Deep Learning and Practice

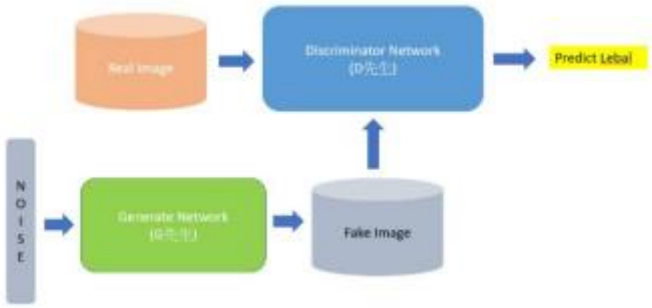
Lab7 - Let's Play GANs

310605009
吳公耀

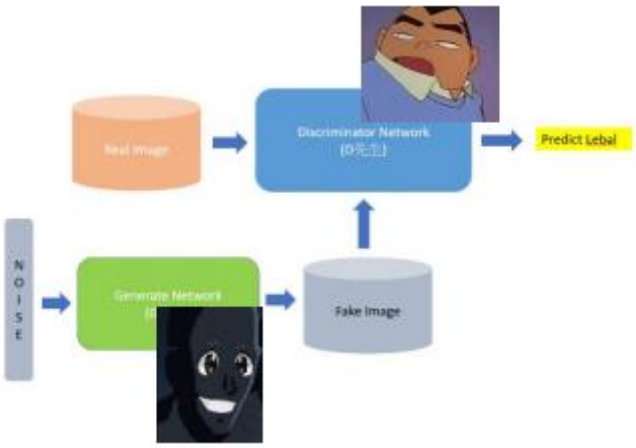
1. Report (50%)

Introduction (5%)

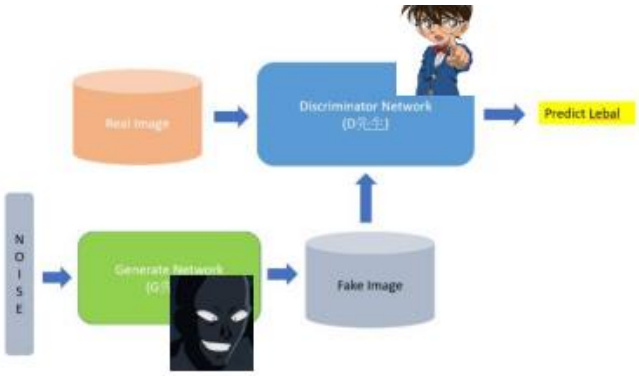
GAN



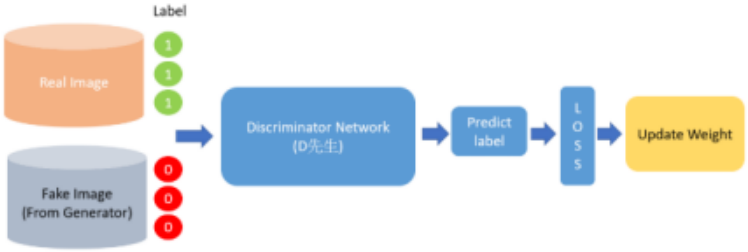
GAN 中有二個 Neural Network 需要去訓練，分別為 Generate Network(生成 器)跟 Discriminator Network (鑑別器)，由 Generate Network 生成圖片，並由 Discriminator Network 判別圖片的真偽來進行互相訓練。



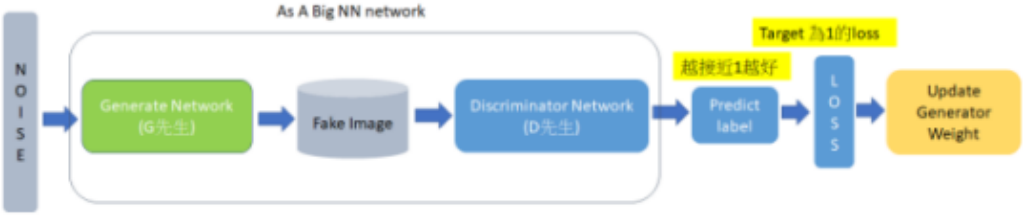
一開始 Generate Network 會生出非常假的圖片，此時 Discriminator Network 的判別能力也非常差。



而 Generate 要透過訓練，想盡辦法生出能夠騙過 Discriminator 的圖片，而相 對的，Discriminator 也會因為 Generate 所生出的假圖片越來越逼真，而提高 其辨別能力。



而訓練過程，即是把 Generator 出來的圖標記為 0(fake image)，然後把真實的 圖標記為 1，這樣的 training data 丟進我們的 Discriminator Network 做訓練。



把 Generator+ Discriminator 看成是一個大的 Neural Network，假設生成器是 前 5 層的 Neural，鑑定器是後 5 層的 Neural，然後這個 10 層的 Neural Network 預估出來的值越接近 1 越好，這裡只 update Generator 的 weight，Discriminator 的 Weight 要 Keep，這樣才可以用更新後 Generator 出來的假圖在 Discriminator 上的值越接近真實的結果，我們可以用一個結論來表示，其實就是更新生成器的 參數讓 Discriminator 接近真實的結果。

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

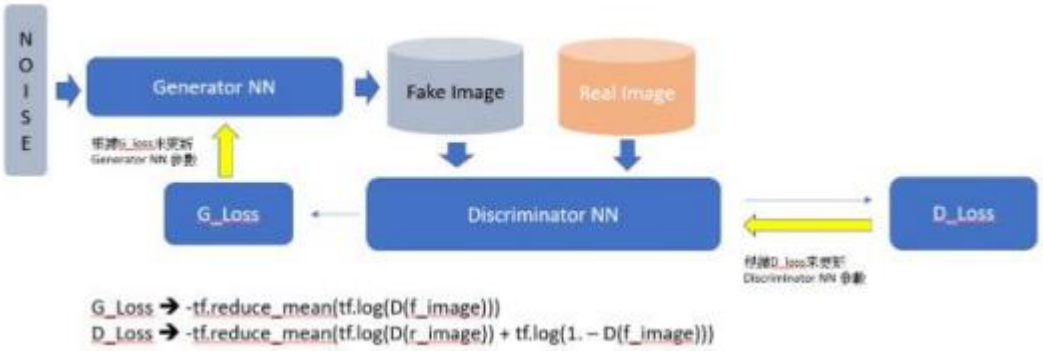
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

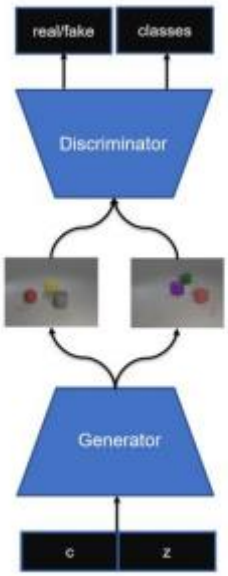
end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

上圖為論文中 GAN 的演算法



根據演算法及上圖可得知： G_Loss 是我生成器產生的偽造圖輸入進鑑定器中的輸出跟 1 的 loss D_Loss 是生成器產生的偽造圖輸入進鑑定器中的輸出跟 0 的 loss+真實的圖輸 入進鑑定器中的輸出跟 1 的 loss。 一般 GAN 為 Unsupervised ，並由 noise 生成新的假圖像，但在這次的 lab 需要加入 conditional，除了 Discriminator 要判斷 Generator 所生成的圖片是真是 假，另外還有與 groundtruth 做比較，來判斷這張圖片是真的真的還是真的假 的，並由 Unsupervised 變為 supervised。



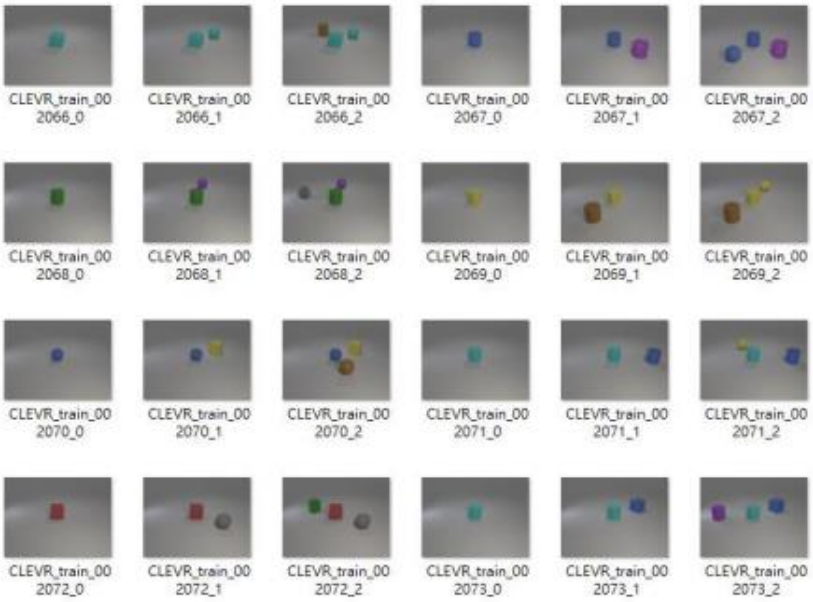
依照此次的架構，除了利用 GAN 訓練出 對應其 label 的顏色跟形狀。

(一)Lab Objective

- implement a conditional GAN
- generate synthetic images according to different conditions.

(二)Dataset Iclevr

為這次的訓練資料集中，其中最多為三個幾何形體，且有不同的顏色。



train： 已經整理好的圖片名稱已及其對應所有的幾何形體


```
{"CLEVR_train_002066_0.png": ["cyan cube"], "CLEVR_train_002066_1.png": ["cyan cube", "cyan sphere"], "CLEVR_train_002066_2.png": ["cyan
 cube", "cyan sphere", "brown cylinder"], "CLEVR_train_000341_0.png": ["gray sphere"], "CLEVR_train_000341_1.png": ["gray sphere", "green
 cylinder"], "CLEVR_train_000341_2.png": ["gray sphere", "green cylinder", "blue sphere"], "CLEVR_train_000051_0.png": ["purple sphere"],
 "CLEVR_train_000051_1.png": ["purple sphere", "red cube"], "CLEVR_train_000051_2.png": ["purple sphere", "red cube", "yellow cube"],
"CLEVR_train_003457_0.png": ["gray cube"], "CLEVR_train_003457_1.png": ["gray cube", "yellow sphere"], "CLEVR_train_003457_2.png": ["gray
 cube", "yellow sphere", "green sphere"], "CLEVR_train_004620_0.png": ["gray sphere"], "CLEVR_train_004620_1.png": ["gray sphere", "green
 sphere"], "CLEVR_train_004620_2.png": ["gray sphere", "green sphere", "blue cylinder"], "CLEVR_train_005789_0.png": ["red sphere"],
"CLEVR_train_005789_1.png": ["red sphere", "gray cylinder"], "CLEVR_train_005789_2.png": ["red sphere", "gray cylinder", "green cylinder"
], "CLEVR_train_001045_0.png": ["blue sphere"], "CLEVR_train_001045_1.png": ["blue sphere", "green sphere"], "CLEVR_train_001045_2.png":
["blue sphere", "green sphere", "yellow cylinder"], "CLEVR_train_003729_0.png": ["blue cube"], "CLEVR_train_003729_1.png": ["blue cube",
"purple cube"], "CLEVR_train_003729_2.png": ["blue cube", "purple cube", "blue sphere"], "CLEVR_train_003354_0.png": ["cyan sphere"],
```


Objects 對於資料集中，所出現的顏色及形體進行分類，共 24 種類


```
{"gray cube": 0, "red cube": 1, "blue cube": 2, "green cube": 3, "brown cube": 4, "purple cube": 5, "cyan cube": 6, "yellow cube": 7, "gray
 sphere": 8, "red sphere": 9, "blue sphere": 10, "green sphere": 11, "brown sphere": 12, "purple sphere": 13, "cyan sphere": 14, "yellow
 sphere": 15, "gray cylinder": 16, "red cylinder": 17, "blue cylinder": 18, "green cylinder": 19, "brown cylinder": 20, "purple cylinder":
 21, "cyan cylinder": 22, "yellow cylinder": 23}
```

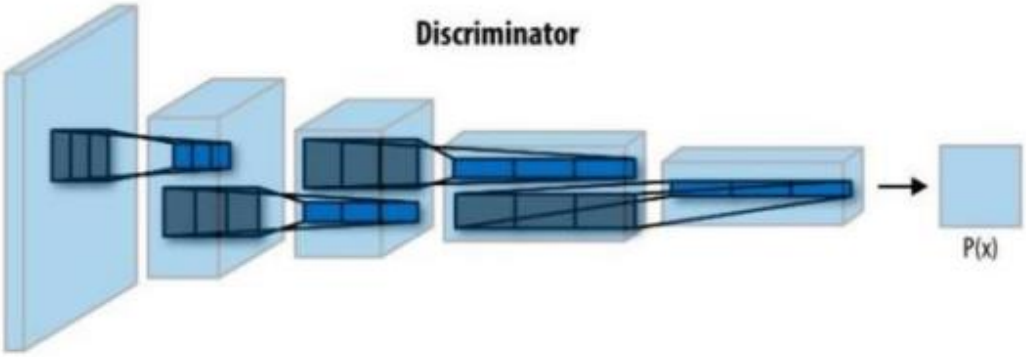

test 利用訓練好的模型與 test 的 label 生成圖片，並計算其準確率


```
[["gray cube"], ["red cube"], ["blue cube"], ["blue cube", "green cube"], ["brown cube", "purple cube"], ["purple cube", "cyan cube"], [
"yellow cube", "gray sphere"], ["blue sphere", "green sphere"], ["green sphere", "gray cube"], ["brown sphere", "red cube", "red cylinder"]
, ["purple sphere", "brown cylinder", "blue cube"], ["cyan sphere", "purple cylinder", "green cube"], ["yellow sphere", "cyan cylinder",
"brown cube"], ["gray cylinder", "yellow cylinder", "purple cube"], ["blue cylinder", "gray cube", "cyan cube"], ["blue cylinder", "red
 cube", "yellow cube"], ["green cylinder"], ["brown cylinder"], ["purple cylinder"], ["cyan cylinder", "purple cylinder"], ["blue cylinder"
, "green cylinder"], ["gray cylinder", "green cube"], ["cyan sphere", "gray cylinder"], ["brown sphere", "green sphere"], ["blue sphere",
"yellow cylinder"], ["red sphere", "cyan cylinder", "cyan cube"], ["gray sphere", "purple cylinder", "blue cube"], ["yellow cube", "brown
 cylinder", "purple cube"], ["cyan cube", "green cylinder", "blue cube"], ["brown cube", "blue cylinder", "blue sphere"], ["green sphere",
"red cylinder", "brown sphere"], ["blue cylinder", "gray cylinder", "cyan sphere"]]
```

Implementation details (15%)

– Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions. (10%)

在這個 LAB，我所選擇的 GAN 為 DCGAN，也就是將 Deep Convolution 與 GAN 結合，在處理 features 的時候使用卷積架構。



也就是需要符合 Convolution 的架構，convolution(經過 kernel，bias 後再經過 activation function 以及 padding)，利用卷積的特徵擷取，來做圖像的生成及訓練。

在 Generator 中，選擇 ReLU 做為 activation function

```
class Generator(nn.Module):
    def __init__(self, noise_size):
        super(Generator, self).__init__()

        self.gen = nn.Sequential(
            nn.ConvTranspose2d(noise_size, 512, 4, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.ReLU(True),

            nn.ConvTranspose2d(512, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.ReLU(True),

            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.ReLU(True),

            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.ReLU(True),

            nn.ConvTranspose2d(64, 32, 4, 2, 1, bias=False),
            nn.BatchNorm2d(32),
            nn.ReLU(True),

            nn.ConvTranspose2d(32, 16, 5, 1, 1, bias=False),
            nn.BatchNorm2d(16),
            nn.ReLU(True),

            nn.ConvTranspose2d(16, 3, 1, 1, 1, bias=False),
            nn.Tanh()
            #nn.Sigmoid()
        )
        self.noise_size = noise_size

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

Generator 的部份輸入 noise(40 維)+condition(24 維)，來生成對應 的圖片，而 Discriminator 的輸入除了 real image 還有 fake image，還 會把圖片對應的 label(也就是 condition，24 維)一起輸入到 D 裡面， 因為 D 原本是輸入 image，而 label 的大小跟 image 相差很大，所以 有先利用一個全連接層將 24 維的 label 放大到 64 x 64 的大小，讓他 有點像是一張新的圖片，在與 image concatenate 在一起，變成 4 個 channel 後輸入到 D 裡面，D 的結構如下，會先把 label 轉變為 1 x 64 x 64，讓他能跟 image concatenate 在一起，在輸入至 discriminator 內， 經過 5 層 CNN 還有 LeakyReLU 跟 batch normalization，最後經過 sigmoid 層，輸出一個 0~1 之間的值，代表他是 real image 或是 fake image 的 機率。
在 Discriminator 中，選擇 LeakyReLU 做為 activation function

```
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.main = nn.Sequential(
                nn.Conv2d(4, 32, 4, 2, 1, bias=False),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Conv2d(32, 64, 4, 2, 1, bias=False),
                nn.BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Conv2d(64, 128, 4, 2, 1, bias=False),
                nn.BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Conv2d(128, 256, 4, 2, 1, bias=False),
                nn.BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True),
                nn.LeakyReLU(0.2, inplace=True),
                nn.Conv2d(256, 1, 4, 1, 0, bias=False),
                nn.Sigmoid()
        )

        self.toinput = nn.Linear(24, 64 * 64, bias=True)

    def forward(self, x, l):
        l = self.toinput(l).view(-1, 1, 64, 64)
        x = torch.cat([x, l], 1)
        return self.main(x).view(-1, 1)
```

補充一些程式碼的問題說明：

(1) feature map（特徵圖）：卷積過程中產生的圖

ngf: Generator 中 feature map 的圖片大小

ndf: Discriminator 中 feature map 的圖片大小

e.g. nn.Conv2d(ndf, ndf * 2, 4, 2, 1, bias=False),

in_channels=ndf

out_channels=ndf * 2

kernel_size=4

stride=1

padding=0 9

(2) noise：一個隨機分佈，給 Generator 產生 fake image
Discriminator 判斷 Generator 用 noise 產生的 fake image，並計算 loss
Discriminator 的 loss 是判斷真圖的 loss + 判斷假圖的
loss errD = errD_real + errD_fake

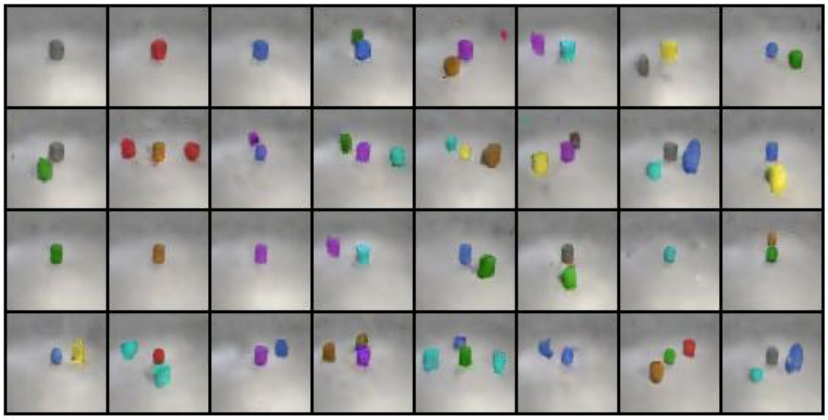
Loss function 的部份 discriminator 使用 nn.BCELoss()，去計算輸入 照片真假預測的錯誤，Generator 的部份也是使用 nn.BCELoss()來想要 用 fake image 騙過 discriminator 的 error，除了這兩個典型的 loss 以 外，還有使用 MSELoss()，來計算 fake image 與 real image 之間的誤 差，因為 generator 在生成照片時用的 condition 跟 real image 相同， 所以 fake image 跟 real image 有對應關係，希望他們可以比較相似， 因為我們的目的不是生成許多不同照片，而是生成對應條件的照片， 所以跟 real image 計算差異想說能給 generator 當作一個參考，而在 infoGAN 的實作下，generator 的確也有比較快學習到 real image 的樣 子，方塊還有顏色會在比較早的 epoch 出現。

– Specify the hyperparameters (learning rate, epochs, etc.) (5%)
batch size: 16
image size: 64 x 64
discriminator learning rate: 0.0002 or 0.0001
generator learning rate: 0.001
condition size: 24
noise size: 64
optimizer: Adam

epoch: 200/300/500

Results and discussion (30%)

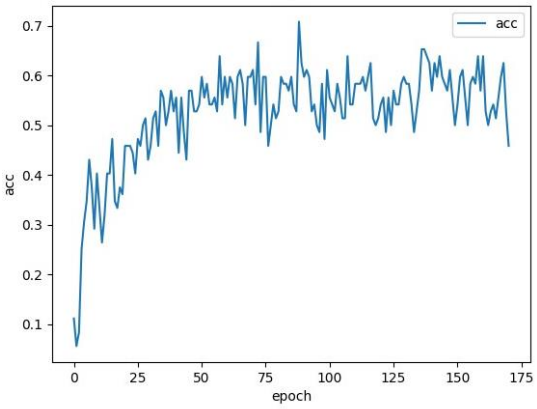
– Show your results based on the testing data. (5%) (including images)

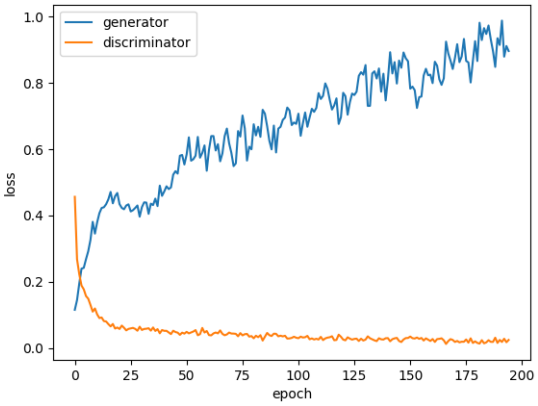


上圖示訓練完 test.jsont 的結果。

```
[88/200] 281/282  loss_g: 0.569  loss_d: 0.415
avg loss_g: 4.336  avg_loss_d: 0.177
testing score: 0.71
```

訓練過程最高有到 0.71





Acc　　Loss

上圖為訓練的 acc 以及 Generator 以及 Discriminator loss 比較圖

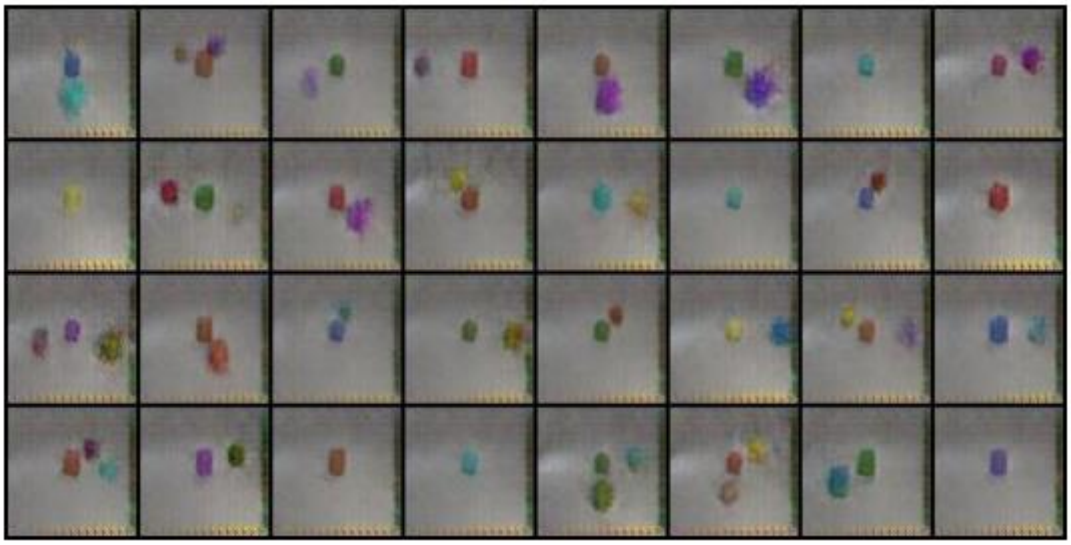
```
g_model.eval()
d_model.eval()
with torch.no_grad():
    gen_imgs=g_model(fixed_z,test_conditions)
score=evaluation_model.eval(gen_imgs,test_conditions)
plot_test_acc.append(score)
if score>best_score:
    best_score=score
    best_model_wts=copy.deepcopy(g_model.state_dict())
    torch.save(best_model_wts,os.path.join('models',f'epoch{epoch}_score{score:.2f}.pt'))
if epoch % 3 == 0:
    Plot_loss(plot_loss_generator, plot_loss_discriminator, None, 0, epoch)
    Plot_acc(plot_test_acc, 0, epoch)
print(f'avg loss_g: {total_loss_g/len(dataloader):.3f}  avg_loss_d: {total_loss_d/len(dataloader):.3f}')
print(f'testing score: {score:.2f}')
print(f'best score: {best_score:.2f}')
print('-----------------------------------------------')
# savefig
save_image(gen_imgs, os.path.join('results', f'epoch{epoch}.png'), nrow=8, normalize=True)
```

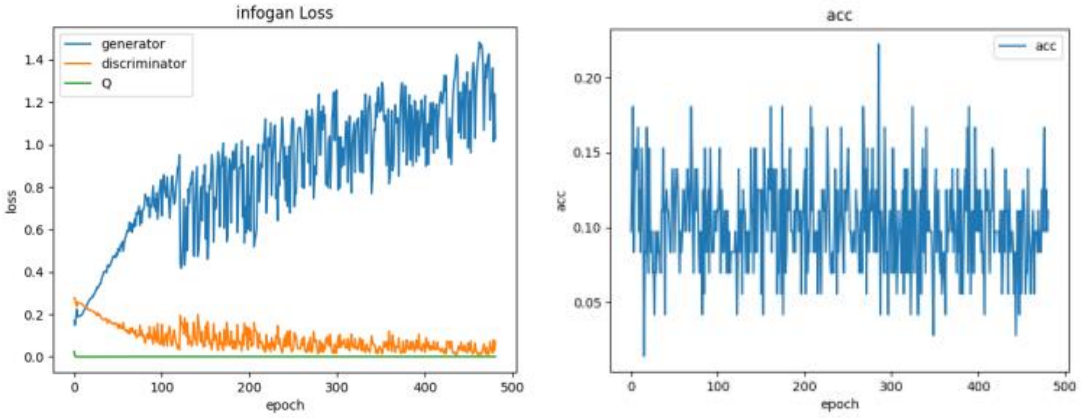
此處 load 已經寫好的 evaluator 來算分數，並於儲存所預測的圖片

– Discuss the results of different models architectures. (25%)

關於特別的 loss，就如我之前提到的有使用 MSELoss 來判斷 real image 跟 fake image 之間的差異，可以讓 model 在比較早的 epoch 就 有 real image 的大概，不會有一堆顏色混在一起不知道是什麼圖像的 image 產生。 不同架構的 model，如我前面嘗試的兩種 model，conditional GAN 以及 info GAN，以我執行的結果來是 conditional GAN 的結果比 info GAN 好很多，對 infoGAN 進行很多改變，在 evaluate 時都無法超過 0.3，而至於 conditional GAN 我在第一個版本 evaluate 就超過 0.4 了， epoch 在 50 左右甚至還到達 0.5，以實作結果來說 conditional GAN 是 比較強的 model 架構。

infoGAN





infoGAN 的 loss 以及 acc，可以看到 acc 幾乎無法超過 0.2，而模 型訓練不起來我一直在想是不是 Discriminator 太強，所以導致 Generator 不管生成什麼照片都會被判斷是 fake，所以它就無法學習 到 real image 的精隨，才會產生不好的結果，不過不斷降低 Discriminator 還是無法成功訓

練 infoGAN，有可能我的程式碼有哪裡寫錯導致，或是 infoGAN 真的不適合在這個 task 上進行。

```
LR_Discriminator = 2e-4
LR_Generator_G = 1e-3
```

在訓練過程中，也發現 Discriminator 的學習速度比 Generator 快太多，以至於無法生出較好的圖，因此把兩者的 learning rate 分開，並依據學習速度給予不同的值，可以得到較好的結果。 在訓練中有查過很多方法，像是，BIG GAN 把 batch size 調成 2048 但是電腦記 憶體不足無法訓練，或是把 Generator 中的 ReLu 全部換為 LeakyReLu， DCGAN 中像是 VGG 的方法更換 kernel size 或是增加層數等等，但是效果都沒 有預期的好，最後的 accuracy 不增反減，最後還是換回最基本的 DCGAN 去做 訓練

2. Experimental results(50%)

test.json

```
[88/200] 281/282  loss_g: 0.569  loss_d: 0.415
avg loss_g: 4.336  avg_loss_d: 0.177
testing score: 0.71
```

new_test.json

```
score: 0.61
score: 0.61
score: 0.58
score: 0.61
score: 0.58
score: 0.62
score: 0.62
score: 0.61
score: 0.61
score: 0.62

avg score: 0.61
```

new test.json 平均 10 次的成績