Usage of CNN for real-time environment recognition in game ‘Spelunky 2’

데이터과학과

2023320302

Team 20. 이유찬

Spelunky 2 is a roguelike 2D platformer game. Every map/level is randomly generated under certain rules. It is known for unforgiving difficulty and very dangerously programmed enemies. It takes lots of tries to even clear the first level. In this report, I used Convolutional Neural Network to recognize the environment around player. With this, I plan to eventually make a functioning ai bot that can beat level 1-1.

Objective

In order to implement fully functional ai bot, maximum 1 second of reaction time is needed (due to heuristic). Therefore, lower than 0.3 seconds of recognition time is preferred.

Gathering Data

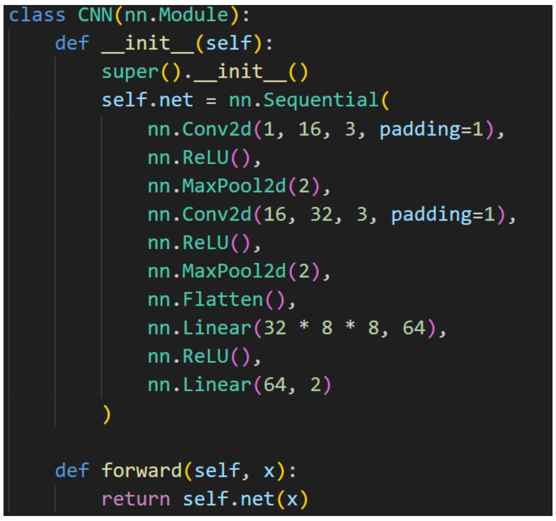
Before training the model, actual data was needed to be gathered. There was not much data online, so I had to gather it myself with the help of macro. The pipeline can be shown as below:

1. Take screenshot of game
2. Make it into grayscale
3. Split it into 32 x 32 sized blocks

Using this, I could gather few hundreds of data. I then manually moved the blocks under directory “air” and “solid”. They each contain about 250~300 block pictures. There might have been more subcategories, but due to lack of data and schedule issues, this had to be left for future improvement.

Implementing CNN

Pytorch and NumPy were used as main libraries. RTX 4070 super was used as hardware. Below is the implementation of CNN.

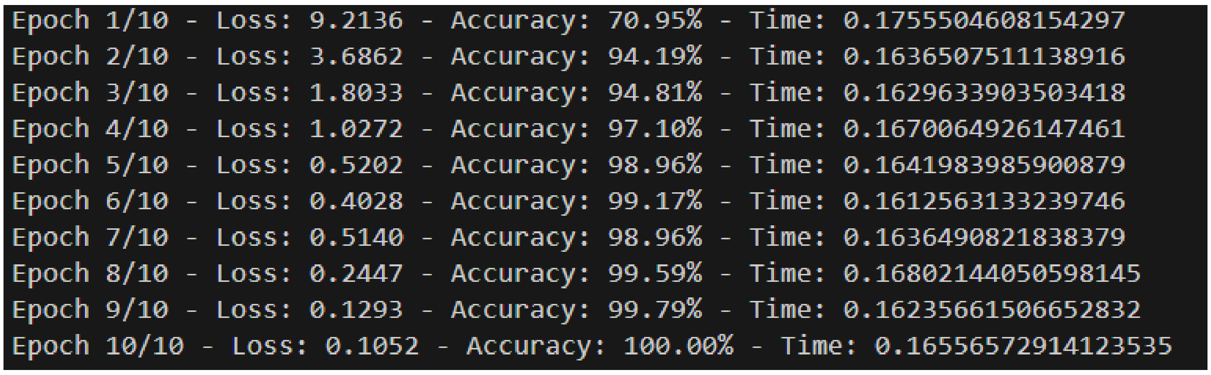


```
class CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Conv2d(1, 16, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Conv2d(16, 32, 3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2),
            nn.Flatten(),
            nn.Linear(32 * 8 * 8, 64),
            nn.ReLU(),
            nn.Linear(64, 2)
        )

    def forward(self, x):
        return self.net(x)
```

Training CNN

I used Adam as optimizer and cross-entropy for loss. The loss and accuracy converged very fast (around epoch 8) as seen below.

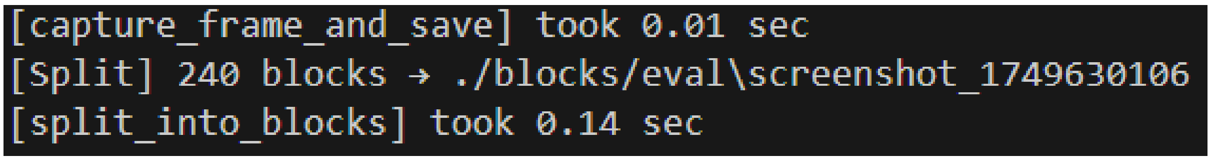


```
Epoch 1/10 - Loss: 9.2136 - Accuracy: 70.95% - Time: 0.1755504608154297
Epoch 2/10 - Loss: 3.6862 - Accuracy: 94.19% - Time: 0.1636507511138916
Epoch 3/10 - Loss: 1.8033 - Accuracy: 94.81% - Time: 0.1629633903503418
Epoch 4/10 - Loss: 1.0272 - Accuracy: 97.10% - Time: 0.1670064926147461
Epoch 5/10 - Loss: 0.5202 - Accuracy: 98.96% - Time: 0.1641983985900879
Epoch 6/10 - Loss: 0.4028 - Accuracy: 99.17% - Time: 0.1612563133239746
Epoch 7/10 - Loss: 0.5140 - Accuracy: 98.96% - Time: 0.1636490821838379
Epoch 8/10 - Loss: 0.2447 - Accuracy: 99.59% - Time: 0.1680214405059814
Epoch 9/10 - Loss: 0.1293 - Accuracy: 99.79% - Time: 0.1623566150665283
Epoch 10/10 - Loss: 0.1052 - Accuracy: 100.00% - Time: 0.1655657291412353
```

This may have two causes. First, the dataset was too small. Second, classification may have been too easy. It had only two labels, and very recurring block shapes.

Inferencing

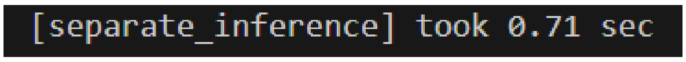
Three methods in total were used for analysis. All of them used the same pipeline for screenshotting and splitting.



```
[capture_frame_and_save] took 0.01 sec
[Split] 240 blocks → ./blocks/eval\screenshot_1749630106
[split_into_blocks] took 0.14 sec
```

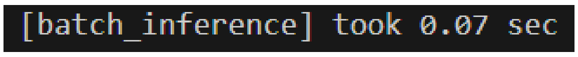
This took around 0.15 seconds, so inferencing had to be done under another 0.15 seconds.

The first method was very simple. Put all split pictures one by one through the network. This took 0.79 seconds, making it totally unusable.



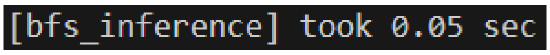
```
[separate_inference] took 0.71 sec
```

The second method was to make a batch matrix of all those split pictures and send it through the network. GPU was able to do parallel computing quick and took 0.07 seconds, achieving the objective.



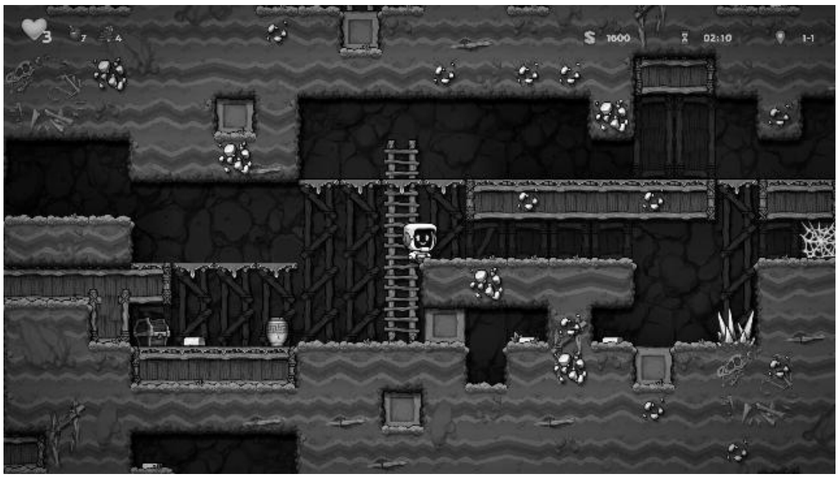
```
[batch_inference] took 0.07 sec
```

However, I took one step further for even shorter inference time. The idea of BFS and FOV was used to only infer the reachable/accessible blocks. The result was not so significant, but surely and consistently showed faster inference by 0.05 seconds.

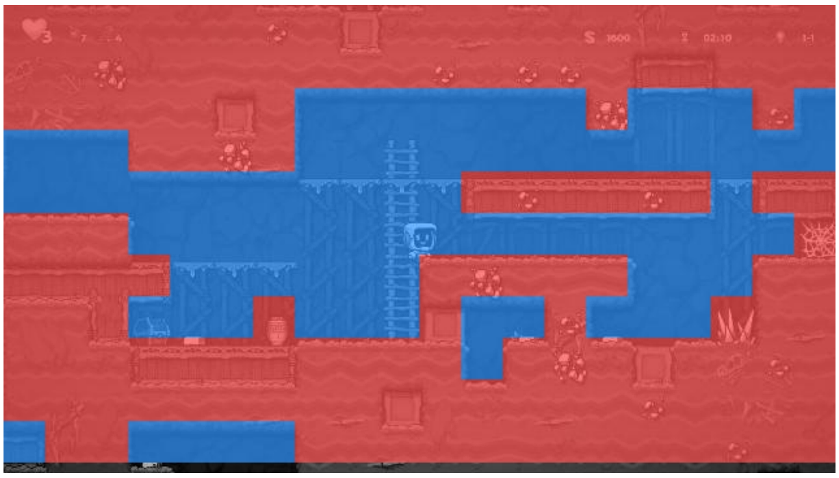


```
[bfs_inference] took 0.05 sec
```

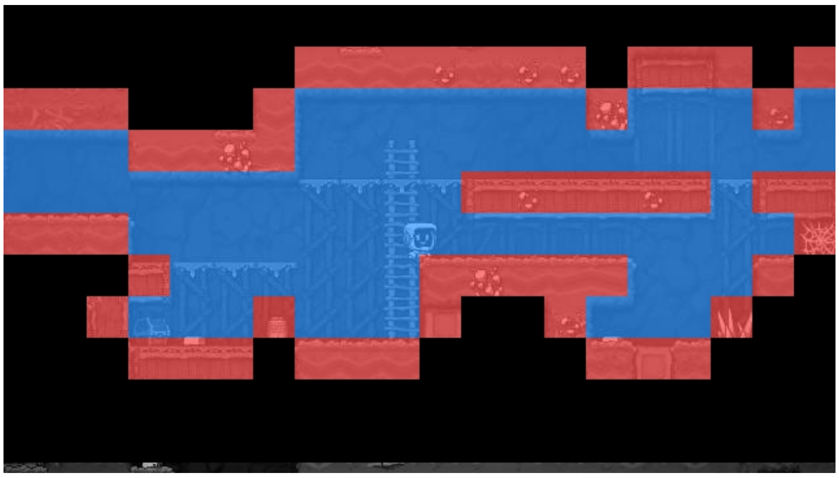
Below are the grayscale map and heatmaps of full/BFS inferences.


3
1600
02:10
1-1

<original grayscale map>



<fully inferenced heatmap>



<BFS inferenced heatmap>

Future improvements

Most importantly, the actual movements of the ai bot should be implemented. Using the BFS map to find a way out from current frame would be nice, while reinforcement learning would be also nice.

More subcategories of environment blocks would be good for higher survival rates. According actions for certain enemies and traps will be very useful.