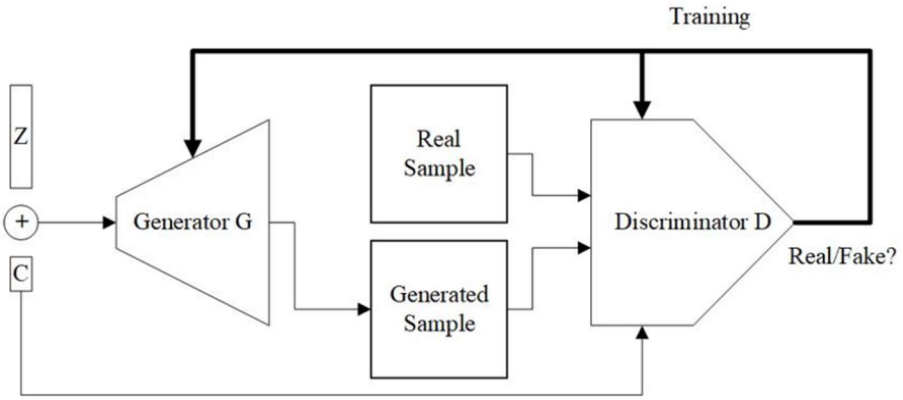
Report Spec

1. Introduction

本次使用 GAN，又稱生成對抗網路。其中整個 neural network，主要分為兩大架構。

1. 生成的部分 Generator G 、 2. 監督的部分 Discriminator D ，整體的邏輯大概是 - 會先有一組希望 Generator 到達的目標 T，然後依造下面的步驟訓練，Generator 生成 GP > Discriminator 取得 GP 將其 label 設為 0 , T 的 label 設為 1，用來使 discriminator 能夠成分辨出 GP 與 T 的差別。再來 Generator 會為了騙過 discriminator 而盡可能的提升自己的能力，以此循環訓練。

而本次 lab 目的是利用 conditional GAN 的方法訓練一個可以生成指定條件的圖片。raining data 為 ICLEVR 的幾何物體圖片，總共有 24 種不同的幾何物體，又 condition 為一個 24-dim 的 vector, 例 : [0,0,0,1,0,0,1,.....0,0,0]

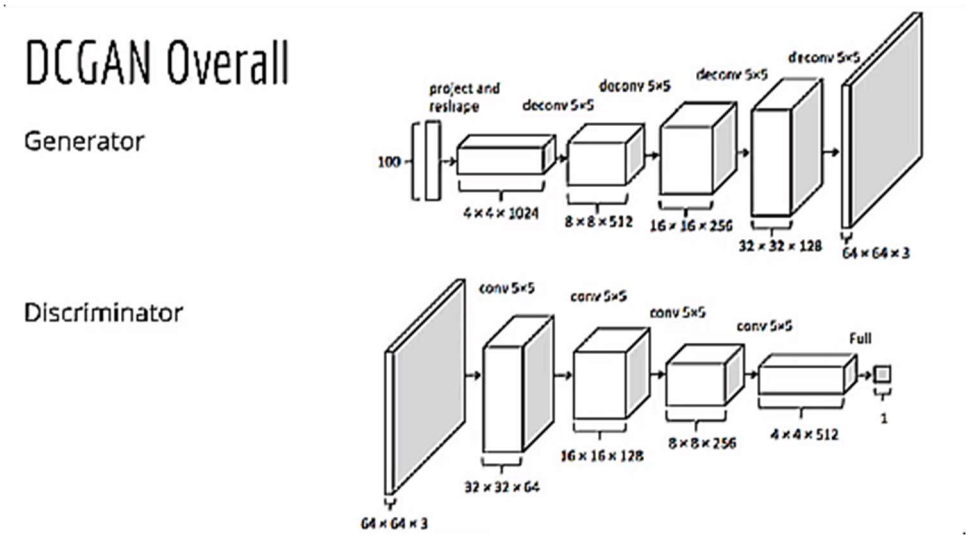


2. Implementation details

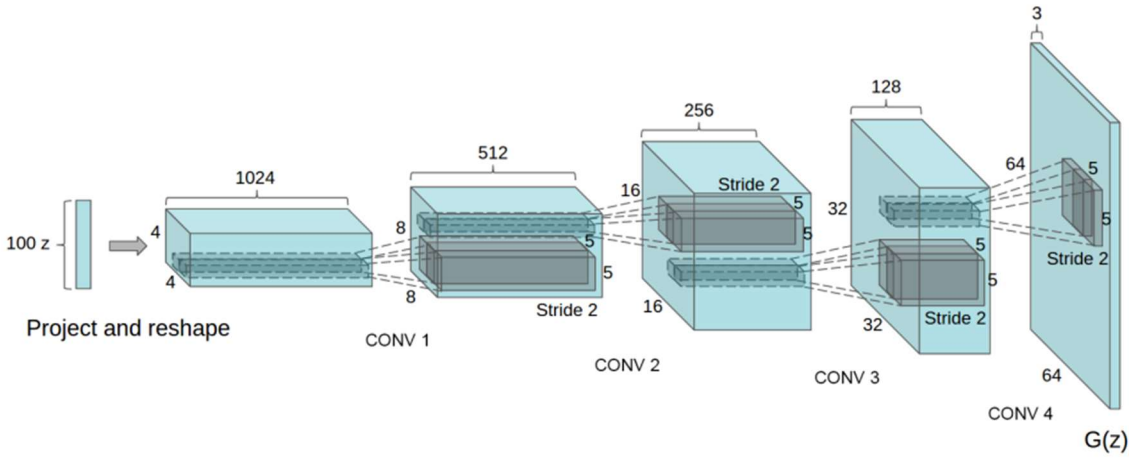
在選擇整個架構的部分，

本次 lab 採用了 DCGAN。算是比較容易且基本的 GAN 架構。

loss function – 選擇了 BCEloss (binary cross entropy)



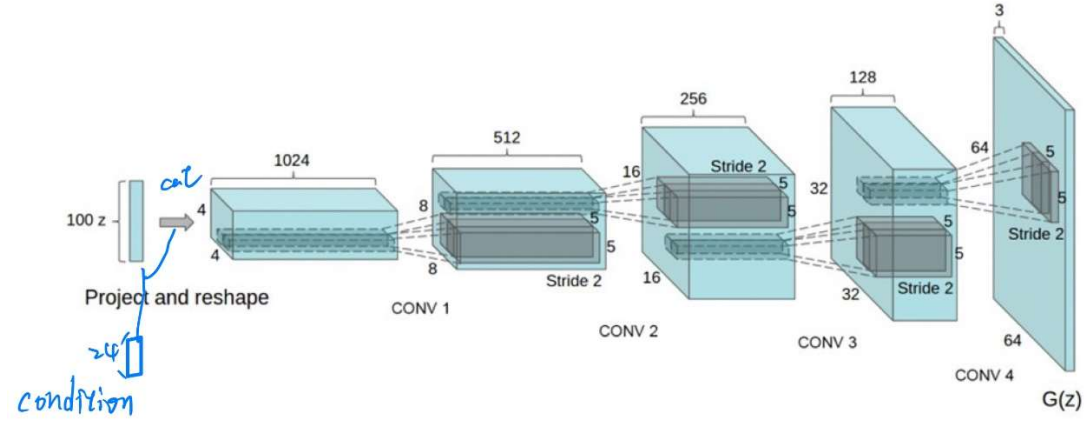
Generator G



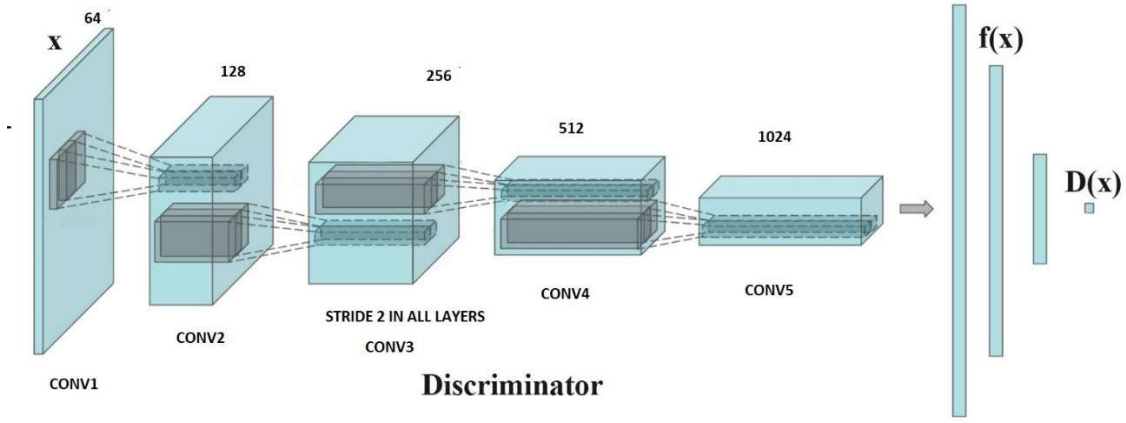

```
self.cotrspos = nn.Sequential(
    nn.ConvTranspose2d(124, 512, 4, 1, 0, bias=False),
    nn.BatchNorm2d(512),
    nn.LeakyReLU(),

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

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

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

    nn.ConvTranspose2d(64, 3, 4, 2, 1, bias=False),
    nn.Tanh()
)
```

1. 整體架構大致上不變，在 input 的部分加入了 condition，這邊是用 torch.cat 的方法 cat 上 normal distribution(100 維的 vector)。
2. 後面使用多層 convtranspose 的方式慢慢把圖片還原出來。
3. 這邊有嘗試使用不同的 convTranspose 架構，總結來說整體每層的架構不要差太多，且 layer 要有漸進的層次感，整體才會有較好的 performance
4. 另外還有重要的一點是需要調整每層 kernel size or stride 甚至是 padding 的部分來達到最後一層是輸出成一張圖片的效果。

我的架構 ：



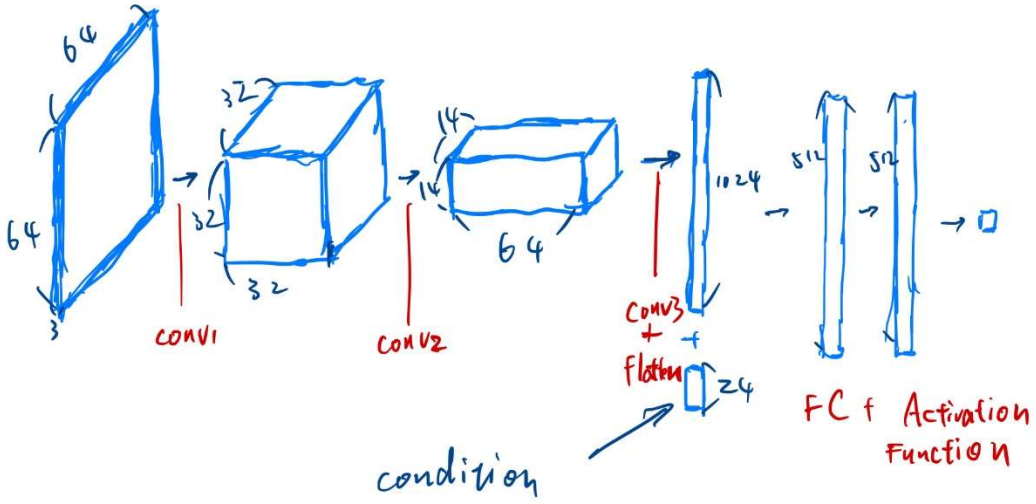
Discriminator D




```
DataParallel(
  (module): Discriminator(
    (conv): Sequential(
      (0): Conv2d(3, 32, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
      (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): Conv2d(32, 64, kernel_size=(8, 8), stride=(2, 2), padding=(1, 1))
      (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (4): LeakyReLU(negative_slope=0.01)
      (5): Conv2d(64, 64, kernel_size=(10, 10), stride=(2, 2), padding=(1, 1))
      (6): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (7): LeakyReLU(negative_slope=0.01)
      (8): Flatten()
    )
    (lin): Sequential(
      (0): Linear(in_features=1048, out_features=512, bias=True)
      (1): LeakyReLU(negative_slope=0.2, inplace=True)
      (2): Linear(in_features=512, out_features=512, bias=True)
      (3): Dropout(p=0.4, inplace=False)
      (4): LeakyReLU(negative_slope=0.2, inplace=True)
      (5): Linear(in_features=512, out_features=512, bias=True)
      (6): Dropout(p=0.4, inplace=False)
      (7): LeakyReLU(negative_slope=0.2, inplace=True)
      (8): Linear(in_features=512, out_features=1, bias=True)
    )
    (sig): Sigmoid()
  )
)
```

1. 這邊我的想法是希望在讀取完 image 的信息後再與 condition cat 起來。
2. 所以我整體架構如左，input img 先用 conv2d 的方法提取 feature map 後，與 condition cat 起來再丟入一個 4 層的 nn 搭配 activation function 架構
3. 另外值得一提的是，若是我 conv2d 的部分設定的太複雜，condition 的部分效果會變得不好，eg 顏色形狀數量分辨的不好
4. 且 full connected 的部分若強度不夠 (layer 太少) 也練不起來。

我的架構：



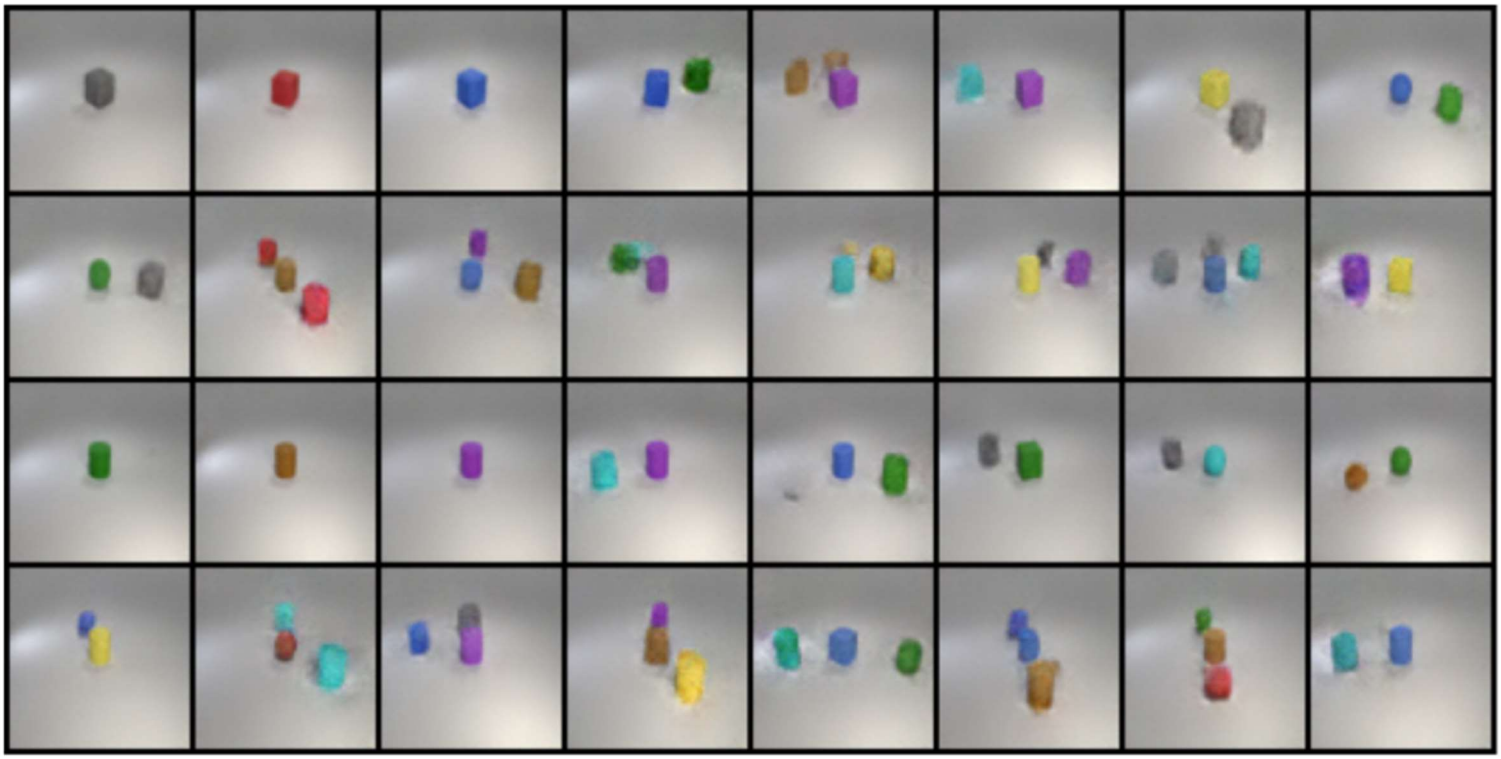
Hyperparameters

batch_size = 32 image_size = 64 num_epochs = 500 lrG = 0.0007 lrD = 0.0002 ngpu = 4 optim = torch.Adam()	Learning Rate 的部分有做出 D 與 G 不同的調整，由於之前設相同時有發現 Generator 的 loss 有降很慢的趨勢，故作出此調整

3. Results and Discussion

Show your results based on the testing data -

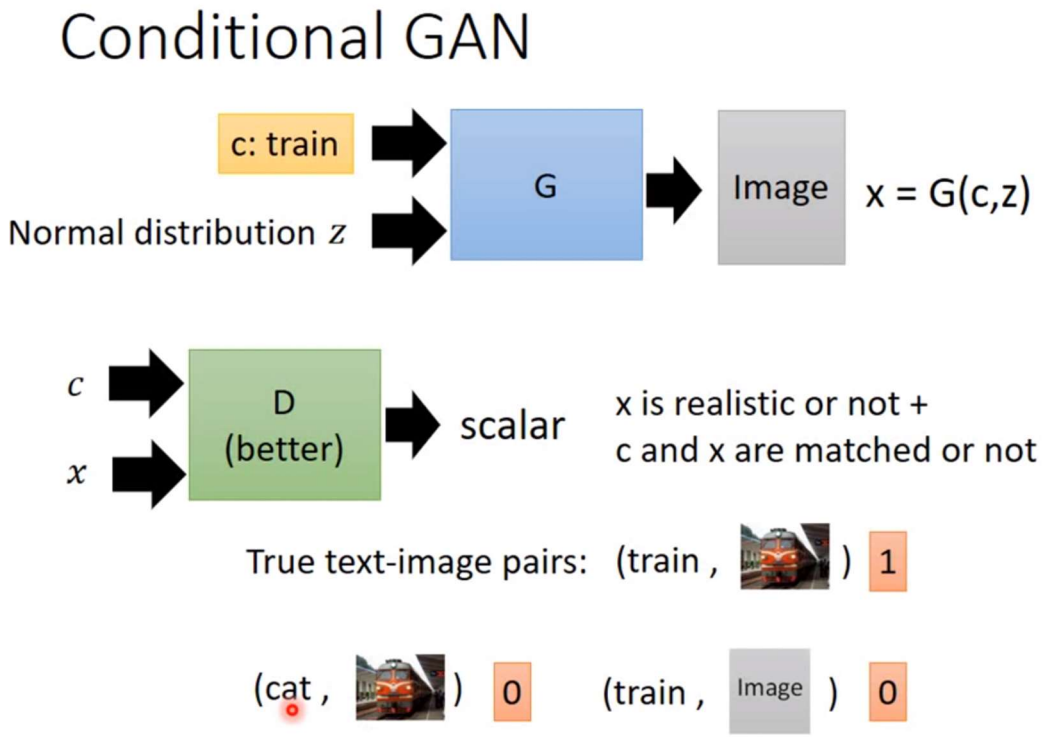
```
[268/600][116/563]      Loss_D: 0.5473  Loss
acc : 76.39
new_max_acc : 83.33
new_max_acc : 84.72
[269/600][ 53/563]      Loss_D: 0.3266  Loss
acc : 84.72
[269/600][553/563]      Loss_D: 0.3565  Loss
```



Discuss the results of different models architectures –

1. 多一個計算訓練 discriminator 的方法

這邊試了一個非常有趣的東西



這邊是我另外聽台大李宏毅教授講解 cGAN 時提到的一個東西，就是通常我們再練 discriminator 的時候。label 不便的前提下，只是將原本對應此 label 的原圖設為 1 且將 Generator 對應此 label 生成的圖設為 0，用這兩個部分的 loss 來做 training。

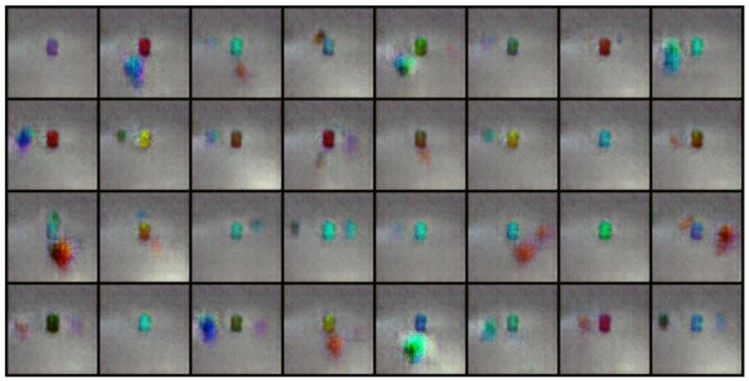
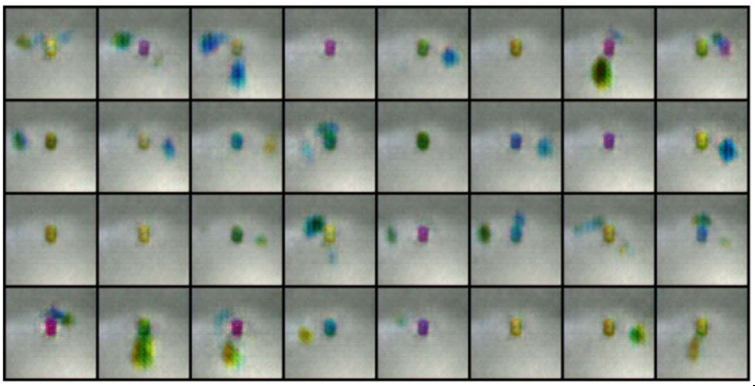
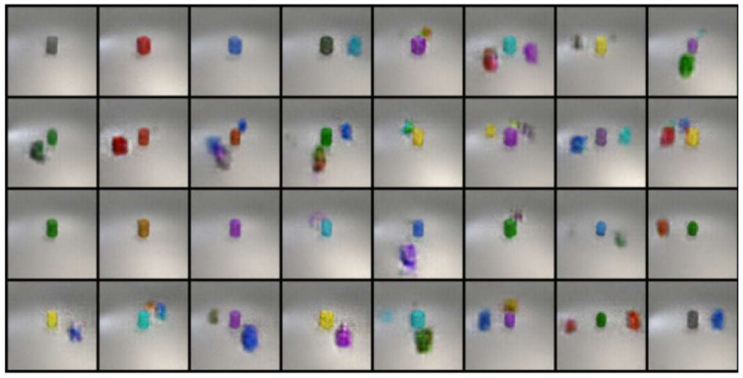
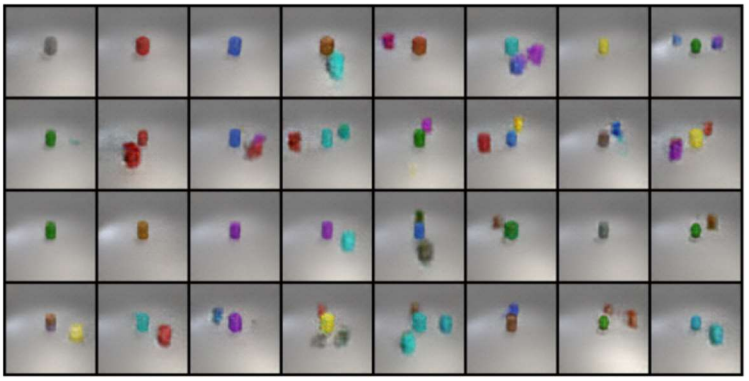
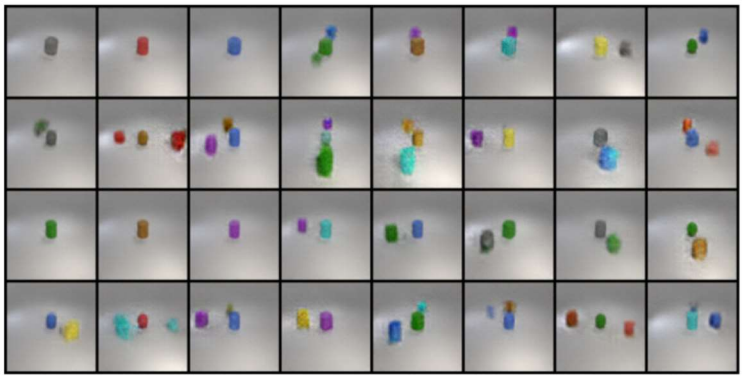
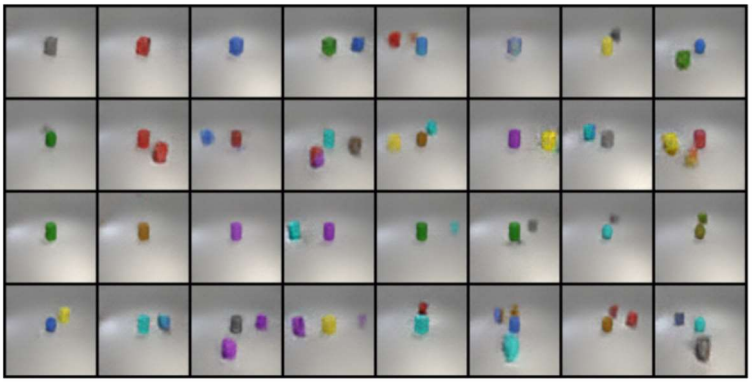
李教授有另外提供一個地方的 loss 需要注意，上圖螢光筆的地方。再丟給正確的圖、錯誤的 label 時，這地方也要需要設為 0。
下為實作的部分 ：

```
def randfake_label(labelloader):
    z = torch.tensor([]).to(device)
    for i in labelloader:
        s = torch.zeros(1,24).to(device, dtype = torch.float)
        for j in range(random.randint(1,3)):
            s[0][random.randint(0,23)] = 1
        while torch.equal(i,s[0]):
            s = torch.zeros(1,24).to(device, dtype = torch.float)
            for j in range(random.randint(1,3)):
                s[0][random.randint(0,23)] = 1

        z = torch.cat((z, s),0)
    return z
```

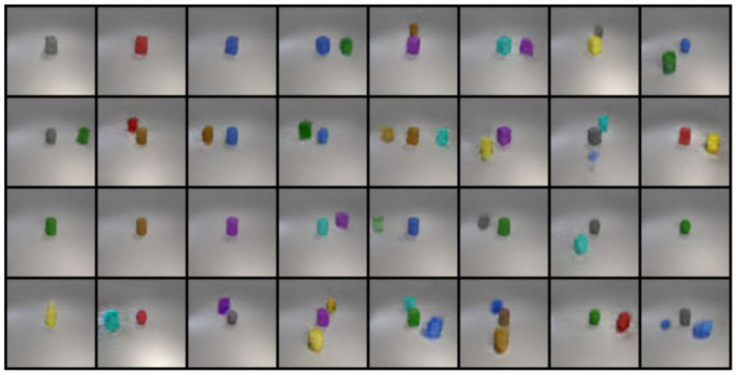
上面為產生錯誤 label 的部分:
由於根據整個 test 與 train 的 data 來看 label 的部分最多不超過三個，所以這邊隨機生成 label 也不超過三個，另外如果剛好生成的 label 是正解的話再隨機生成一次。

```
output = netD(real_cpu, randfake_label(condition)).view(-1)
errD_fake1 = criterion(output, label)
errD_fake1.backward()
```

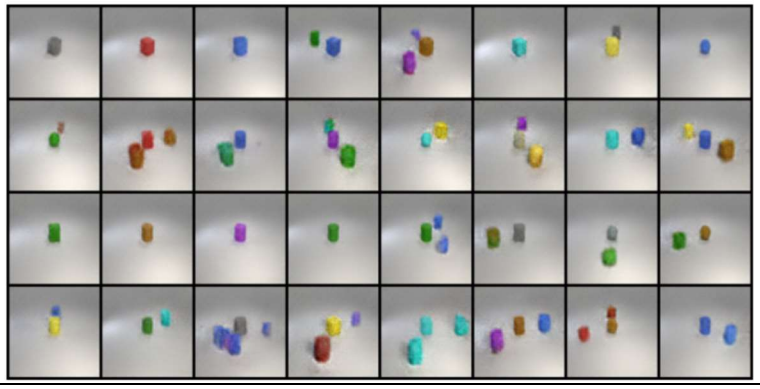
在 train 的部分與其他兩種常見的一樣，丟入 Discriminator 得到 loss
最後，在其他參數完全一樣的情況下，分別各跑了 150 個 epoch，下面是比較的部分。

epoch 1 	epoch 1 
epoch 25 	epoch 25 
epoch 50 	epoch 50 

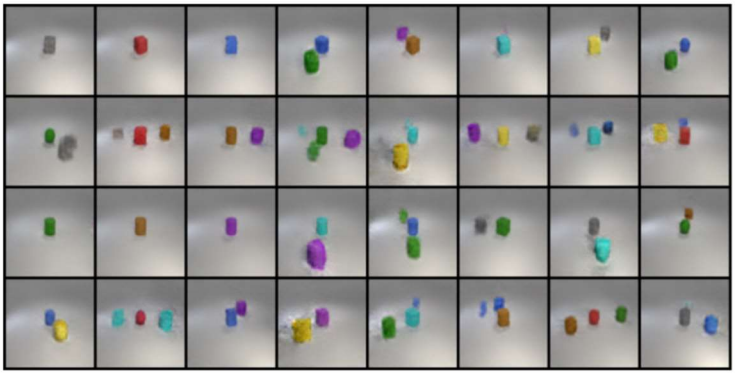
epoch 100



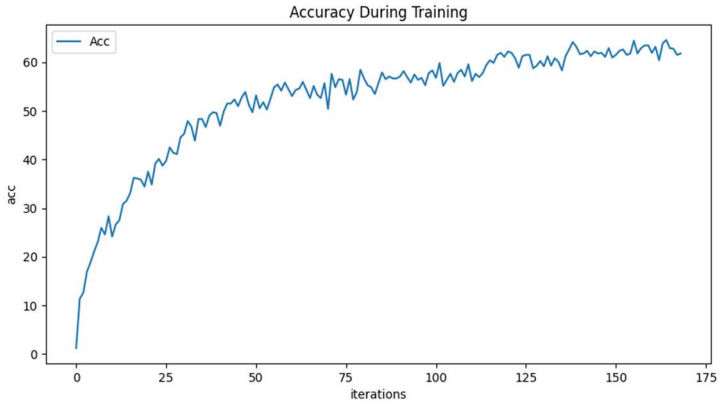
epoch 100

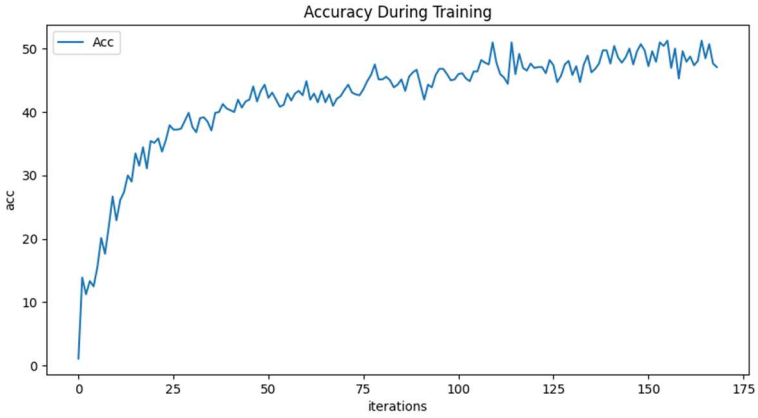


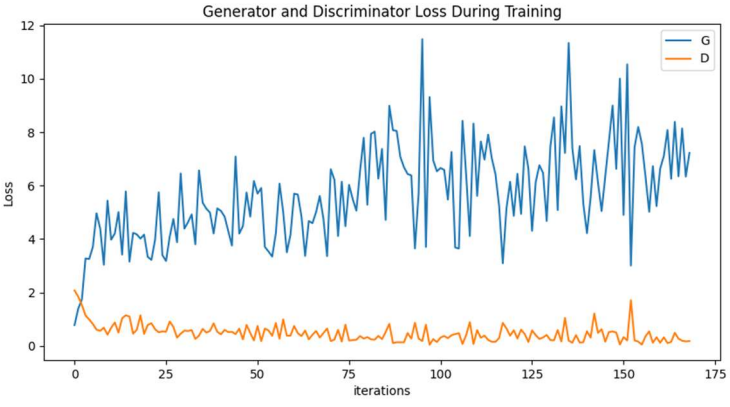
epoch 150

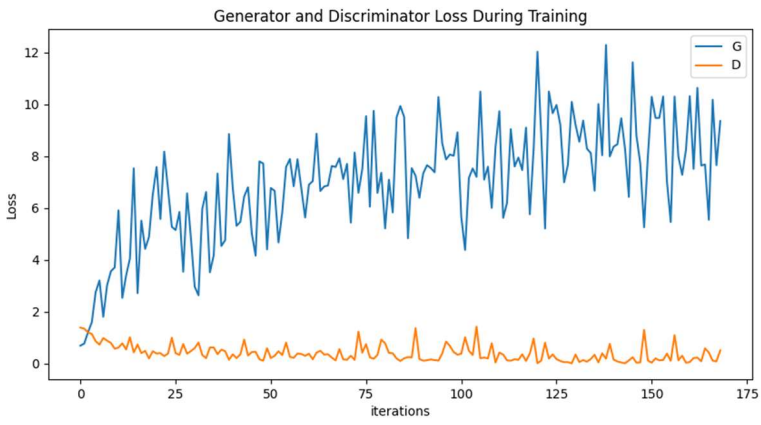


epoch 150









從 acc 的部分更可以看出來差異，左邊已經到達 65 的 acc 了

在真實圖片的部分 分別顯示了 epoch1, 25, 50, 100, 150 可以看到有加此種 Loss 的 model，對於整個物體 顏色、形狀、數量的掌控度，是較沒加此 Loss 的強的

2. 使用較大的 image size 來訓練

另外由於圖片一直很模糊的緣故，想說使用解析度較好的 image，model 可以提取出更多有用的資訊，所以使用了 3*128*128 為 image size 丟進去訓練。
但不知道是否因為圖片較大所以 generator 也需要詳細設計的緣故，整體並沒有我想像的那麼好。甚至 acc 的部分(有 resize64)，也比 3*64*64 來的差

為訓練 500 個 epoch 後產生的圖