Deep Learning Homework 3

0853412 吳宛儒

1 Generative adversarial network (GAN)

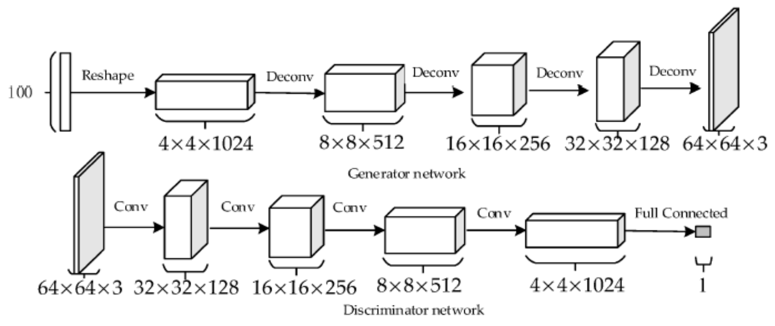
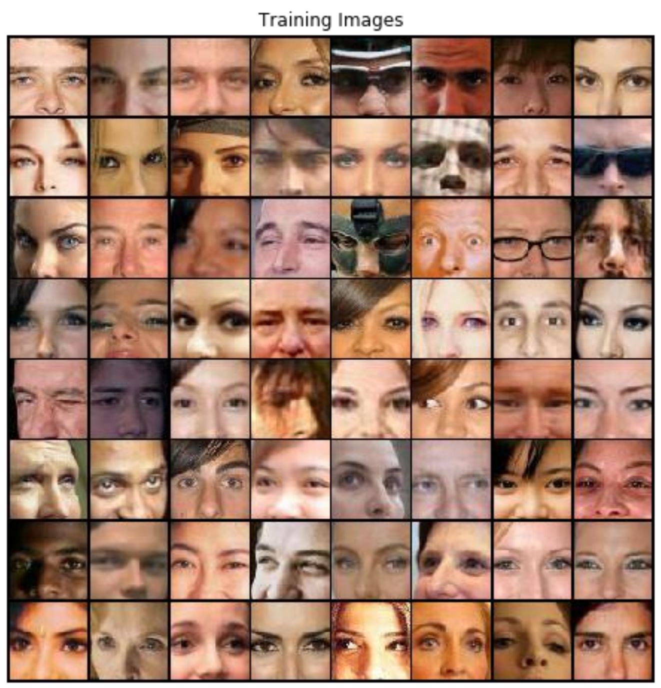
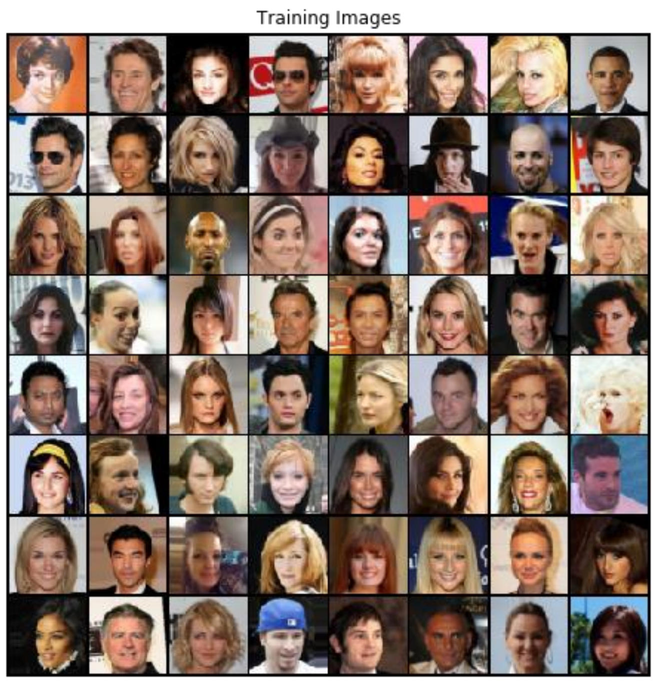
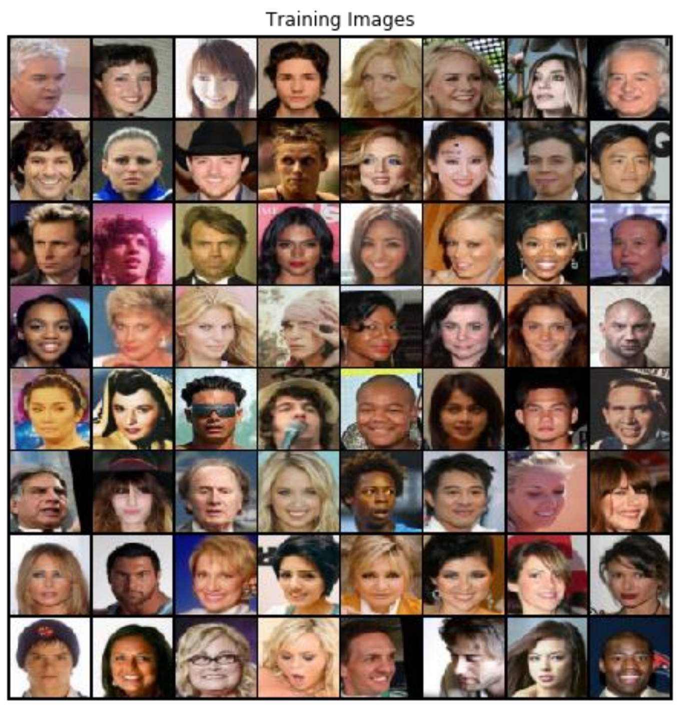
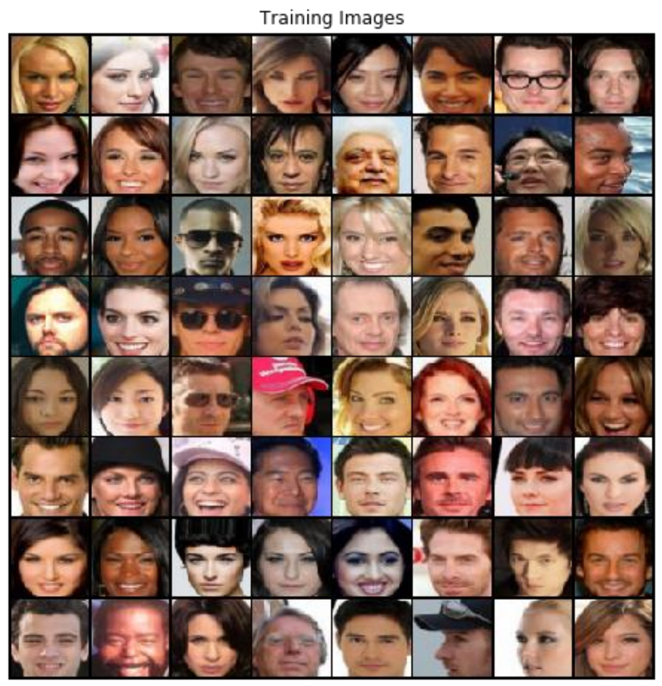


Figure 1: Structure of DCGAN

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.

在圖片前處理的部分，主要參考網路上的文章：[PyTorch 學習筆記（三）：transforms 的二十二個方法]。圖片的前處理大致上可以分成裁剪、翻轉、旋轉和圖像變化等等。針對模型任務，前處理著重在裁剪上，因為發現資料集中的圖片雖然多樣，但其人臉幾乎集中在畫面中間，因此在裁剪上選擇使用 CenterCrop 的方式。然而直接套用 CenterCrop 會發現人臉被切半，多集中在鼻子以上的部分，因此前面需要先對圖片做 Resize。Resize 的部分則嘗試了 img_size(設定 img_size=64)、img_size+15 以及 img_size+45 三種作法。當針對 img_size 去作 CenterCrop 時，會發現其包含在畫面內的背景非常多樣，然而背景並不是我們的模型需要費心學習的部分，因此需進一步裁切。

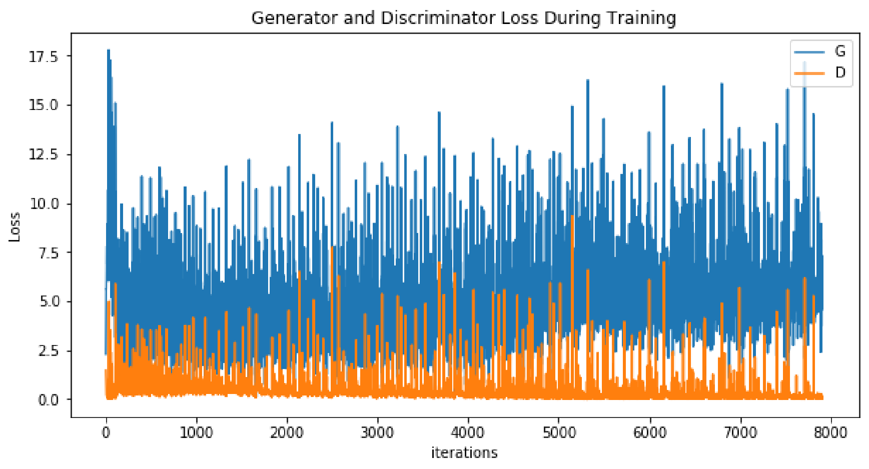
不做 Resize 直接 CenterCrop	**使用 img_size 做 Resize**
	
人臉被切到只剩下上半部	背景多樣
使用 img_size + 15 做 Resize	**使用 img_size + 45 做 Resize**

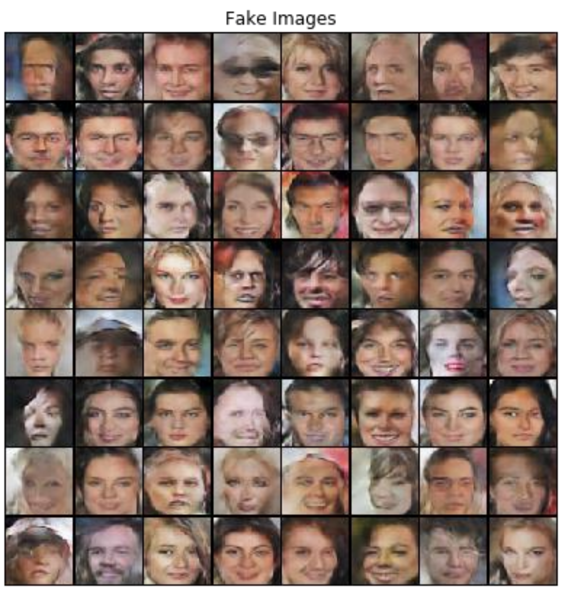
	
比較專注於人臉，但仍有部分背景成分	讓整個畫面幾乎只剩人臉

若先做 resize 到 79*79(64+15=79)可以發現 CenterCrop 後背景占比的部分減少許多，比較專注於人臉。而若是先做 resize 到 109*109(64+45=109)再 CenterCrop 後幾乎只剩下人臉，也是模型最主要學習目標，最後決定 resize 成 img_size+45 再進行 CenterCrop。除了對圖片的裁剪以外，也在 transform 的時候對圖片做 normalize 處理，讓其值在三個維度上皆是 mean 為 0.5，std 為 0.5。

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.

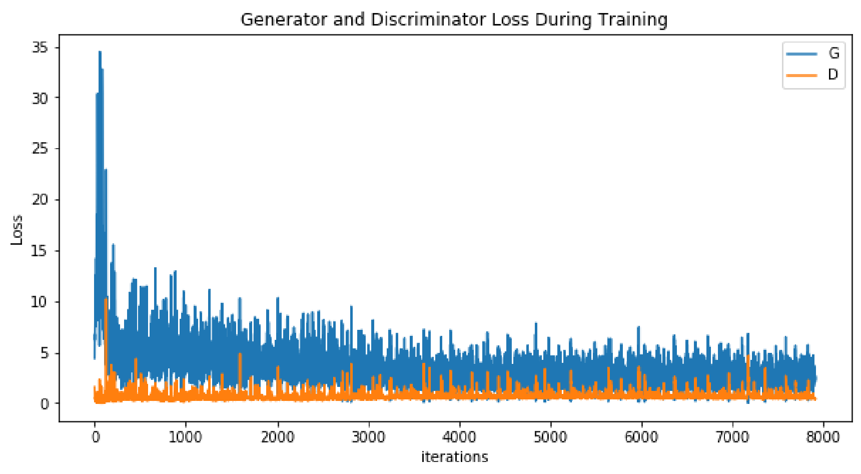
$$\begin{aligned}\max_{D}\ \mathcal{L}(D) &= \mathbb{E}_{x\sim p_{\text{data}}}\log D(x)+\mathbb{E}_{z\sim p_z}\log(1-D(G(z)))\\ \min_{G}\ \mathcal{L}(G) &= \mathbb{E}_{z\sim p_z}\log(1-D(G(z)))\end{aligned}$$

[實驗一]：當generator和discriminator之初始weight皆設為0(最初始的code)	
Loss record of Generator and Discriminator	**觀察**
	一直到最後的iteration仍非常震盪，generator和discriminator有generator稍微較弱(loss較大)的感覺。兩者相互拮抗並未達到平衡。圖片生成上，有些人臉輪廓跑掉或是沒有輪廓。
Real images and fake images	



[實驗二]：當generator和discriminator之初始weight為隨機數字

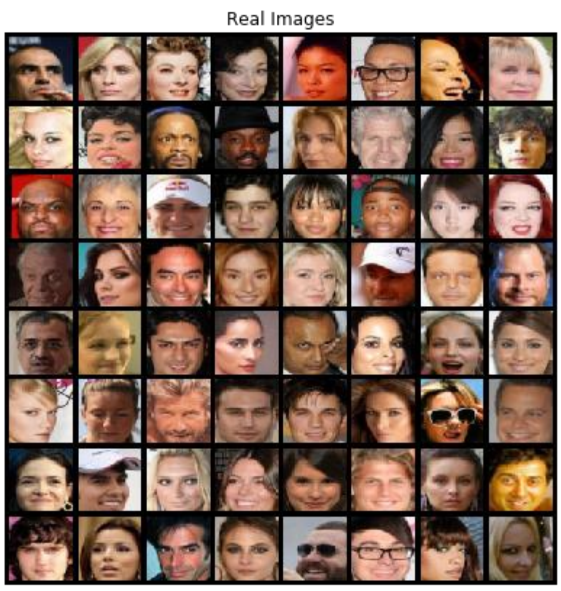
Loss record of Generator and Discriminator

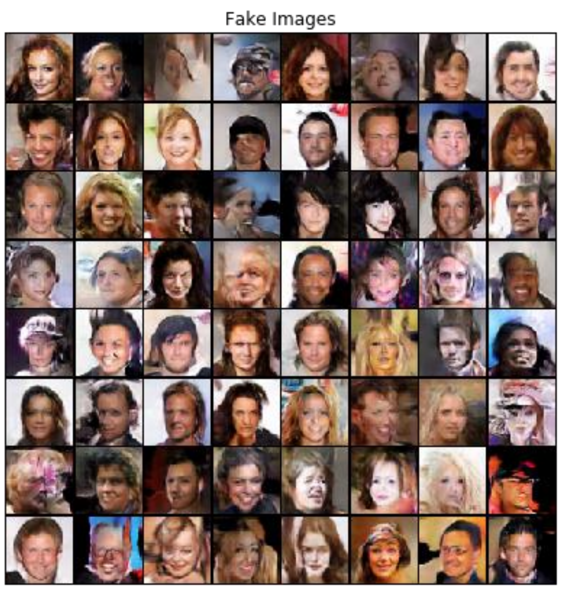


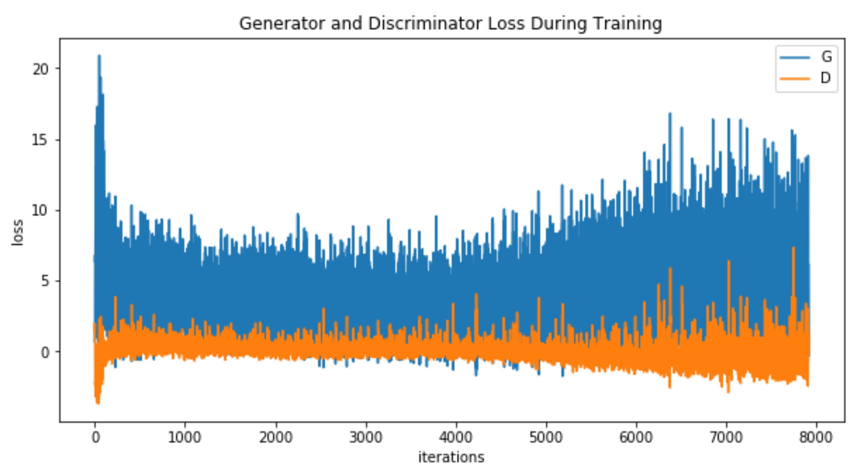
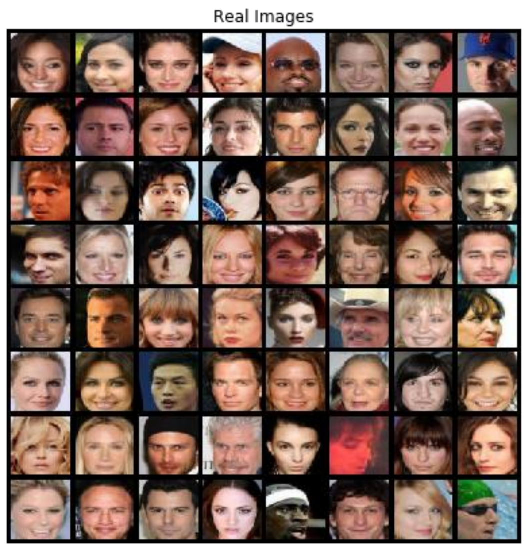
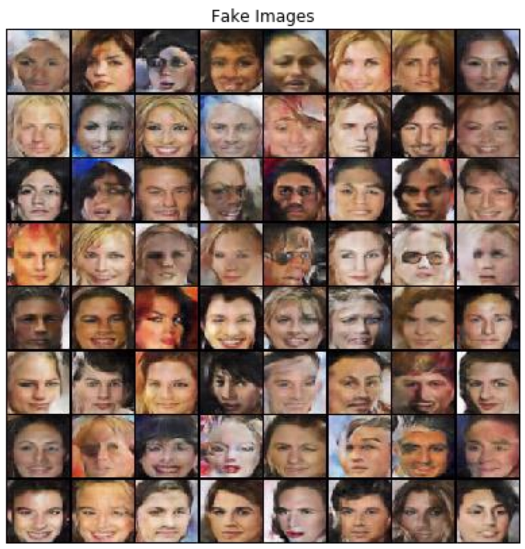
觀察

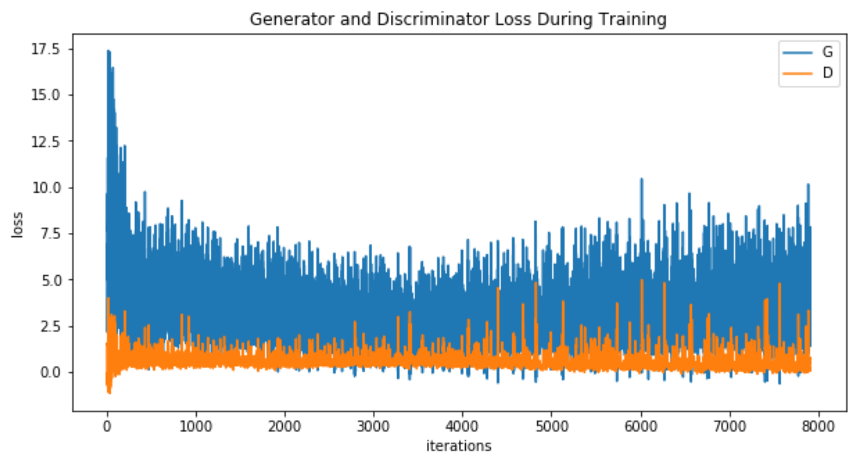
收斂速度較實驗一來得快，收斂效果也比較好。不會像實驗一如此震盪。圖片生成上，相比於實驗一，在輪廓上生成大部分有改善，然而仍有色塊不平均、五官些微扭曲的情況。

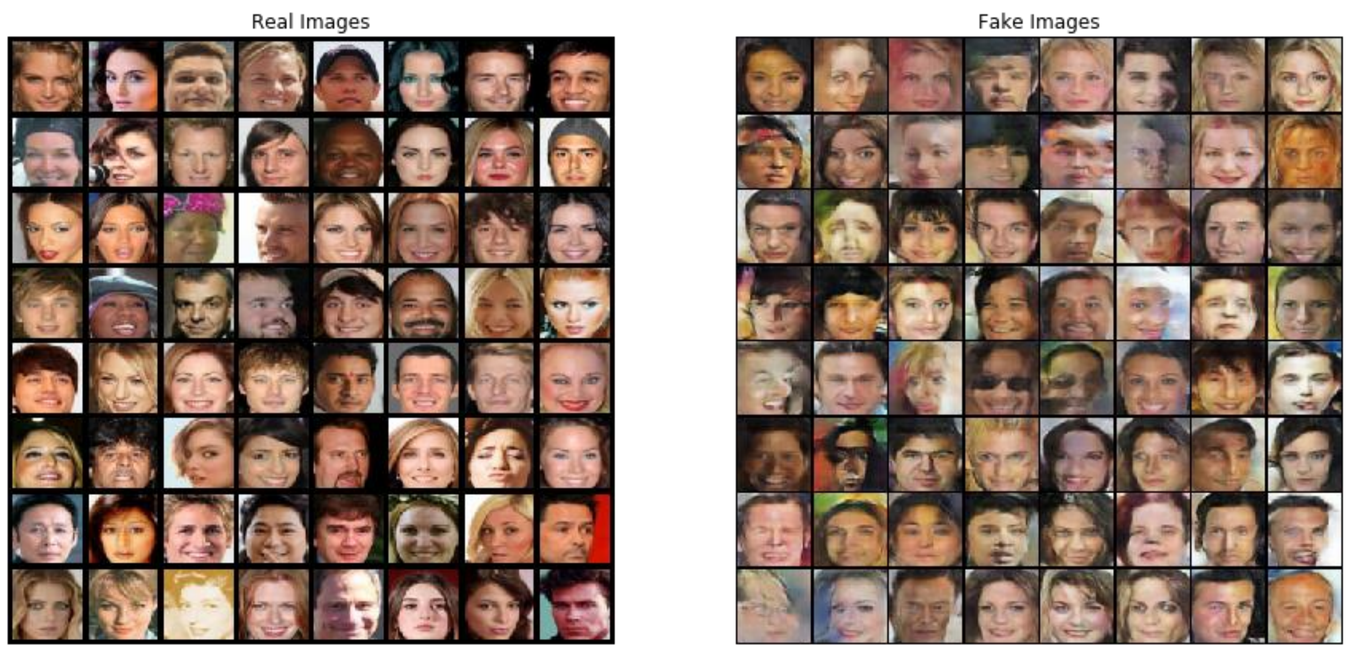
Real images and fake images

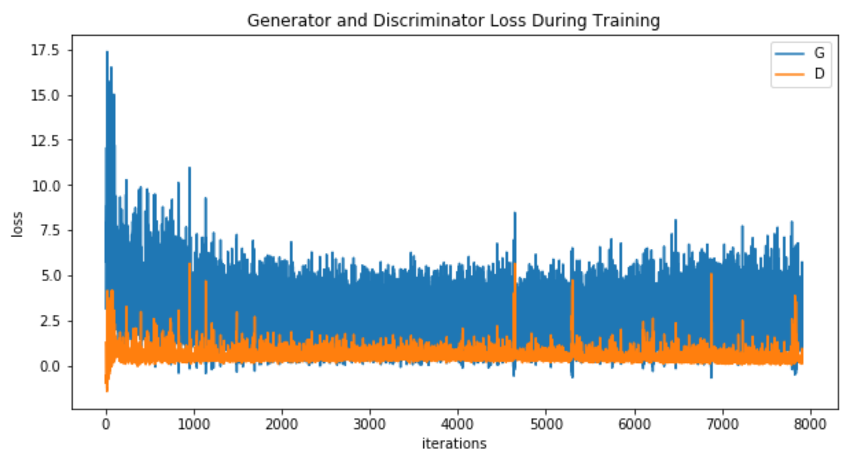




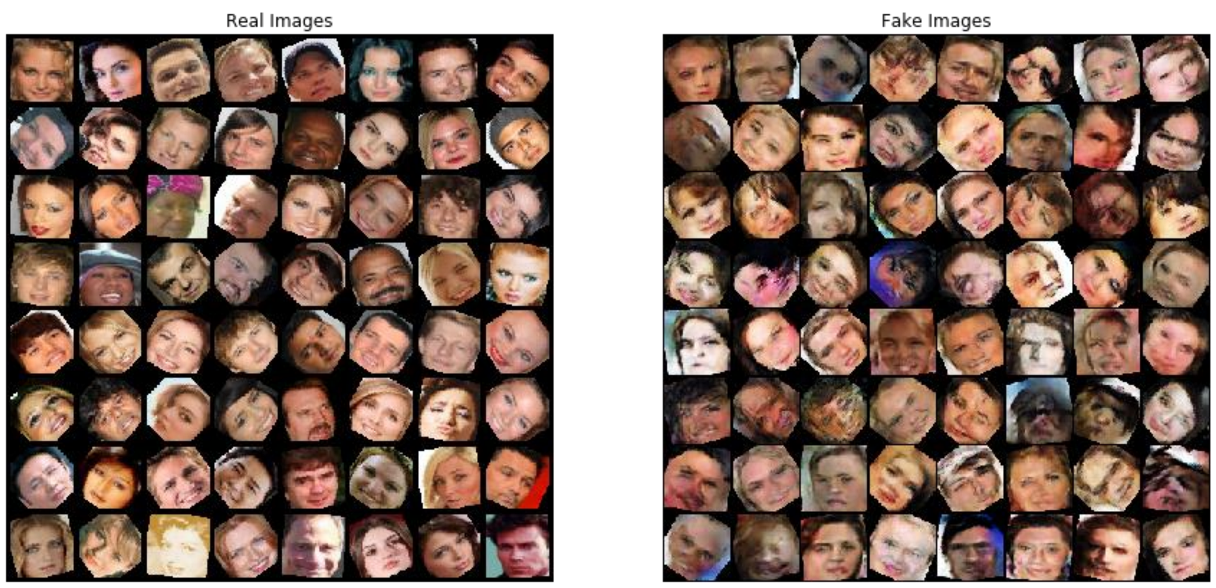
[實驗三]：real和fake的label改為soft label version 1	
Loss record of Generator and Discriminator	**觀察**
	把原本real=1, fake=0的label轉換為soft label，這邊使用real label = 0.9~1.5之間，fake label=0.0~0.5之間。發現雖然生成圖片效果普通，但相比實驗二，其loss到最後有逐漸增大的趨勢，G和D相互拮抗到最後D比較強。想嘗試把soft label的範圍調小一些。
Real images and fake images	
	

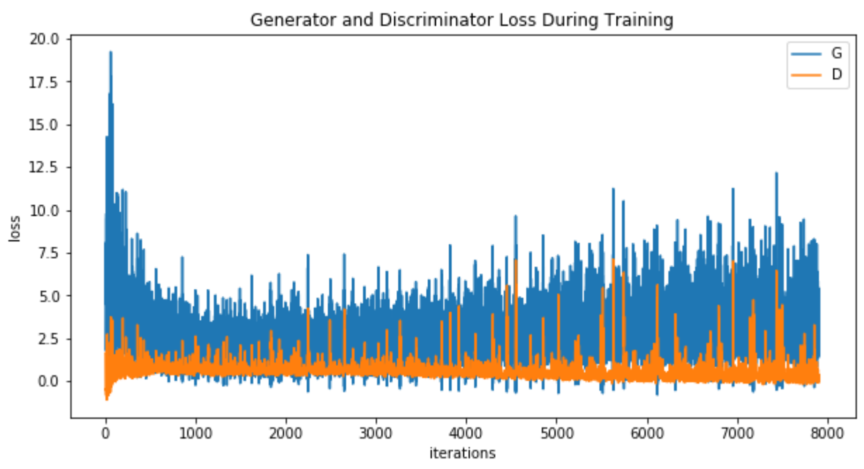
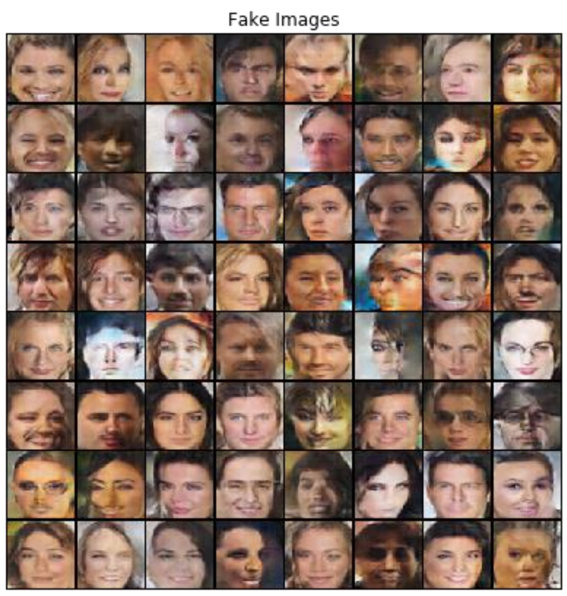
[實驗四]：real和fake的label改為soft label version 2	
Loss record of Generator and Discriminator	**觀察**
	震盪的情形看起來比實驗三還要小一些，但是後期也有稍微增強的趨勢。在生成圖片的部分，則出現了比較多偏白的且輪廓不清楚的人臉，估計也和label設定的值域相關。
Real images and fake images	

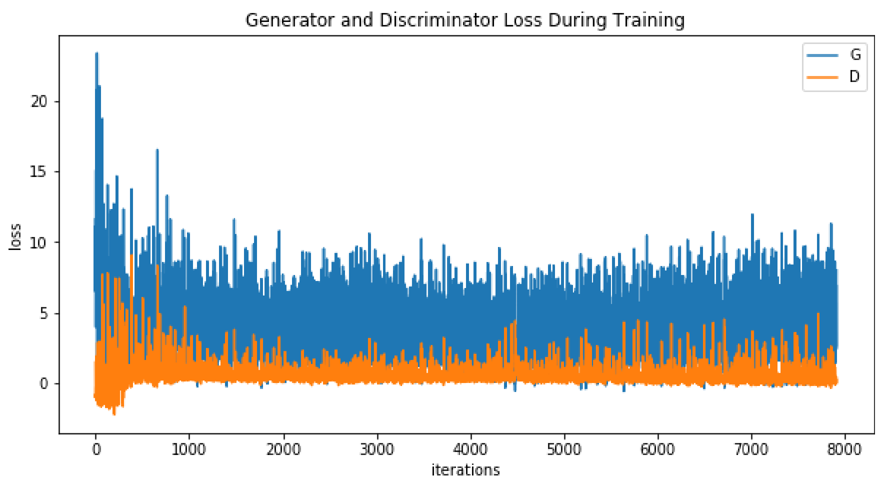


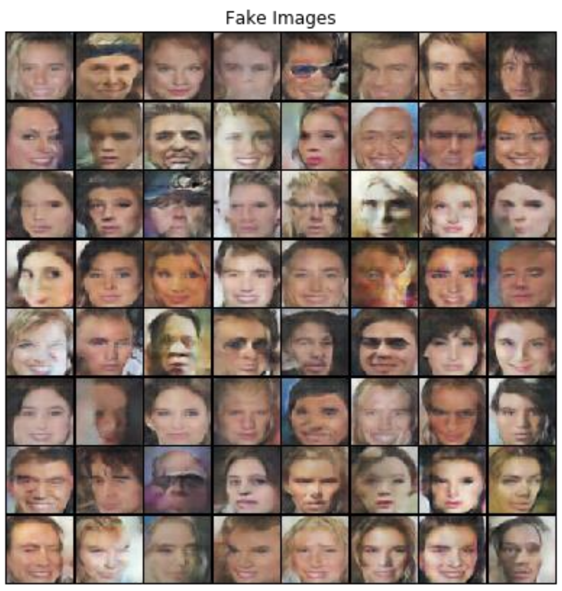
[實驗五]：更改transform讓圖片在-60~60度間隨機旋轉	
Loss record of Generator and Discriminator	**觀察**
	這部分本來預期在不複雜化模型的前提下，應該是比較難train到比較好的結果。上圖為training data的模樣。可以看到一開始loss就能比較低，但到後期G的loss的震盪仍偏大，推測應該就是和先前的transform有關，G比較難產圖，下面的圖也相應證：效果並沒有前面幾次來得好。

Real images and fake images



[實驗六]：更改模型—更改generator的activation function	
Loss record of Generator and Discriminator	**觀察**
	這部分將G的activation function由ReLU改成同D的LeakyReLU，然而其震盪仍大，生成的圖片雖然輪廓鰻清晰，但也有色塊不均勻的情況，相比於原本未更動前的版本，這個版本的收斂程度比較差。因此推測還是保持原本的activation function才會有比較好的模型效果。
Real images and fake images	
	

[實驗七]：更改模型learning rate 讓D的lr比較大	
Loss record of Generator and Discriminator	**觀察**
	有參考網路上提供的建議，雖然G&D兩者應該相互平衡，但一般會希望D略強一些，因此調動D的lr到0.0004。相比於實驗二的結果，雖然模型看起來有在逐漸收斂，但是效果不如實驗二，且生成的圖片有霧感，輪廓也不如實驗二清晰。

Real images and fake images


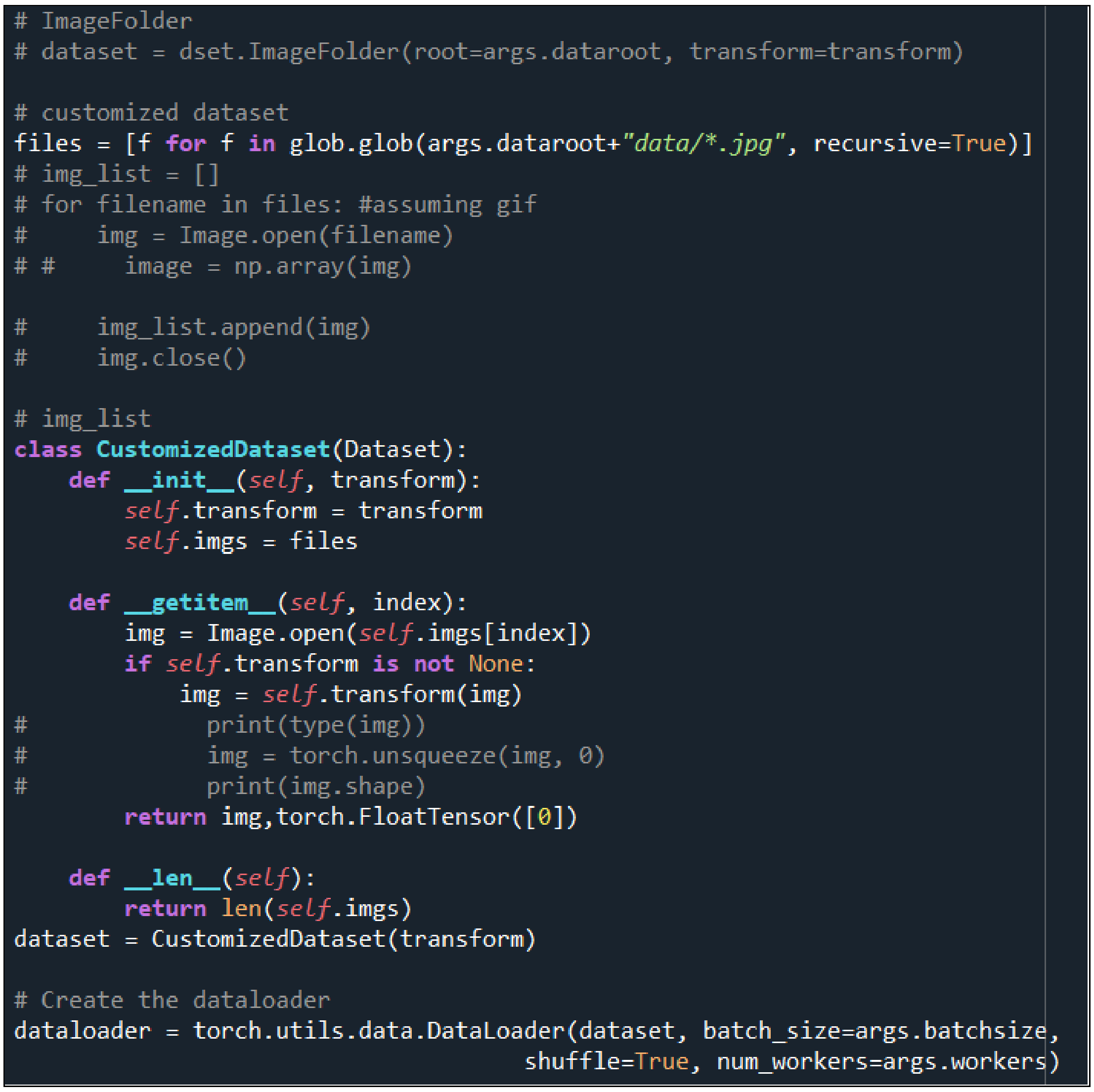
3. Implementation details are addressed as follows

(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)

更改模型中的activation function如上述的實驗六。主要是因為之前經驗是LeakyReLU效果普遍比ReLU還要好，但是助教提供原始的model裡面generator是使用ReLU作為activation function，因此想改成和discriminator一樣是LeakyReLU並比較效果。原本也有想要更改CNN的層數、filter數或是stride大小等等，但查了許多網路上的資料，DCGAN最好效果幾乎都是用這一組參數與架構，因此這部分沒有多做更動。實驗七則是**更改learning rate**觀察變化。

(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)

自己寫dataset：原本是使用ImageFolder來直接作為dataset，後來使用自定義的dataset來讀取圖片資料集。首先先將資料夾底下所有圖片的路徑皆存在files這個list底下，接著使用getitem的方式來取得圖片。由於是第一次寫還不太熟悉，中間過程遇到蠻多困難，例如究竟要在init就開好圖片還是要在getitem才開好圖片？如果一開始在init就開好(而非只是給路徑)，就會遇到無法對closed image操作的問題；另外也有遇到像是getitem要return的type以及資料要是什麼，type的部分在Image、np array以及tensor上兜轉了許久才確定是回傳tensor，而回傳資料的部分則是一開始只有寫return img的時候出來的shape會錯誤，後來才發現是需要return兩個資料，一個是data一個是label，納因為我們模型不需要label，雖然不會用到但還是需要有個東西去接它，才不會使得dataloader出來的東西出錯，如果只有return img的話，dataloader出來只會有一張圖片，而不是batchsize的數量的圖片。



```python
# ImageFolder
# dataset = dset.ImageFolder(root=args.dataroot, transform=transform)

# customized dataset
files = [f for f in glob.glob(args.dataroot+"data/*.jpg", recursive=True)]
# img_list = []
# for filename in files: #assuming gif
#     img = Image.open(filename)
# #     image = np.array(img)

#     img_list.append(img)
#     img.close()

# img_list
class CustomizedDataset(Dataset):
    def __init__(self, transform):
        self.transform = transform
        self.imgs = files

    def __getitem__(self, index):
        img = Image.open(self.imgs[index])
        if self.transform is not None:
            img = self.transform(img)
#             print(type(img))
#             img = torch.unsqueeze(img, 0)
#             print(img.shape)
        return img,torch.FloatTensor([0])

    def __len__(self):
        return len(self.imgs)
dataset = CustomizedDataset(transform)

# Create the dataloader
dataloader = torch.utils.data.DataLoader(dataset, batch_size=args.batchsize,
                                         shuffle=True, num_workers=args.workers)
```

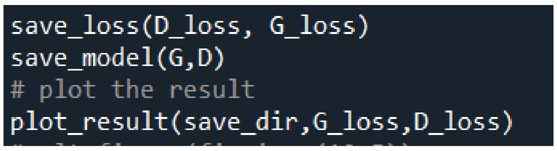
(c) In main.py, you have to complete three functions. main(), train(), and visualizaion.

- **main() : you have to set up dataset (dataloader), models, optimizers, and the criterion. After preparation, the train function is called to start the training procedure.**
- **train() : In every iteration, you have to perform the following jobs**
  - **send true data into the discriminator, and update the discriminator.**
  - **use generator to create fake data, and send them to the discriminator. Calculate loss for both models and update them.**
  - **record the loss in every iteration and draw some samples by the current generator after the fixed number of iterations.**
  - **after finishing all epochs, you have to save the files of your models, losses, and samples. (sampling should not be more than 20 times)**

(d) Visualization(If you don't use jupyter notebook, attach your visualization code in the folder and chart in report.)

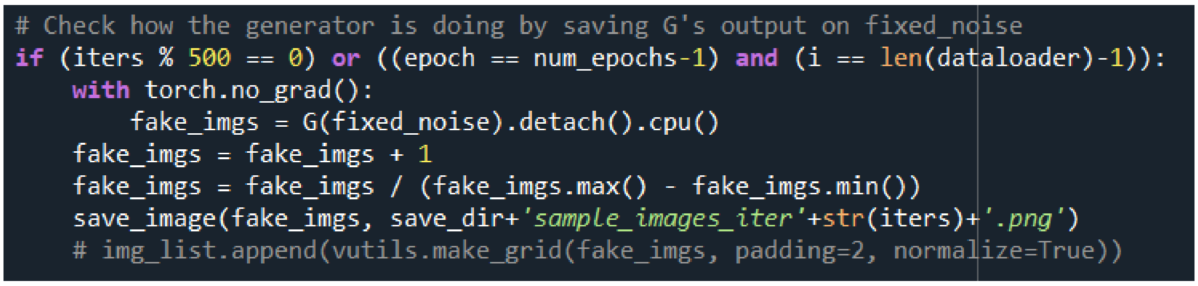
- **write down the visualization code in Visualization.ipynb, you can open it by using jupyter notebook.**

  這部分我在main和其引入的Visualization中的程式碼包含了所有visualization的code，並沒有使用到jupyter notebook。



```
save_loss(D_loss, G_loss)
save_model(G,D)
# plot the result
plot_result(save_dir,G_loss,D_loss)
```

這是在main當中，跑完所有的iterations以後將losses, models和loss的圖都存下來。



```
# Check how the generator is doing by saving G's output on fixed_noise
if (iters % 500 == 0) or ((epoch == num_epochs-1) and (i == len(dataloader)-1)):
    with torch.no_grad():
        fake_imgs = G(fixed_noise).detach().cpu()
    fake_imgs = fake_imgs + 1
    fake_imgs = fake_imgs / (fake_imgs.max() - fake_imgs.min())
    save_image(fake_imgs, save_dir+'sample_images_iter'+str(iters)+'.png')
    # img_list.append(vutils.make_grid(fake_imgs, padding=2, normalize=True))
```

而上面這部份則是每500個iterations會把生成的假的圖片存起來。

- **plot the following two charts and show them in the report**
  - **original samples and generated samples**
  - **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.**

(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.

DCGAN總結：

在這次GAN的實作中，發現GAN的training蠻困難的，先前僅有使用keras實作過簡單的GAN，而這次第一次使用Pytorch，也第一次訓練DCGAN。有可能因為原本的Model架構與參數都已經是多人經驗中最好的tuning，因此在前面七次實驗中，使用同樣的model架構下，似乎都沒辦法有效地提升模型訓練結果，往往都只會讓效果更差。也發現在改善圖片生成上，無法確定究竟更改哪個部分會對應到圖片生成的哪個部分？例如色塊不均、輪廓不明顯或是有霧感，都沒辦法確切找出究竟是模型中哪項因子影響到，或是說其實這些都是整體綜合下的結果，因此在訓練這種generative model的時候，評估生成圖片的好壞其實是相對困難的，之前也有看過有人試著將圖片生成之好壞量化，但詳細作法還要再去瞭解。而模型效能提升的部分應該還有待更進一步去survey近年有針對GAN上作improvement的paper，方能找到更有效的方法去改善實驗結果。

2 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

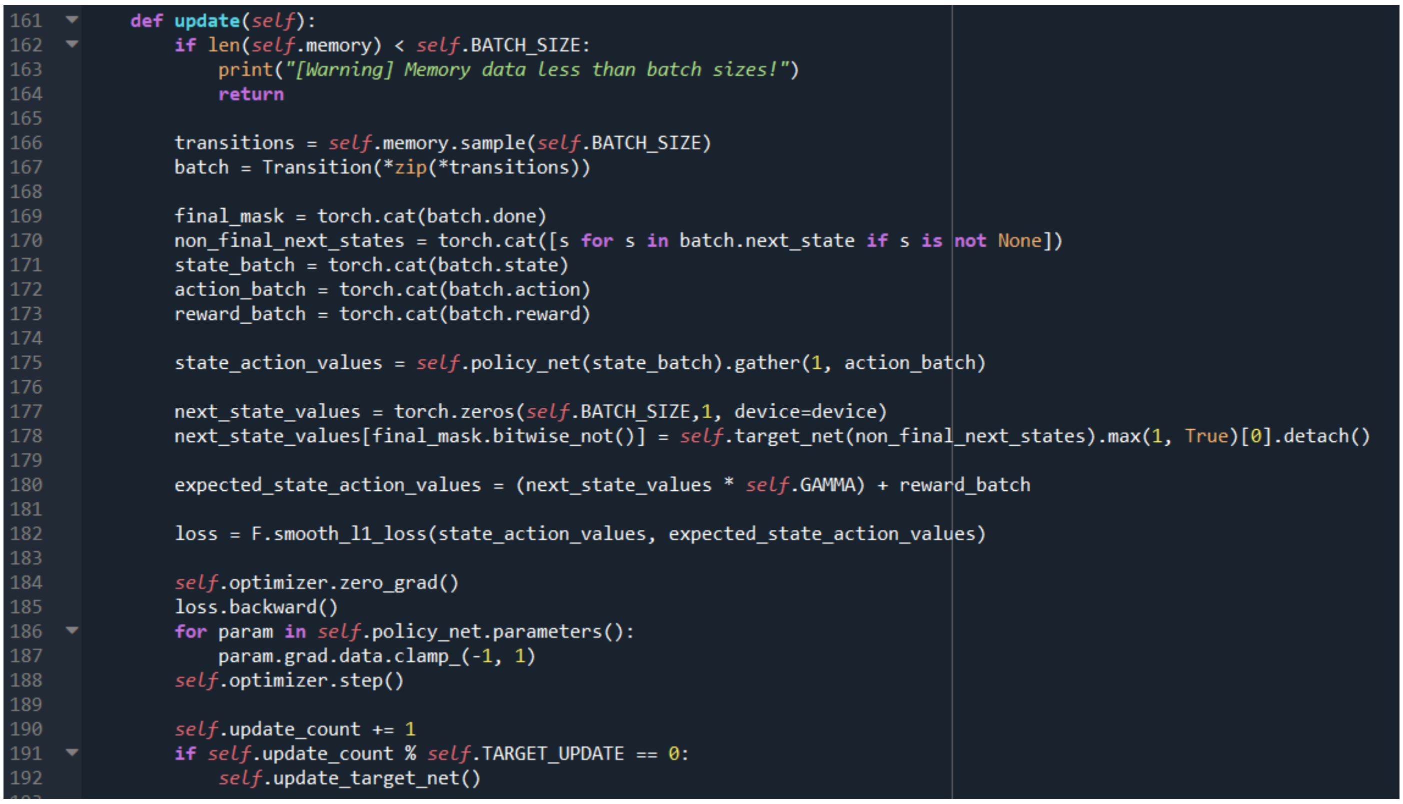
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \cdot \max_a Q(s_{t+1}, a_t) - Q(s_t, a_t)).$$

$$Q^*(s,a) = Q(s,a) + \alpha(\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{Target Q}} - Q(s,a))$$

$$L(w) = \mathbb{E}[(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{Target} - Q(s, a, w))^2]$$

目標是尋找最佳的Q，也就是Q*。可以把這樣的target Q視為一個label，在不斷更新的過程中，讓Q function可以逼近target Q function。在定義好這樣的loss以後就能進行NN訓練。

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$


```python
    def update(self):
        if len(self.memory) < self.BATCH_SIZE:
            print("[Warning] Memory data less than batch sizes!")
            return

        transitions = self.memory.sample(self.BATCH_SIZE)
        batch = Transition(*zip(*transitions))

        final_mask = torch.cat(batch.done)
        non_final_next_states = torch.cat([s for s in batch.next_state if s is not None])
        state_batch = torch.cat(batch.state)
        action_batch = torch.cat(batch.action)
        reward_batch = torch.cat(batch.reward)

        state_action_values = self.policy_net(state_batch).gather(1, action_batch)

        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, True)[0].detach()

        expected_state_action_values = (next_state_values * self.GAMMA) + reward_batch

        loss = F.smooth_l1_loss(state_action_values, expected_state_action_values)

        self.optimizer.zero_grad()
        loss.backward()
        for param in self.policy_net.parameters():
            param.grad.data.clamp_(-1, 1)
        self.optimizer.step()

        self.update_count += 1
        if self.update_count % self.TARGET_UPDATE == 0:
            self.update_target_net()
```

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

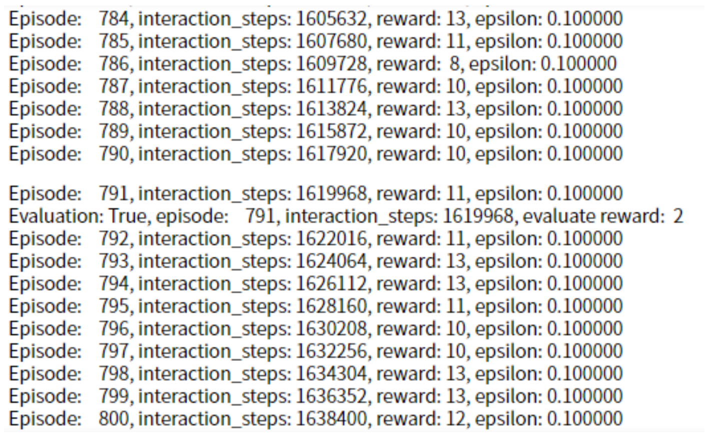
Updating step(α): 為0~1之間的值。α=0代表新的Q(s,a)會直接使用舊的Q(s,a)，表示沒有學習能力、Q完全不改變，而若α=1則是代表新的Q(s,a)會採取「當下獎勵值R(s,a)和後續新狀態之最大可能獎勵總和maxQ」來共同決定，而不考慮舊的Q(s,a)。

Discount factor(γ): 為0~1之間的值。通常是小於1的值。當下的反饋是相對重要的，當時間間隔越久，影響就該越小。

<u>Target network update period(τ):</u> DQN算法中的target跟policy network是分開的，policy network做一段時間以後複製到target network，Target network update period即是決定多久更新一次target network。

<u>ϵ-greedy policy(ϵ):</u> ϵ介於0~1之間，代表在決定下一個action的時候，有ϵ的機會會採用隨機的方式挑選動作，有(1- ϵ)的機率會挑選具有最大reward的動作。所以ϵ代表了隨機性的策略機率。

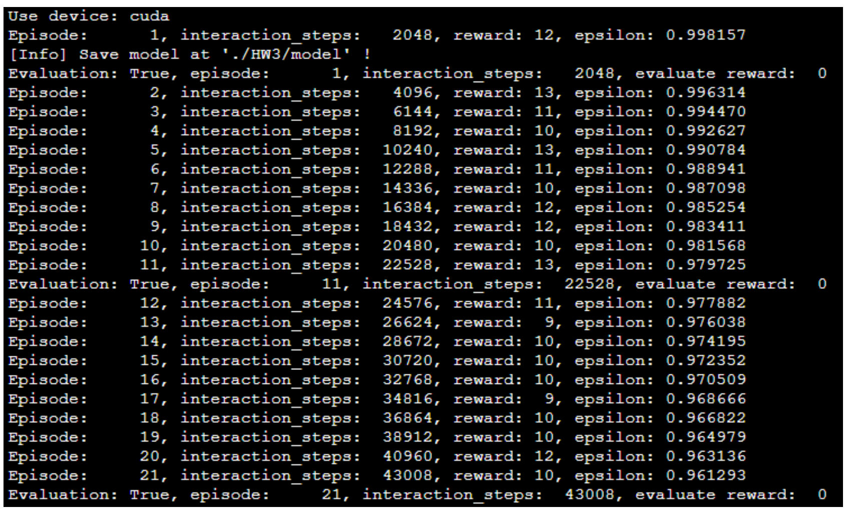
2. For deep Q-Learning, exploring the environment is an important procedure. Because the reward signals in the given environment is very sparse, it takes a lot of computation time to explore the state space and perform the updating. To speed up the training process, you can simply change the probability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



```
Episode:   784, interaction_steps: 1605632, reward: 13, epsilon: 0.100000
Episode:   785, interaction_steps: 1607680, reward: 11, epsilon: 0.100000
Episode:   786, interaction_steps: 1609728, reward:  8, epsilon: 0.100000
Episode:   787, interaction_steps: 1611776, reward: 10, epsilon: 0.100000
Episode:   788, interaction_steps: 1613824, reward: 13, epsilon: 0.100000
Episode:   789, interaction_steps: 1615872, reward: 10, epsilon: 0.100000
Episode:   790, interaction_steps: 1617920, reward: 10, epsilon: 0.100000

Episode:   791, interaction_steps: 1619968, reward: 11, epsilon: 0.100000
Evaluation: True, episode:   791, interaction_steps: 1619968, evaluate reward:  2
Episode:   792, interaction_steps: 1622016, reward: 11, epsilon: 0.100000
Episode:   793, interaction_steps: 1624064, reward: 13, epsilon: 0.100000
Episode:   794, interaction_steps: 1626112, reward: 13, epsilon: 0.100000
Episode:   795, interaction_steps: 1628160, reward: 11, epsilon: 0.100000
Episode:   796, interaction_steps: 1630208, reward: 10, epsilon: 0.100000
Episode:   797, interaction_steps: 1632256, reward: 10, epsilon: 0.100000
Episode:   798, interaction_steps: 1634304, reward: 13, epsilon: 0.100000
Episode:   799, interaction_steps: 1636352, reward: 13, epsilon: 0.100000
Episode:   800, interaction_steps: 1638400, reward: 12, epsilon: 0.100000
```

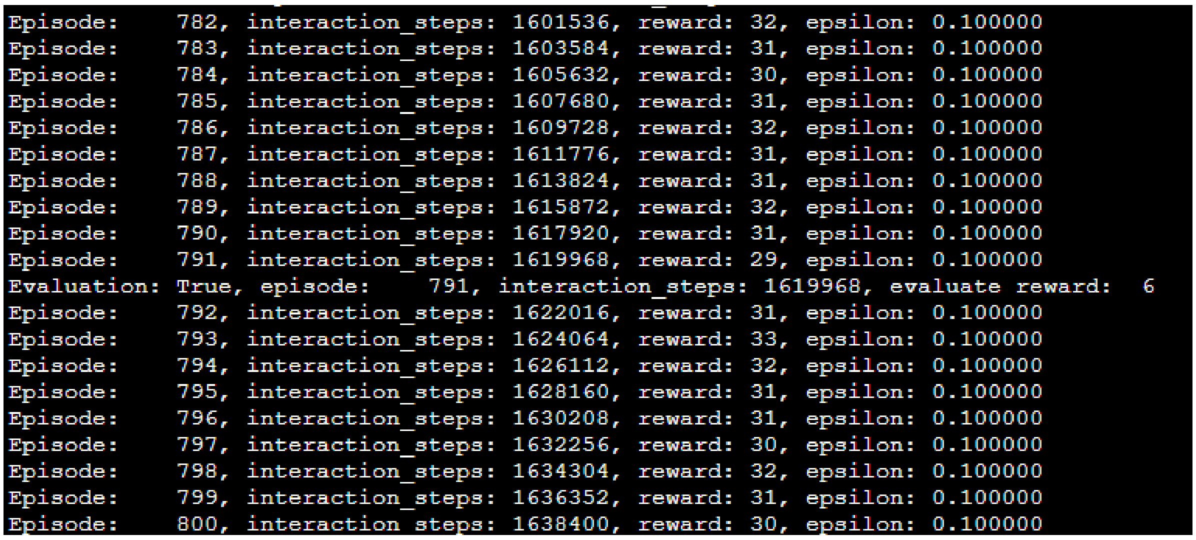
<u>原始的code：</u> 三種action操作的機率均等。訓練到最後，episode結果如下：
在這樣的設定狀況下，前面有很長一段時間reward皆是0。一直到800 episode附近的時候才有約莫10的reward。效果不甚好。



```
Use device: cuda
Episode:      1, interaction_steps:   2048, reward: 12, epsilon: 0.998157
[Info] Save model at './HW3/model' !
Evaluation: True, episode:      1, interaction_steps:   2048, evaluate reward:  0
Episode:      2, interaction_steps:   4096, reward: 13, epsilon: 0.996314
Episode:      3, interaction_steps:   6144, reward: 11, epsilon: 0.994470
Episode:      4, interaction_steps:   8192, reward: 10, epsilon: 0.992627
Episode:      5, interaction_steps:  10240, reward: 13, epsilon: 0.990784
Episode:      6, interaction_steps:  12288, reward: 11, epsilon: 0.988941
Episode:      7, interaction_steps:  14336, reward: 10, epsilon: 0.987098
Episode:      8, interaction_steps:  16384, reward: 12, epsilon: 0.985254
Episode:      9, interaction_steps:  18432, reward: 12, epsilon: 0.983411
Episode:     10, interaction_steps:  20480, reward: 10, epsilon: 0.981568
Episode:     11, interaction_steps:  22528, reward: 13, epsilon: 0.979725
Evaluation: True, episode:     11, interaction_steps:  22528, evaluate reward:  0
Episode:     12, interaction_steps:  24576, reward: 11, epsilon: 0.977882
Episode:     13, interaction_steps:  26624, reward:  9, epsilon: 0.976038
Episode:     14, interaction_steps:  28672, reward: 10, epsilon: 0.974195
Episode:     15, interaction_steps:  30720, reward: 10, epsilon: 0.972352
Episode:     16, interaction_steps:  32768, reward: 10, epsilon: 0.970509
Episode:     17, interaction_steps:  34816, reward:  9, epsilon: 0.968666
Episode:     18, interaction_steps:  36864, reward: 10, epsilon: 0.966822
Episode:     19, interaction_steps:  38912, reward: 10, epsilon: 0.964979
Episode:     20, interaction_steps:  40960, reward: 12, epsilon: 0.963136
Episode:     21, interaction_steps:  43008, reward: 10, epsilon: 0.961293
Evaluation: True, episode:     21, interaction_steps:  43008, evaluate reward:  0
```

<u>更改過後：</u> 將三種action操作的機率改為[**NOOP (0.3), UP (0.6), DOWN(0.1)**]。結果如下：

可以發現在最一開始就可以將reward來到10上下，而到最後快要結束的時候，則可以來到平均31左右的reward。明顯效果有提升，agent玩遊戲玩得比較好。

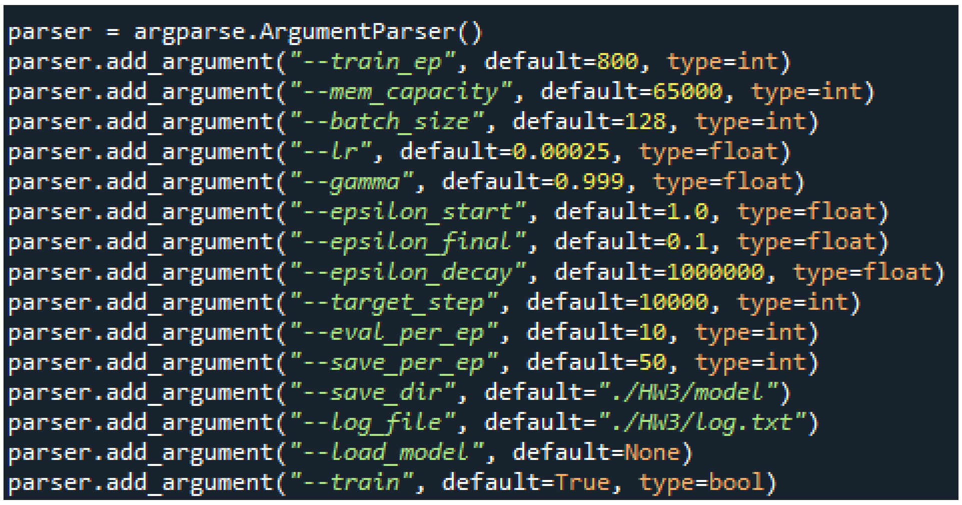


```
Episode:     782, interaction_steps: 1601536, reward: 32, epsilon: 0.100000
Episode:     783, interaction_steps: 1603584, reward: 31, epsilon: 0.100000
Episode:     784, interaction_steps: 1605632, reward: 30, epsilon: 0.100000
Episode:     785, interaction_steps: 1607680, reward: 31, epsilon: 0.100000
Episode:     786, interaction_steps: 1609728, reward: 32, epsilon: 0.100000
Episode:     787, interaction_steps: 1611776, reward: 31, epsilon: 0.100000
Episode:     788, interaction_steps: 1613824, reward: 31, epsilon: 0.100000
Episode:     789, interaction_steps: 1615872, reward: 32, epsilon: 0.100000
Episode:     790, interaction_steps: 1617920, reward: 31, epsilon: 0.100000
Episode:     791, interaction_steps: 1619968, reward: 29, epsilon: 0.100000
Evaluation: True, episode:     791, interaction_steps: 1619968, evaluate reward:  6
Episode:     792, interaction_steps: 1622016, reward: 31, epsilon: 0.100000
Episode:     793, interaction_steps: 1624064, reward: 33, epsilon: 0.100000
Episode:     794, interaction_steps: 1626112, reward: 32, epsilon: 0.100000
Episode:     795, interaction_steps: 1628160, reward: 31, epsilon: 0.100000
Episode:     796, interaction_steps: 1630208, reward: 31, epsilon: 0.100000
Episode:     797, interaction_steps: 1632256, reward: 30, epsilon: 0.100000
Episode:     798, interaction_steps: 1634304, reward: 32, epsilon: 0.100000
Episode:     799, interaction_steps: 1636352, reward: 31, epsilon: 0.100000
Episode:     800, 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.

使模型收斂的方法：參考論文使用的超參數設定，並適度更改mem_capacity。

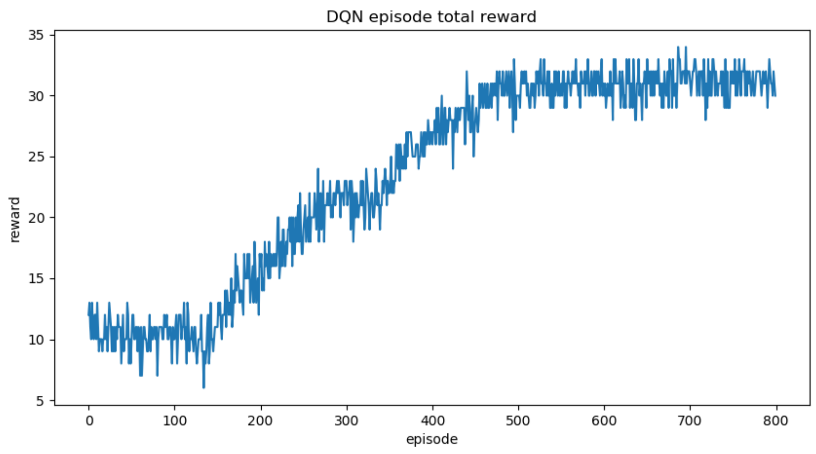
1. **參考論文中的超參數設定：**

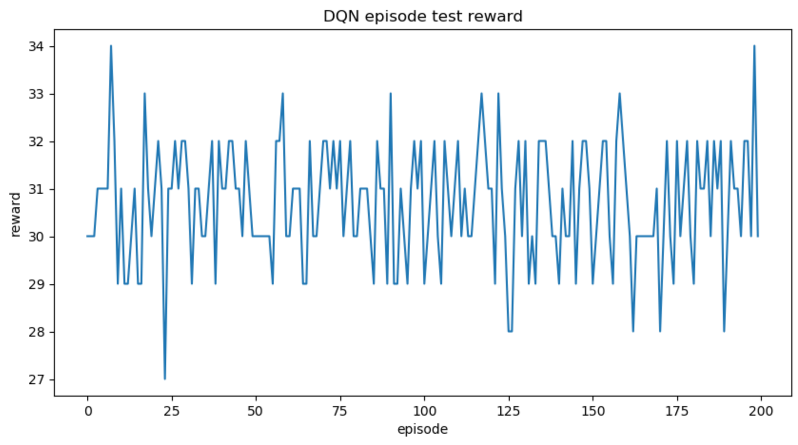


```
parser = argparse.ArgumentParser()
parser.add_argument("--train_ep", default=800, type=int)
parser.add_argument("--mem_capacity", default=65000, type=int)
parser.add_argument("--batch_size", default=128, type=int)
parser.add_argument("--lr", default=0.00025, type=float)
parser.add_argument("--gamma", default=0.999, type=float)
parser.add_argument("--epsilon_start", default=1.0, type=float)
parser.add_argument("--epsilon_final", default=0.1, type=float)
parser.add_argument("--epsilon_decay", default=1000000, type=float)
parser.add_argument("--target_step", default=10000, type=int)
parser.add_argument("--eval_per_ep", default=10, type=int)
parser.add_argument("--save_per_ep", default=50, type=int)
parser.add_argument("--save_dir", default="./HW3/model")
parser.add_argument("--log_file", default="./HW3/log.txt")
parser.add_argument("--load_model", default=None)
parser.add_argument("--train", default=True, type=bool)
```

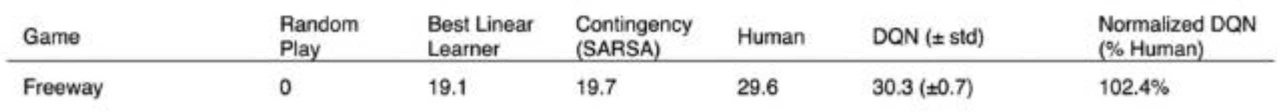
2. **調整mem_capacity：**

因為使用實驗室的 server 上的 GPU，發現原本 default 設定的 mem_capacity=90000 容易在幾次 episode 之後產生 Runtime error，會發生不足使用的問題，因此最後調整為 mem_capacity=65000 讓實驗能夠順利跑完，但也要花上 4~5 小時之久。



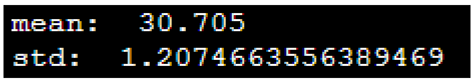


4. After training, you will obtain the model parameters for the agent. Show total reward in some episodes for deep Q-network agent.
在testing phase中，將原本訓練好的agent之model parameters load進來，呈現episode結果，並和paper中的效果作對照：
Paper中的結果：



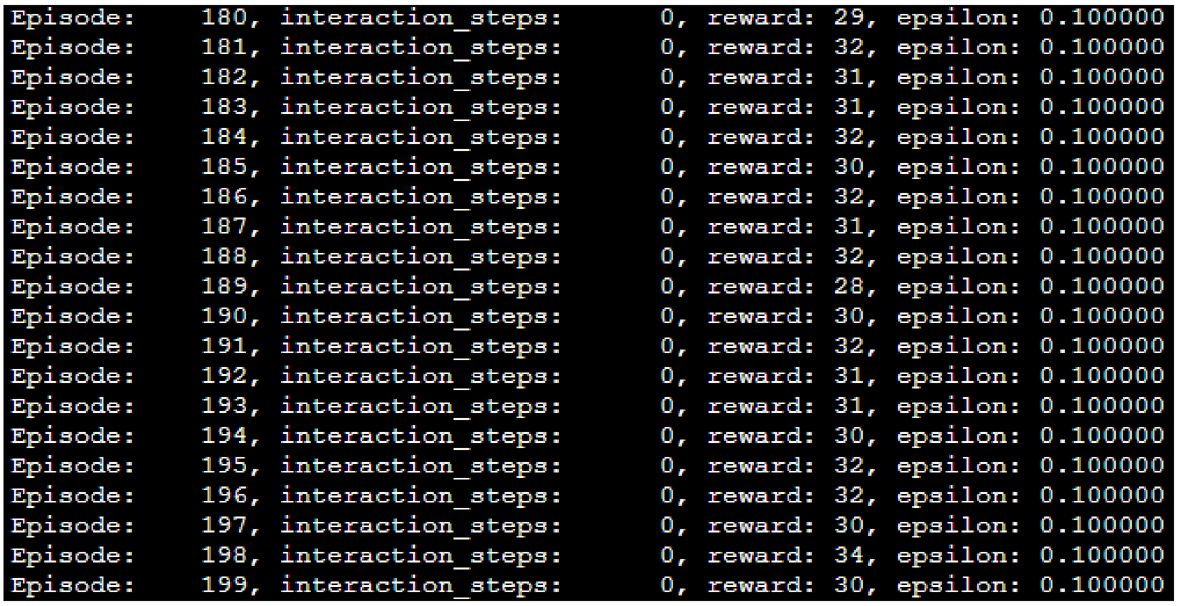
Game	Random Play	Best Linear Learner	Contingency (SARSA)	Human	DQN (± std)	Normalized DQN (% Human)
Freeway	0	19.1	19.7	29.6	30.3 (±0.7)	102.4%

模型結果：



```
mean:   30.705
std:  1.2074663556389469
```

Mean: 30.705
Std: 1.2074663556389469

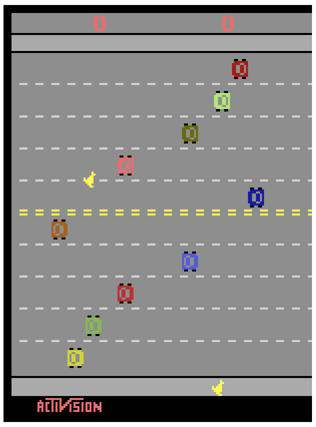


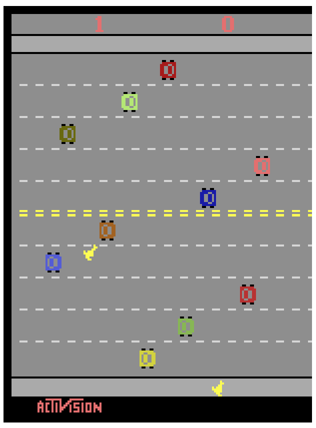
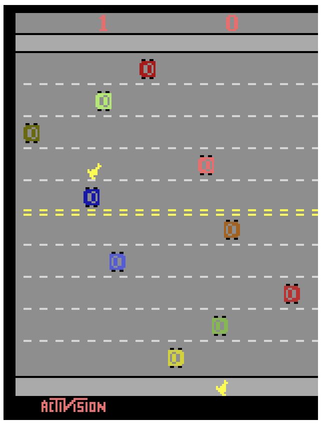
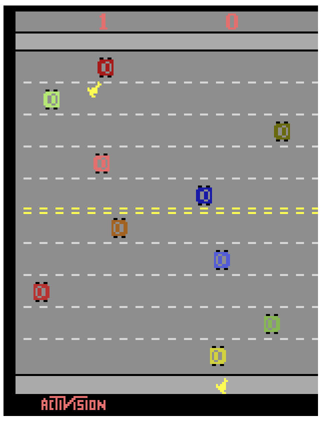
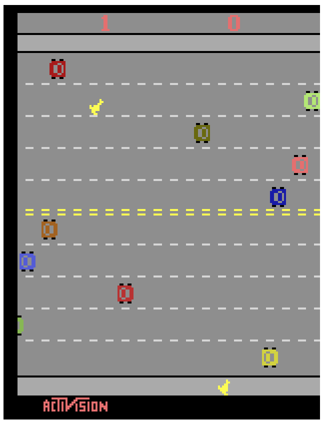
```
Episode:     180, interaction_steps:       0, reward: 29, epsilon: 0.100000
Episode:     181, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     182, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:     183, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:     184, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     185, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:     186, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     187, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:     188, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     189, interaction_steps:       0, reward: 28, epsilon: 0.100000
Episode:     190, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:     191, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     192, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:     193, interaction_steps:       0, reward: 31, epsilon: 0.100000
Episode:     194, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:     195, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     196, interaction_steps:       0, reward: 32, epsilon: 0.100000
Episode:     197, interaction_steps:       0, reward: 30, epsilon: 0.100000
Episode:     198, interaction_steps:       0, reward: 34, epsilon: 0.100000
Episode:     199, interaction_steps:       0, reward: 30, epsilon: 0.100000
```

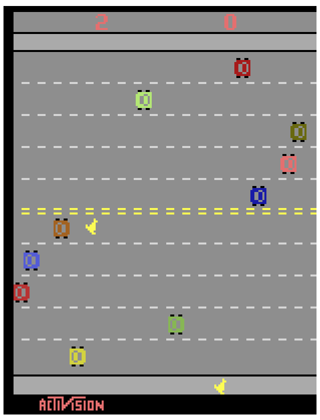
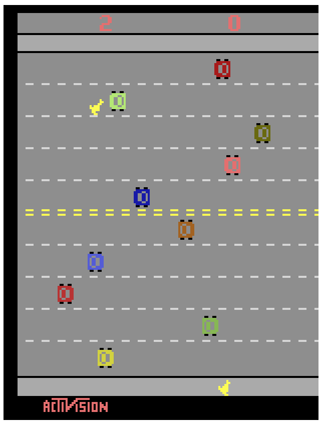
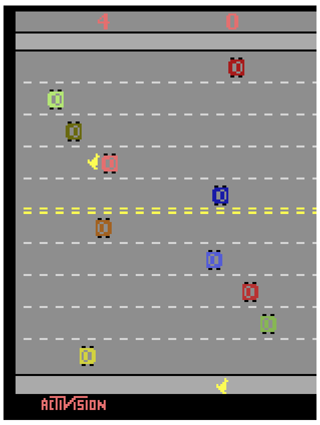
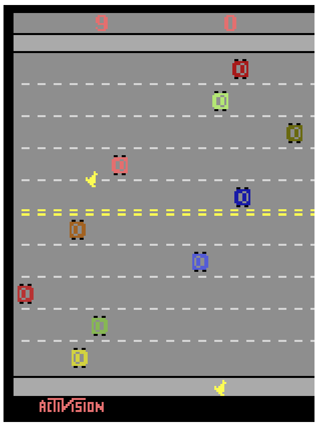
在testing的時候模型平均獲得的reward為30.705左右，略好於paper上的紀錄。在後面遊戲畫面的截圖上也可以看到agent有許多策略是偏向高難度、高風險，是人在遊玩的時候比較不會做出的、卻可以大幅提升reward的操作與決定。

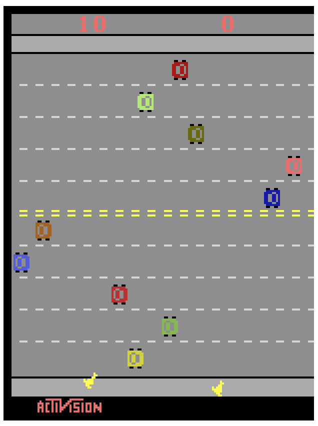
5. Sample some states, show the Q values for each action, analyze the results, and answer

- **Is DQN decision in the game the same as yours? Any good or bad move?**

遊戲畫面	Q value	choice	avg	我的 不同於我的決定
	[1.4895419 1.5018126 1.3285295]	UP	1.4399613	車子距離我的前方還有些微距離，選擇前進。

	[0.7968872 0.86689264 0.746572]	UP	0.8034506	前面有車子靠近(褐色)但agent決定要往前。
	[0.86951584 0.847171 0.83254826]	NOOP	0.84974504	前方車子剛離去但agent沒打算前進反而是經歷了兩次的NOOP決定。
	[1.5487285 1.477078 1.5487279]	NOOP	1.5248448	當逼近車子的時候，選擇NOOP。在此前一個動作為DOWN來跟車子保持一定的距離。
	[1.6127053 1.6073669 1.6194779]	DOWN	1.6131834	最後一段路程，agent上上下下了很多次，但是這時車道其實是相對空曠安全的，卻花費了很多時間與步伐在最後兩道上。

	[0.6470836 0.98710275 0.65066534]	UP	0.7616172	在連續三道都即將有車子行進過來的狀態下，agent直接火力全開向前行，毫不猶豫。
	[1.4956856 1.5925587 1.4161783]	UP	1.5014743	類似上一個，這次的agent幾乎一路衝到最後，即將驚險閃過亮綠色的車子。
	[1.206667 1.4406475 0.86545074]	UP	1.1709218	發現agent可以許多次都能夠驚險地度過難關，例如這張截圖，通常人在玩的時候可能沒辦法很精準知道怎樣是「允許的邊界」，因此不會做這麼極端的決定。
	[1.1888151 1.3931128 0.9849134]	UP	1.1889471	發現agent在兩道都有車子剛好經過的時候，很喜歡從中間剛好穿過去。我想這是人比較不會做的決定，也會是相對較困難、具有風險的決定。

	[0.20109983 0.2006028 0.2055704]	DOWN	0.20242435	前方兩道車子都剛好像又離去，但是agent卻連做了四次DOWN的決定。

- **Why the averaged Q-value of three actions in some state is larger or less than those of the other states?**
  從上面的遊戲截圖可以看出來，三種action的Q value平均較高者(avg>1)，agent在畫面上都已經走完了一半，也就是超過中間的雙黃線，而平均較低者(avg<1)，則是大多還沒走到一半(如上面表格中最後一個遊戲截圖)，也就是距離成功獲得reward還有一大段距離，因此由此推測，當越接近獲得reward的時候，其平均的Q value也會越高，當遊戲在離獲得reward還有段距離時，平均Q value會較低。

DQN總結：
在這次DQN的實作當中，透過助教給予的輔助程式碼，能讓我更快理解實作細節。原本在理論上比較不清楚的部分也可以透過code來去驗證。之前總是耳聞RL很難訓練，這次可能是因為有了現有的固定程式以及架構，才會讓reward在改變機率以後就有很好的提升。日後希望有機會能再survey有關DQN的實作細節以及近年來更多的改善方法，並試著用在更多的應用層面上。

GAN以及DQN model參數連結：
https://drive.google.com/drive/folders/1WuuSSY5m-BpI2b41BTaWpIzouYNT1rmB?usp=sharing