Flappy Bird

Deep Learning & Image Recognition

June 2020

Daniela Matinho | Jonathan Huff | Zeyang (Roy) Xie

Agenda

- ✓ Flappy Bird game
- ✓ Reinforcement Learning
- ✓ Models: Q-learning, DQN, and YOLOv3 & DQN
- ✓ Demo of the game
- ✓ Future Work

Flappy Bird game

Side-scroller where the player controls a bird, attempting to fly between columns of pipes without hitting them

Main Goal of the Project

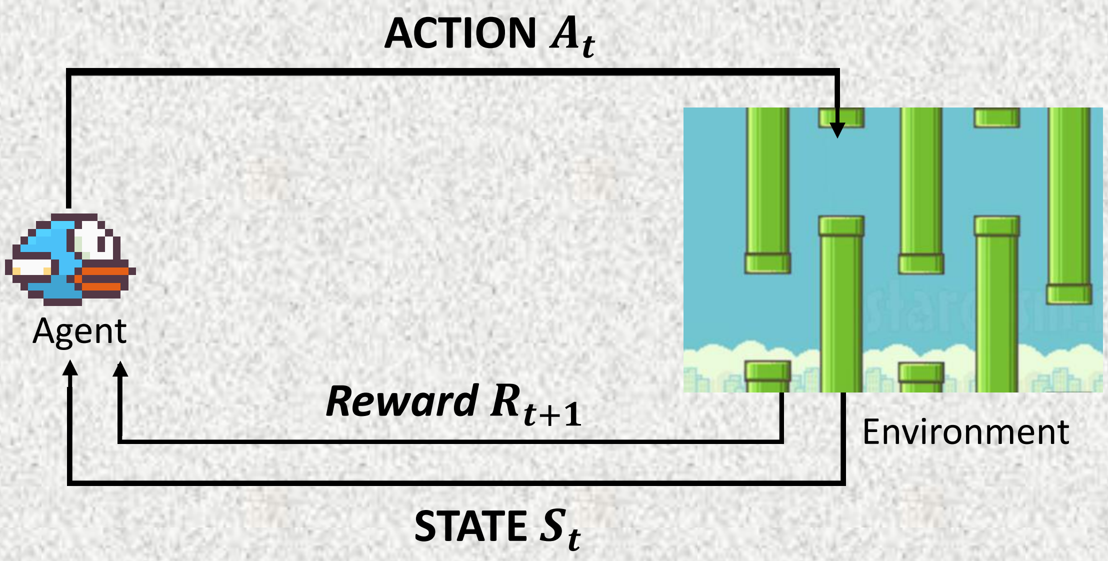
Teach the game how to play by itself using reinforcement learning techniques

Introduction to Reinforcement Learning (RL)

Machine Learning technique that enables an agent to learn in an interactive environment by trail and error using feedback from its own actions.

Input: Initial state vs **Output:** Maximize the reward
Training: Upon the input, the model returns a state and the user will get reward or punish
Model: Learn every iteration - decisions are dependent

The agent (bird) learns to achieve a goal (passing the pipe) in an uncertain environment. The bird gets either rewards or penalties for the actions the bird performs. The final goal is to maximize the number of pipes the bird passes through.



Reinforcement Learning using Q-learning

Q-Learning definition

Q = ‘quality’ of a given state and action pair in gaining some reward

Seeks to find the best action to take given the current state n order to maximize the total reward

Q-table [state, action] with initial values of zero

Initialize Q-table
↓
Choose an action
↓
Perform action
↓
Measure reward
↓
Update Q-table

Q-learning algorithm

State Space

- Vertical distance from lower pipe
- Horizontal distance from next pair of pipes
- Velocity

Actions

- Jump
- Do nothing

Rewards

- Game score

Reinforcement Learning using Q-learning


```
{"-40_-300_-10": [0, 0], "-40_-300_-9": [0, 0], "-40_-300_
0], "-40_-300_0": [0, 0], "-40_-300_1": [0, 0], "-40_-300_
"-40_-300_10": [0, 0], "-40_-290_-10": [0, 0], "-40_-290_
0], "-40_-290_-1": [0, 0], "-40_-290_0": [0, 0], "-40_-29
"-40_-290_9": [0, 0], "-40_-290_10": [0, 0], "-40_-280_-1
 "-40_-280_-2": [0, 0], "-40_-280_-1": [0, 0], "-40_-280_
"-40_-280_8": [0, 0], "-40_-280_9": [0, 0], "-40_-280_10"
"-40_-270_-3": [0, 0], "-40_-270_-2": [0, 0], "-40_-270_-
"-40_-270_7": [0, 0], "-40_-270_8": [0, 0], "-40_-270_9":
"-40_-260_-4": [0, 0], "-40_-260_-3": [0, 0], "-40_-260_-
"-40_-260_6": [0, 0], "-40_-260_7": [0, 0], "-40_-260_8":
"-40_-250_-5": [0, 0], "-40_-250_-4": [0, 0], "-40_-250_-
"-40_-250_5": [0, 0], "-40_-250_6": [0, 0], "-40_-250_7":
"-40_-240_-6": [0, 0], "-40_-240_-5": [0, 0], "-40_-240_-
"-40_-240_4": [0, 0], "-40_-240_5": [0, 0], "-40_-240_6":
"-40_-230_-7": [0, 0], "-40_-230_-6": [0, 0], "-40_-230_-
"-40_-230_3": [0, 0], "-40_-230_4": [0, 0], "-40_-230_5":
"-40_-220_-8": [0, 0], "-40_-220_-7": [0, 0], "-40_-220_-
"-40_-220_2": [0, 0], "-40_-220_3": [0, 0], "-40_-220_4":
"-40_-210_-9": [0, 0], "-40_-210_-8": [0, 0], "-40_-210_-
"-40_-210_1": [0, 0], "-40_-210_2": [0, 0], "-40_-210_3":
"-40_-200_-10": [0, 0], "-40_-200_-9": [0, 0], "-40_-200_
0], "-40_-200_0": [0, 0], "-40_-200_1": [0, 0], "-40_-200_
"-40_-200_10": [0, 0], "-40_-190_-10": [0, 0], "-40_-190_
0], "-40_-190_-1": [0, 0], "-40_-190_0": [0, 0], "-40_-19
"-40_-190_9": [0, 0], "-40_-190_10": [0, 0], "-40_-180_-1
 "-40_-180_-2": [0, 0], "-40_-180_-1": [0, 0], "-40_-180_
```


Updating the q-table

(1- Learning rate) * old value

+

Learning rate

*

(Current reward + Discount factor * Maximum expected future reward applied to current q-table)

Bellman equation

Definition of the Parameters

Learning Rate:
How quickly that agent abandons the previous Q-value

Discount factor:
A value less than 1, which makes the Q function converge.

Reward:
1 for passing a pipe
-1000 for crashing

Reinforcement Learning using Deep Q-Networks (DQN)

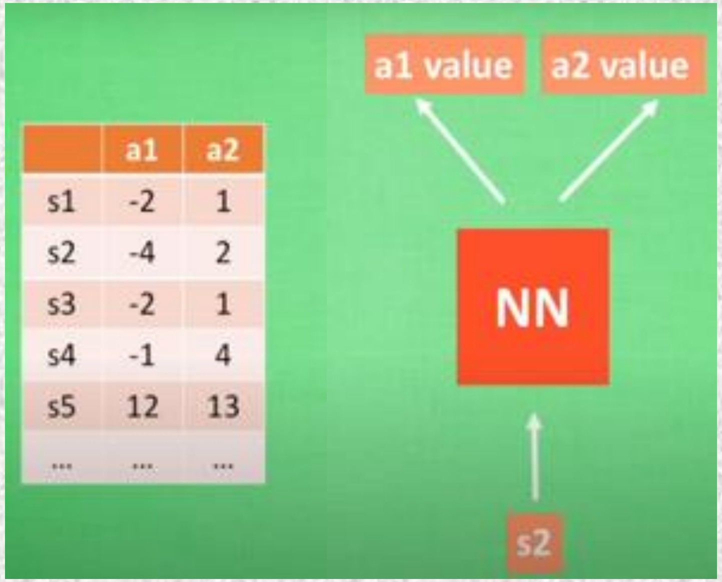
DQN definition

A reinforcement learning algorithm that combines Q-Learning with deep neural networks to let RL work for complex, high-dimensional environments.

DQN uses a neural network to approximate the Q-values function to generalize unseen states

DQN Implementation

- For DQN, it uses the state from Q-learning as inputs
- DQN uses the q-values from Q-learning as the targeted output
- In DQN, a Q-table is no longer needed, and it can be replaced by a fixed size queue
- Loss function: min(Q*-Q)^2



Data Pre-Processing for DQN

Pre-Processing

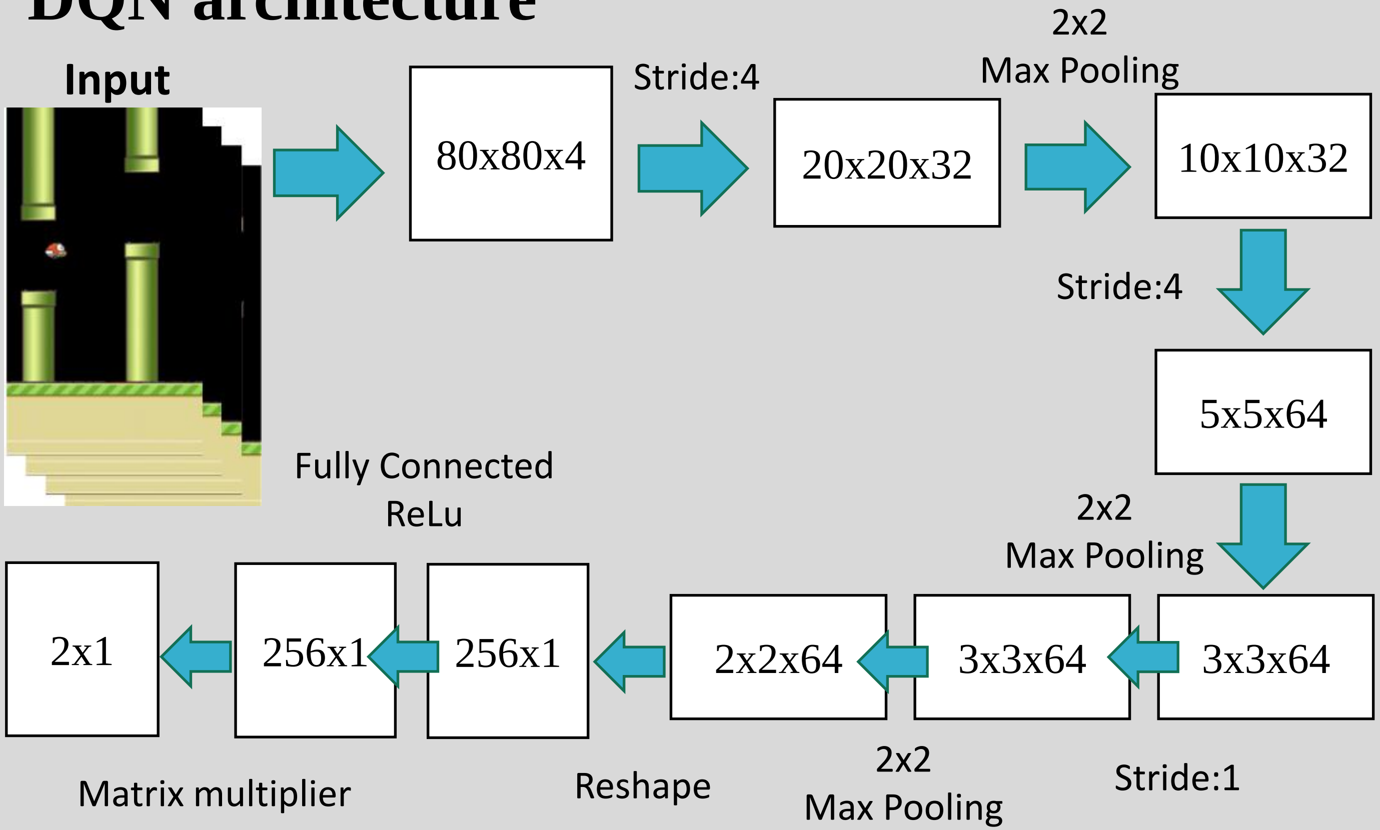
Input image (288x512) → 64x64

0-255 color → background removed replaced by black image

To process several frames → Current frame is overlapped with previous frames

Estimate Q-values from images

DQN architecture



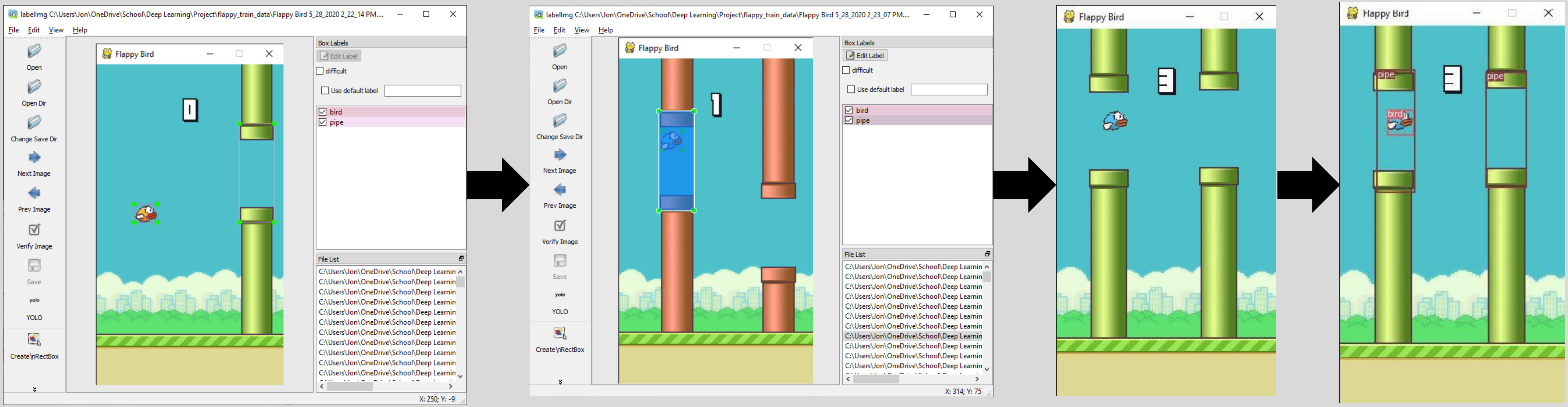
Source: Playing FlappyBird with Deep Reinforcement Learning

DQN with YOLOv3

Data Pre-Processing using LabelImg

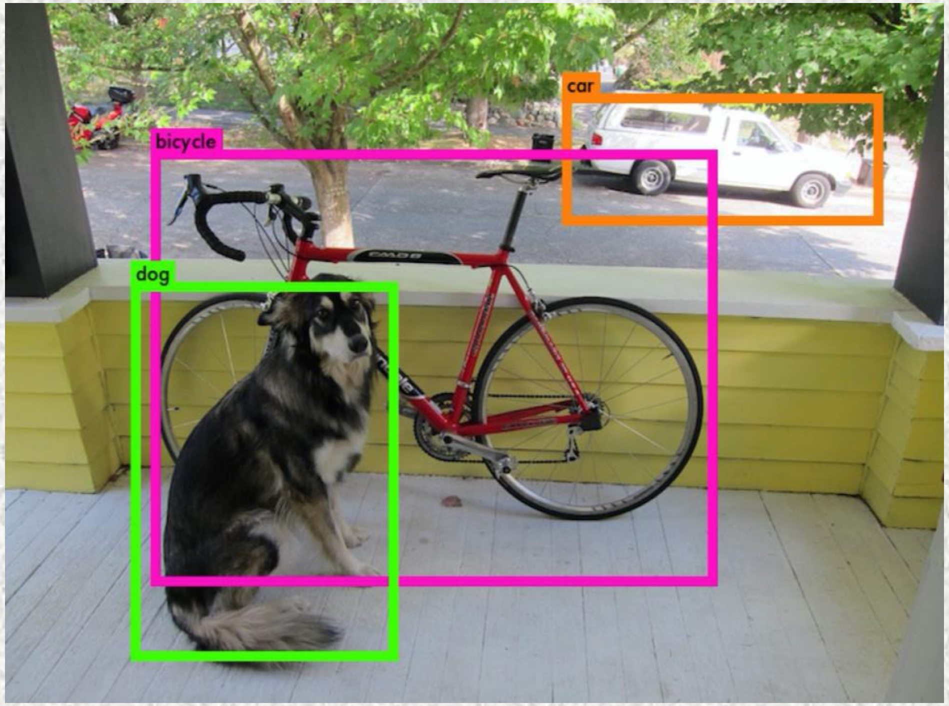
Input = Image size 288x512

200 screenshots of the game



DQN with YOLOv3

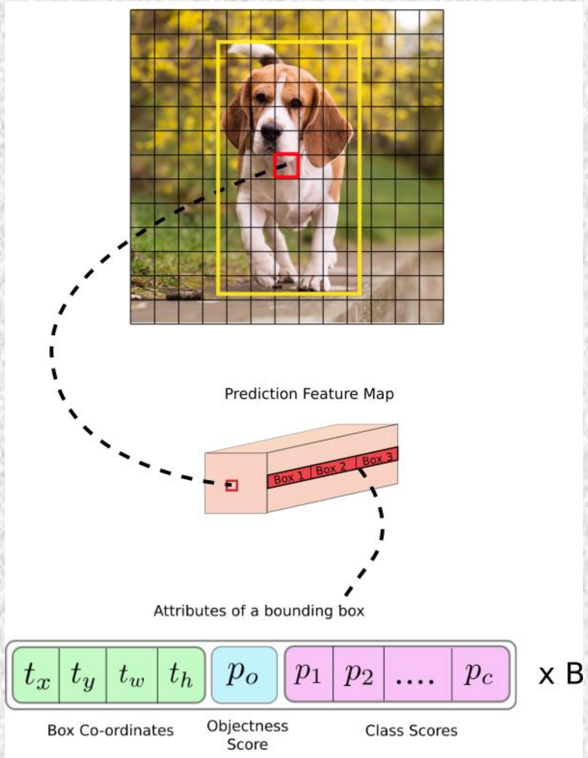
- YOLOv3 was used to detect the bird and pipe openings
- This algorithm was chosen due to its superior performance over faster R-CNN
- YOLO started as a fully convolutional network developed by Joseph Reddie, now utilizes an architecture called Darknet
- Transfer learning with Darknet was used, where the last layers were unfrozen and trained on the Flappy Bird screenshots and manually drawn bounding boxes

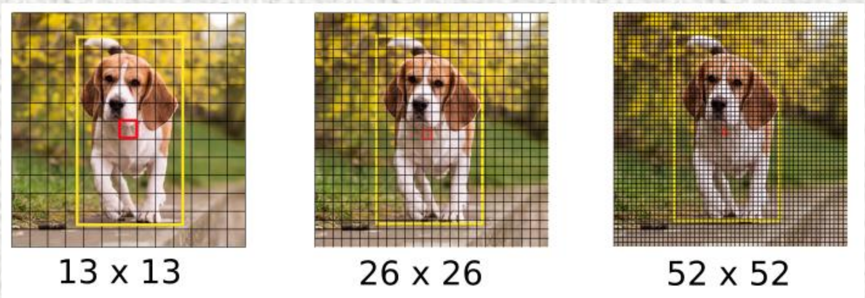


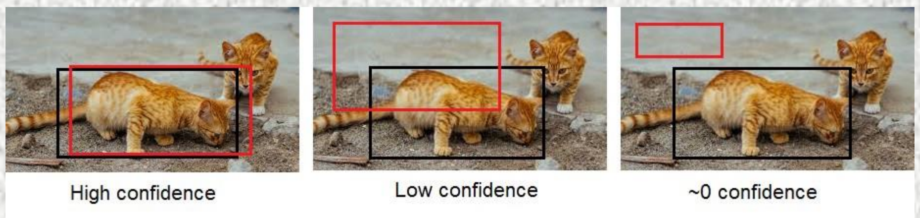
https://github.com/cfotache/pytorch_objectdetecttrack
https://github.com/cfotache/pytorch_custom_yolo_training

DQN with YOLOv3

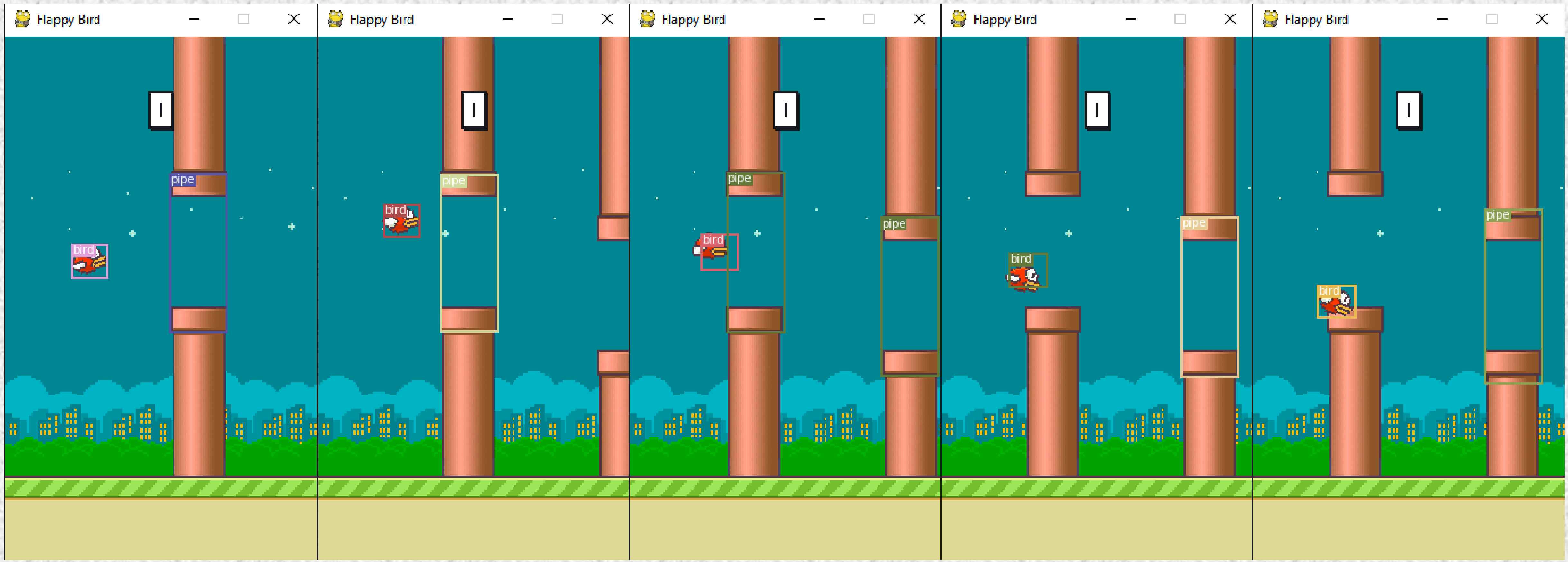
- Splits the input into a grid of cells
- Invariant to input image dimensions
- 53 convolutional layers, each containing batch normalization and leaky ReLU
- Instead of pooling layers, filters with stride 2 are used to downsample feature maps throughout propagation
- The cell in the image grid containing the ground truth is responsible for predicting the object which can predict multiple bounding boxes
- Across multiple grids, bounding boxes are drawn and objectness score threshold is applied where boxes with low scores are ignored
- Non-max suppression is then used to ignore multiple detections on the same object
- IoU: Intersection over Union
  - The degree to which the bounding box overlaps the ground truth; drives the confidence score of the object detection







DQN with YOLOv3



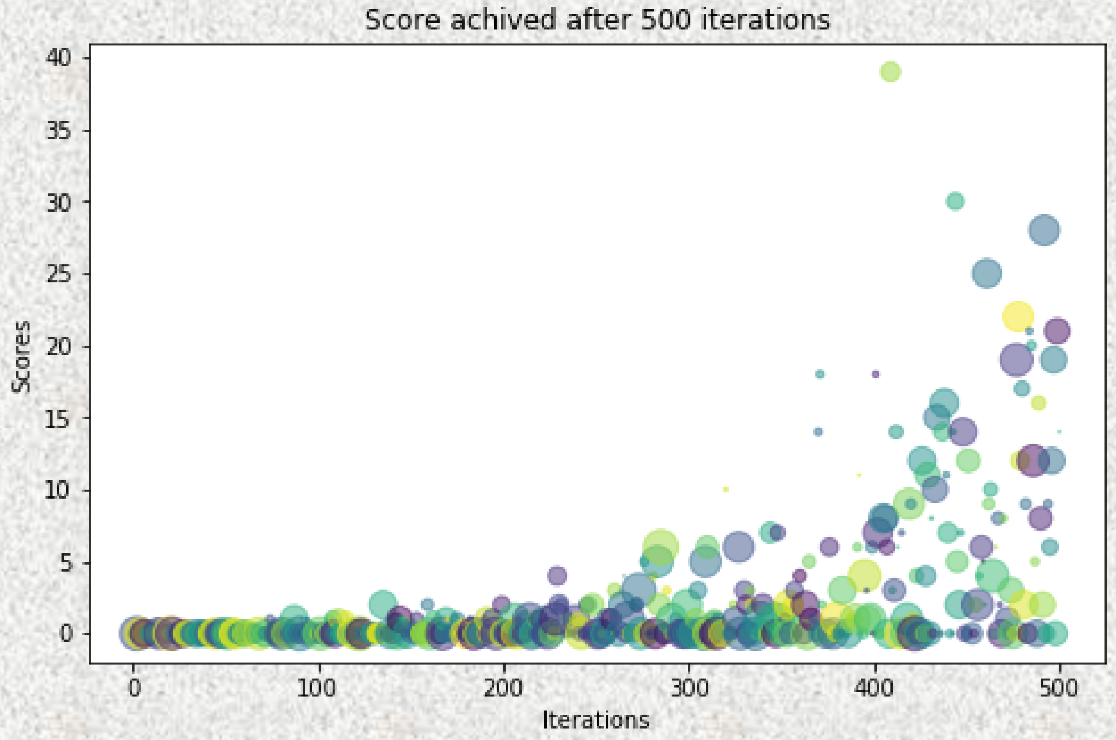
DQN with YOLOv3

- PyTorch was used for the entire implementation
- Within the Flappy Bird game, the user screen was extracted as a numpy array and fed into a call to the YOLOv3 model
- The YOLO model detected the bird and pipe openings, calculated bounding boxes, and returned a tuple of (x,y) coordinates (4 element vector) of the two objects. Non-maximum suppression parameter was tuned to determine optimal detection across multiple frame samples
- Five consecutive frames were stacked together to give context to the bird's movement which was then fed into a 20-neuron multilayer perceptron
- Due to the fact that EACH training iteration required calling the object detection framework, the training process was painfully slow, and therefore impractical based on this implementation.

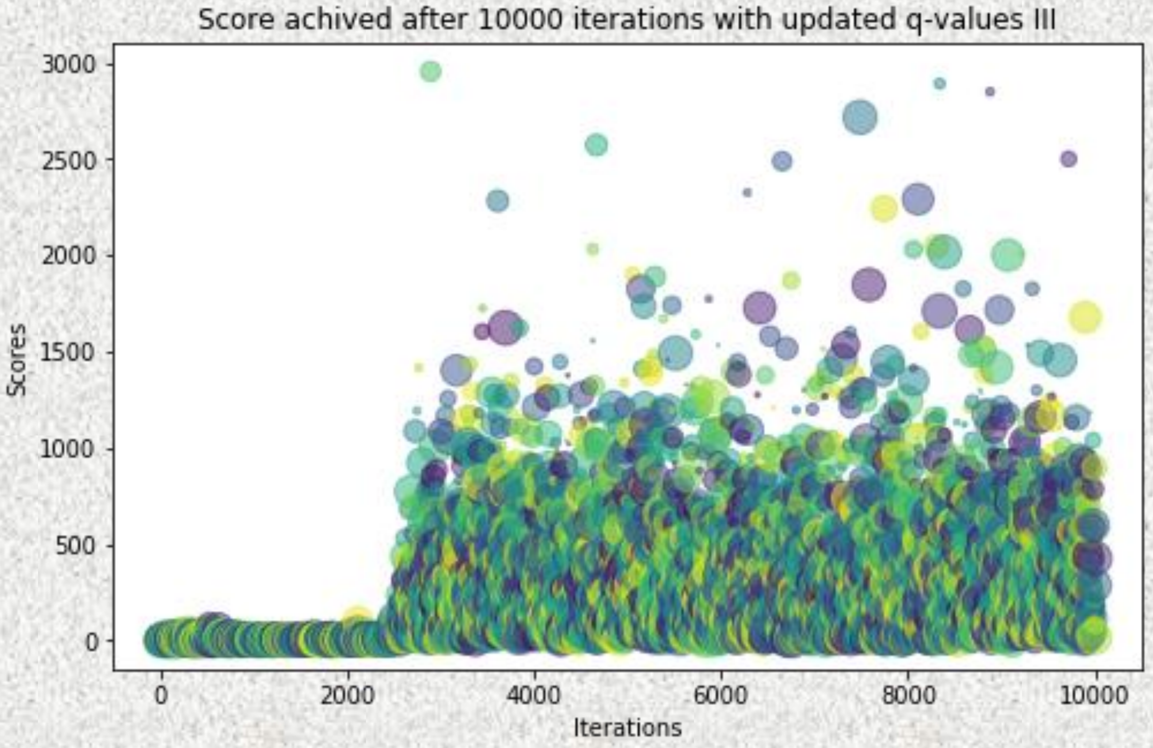
```
Running Loss:  377.06744755864946
Batch Loss:  377.06744755864946
Game Ends:  1426
Game Iterations:  100


Running Loss:  377.06712764873873
Batch Loss:  377.06712764873873
Game Ends:  1427


Running Loss:  377.066974583025
Batch Loss:  377.066974583025
Game Ends:  1427
Game Iterations:  100


Running Loss:  377.06165210954595
Batch Loss:  377.06165210954595
Game Ends:  1428


Running Loss:  377.0568306727977
Batch Loss:  377.0568306727977
Game Ends:  1428
Game Iterations:  100


Running Loss:  377.05983390433045
Batch Loss:  377.05983390433045
Game Ends:  1429


Running Loss:  377.05743649536396
Batch Loss:  377.05743649536396
Game Ends:  1429
Game Iterations:  100
```

Evaluating the performance of the models

Using Q-Learning

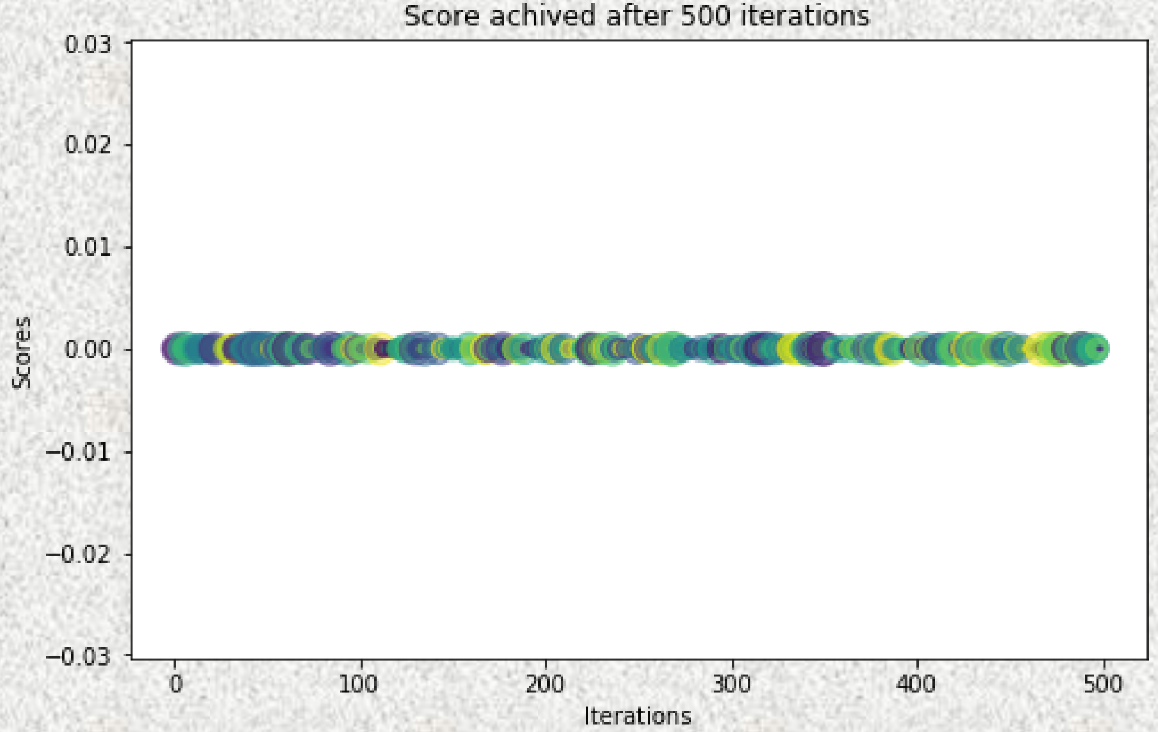


Using Q-Learning II

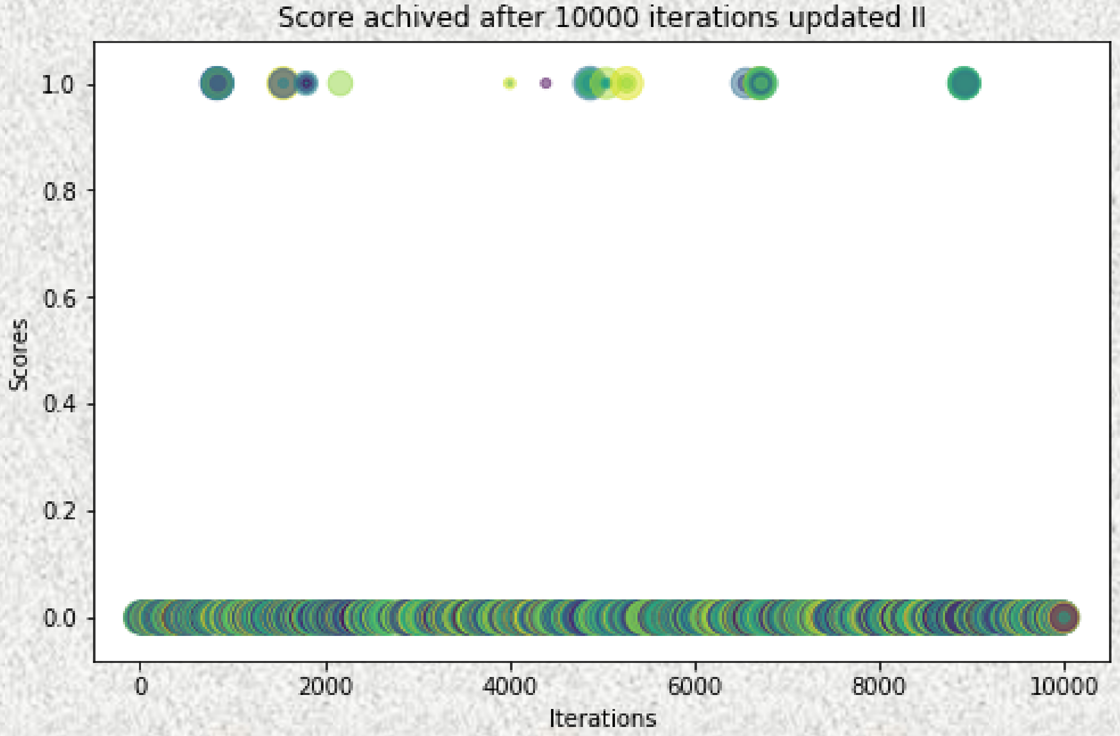


Evaluating the performance of the models

Using DQN



Using DQN II



Measure the success of the two approaches

MODELS	AVERAGE	MAX
Q-Learning	220	2955
DQN	0.009	1

The most suitable approach because we have a very small environment. Q-Learning can easily lose feasibility with larger number of states and actions.

Future Work

- ✓ Implement DQN using the input images from YOLOv3
- ✓ Test object tracking techniques to improve the game performance
- ✓ Use the Q-learning code as a baseline to train similar games


THANK
YOU