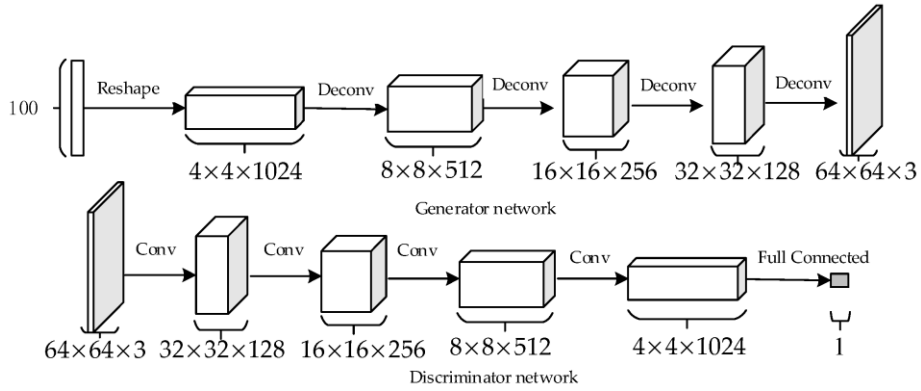
Deep Learning Homework3

統研所碩一 曾鈺評 0852617

一、 Generative adversarial network (GAN)

本次 GAN 架構如下：



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.

(1) 一開始我們先把全部圖片resize成64*64，以統一圖片大小並且可以減少資料量

(2) 把所有圖片normalize到(-1,1)的範圍，因為我們在generator中架構設計最後一層是tanh，理論上應該輸出為(-1,1)的範圍，另外有嘗試不要normalize的版本，即範圍設在(0,255)，發現這樣會導致train出來的風格跟原本圖片很不一樣，推測是因為(0,255)只會被取到(0,1)的範圍，在某種程度上過濾掉某些特徵，詳細的結果請見下面

(3) 隨機選取一半的圖片進一步作水平翻轉，因為根據猜測，水平翻轉就有如人照鏡子作反射一般，會看到左右相反的圖像以增加圖片的多樣性。

2. Construct a DCGAN with vanilla GAN objective, plot the learning curves for both generator and discriminator, and draw some samples generated from your model.

$$\max_D \mathcal{L}(D) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}} \log(1 - D(G(\boldsymbol{z})))$$

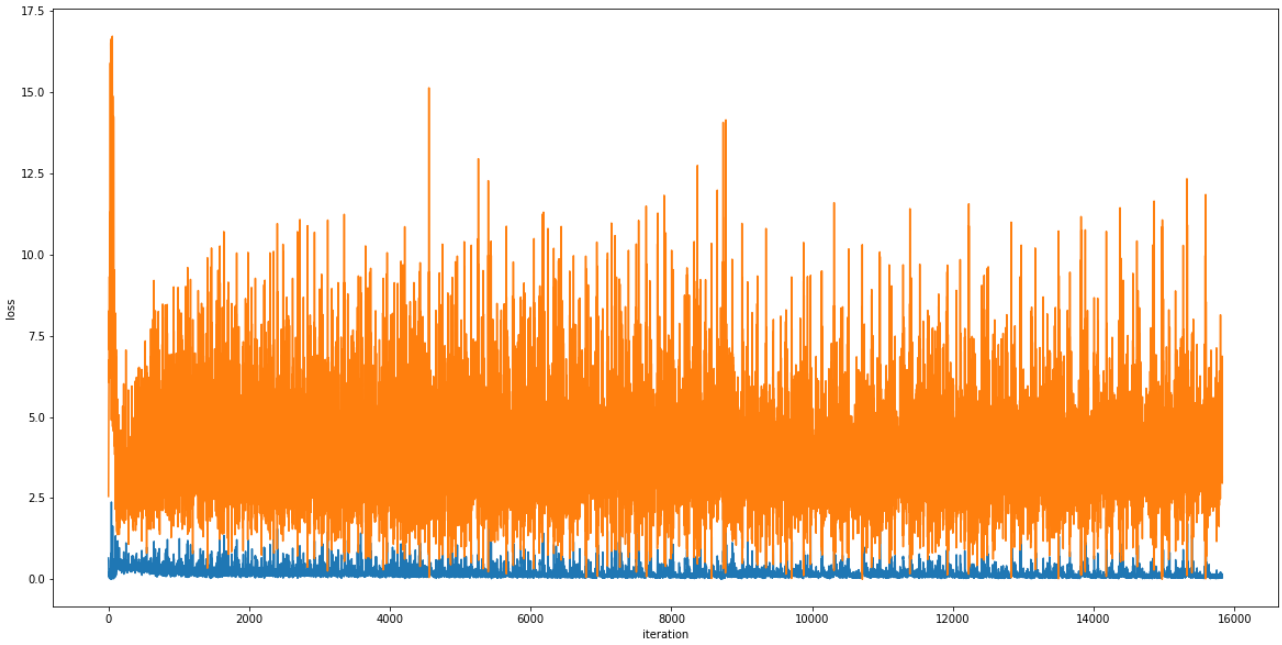
$$\min_G \mathcal{L}(G) = \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}} \log(1 - D(G(\boldsymbol{z}))$$

以下我們進行 3 種不同的調整：

版本	optimizer	normalize
1	Adam	有
2	RMSprop	有
3	RMSprop	無

[版本 1]

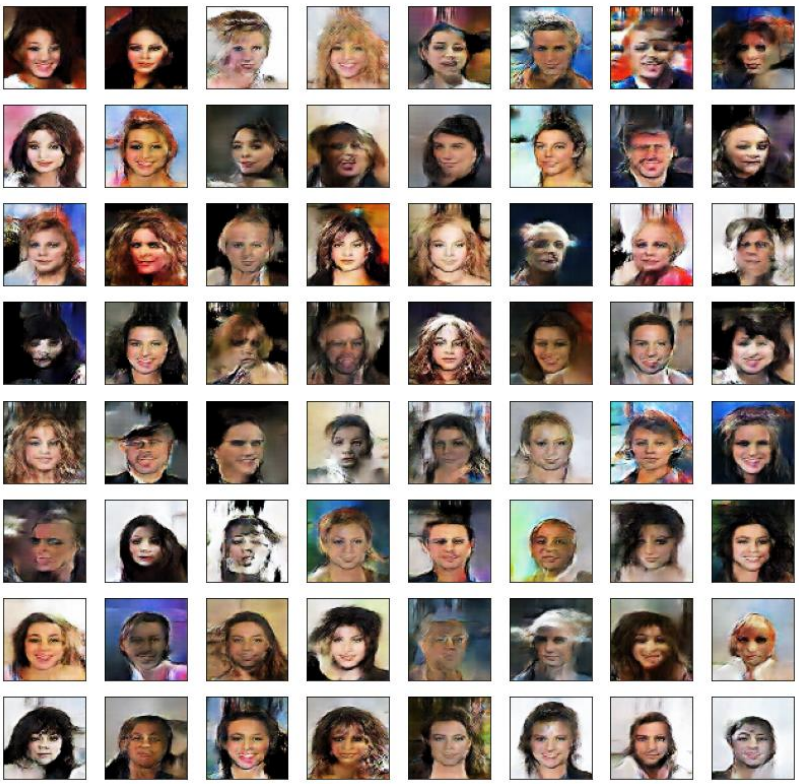
在原始架構下，optimizer 選擇使用 Adam，圖片資料轉成數值資料後作 normalize，範圍為(-1,1)之間，我們可以發現 gernerator 的 Loss 大部分會一直在 2.5~5 之間震盪，有時會有大於 10 的情況，discriminator 的 loss 則一直很穩定在 0~2.5 左右。



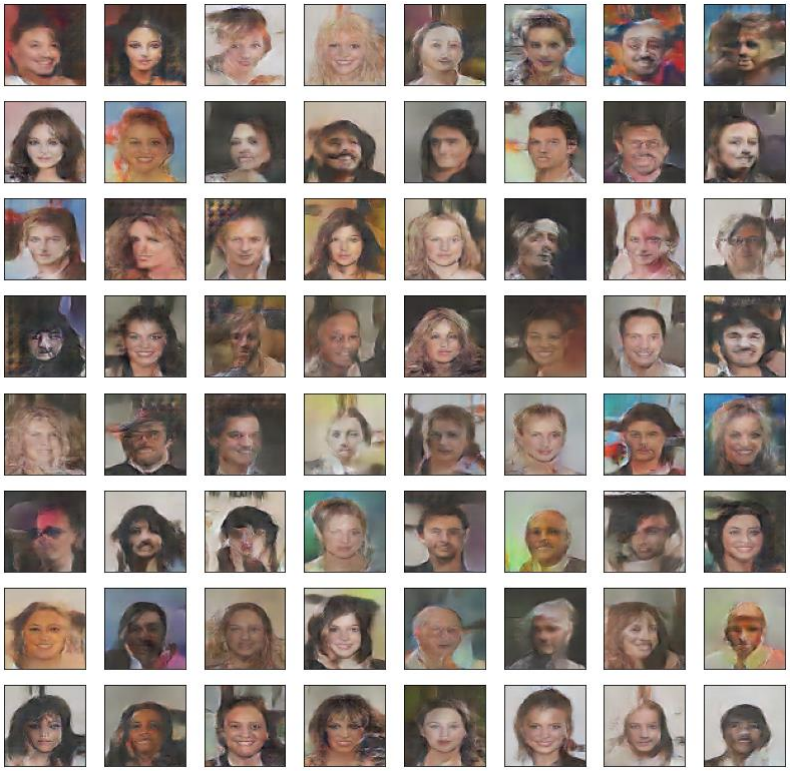
原始圖片如下：

我們可以實際抽出一些 sample 來看，以下依序是第 1 到第 5 個 epoch 的範圍，可以發現到在第一個 epoch 時，大部分的人臉已經初步呈現出形狀，但仔細看會發現臉部特徵仍然不是非常清楚，比如最右邊下面兩張的臉扭曲的有些嚴重。

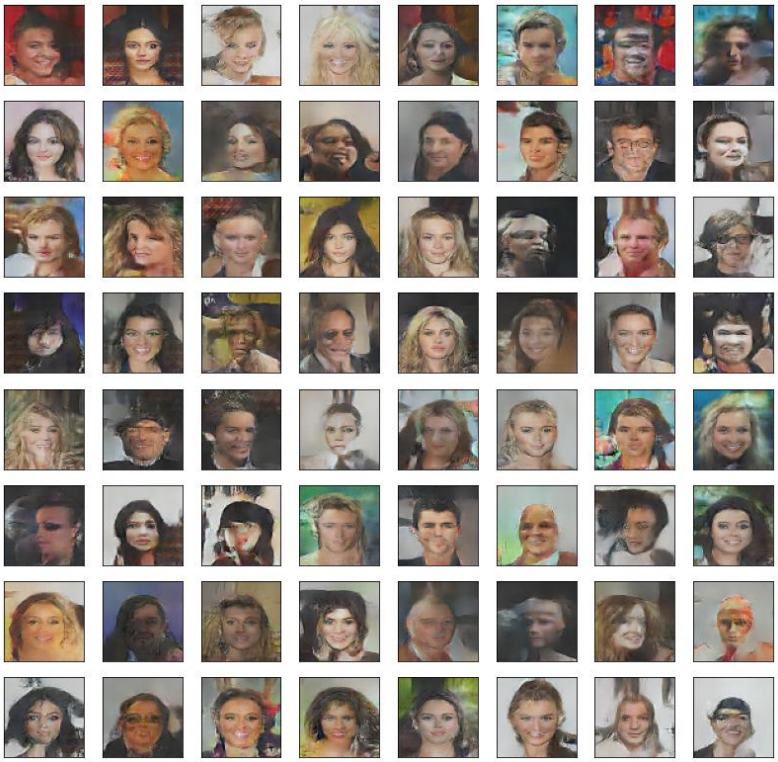
接著以下依序是第 2 個 epoch：



第 3 個 epoch：



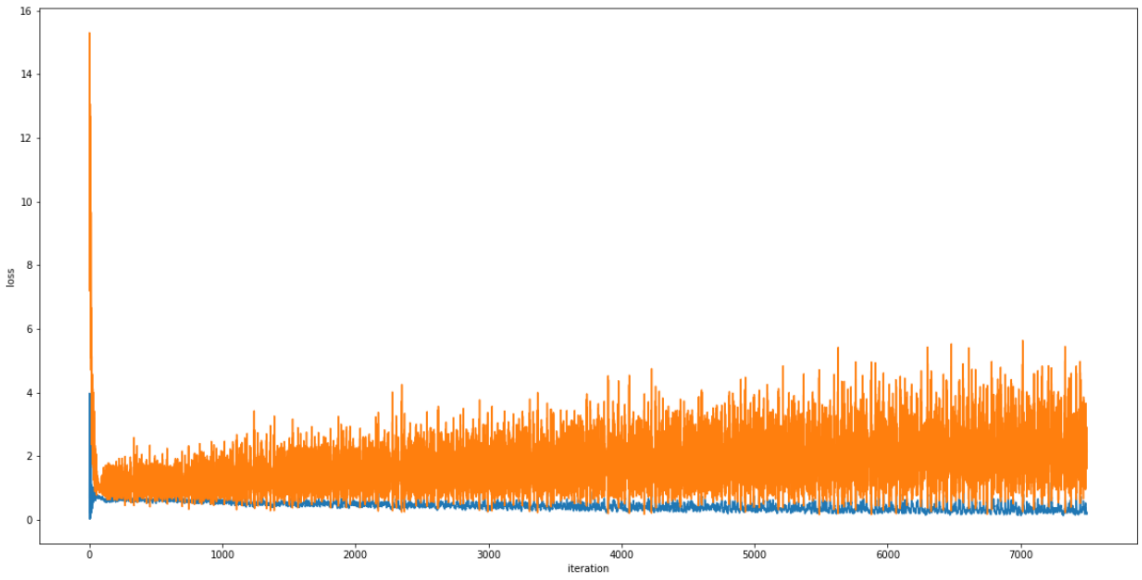
第 4 個 epoch：



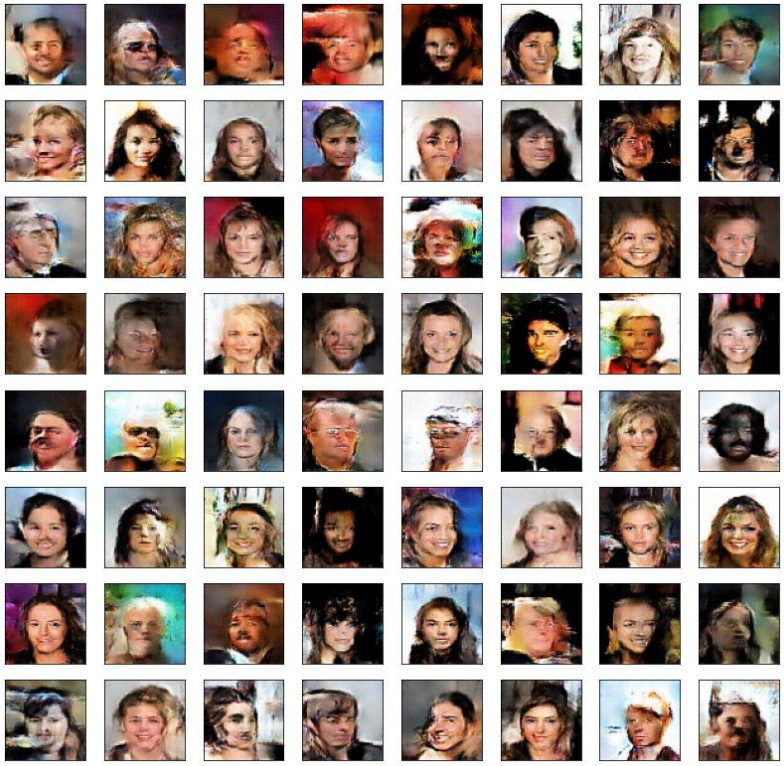
當 train 到第 5 個 epoch 時，我們可發現有些人臉相當的完整，但有些仍然無法描繪出清楚的人臉輪廓，將人與背景混雜在一起

全部 5 個 epoch 的動態圖片請見下列連結：
https://drive.google.com/file/d/1CDAERN9na5ZZisU-jCzLz56jcW3rz4eV/view?usp=sharing

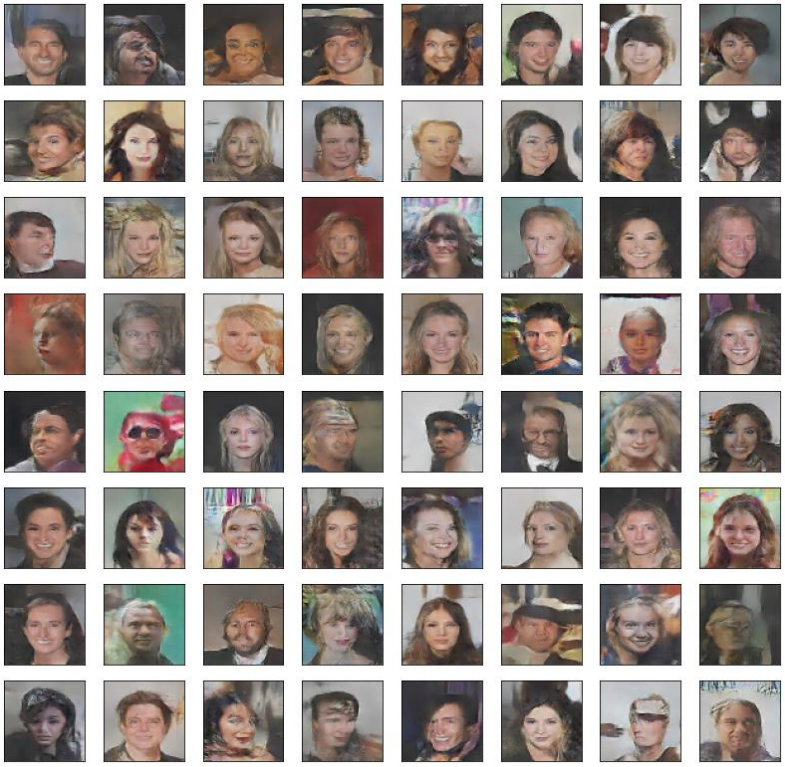
[版本 2]
版本 2 相較於版本 1 是將原本 optimizer 的 Adam 改為 RSMprop (Root Mean Square Prop)，差別在於少了 momentum 與 bias correction，這樣得到的震幅較小的 loss 結果，但會發現 train 到後期整體平均 loss 會有上升的趨勢，似乎越訓練反而越不收斂



第一個 epoch 生成圖片如下，與前面一樣還是有一些背景與人臉混在一起的結果：

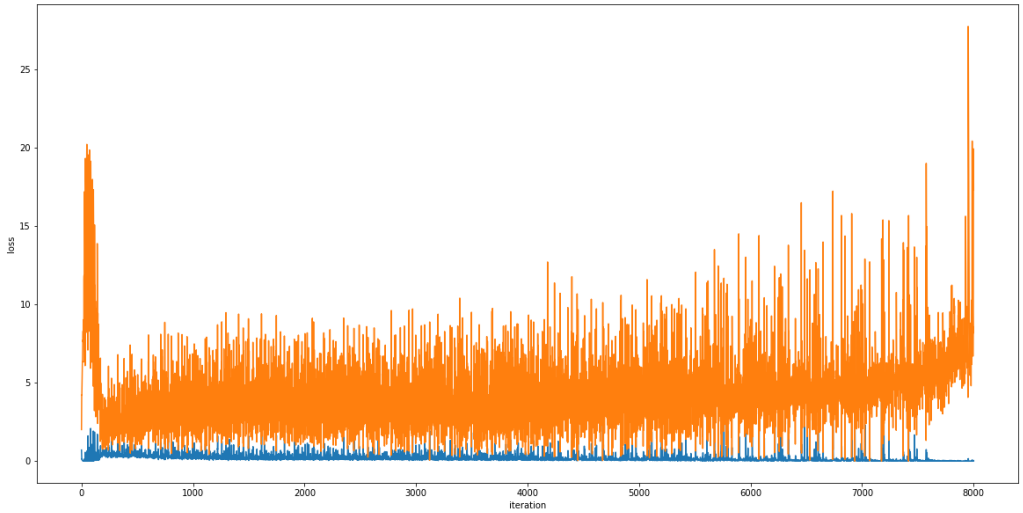


第 2,3,4 個 epoch 的圖片在此就不再附上，詳細結果可參考下面動畫圖的連結，train 到第 5 個 epoch 的結果如下，跟上面用 Adam 訓練出來的結果相比，我認為臉部輪廓清楚很多，但還是有一些圖片沒有訓練得很成功，出現一些不相干的東西：



全部 5 個 epoch 的動態圖片請見下列連結：
https://drive.google.com/file/d/102VoPUIRbRIxkrvOFQ93ytmhK1-bu9mR/view?usp=sharing

[版本 3]
第 3 個版本是未將資料 normalize 過去跑 model 所 train 出來的結果，一開始是因為忘了將資料作 Normalize 的前處理，但意外的是似乎能得到另一種風格。Loss 幾乎都在 2.5~7.5 之間震盪，到後期(大約第 7500 個 iteration)之後會有逐漸上升的趨勢



接著來看一下隨機抽樣產生的圖片，以下是第一個 epoch 的，train 出來的結果風格呈現淺色系，人臉輪廓相比前面變的更不清楚，變得有點像是手繪版本，基本上這還算蠻合理的，因為我們的資料範圍是(0,255)，但是模型只用到(0,1)的範圍，所以有很多原本應該保留的特徵都被去除掉了，也可以大致上推論資料數值範圍較小的部分應該是比較淺的色彩：

Train 到第 5 個 epoch 時，結果比我想像中預期的還要好，和第一個 epoch 一樣維持同樣淺色風格，我們可以發現除了有一些人臉輪廓不太清楚，但有一些圖片仍可看得出來有人臉輪廓與五官，有如手繪圖片：

全部 5 個 epoch 的動態圖片請見下列連結：
https://drive.google.com/file/d/1Wp3CcRKJzBldUucLWmN1UR3iLBKnYUqg/view?usp=sharing

3. Implementation details are addressed as follows
(a) Models has already been designed in Model.py. Feel free to modify the generator and discriminator, and you can write down how you design your model and why. (bonus 5 points)

原本optimizer不是選用Adam而是選擇RMSprop (Root Mean Square Prop)，差別在於少了momentem與bias correction，相較之下RSMprop在沒有momentum的情況下也可以訓練得起來，只不過loss最後似乎有些微上升的趨勢。結果請見上述[版本2]。

(b) In data preprocessing, the ImageFolder and Dataloader provided by pytorch are recommended. The customized dataset (without using ImageFolder) can be implemented for extra points. (bonus 5 points)

在資料前處理的部分並沒有使用ImageFolder進行，而是先用cv2.imread()將照片讀進來，轉成numpy array的數值資料後再用transform作normalize到(-1,1)的範圍，而在前處理的過程中因為

不小心忘記作normalize也能夠train得起來，請見上述[版本3]

(c) In main.py, you have to complete three functions. main(), train(), and visualization.

詳細code請見檔案"HW3-1_0852617_曾鈺評_main.py"，　"Model.py", "HW3-1_0852617_曾鈺評_Visualizaion.py"

(d) Visualization

- plot the following two charts and show them in the report

i. original samples and generated samples

ii. learning curves for generator and discriminator

- training procedure of GANs is unstable, when visualizing the loss curve you can do moving average every N steps (smooth the curve) to observe the trend easily.
- we provided an extra visualizaion code in Visualization.ipynb, you can use it to see what does the generator generates through the training procedure.

詳細code請見檔案"HW3-1_0852617_曾鈺評_Visualizaion.py"
這部分已於第2小題陳述過，請參見上面。

(e) Please do some discussion about your implementation. You can write down the difficulties you face in this homework, e.g. hyperparameter settings, analysis of the generated images, or anything you want to address.

(1) GAN 跟前面所教過的 CNN、RNN 等相比起來，它比較沒有一個標準評判的準則來判斷訓練結果到底好不好，所以在訓練的時候我們通常都只能很主觀的決定是不是有訓練好
(2) GAN 如果訓練太久反而會使 discriminator 壞掉，看文獻是因為這樣會造成梯度消失，可是通常我們都要等到看到壞掉的圖片才知道訓練太久了，這時候就又必須在從前面開始重跑模型，所以把每個 epoch 的模型參數都記下來還蠻重要的
(3) 在參數設定上，learning rate 通常在我們訓練的 neural network 扮演蠻重要的角色，尤其是在 GAN 中，因為 discriminator 和 generator 兩者是息息相關的，需要互相配合，所以在調整 learning rate 要小心，比如我在 optimizer 使用 Adam 的情況下 discriminator 和 generator 的 learning rate 是使用一樣為 0.0002，但在 RMSprop 的情況下 discriminator 的 learning rate 則設成 generator 的 0.5 倍會比較適合。
(4) 我在此次訓練結果中還有一定程度的進步空間，當前是更新 discriminator 一次即更新 generator 一次，但根據文獻與理論，理想上更新 discriminator 多次一點再更新 generator 一次會比較好，這樣可以使我們訓練的比較穩定。

二、 Deep Q Network (DQN)

1. Please indicate the code paragraph about the updating based on the temporal difference learning in your implementation or from the given source code

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)).$$

```
def select_action(self, state):
    self.interaction_steps += 1
        #隨著 interaction step 增加會漸漸遞減 ϵ
    self.epsilon = self.EPS_END + np.maximum( (self.EPS_START-
self.EPS_END) * (1 - self.interaction_steps/self.EPS_DECAY), 0)
        #有 ϵ 的機率會選擇 random agent 隨機抽出 NOOP,UP, DOWN 其中一個 action
    if random.random() < self.epsilon:
        return torch.tensor([[random.randrange(self.action_dim)]], device=device, dtype=torch.lon
g)
        #有 1-ϵ 的機率選擇能夠最大化 Q-value 的那個 action
    else:
        with torch.no_grad():
            return self.policy_net(state).max(1)[1].view(1, 1)
#用能最大化 Q-value 的那個 action 計算 Q(st+1; at)
next_state_values = torch.zeros(self.BATCH_SIZE,1, device=device)
next_state_values[final_mask.bitwise_not()] = self.target_net(non_final_next_states).max(1, Tr
ue)[0].detach()
#這裡 α 設定為 1，更新 Q(st; at)
expected_state_action_values = (next_state_values * self.GAMMA) + reward_batch
```

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: 決定每次更新要更新的比例有多少，通常是一個介於0~1的數，如果越接近0代表學習速度越慢，如果越接近1代表學習速度越快。這邊設定為1，因為我們希望可以讓network更新全部學到的比例
- discount factor γ: discount factor是乘在Q(s_{t+1}; a_t)之前的權重，當action對於未來某個動作期望越大，理論上我們應該會給它一個比較大的reward，所以對時間發生越早的reward給予一個小於1的權重，但通常也不會設成太小，通常discount factor設為0.8~0.99

- target network update period τ: 在訓練DQN的過程中有兩個Q network，一個是我們真正用來訓練一直在更新參數的policy network(Q)，一個是我們最為計算MSE的目標target network($\hat{Q}$)，update period τ是指每隔多久時間要將policy network(Q)的參數θ更新到target network($\hat{Q}$)的參數θ^-，在計算MSE時如果我們的target一直在改變，這樣會造成訓練過程變得很不穩定，所以才會設計成隔一段時間再更新。這裡設定update period τ為10000。
- ϵ-greedy policy: agent會根據ϵ-greedy policy去做決策，我們現有的policy是讓agent去選擇能夠最大化Q的那個action(即exploitation)，另一方面我們希望agent能夠大膽嘗試新的policy(即exploration)，ϵ即是我們設定用來作出不同選擇的機率，使agent有ϵ的機率選擇exploration，有1-ϵ的機率選擇exploitation。這裡設定ϵ初始為1，逐漸遞減至0.1，最後我們不設定成0是因為仍然希望network可以有一些隨機的嘗試去做exploration。

2. For deep Q-Learning, exploring the environment is an important procedure. Because the reward signals in the given environment is very sparse, it takes a lot of computation time to explore the state space and perform the updating. To speed up the training process, you can simply change the probability of random agent: [NOOP (0.3), UP (0.6), DOWN (0.1)]. Please show the total reward of sample episodes for this configuration.

   用修改過的random agent去作選擇，每次抽出一個action，原地不動的機率為0.3，往前的機率為0.6，後退的機率為0.1。在過馬路的過程中，往前越多可以讓我們越容易抵達終點，但因為會遇到某些情況是只能原地不動或後退，所以還是要保留一部份的機率讓小雞原地不動或後退。在這樣的設計下，DQN一開始就能得到平均12左右的reward。

```
Episode:      0, interaction_steps:   2048, reward: 11, 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: 11, epsilon: 0.994470
Episode:      3, interaction_steps:   8192, reward: 11, epsilon: 0.992627
Episode:      4, interaction_steps:  10240, reward: 12, epsilon: 0.990784
Episode:      5, interaction_steps:  12288, reward: 11, epsilon: 0.988941
Episode:      6, interaction_steps:  14336, reward: 12, epsilon: 0.987098
Episode:      7, interaction_steps:  16384, reward: 14, epsilon: 0.985254
Episode:      8, interaction_steps:  18432, reward: 12, epsilon: 0.983411
Episode:      9, interaction_steps:  20480, reward: 12, epsilon: 0.981568
Episode:     10, interaction_steps:  22528, reward: 12, epsilon: 0.979725
Evaluation: True, Episode:     10, Interaction_steps:  22528, evaluate reward: 0.000000
```

   訓練到最後平均大概可以得到30~32的reward

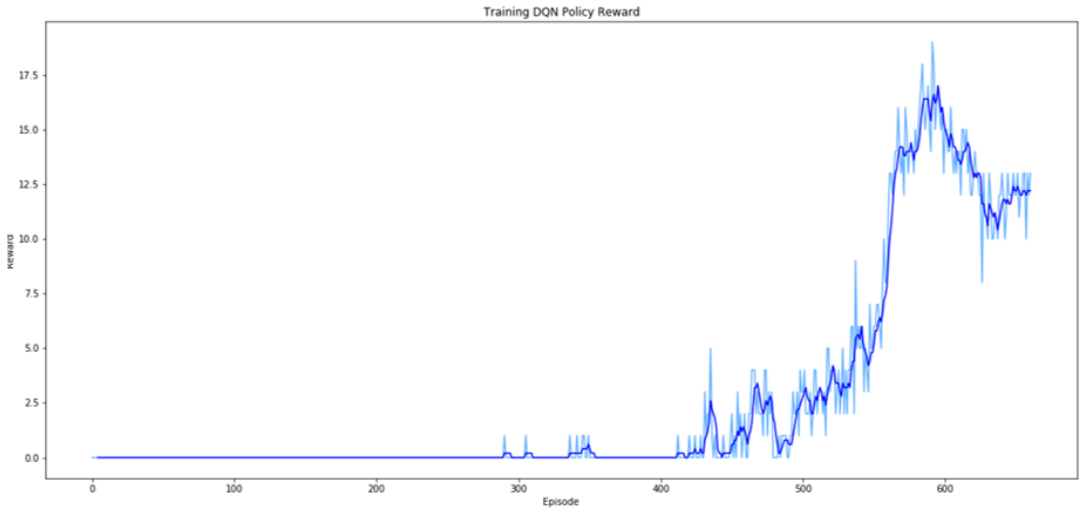
```
Evaluation: True, Episode:    790, Interaction_steps: 1619968, evaluate reward: 31.600000
Episode:    791, interaction_steps: 1622016, reward: 31, epsilon: 0.100000
Episode:    792, interaction_steps: 1624064, reward: 32, epsilon: 0.100000
Episode:    793, interaction_steps: 1626112, reward: 32, epsilon: 0.100000
Episode:    794, interaction_steps: 1628160, reward: 32, epsilon: 0.100000
Episode:    795, interaction_steps: 1630208, reward: 30, epsilon: 0.100000
Episode:    796, interaction_steps: 1632256, reward: 30, epsilon: 0.100000
Episode:    797, interaction_steps: 1634304, reward: 31, epsilon: 0.100000
Episode:    798, interaction_steps: 1636352, reward: 31, epsilon: 0.100000
Episode:    799, interaction_steps: 1638400, reward: 30, epsilon: 0.100000
```

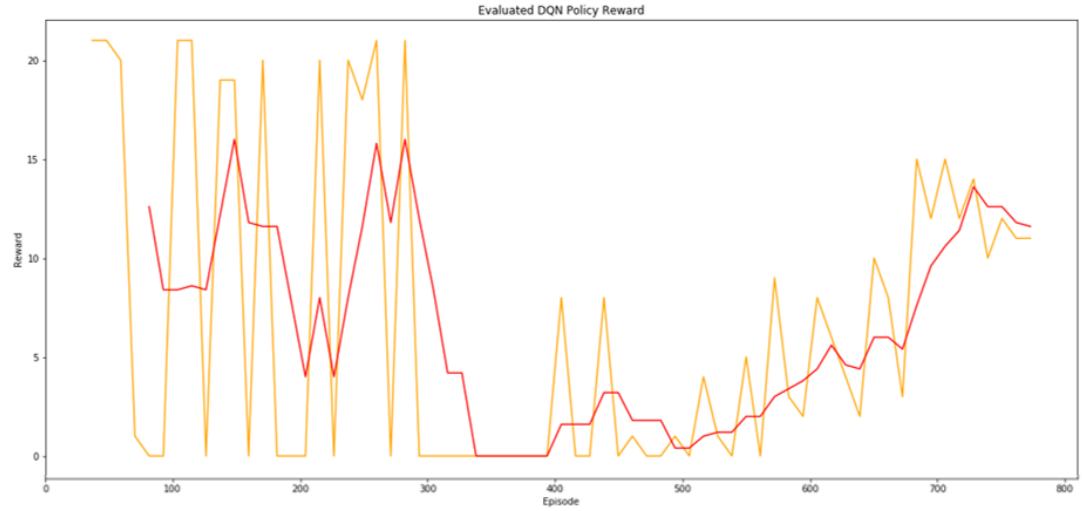
3. Use the modified random agent in the ϵ-greedy and keep training. Here, you should tune hyperparameters to let model converge. Plot the episode reward in learning time and evaluation time (ϵ = final epsilon) (2 charts). Show your configuration and discuss what you find in training phase.

   [uniform random agent]

   Learning time: 以下是用uniform random的agent去學習的情況，隨著ϵ從1慢慢減少到0.1，total reward會先維持在0的情況，要等到有reward要花很長一段時間，大約到第400個Episode才開始有比較多非0的reward。這是因為ϵ-greedy的策略會使得我們有ϵ的機率用uniform agent去作更新，初期ϵ大，會有ϵ的機率是從停止不動、往前、退後這三種action隨機選取，但從我們玩遊戲的直觀想法去思考的話這其實不太合理，因為理想上往前的機率應該要大一些、 才能讓我們比較容易順利抵達馬路終點。從圖可以發現到即使學習到第800個epsisode還沒收斂，可能需要訓練更多的時間。

   Evaluation time: 這裡看的是network訓練好以後獲得reward可能的情況，用的是target network的greedy決策，也就是ϵ=0.1的情況，一開始reward震盪非常大，表示DQN其實還沒有被訓練好，偶爾得到比較大的reward可能是剛好運氣好

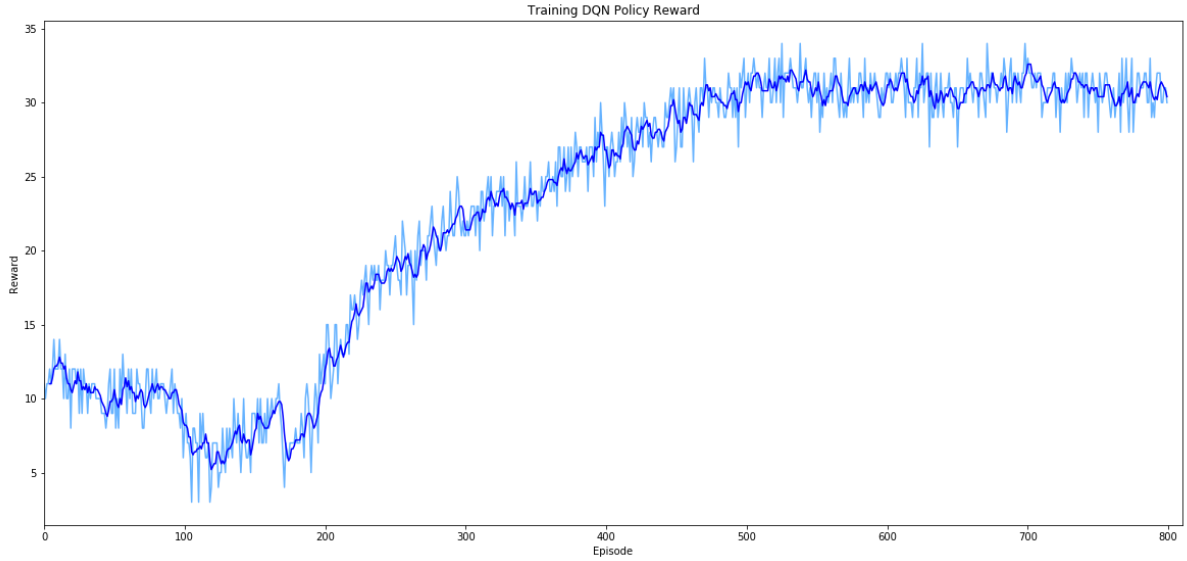


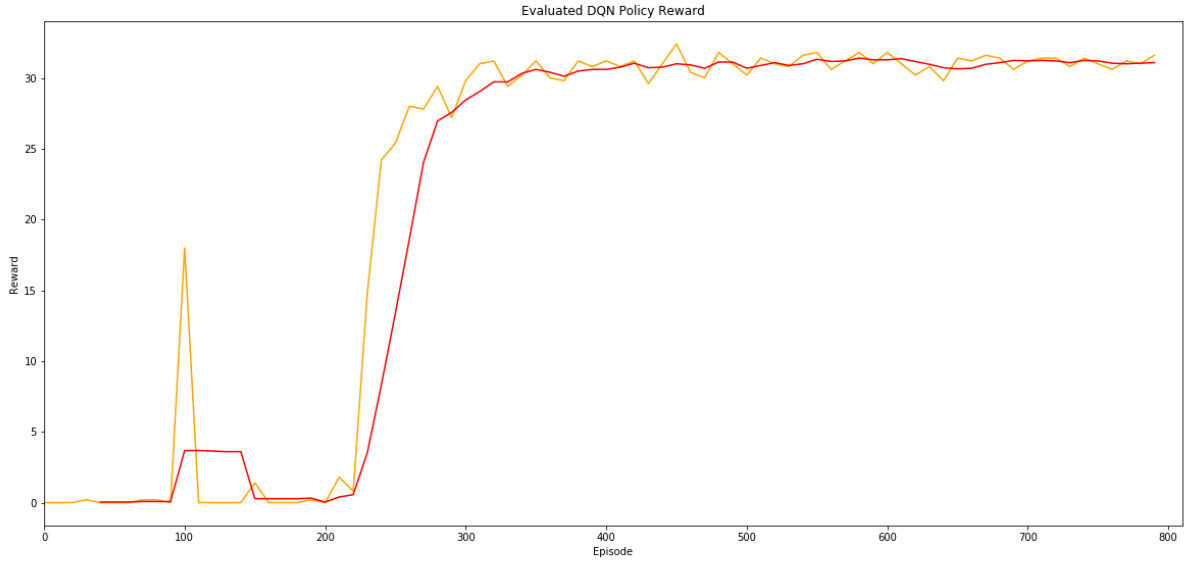


[modified random agent]

Learning time: 為了讓訓練速度加快，我們會把uniform的agent改為modified random agent，即原地不動的機率為0.3，往前的機率為0.6，後退的機率為0.1，這樣可以讓一開始的reward就直接得到10~15左右，在100~200個episode的reward稍微往下降，我推測可能是DQN在學習時遇到了撞牆期，因為**車子原本是像右行的，過了黃線之後變成像左行，原本學得還不錯的策略突然不管用了**，所以需要重新調整，大概過了第200個episode後reward就幾乎又是往上的趨勢，到最後趨勢趨於平緩，表示我們的DQN差不多收斂了

Evaluation time: 這裡可以看出來大致上和learning time的圖有差不多的趨勢，從第200個episode開始上升到最後趨於平緩，比較有趣的是在第200個episode以前大部分皆為0，對照上面uniform agent，相比之下，反而是初期用uniform agent效果比較好，我推測是因為往前走的比例太高了，造成在遊戲中常常撞車無法抵達對面，但也因為迅速大量累積這些失敗的經驗，使得使用modified random agent的DQN可以快速從失敗中學習，以致於在後期比用uniform random agent的DQN還要快收斂。





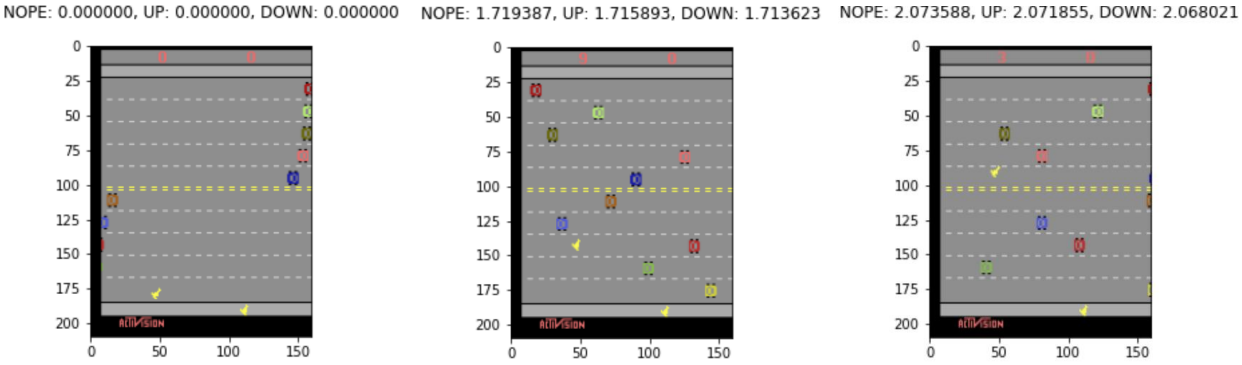
4. After training, you will obtain the model parameters for the agent. Show total reward in some episodes for deep Q-network agent.
   用模型訓練好的參數實際跟環境互動，reward平均大概落在28~32的範圍內，與paper平均結果30.3差不多，可以確認我們的模型訓練應該是有達到收斂狀態。

```
[Info] Restore model from './model/q_target_checkpoint_1538048.pth' !
Episode:      0, interaction_steps:      0, reward: 32, epsilon: 0.100000
Episode:      1, interaction_steps:      0, reward: 29, epsilon: 0.100000
Episode:      2, interaction_steps:      0, reward: 33, epsilon: 0.100000
Episode:      3, interaction_steps:      0, reward: 31, epsilon: 0.100000
Episode:      4, interaction_steps:      0, reward: 32, epsilon: 0.100000
Episode:      5, interaction_steps:      0, reward: 30, epsilon: 0.100000
Episode:      6, interaction_steps:      0, reward: 31, epsilon: 0.100000
Episode:      7, interaction_steps:      0, reward: 32, epsilon: 0.100000
Episode:      8, interaction_steps:      0, reward: 28, epsilon: 0.100000
Episode:      9, interaction_steps:      0, reward: 32, epsilon: 0.100000
```

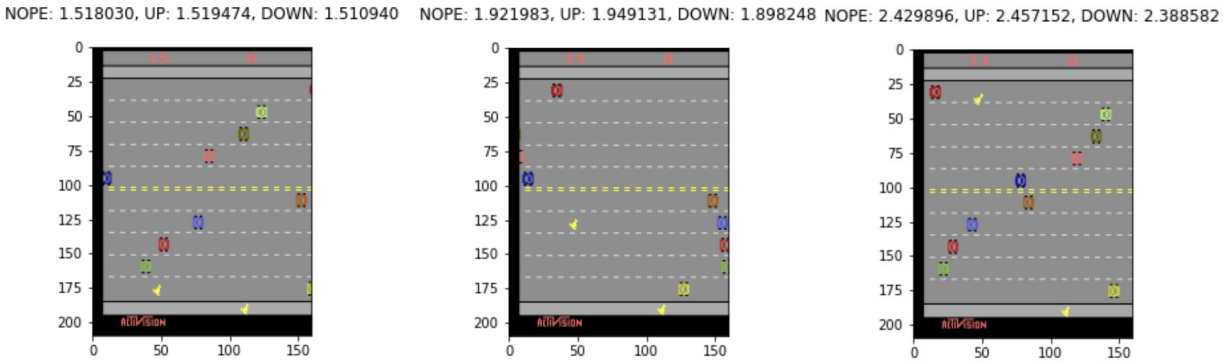
5. Sample some states, show the Q values for each action, analyze the results, and answer:

以下分別擷取NOOP, UP, DOWN各三張圖：

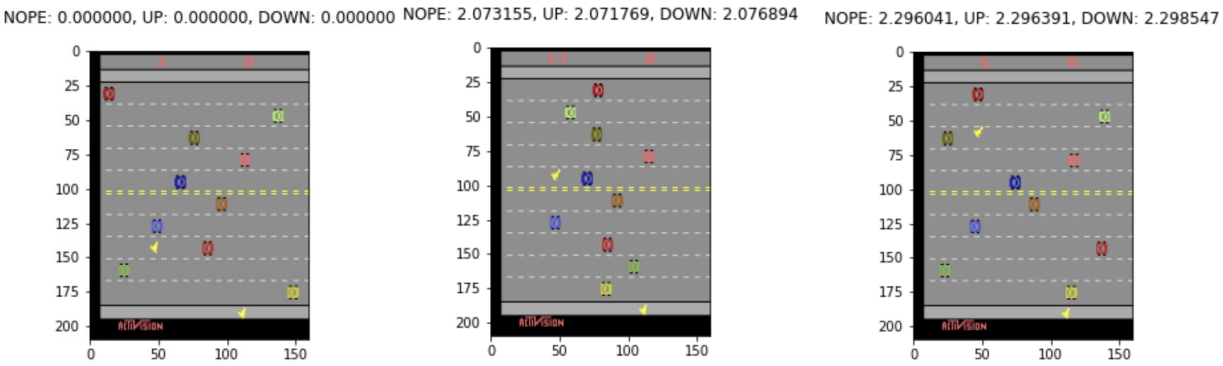
NOOP



UP



DOWN



- Is DQN decision in the game the same as yours? Any good or bad move?
  我的選擇和DQN的選擇大致上相同，除了上面NOOP的第一張我會選擇往前，因為看起來還有往前的空間；UP的第一張我會選擇原地不動以避免被撞；DOWN的第一張我會選擇原地不動以避免被撞，但也有可能DQN是因為考量到後面的state才作出這樣的選擇。另外我覺得DQN在DOWN的第二張表現得很好，它已經學習到在黃線之上的車會往左，黃線之下的車會往右，所以在這個state下，後退是最好的選擇

- Why the averaged Q-value of three actions in some state is larger or less than those of the other states?
  得到比較大的平均Q-value可能是因為小雞已經跑到比較靠近終點的地方，如UP和DOWN的第三張，或是環境中出現很少車，黃線以下的車幾乎都已經向右離去，黃線以上的車幾乎都已經向左離去，如UP的第二張，這樣就會越傾向得到越大的reward；而一開始DQN還沒有很多經驗時，或是小雞還站在離起點很靠近的地方就會得到比較小的平均Q-value。