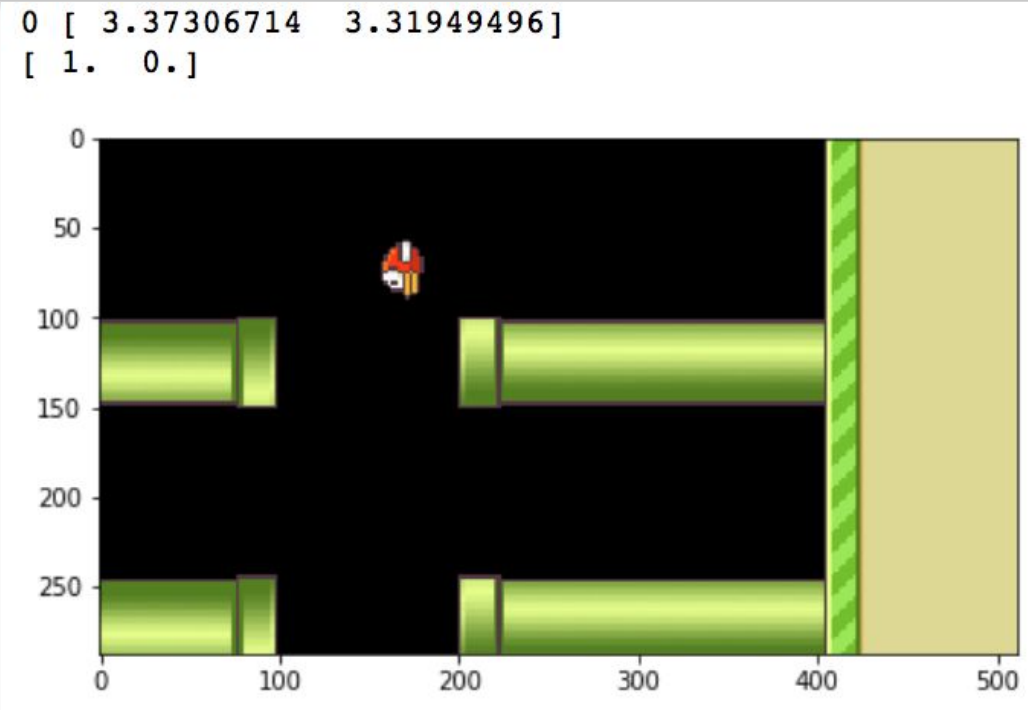
Results Cont

- Secondly, it also recognizes when it is nearing the pipe. This image shows that as it is reaching the pipe and notices it needs to move up to avoid it, it does so.



Results Cont

- Once it reaches an altitude higher than the pipe, it stops moving upwards and attempts to go through the section between the upper and lower pipe.
- The agent trained through this project is able to get through the middle of the first set of pipes, but not much further than that.



Results Cont. and Future Work

- As a comparison, the original article stated that its agent reached superhuman score with roughly one million frames of exploration (on a Titan X GPU), which is 20 times more data than what this project used. By a rough estimate, this would have taken 80 hours of training time, or 40 hours on AWS, which would be very costly. Due to lack of resources, we cut the training short here after achieving some very interesting results depicted in previous slides.
- Future work can include finding out ways to achieve convergence of the network quicker (using prioritized experience replay, double DQN structure), playing around with different CNN parameters and different reward values.