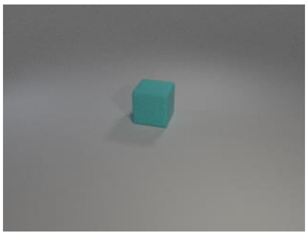
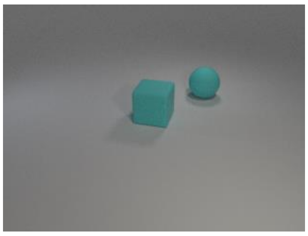
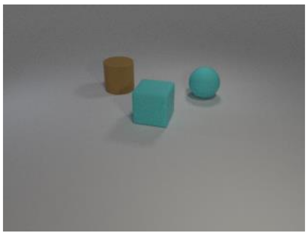
DLP Lab5 Let’s Play GANs

0711529 陳冠儒

1. Introduction (5%)
這次的 Lab 是用 GAN 架構與 multi-label condition 來產生 synthetic images，並且要 (1)自行選擇要使用的 conditional GAN (e.g., conditional GAN、Auxiliary GAN、Projection discriminator)；(2)自行選擇 discriminator 與 generator 的設計 (e.g., DCGAN、Super resolution GAN、Self-attention GAN、Progressive growing of GAN)；(3)自行選擇要使用的 GAN loss function (e.g., normal GAN、LSGAN、WGAN、WGAN-GP)。最後再去比較結果，像是各個架構間的區別等等。

    A. Dataset
    這此使用的 Dataset 是 i-CLEVR，裡面共有 24 種的 objects，包含 8 種的顏色和 3 種的形狀，而在 training data 中會是一張圖片對應到一組的 object labels，而在 testing data 的部分則只會給 object labels，以此搭配上 random noise 去 generator 出圖像。

    training data example：

		
["cyan cube"]	["cyan cube", "cyan sphere"]	["cyan cube", "cyan sphere", "brown cylinder"]

2. Implementation details (15%)
我最後選擇的是 DCGAN + Auxiliary GAN + WGAN-GP 的組合。
    A. Overall Model Architecture
        - Discriminator

```
================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
├─Sequential: 1-1                        [-1, 512, 4, 4]           --
|    └─Conv2d: 2-1                       [-1, 64, 32, 32]          3,072
|    └─LeakyReLU: 2-2                    [-1, 64, 32, 32]          --
|    └─Conv2d: 2-3                       [-1, 128, 16, 16]         131,072
|    └─LeakyReLU: 2-4                    [-1, 128, 16, 16]         --
|    └─Conv2d: 2-5                       [-1, 256, 8, 8]           524,288
|    └─LeakyReLU: 2-6                    [-1, 256, 8, 8]           --
|    └─Conv2d: 2-7                       [-1, 512, 4, 4]           2,097,152
|    └─LeakyReLU: 2-8                    [-1, 512, 4, 4]           --
├─Sequential: 1-2                        [-1, 1, 1, 1]             --
|    └─Conv2d: 2-9                       [-1, 1, 1, 1]             8,192
├─Sequential: 1-3                        [-1, 24]                  --
|    └─Linear: 2-10                      [-1, 24]                  196,632
|    └─Sigmoid: 2-11                     [-1, 24]                  --
================================================================================
Total params: 2,960,408
Trainable params: 2,960,408
Non-trainable params: 0
Total mult-adds (M): 106.97
================================================================================
```

input：image，shape 為 [3, 64, 64]

output：

(1) 判別 real、fake，shape 為 [1]

(2) 判別 label，shape 為 [24]

- Generator

```
================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
├─Linear: 1-1                            [-1, 1, 128]              16,000
├─Sequential: 1-2                        [-1, 3, 64, 64]           --
|    └─ConvTranspose2d: 2-1              [-1, 512, 4, 4]           1,048,576
|    └─BatchNorm2d: 2-2                  [-1, 512, 4, 4]           1,024
|    └─ReLU: 2-3                         [-1, 512, 4, 4]           --
|    └─ConvTranspose2d: 2-4              [-1, 256, 8, 8]           2,097,152
|    └─BatchNorm2d: 2-5                  [-1, 256, 8, 8]           512
|    └─ReLU: 2-6                         [-1, 256, 8, 8]           --
|    └─ConvTranspose2d: 2-7              [-1, 128, 16, 16]         524,288
|    └─BatchNorm2d: 2-8                  [-1, 128, 16, 16]         256
|    └─ReLU: 2-9                         [-1, 128, 16, 16]         --
|    └─ConvTranspose2d: 2-10             [-1, 64, 32, 32]          131,072
|    └─BatchNorm2d: 2-11                 [-1, 64, 32, 32]          128
|    └─ReLU: 2-12                        [-1, 64, 32, 32]          --
|    └─ConvTranspose2d: 2-13             [-1, 3, 64, 64]           3,072
|    └─Tanh: 2-14                        [-1, 3, 64, 64]           --
================================================================================
Total params: 3,822,080
Trainable params: 3,822,080
Non-trainable params: 0
Total mult-adds (M): 435.84
================================================================================
```

input：

(1) noise，shape 為 [100]

(2) label，shape 為 [24]

output：generated image，shape 為 [3, 64, 64]

- Loss function

(1) Discriminator Loss

```
auxiliary_loss = nn.BCELoss()
# Real images
real_pred, real_aux = discriminator(real_imgs)
d_real_loss = - torch.mean(real_pred)
d_real_cls_loss = auxiliary_loss(real_aux, real_labels.float())

# Fake images
fake_pred, fake_aux = discriminator(gen_imgs.detach())
d_fake_loss = torch.mean(fake_pred)
d_fake_cls_loss = auxiliary_loss(fake_aux, real_labels.float())

# gradient penalty
gradient_penalty = compute_gradient_penalty(discriminator, real_imgs, gen_imgs)

 # Total discriminator loss
d_loss = d_real_loss + d_fake_loss + lambda_cls*d_real_cls_loss + lambda_gp*gradient_penalty
```

(2) Generator Loss

```
auxiliary_loss = nn.BCELoss()
# Loss measures generator's ability to fool the discriminator
gen_imgs = generator(z, real_labels)

# Loss measures generator's ability to fool the discriminator
fake_validity, pred_label = discriminator(gen_imgs)

g_loss_fake = -torch.mean(fake_validity)
g_loss_cls = auxiliary_loss(pred_label, real_labels.float())
g_loss = g_loss_fake + lambda_cls * g_loss_cls
```

B. DCGAN implementation details

- Architecture

(1) 在 discriminator 上用 strided convolutions，在 generator 上用 fractional-strided convolutions。

(2) 在 generator 和 discriminator 上都使用 batchnorm。
(但是 discriminator 的 batchnorm 因為用了 wgan-gp 所以拿掉了)

(3) Generator 的除了輸出層外的所有層使用 ReLU，輸出層採用 tanh。

(4) Discriminator 的所有層上使用 LeakyReLU。

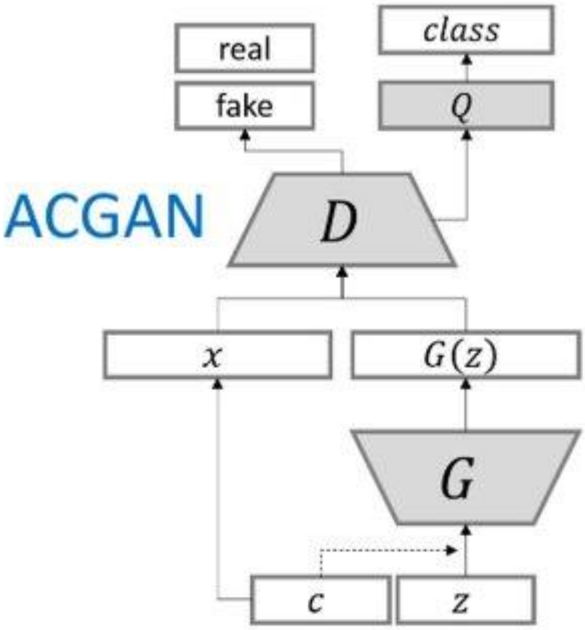
- Training

(1) 預處理環節，將圖像 scale 到 tanh 的[-1, 1]。

(2) 所有的參數初始化由(0, 0.02)的正態分佈中隨即得到

(3) 使用 Adam optimizer，並且將 momentum 參數 beta 從 0.9 降為 0.5 來防止震盪和不穩定。

C. Auxiliary GAN implementation details



- Architecture
  (1) Generator 將 one-hot 處理過的 condition 跟 noise(z) concatenate 在一起。
  (2) Discriminator output 出兩個，一個是判斷 real 或 fake 的機率，一個是判斷是哪個 class 的機率。

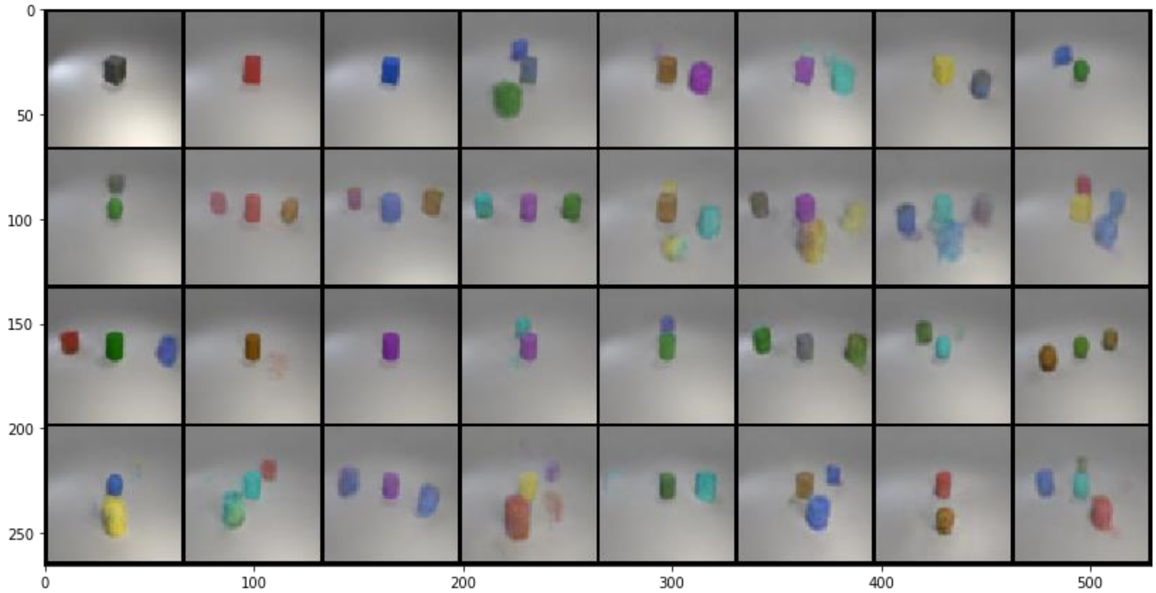
- Training
  (1) 因為這次使用的是 multi-label classes，所以在 discriminator 最後一層判斷 classes 的時候要用 sigmoid，而在 loss function 的地方要用 BCELoss。

D. WGAN-GP implementation details
- Architecture
  (1) 用 gradient penalty 取代 weight clipping
  (2) 刪掉 critic 中的 batchnorm

E. Hyperparameters
- batch_size = 128
- latent_dim = 100
- Optimizer：Adam(lr = 0.0002, betas = (0.5, 0.999))
- epochs = 600
- n_critic = 5 # number of training steps for discriminator per iter
- lambda_gp = 10 # Loss weight for gradient penalty
- lambda_cls = 5 # Loss weight for auxiliary

3. Results and discussion (20%)

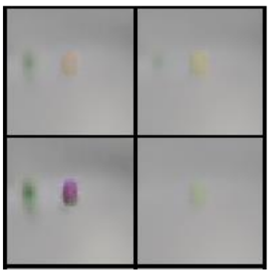
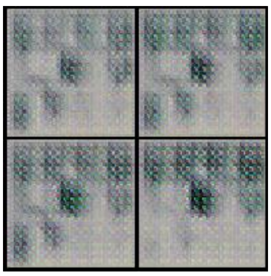
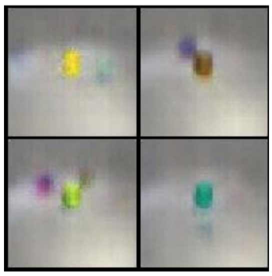
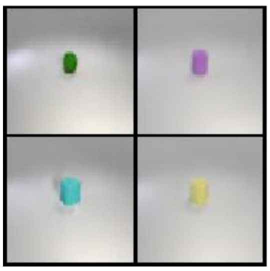
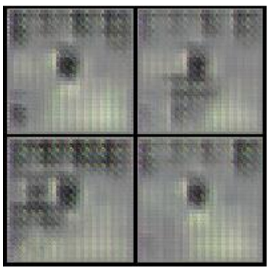
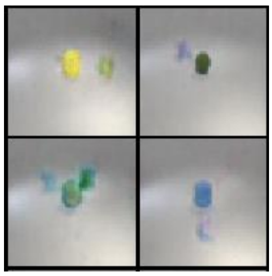
A. Results based on testing data

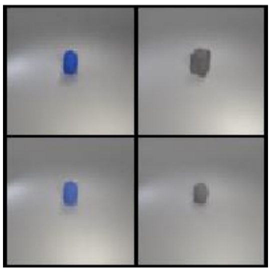
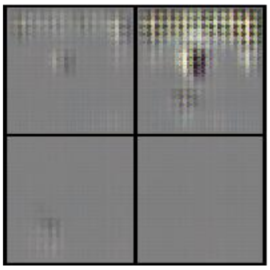
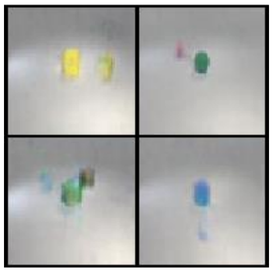


Score：0.7638

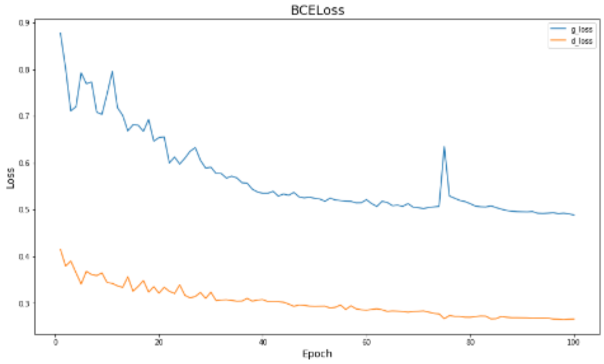
B. Discuss

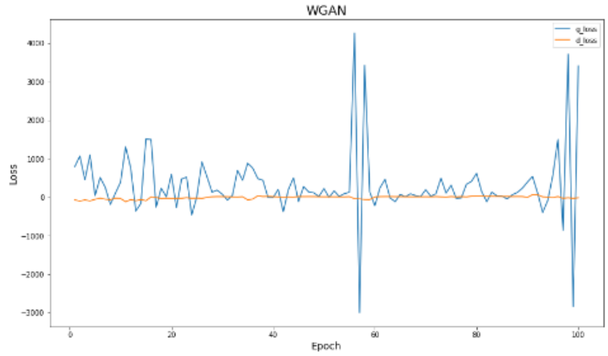
- ACGAN + DCGAN with different loss function (BCELoss, WGAN, WGAN-GP)
    - Testing result (擷取右上角的四個結果)

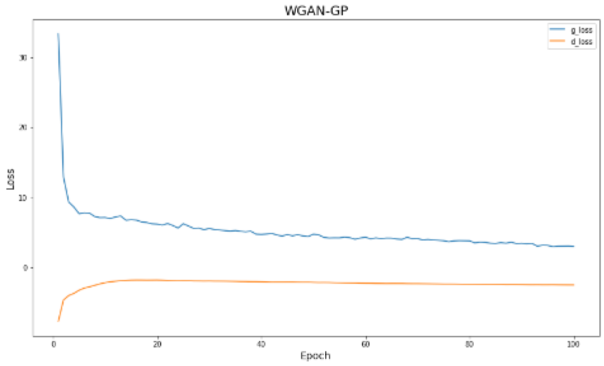
BCELoss	WGAN	WGAN-GP
		
epoch 10	epoch 10	epoch 10
		
epoch 50	epoch 50	epoch 50

		
epoch 100	epoch 100	epoch 100

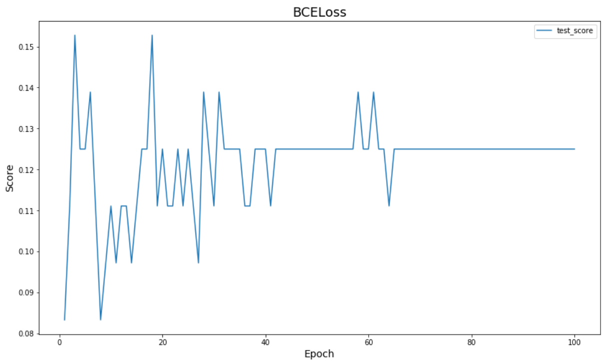
- Training Loss

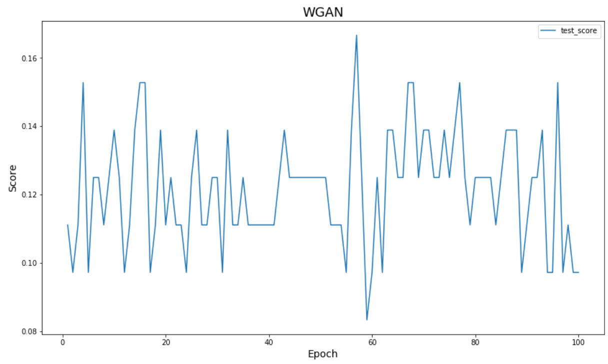


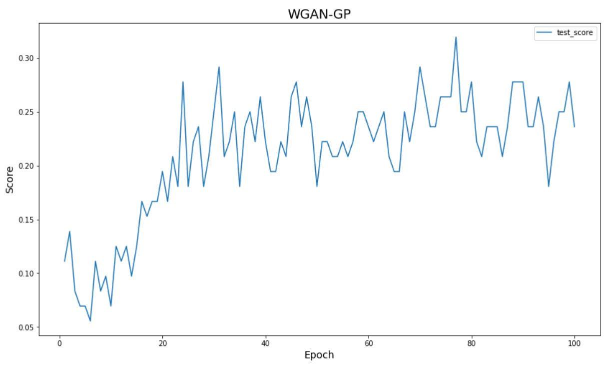




- Testing score





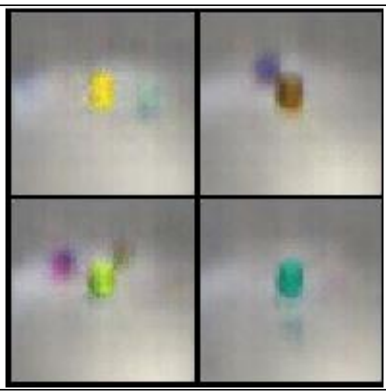
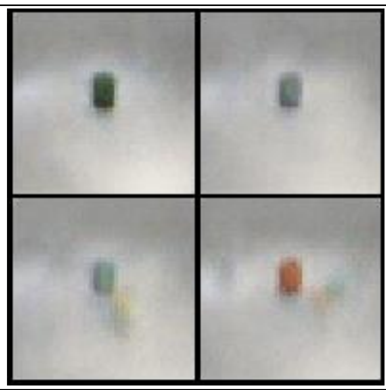


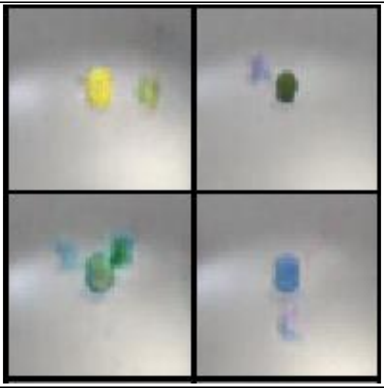
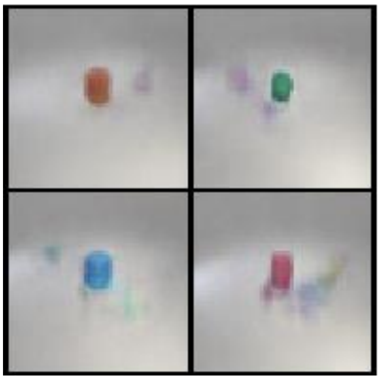
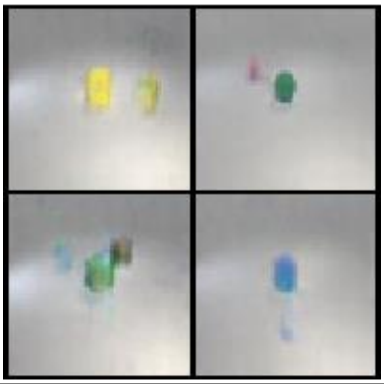
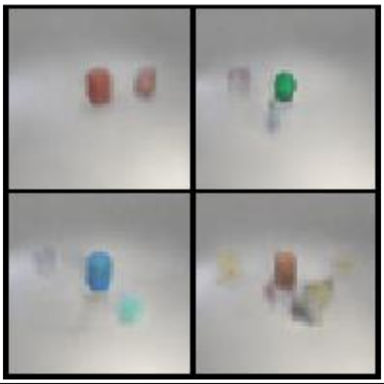
- Discuss

  從上面的結果可以證明用 WGAN-GP loss function 最能夠進行有效的收斂及穩定的訓練；而 WGAN，可能因為使用的 weight clipping 只是當初為了提出論文時的權宜之計，並不能很好的實現出 wasserstein distance，所以結果也並沒有很理想。

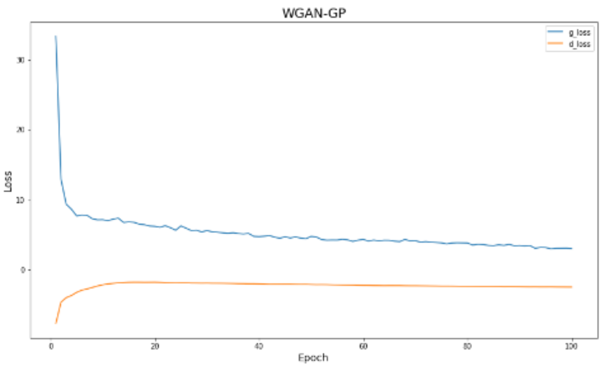
  至於原始的 ACGAN 可以發現訓練到最後會只剩下單一 label，而不是原先 multi-labels 的情況，這部分的原因我還不是很清楚，因為在這個訓練集中每個 label 都是獨立的，所以應該是使用 Binary Cross Entropy 而不是 categorical cross entropy 進行訓練，但最終結果卻也依然只有一個，這是比較困擾我的地方。

- Auxiliary GAN compare with Conditional GAN (both using WGAN-GP as loss function)

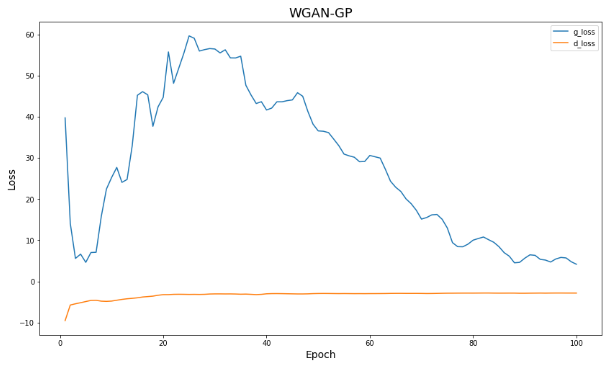
Auxiliary	Conditional
	
epoch 10	epoch 10

	
epoch 50	epoch 50
	
epoch 100	epoch 100

- Training Loss

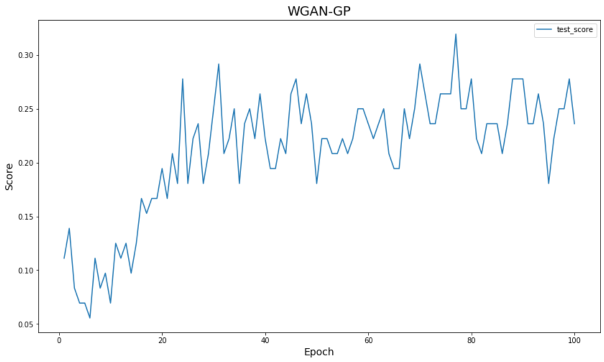


Auxiliary

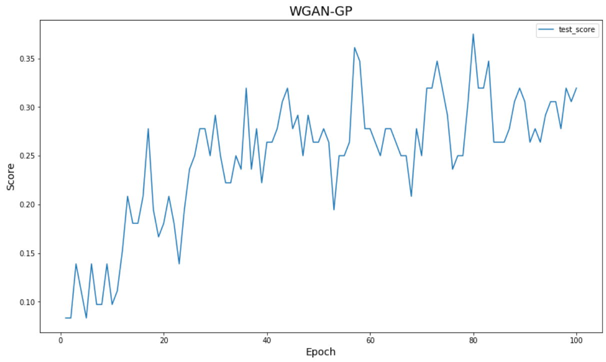


Conditional

- Testing Score



Auxiliary



Conditional

- Discuss

  雖然從前 100 epoch 中，我訓練出來的結果是 conditional 優於 auxiliary，但是從 paper 或是其他文獻中都證明說 Auxiliary 的結果會較 conditional 好，所以我最後還是使用 auxiliary 進行訓練。

  Auxiliary 跟 Conditional 的差異在於 auxiliary 的 discriminator 兼具分類的功能，因此他只要給進一張 image 就能輸出它是屬於哪一類，而 conditional 則是由我們給定 discriminator 它是甚麼類別，因此直觀來說 conditional 在前期訓練時會比較優異(因為要學習參數的比較少)，但若是收斂後，auxiliary 的成效應該會比較優異(因為時間關係所以就沒有訓練 conditional 到收斂)。