# Deep_learning_HW3_Report

林若瑜_0853420

## Problem_1:

Data augmentation can be used to enhance GAN training. Describe how you preprocess the dataset (such as resize, crop, rotate and flip) and explain why.

```
transform = transforms.Compose([transforms.Resize((64, 64)),
                                transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5),
                                                      (0.5, 0.5, 0.5))])

#data_loc = 'C:\Users\user\Desktop\HW3_data'


dataset = datasets.ImageFolder(r'C:\Users\user\Desktop\HW3_data', transform)
# Create the dataloader
dataloader = torch.utils.data.DataLoader(dataset=dataset, batch_size=BATCH_SIZE, shuffle=True)
print('done')
```

把照片轉成 64*64 的大小，用 ImageFolder, dataloader 來處理資料，實踐數據讀取。

```
def main(args):
    # Create the dataset by using ImageFolder(get extra point by using customized dataset)
    # remember to preprocess the image by using functions in pytorch
    transform = transforms.Compose([transforms.Resize((64, 64)), transforms.ToTensor(), transforms.Normalize((0.5, 0.5, 0.5), (0

    data_loc = r'C:\Users\user\Desktop\HW3_data'
    dataset = datasets.ImageFolder(data_loc, transform)
    # Create the dataloader
    dataloader = torch.utils.data.DataLoader(dataset=dataset, batch_size=BATCH_SIZE, shuffle=True)
    print('done')


    # Create the generator and the discriminator()
    # Initialize them
    # Send them to your device
    netG = Generator(ngpu).to(device)
    netD = Discriminator(ngpu).to(device)


    # # Loss fuG_out_D_intion

    fixed_noise = torch.randn(64, nz, 1, 1, device=device)

    real_label = 1
    fake_label = 0



    # Setup optimizers for both G and D and setup criterion at the same time
    optimizer_g = optim.Adam(netG.parameters(), lr=lr, betas=(beta1, 0.999))
    optimizer_d = optim.Adam(netD.parameters(), lr=lr, betas=(beta1, 0.999))
    criterion = nn.BCELoss()


    # Start training~~

    # train(dataloader, netG, netD, optimizer_g, optimizer_d, criterion, args.num_epochs)
    G_losses, D_losses, img_list = train(dataloader, netG, netD, optimizer_g, optimizer_d, criterion, 5)

    return G_losses, D_losses, img_list
```

Main 的部分會處理照片資料，產出一個 generator 和一個 discriminator，設定 real_label 設成 1，fake_label 設成 0，把 optimizer 設 Adam，criterion 設 nn.BCELoss()

```python
# Each epoch, we have to go through every data in dataset
for epoch in range(num_epochs):
    # Each iteration, we will get a batch data for training
    for i, data in enumerate(dataloader, 0):
        # Update D network
        # initialize gradient for network
        # send the data into device for computation
        netD.zero_grad()
        real_cpu = data[0].to(device)
        b_size = real_cpu.size(0)
        label = torch.full((b_size,), real_label, device=device)
        output = netD(real_cpu).view(-1)

        # Send data to discriminator and calculate the loss and gradient
        # For calculate loss, you need to create label for your data
        errD_real = criterion(output, label)
        errD_real.backward()
        D_x = output.mean().item()

        ## Using Fake data, other steps are the same.
        # Generate a batch fake data by using generator
        noise = torch.randn(b_size, nz, 1, 1, device=device)
        fake = netG(noise)
        label.fill_(fake_label)
        output = netD(fake.detach()).view(-1)

        errD_fake = criterion(output, label)
        errD_fake.backward()

        D_G_z1 = output.mean().item()
        errD = errD_real + errD_fake
        optimizer_d.step()

        # Update G network
        netG.zero_grad()
        label.fill_(real_label)
        output = netD(fake).view(-1)
        errG = criterion(output, label)
        errG.backward()
        D_G_z2 = output.mean().item()
        optimizer_g.step()
```

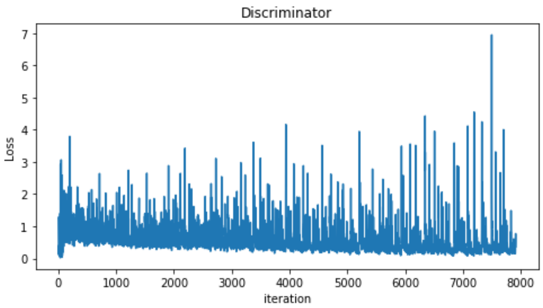
在 train 裡面，預設跑 5 個 epoch，batch_size = 128

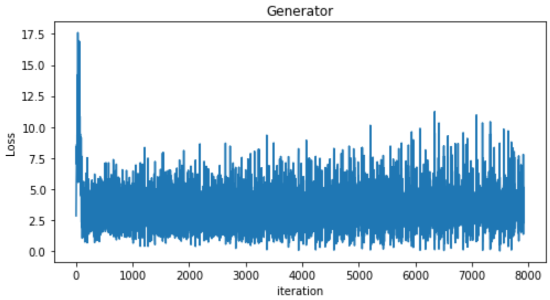
```python
# Start training~~

# train(dataloader, netG, netD, optimizer_g, optimizer_d, criterion, args.num_epochs)
G_losses, D_losses, img_list = train(dataloader, netG, netD, optimizer_g, optimizer_d, criterion, 5)

return G_losses, D_losses, img_list
```

在 main 裡面會執行 train()，並把 G_losses, D_losses, img_list 取出，用來畫圖形。

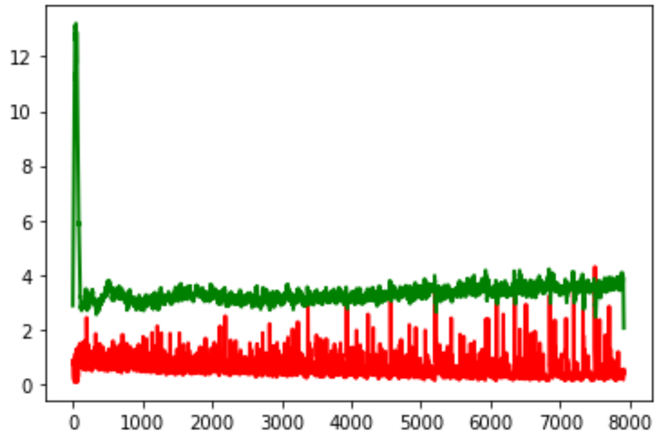
訓練結果：

```
[4/5][1250/1583]        Loss_D: 0.3597  Loss_G: 2.3615  D(x): 
0.8180    D(G(z)): 0.1125 / 0.1409
[4/5][1300/1583]        Loss_D: 0.2358  Loss_G: 4.0850  D(x): 
0.9347    D(G(z)): 0.1431 / 0.0241
[4/5][1350/1583]        Loss_D: 0.2701  Loss_G: 4.6404  D(x): 
0.9437    D(G(z)): 0.1754 / 0.0152
[4/5][1400/1583]        Loss_D: 0.3619  Loss_G: 3.1457  D(x): 
0.8419    D(G(z)): 0.1406 / 0.0611
[4/5][1450/1583]        Loss_D: 0.1406  Loss_G: 3.8360  D(x): 
0.9181    D(G(z)): 0.0487 / 0.0312
[4/5][1500/1583]        Loss_D: 0.9752  Loss_G: 7.6671  D(x): 
0.9814    D(G(z)): 0.5582 / 0.0008
[4/5][1550/1583]        Loss_D: 0.1727  Loss_G: 3.3429  D(x): 
0.9057    D(G(z)): 0.0611 / 0.0542
```

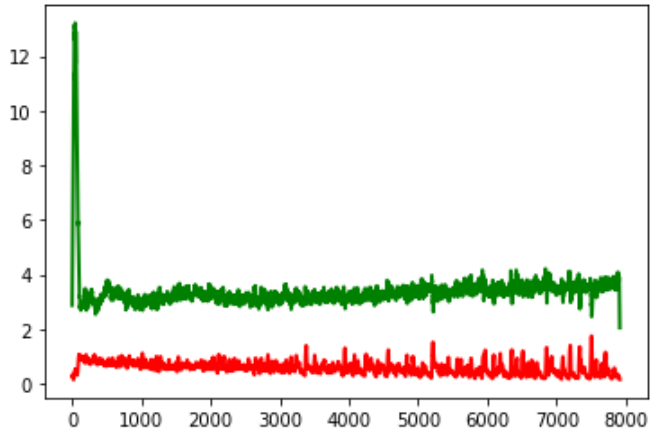



Discriminator 和 generator 的 loss 圖，因為畫的點有點多，看不太出變化，因此讓 Generator （綠）19 個才畫點一次、Discriminator （紅）3 個才畫點一次。

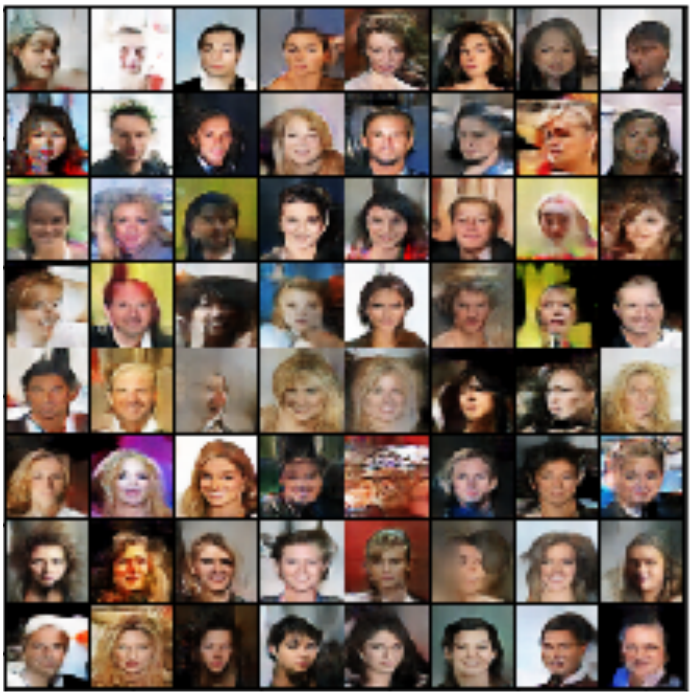


Generator（綠）和 Discriminator（紅）都 19 個才畫點一次。



可以比較清楚看出 loss 收斂的位置。

draw some samples generated from your model：



從結果看來 DCGAN 所產生出來的照片還沒有很全面，可以看到 GAN 模型並不是個很好控制，有可能 Discriminator 分辨率太過強大，漸漸導致 Generator 的權重不管怎麼更新 Loss 還是沒用，使 Training 越來越沒有效果。

## Problem_2:

Explain the purpose of the following hyperparameters: updating step α, discount factor γ, target network update period τ, and ε for ε-greedy policy.

- updating step α：是學習效率 learning rate，為小於 1 的值，決定這次誤差有多少要被學習。
- ε-greedy：是一種決策策略，例如他等於 0.9 時，就代表有 0.9 的機率會按照最優 reward 來當 action，0.1 的機率隨機選擇 action
- discount factor γ：是對未來獎勵的衰減值，離狀態 1 越遠，衰退得越多。γ =1 時代表可以清楚估算所有獎勵，=0 時代表只能知道最接近的 state 獎勵值。
- target network update period τ：要限制 target network 的更新時間，因為如果每次訓練都更新參數的話，reward 會變成一個變數，這樣訓練起來會有問題。

To speed up the training process, you can simply change the probability of random agent:
原本沒有設定 action 的機率，reward 到第 150 個左右的 episode 時仍然會一直

是 0，要訓練很多次才會開始有 reward 值：

```
Episode:    150, interaction_steps: 309248, reward:  0, epsilon: 0.721677
[Info] Save model at './model' !
Evaluation: True, Episode:    150, Interaction_steps: 309248, evaluate reward: 0.000000
Episode:    151, interaction_steps: 311296, reward:  0, epsilon: 0.719834
Episode:    152, interaction_steps: 313344, reward:  0, epsilon: 0.717990
Episode:    153, interaction_steps: 315392, reward:  0, epsilon: 0.716147
Episode:    154, interaction_steps: 317440, reward:  0, epsilon: 0.714304
Episode:    155, interaction_steps: 319488, reward:  0, epsilon: 0.712461
Episode:    156, interaction_steps: 321536, reward:  0, epsilon: 0.710618
Episode:    157, interaction_steps: 323584, reward:  0, epsilon: 0.708774
Episode:    158, interaction_steps: 325632, reward:  0, epsilon: 0.706931
Episode:    159, interaction_steps: 327680, reward:  0, epsilon: 0.705088
```

但當設定 NOOP (0.3), UP (0.6), DOWN (0.1)後，reward 明顯升高比較多，從一開始就有 reward 值。

```
Episode:      0, interaction_steps:   2048, reward: 10, epsilon: 0.998157
[Info] Save model at './model' !
Evaluation: True, Episode:      0, Interaction_steps:   2048, evaluate reward: 0.000000
Episode:      1, interaction_steps:   4096, reward: 10, epsilon: 0.996314
Episode:      2, interaction_steps:   6144, reward: 10, epsilon: 0.994470
Episode:      3, interaction_steps:   8192, reward: 12, epsilon: 0.992627
Episode:      4, interaction_steps:  10240, reward: 12, epsilon: 0.990784
Episode:      5, interaction_steps:  12288, reward: 13, epsilon: 0.988941
Episode:      6, interaction_steps:  14336, reward:  9, epsilon: 0.987098
Episode:      7, interaction_steps:  16384, reward: 11, epsilon: 0.985254
Episode:      8, interaction_steps:  18432, reward: 11, epsilon: 0.983411
Episode:      9, interaction_steps:  20480, reward: 13, epsilon: 0.981568
Episode:     10, interaction_steps:  22528, reward: 12, epsilon: 0.979725
Evaluation: True, Episode:     10, Interaction_steps:  22528, evaluate reward: 0.000000
Episode:     11, interaction_steps:  24576, reward: 12, epsilon: 0.977882
Episode:     12, interaction_steps:  26624, reward: 11, epsilon: 0.976038
Episode:     13, interaction_steps:  28672, reward: 11, epsilon: 0.974195
Episode:     14, interaction_steps:  30720, reward: 12, epsilon: 0.972352
Episode:     15, interaction_steps:  32768, reward: 15, epsilon: 0.970509
Episode:     16, interaction_steps:  34816, reward: 10, epsilon: 0.968666
Episode:     17, interaction_steps:  36864, reward: 13, epsilon: 0.966822
Episode:     18, interaction_steps:  38912, reward: 13, epsilon: 0.964979
Episode:     19, interaction_steps:  40960, reward: 13, epsilon: 0.963136
```

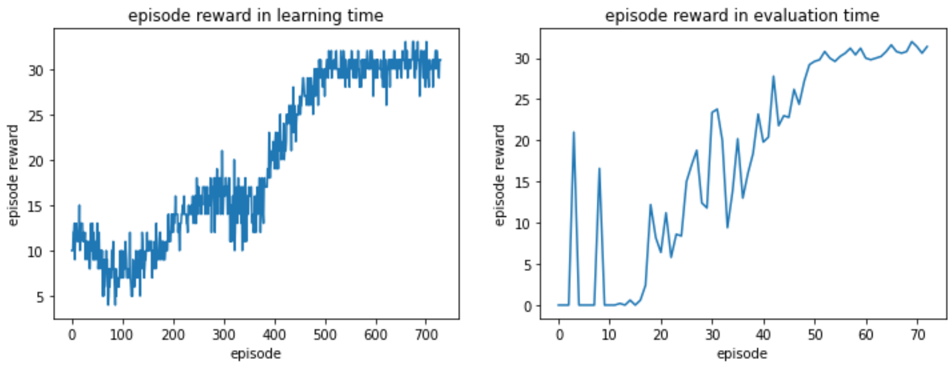
```
Episode:    425, interaction_steps: 872448, reward: 29, epsilon: 0.214797
Episode:    426, interaction_steps: 874496, reward: 27, epsilon: 0.212954
Episode:    427, interaction_steps: 876544, reward: 25, epsilon: 0.211110
Episode:    428, interaction_steps: 878592, reward: 27, epsilon: 0.209267
Episode:    429, interaction_steps: 880640, reward: 28, epsilon: 0.207424
Episode:    430, interaction_steps: 882688, reward: 25, epsilon: 0.205581
Evaluation: True, Episode:    430, Interaction_steps: 882688, evaluate reward: 29.800000
Episode:    431, interaction_steps: 884736, reward: 28, epsilon: 0.203738
Episode:    432, interaction_steps: 886784, reward: 28, epsilon: 0.201894
Episode:    433, interaction_steps: 888832, reward: 25, epsilon: 0.200051
Episode:    434, interaction_steps: 890880, reward: 27, epsilon: 0.198208
Episode:    435, interaction_steps: 892928, reward: 29, epsilon: 0.196365
Episode:    436, interaction_steps: 894976, reward: 29, epsilon: 0.194522
Episode:    437, interaction_steps: 897024, reward: 26, epsilon: 0.192678
Episode:    438, interaction_steps: 899072, reward: 27, epsilon: 0.190835
Episode:    439, interaction_steps: 901120, reward: 28, epsilon: 0.188992
Episode:    440, interaction_steps: 903168, reward: 28, epsilon: 0.187149
Evaluation: True, Episode:    440, Interaction_steps: 903168, evaluate reward: 31.400000
Episode:    441, interaction_steps: 905216, reward: 29, epsilon: 0.185306
Episode:    442, interaction_steps: 907264, reward: 29, epsilon: 0.183462
Episode:    443, interaction_steps: 909312, reward: 28, epsilon: 0.181619
Episode:    444, interaction_steps: 911360, reward: 26, epsilon: 0.179776
Episode:    445, interaction_steps: 913408, reward: 31, epsilon: 0.177933
```

```
Evaluation: True, Episode:    710, Interaction_steps: 1456128, evaluate reward: 30.600000
Episode:     711, interaction_steps: 1458176, reward: 30, epsilon: 0.100000
Episode:     712, interaction_steps: 1460224, reward: 30, epsilon: 0.100000
Episode:     713, interaction_steps: 1462272, reward: 31, epsilon: 0.100000
Episode:     714, interaction_steps: 1464320, reward: 28, epsilon: 0.100000
Episode:     715, interaction_steps: 1466368, reward: 30, epsilon: 0.100000
Episode:     716, interaction_steps: 1468416, reward: 31, epsilon: 0.100000
Episode:     717, interaction_steps: 1470464, reward: 30, epsilon: 0.100000
Episode:     718, interaction_steps: 1472512, reward: 31, epsilon: 0.100000
Episode:     719, interaction_steps: 1474560, reward: 32, epsilon: 0.100000
Episode:     720, interaction_steps: 1476608, reward: 30, epsilon: 0.100000
Evaluation: True, Episode:    720, Interaction_steps: 1476608, evaluate reward: 31.400000
Episode:     721, interaction_steps: 1478656, reward: 32, epsilon: 0.100000
Episode:     722, interaction_steps: 1480704, reward: 32, epsilon: 0.100000
Episode:     723, interaction_steps: 1482752, reward: 31, epsilon: 0.100000
Episode:     724, interaction_steps: 1484800, reward: 31, epsilon: 0.100000
Episode:     725, interaction_steps: 1486848, reward: 30, epsilon: 0.100000
Episode:     726, interaction_steps: 1488896, reward: 29, epsilon: 0.100000
Episode:     727, interaction_steps: 1490944, reward: 31, epsilon: 0.100000
```

可以看到到 700 左右的 episode 時 reward 已經可以到 30 以上，已經比論文上的結果更好。

Episode 的 reward 變化圖：

episode reward in learning time and evaluation time




After training, you will obtain the model parameters for the agent. Show total reward in some episodes for deep Q-network agent.使用訓練好的模型跑 test 玩 10 個 episode：

```
args = parser.parse_args(args=[]) #For jupyter notebook

model_path = "./model/q_target_checkpoint_1435648.pth"
test(model_path)
```

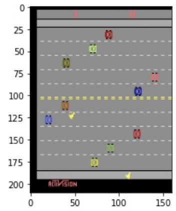
```
[Info] Restore model from './model/q_target_checkpoint_1435648.pth' !
Episode:       0, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:       1, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:       2, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:       3, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:       4, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:       5, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:       6, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:       7, interaction_steps:       0, reward: 33, epsilon: 0.100000
Episode:       8, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:       9, interaction_steps:       0, reward: 29, epsilon: 0.100000
```

Reward 幾乎都在 30 以上，比論文上的結果更好。

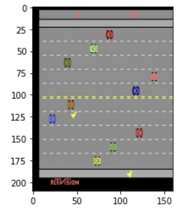
Sample some states, show the Q values for each action, analyze the results, and answer：

## NOOP

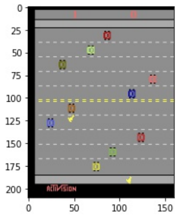
Step: 0091
NOPE: 1.871479, UP: 1.864083, DOWN: 1.863474
Average Q: 1.866345, Action: 0



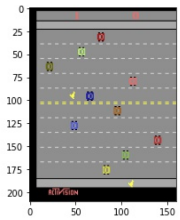
Step: 0092
NOPE: 1.885095, UP: 1.728961, DOWN: 1.879105
Average Q: 1.831053, Action: 0



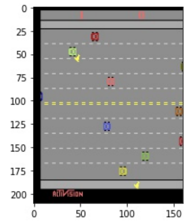
Step: 0093
NOPE: 1.912837, UP: 1.691479, DOWN: 1.901236
Average Q: 1.835184, Action: 0



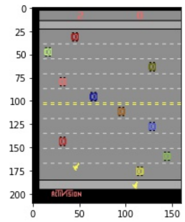
Step: 0105
NOPE: 0.000000, UP: 0.000000, DOWN: 0.000000
Average Q: 0.000000, Action: 0



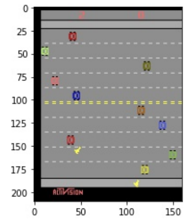
Step: 0120
NOPE: 2.521756, UP: 2.090146, DOWN: 2.499797
Average Q: 2.370566, Action: 0



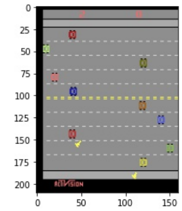
Step: 0145
NOPE: 1.700208, UP: 1.697618, DOWN: 1.697567
Average Q: 1.698464, Action: 0



Step: 0150
NOPE: 1.712855, UP: 1.701951, DOWN: 1.712389
Average Q: 1.709065, Action: 0

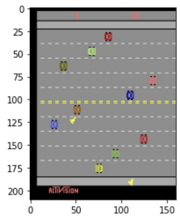


Step: 0151
NOPE: 1.721436, UP: 1.716186, DOWN: 1.719081
Average Q: 1.718901, Action: 0

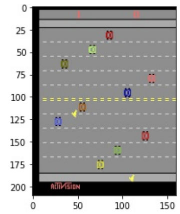


## UP

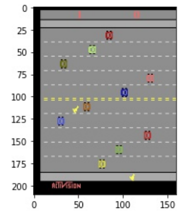
Step: 0094
NOPE: 1.906144, UP: 1.915945, DOWN: 1.893842
Average Q: 1.905310, Action: 1



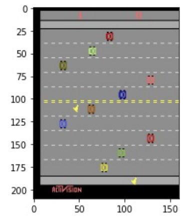
Step: 0095
NOPE: 1.909271, UP: 1.912442, DOWN: 1.897277
Average Q: 1.906330, Action: 1



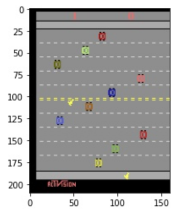
Step: 0096
NOPE: 1.919047, UP: 1.930493, DOWN: 1.904922
Average Q: 1.918154, Action: 1



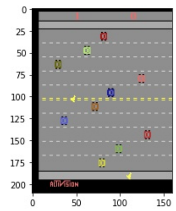
Step: 0097
NOPE: 1.949683, UP: 1.954647, DOWN: 1.939605
Average Q: 1.947979, Action: 1



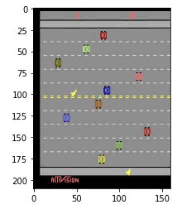
Step: 0098
NOPE: 1.946533, UP: 1.981425, DOWN: 1.941709
Average Q: 1.956556, Action: 1



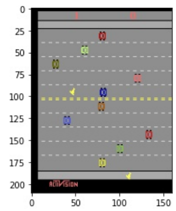
Step: 0099
NOPE: 1.990270, UP: 1.994411, DOWN: 1.983897
Average Q: 1.989526, Action: 1



Step: 0100
NOPE: 1.994772, UP: 2.014825, DOWN: 1.990869
Average Q: 2.000155, Action: 1

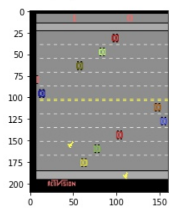


Step: 0101
NOPE: 2.035492, UP: 2.049777, DOWN: 2.027794
Average Q: 2.037688, Action: 1

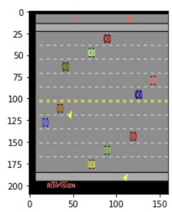


## DOWN

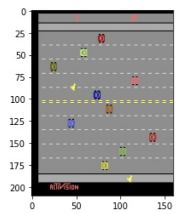
Step: 0078
NOPE: 0.000000, UP: 0.000000, DOWN: 0.000000
Average Q: 0.000000, Action: 2



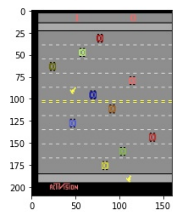
Step: 0090
NOPE: 1.875835, UP: 1.714592, DOWN: 1.877777
Average Q: 1.822735, Action: 2

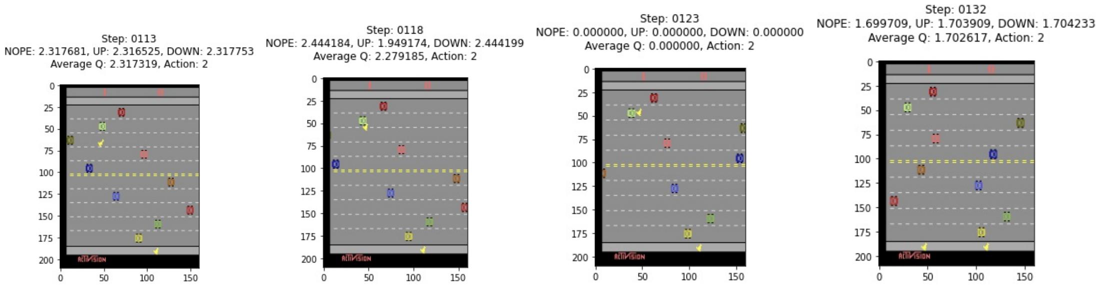


Step: 0103
NOPE: 2.062688, UP: 2.064221, DOWN: 2.066136
Average Q: 2.064348, Action: 2



Step: 0104
NOPE: 2.077536, UP: 2.080404, DOWN: 2.080745
Average Q: 2.079561, Action: 2






1.大部分的狀況都判斷的蠻好的，但有些狀況還是沒有人判斷的準確，如 up 的第一個圖。

2.因爲演算法會有 exploration 的機制，所以大部分時間會取學習過程中Ｑvalue 比較高的 action 來做，但還是會有 epsilon 的機率他選其他的 action，所以值會時高時低。