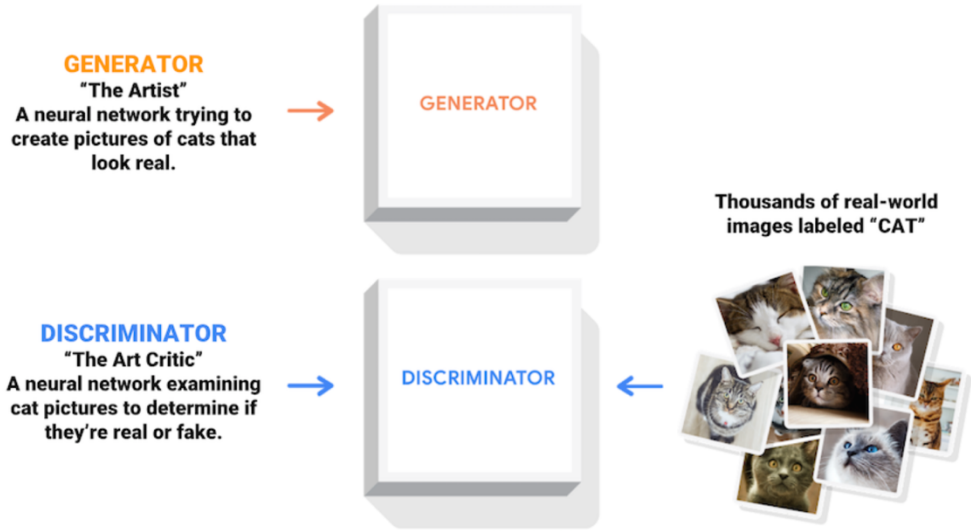
DLP_LAB5_30955124_涂景智_Report

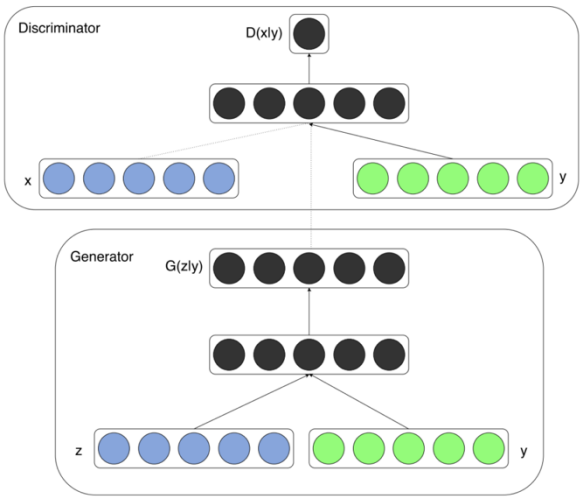
- Introduction

此次的 LAB 是要實作 cGAN (conditional GAN)。GAN 為一種訓練生成模型的方法。它包含了二個模型：「Generator」和「Discriminator」。前者負責生成資料，而後者則負責判別樣本的「真實性」。



至於 cGAN，則是原始 GAN 的擴充。Generator 和 Discriminator 都會額外加入信息（通常是類別信息），使模型可以依我們的期望而生成特定的資料。

如右圖，x 代表我們原始的 data，y 是我們額外加入的信息，z 則是隨機生成的 noise 。在 cGAN 中，我們將原始的 data 連同 data 本身的信息一併加入到 Discriminator 中，並也將隨機生成的 noise 與我們希望生成的類別信息一起丟入我們的 Generator。



- Implement details

A. Describe how you implement your model, including your choice of cGAN, model architectures, and loss functions.

此次 model 的 architecture 只用了最簡單的 conditional DAN。Generator 和 Discriminator 則是用了 DCGAN。整個 Generator 和 Discriminator 的程式分別如下：

```python
class Generator(nn.Module):
    def __init__(self):
        super(Generator, self).__init__()
        self.main = nn.Sequential(
            nn.ConvTranspose2d(nz + nclass, 512, 4, 1, 0, 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, nc, 4, 2, 1, bias=False),
            nn.Tanh(),
        )

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

```python
class Discriminator(nn.Module):
    def __init__(self):
        super(Discriminator, self).__init__()
        self.feature_input = nn.Linear(nclass, 64 * 64)
        self.main = nn.Sequential(
            nn.Conv2d(nc + 1, 64, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(256, 512, 4, 2, 1, bias=False),
            nn.BatchNorm2d(512),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Conv2d(512, 1, 4, 1, 0, bias=False),
        )

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

DCGAN 簡而言之就是用 CNN 得架構去實踐 Generator 和 Discriminator 。左圖中 Generator 的 “nz” 是 latent vector，也是我們隨機生成的 noise 的大小（Generator input）。

另外，本次 model 中的 loss function，是用了原本 GAN 論文中的 loss function：

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[logD(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[log(1-D(G(z)))\right]$$

實作的方式如下圖：

```python
noise.data.resize_(_batch_size, nz, 1, 1).normal_(0, 1)

data = data.to(device)
attr = attr.to(device)

d_real = netD(data, attr)
fake = netG(noise, attr)
d_fake = netD(fake.detach(), attr)

d_loss = criterion(d_real, label_real) + criterion(d_fake, label_fake)
d_loss.backward()
optimizerD.step()

netG.zero_grad()
d_fake = netD(fake, attr)
g_loss = criterion(d_fake, label_real) # trick the fake into being real
g_loss.backward()
optimizerG.step()
```

B. Specify the hyperparameters

1. Batch size = 32
2. latent size = 25 (Generator input size)
3. Learning rate:
   I. Generator: 0.0002
   II. Discriminator: 0.00001
4. Training epochs = 75

- Results and discussion

A. Show your results based on the testing data.

最後 model testing 最好的 accuracy 為 0.5277777777777778，結果如下圖：

The

synthetic images of F1-score of 0.5278

B. Discuss the results of different model architectures.

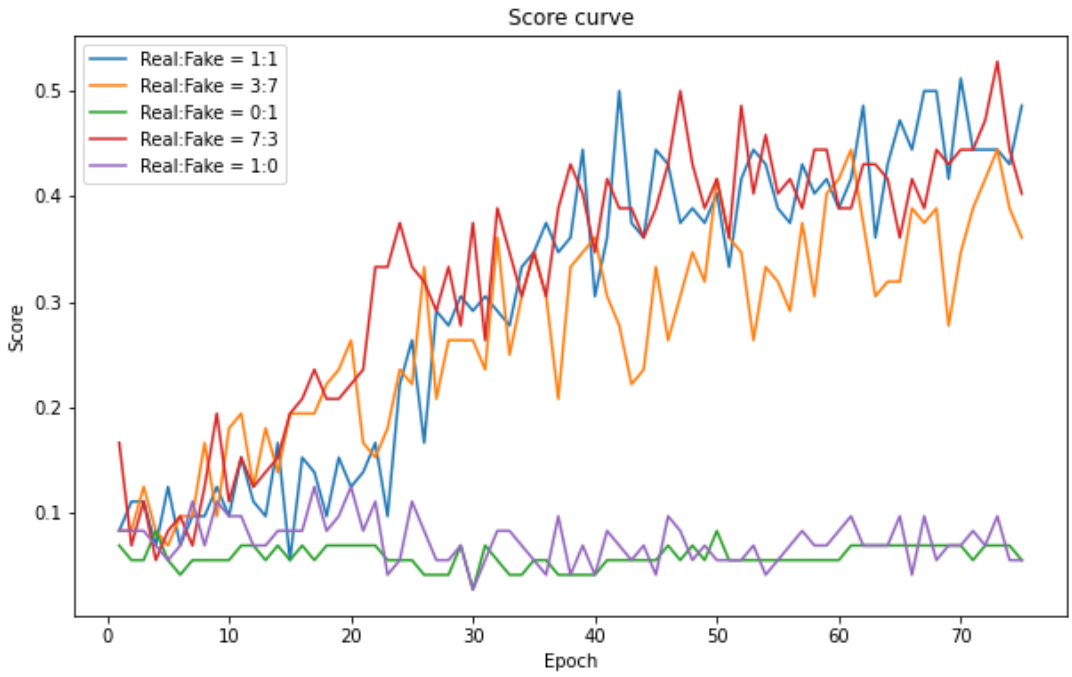
在原先的 model 中，我們 Discriminator 的其實就是一個可以辨別假圖片與真圖片的 binary classification network，而原先的 Discriminator 的 loss function 是長這樣：

```
d_loss = criterion(d_real, label_real) + criterion(d_fake, label_fake)
d_loss.backward()
optimizerD.step()
```

Discriminator 的 loss function，可以解讀為分成二個部分，分別是 real batches 的 loss 和 fake batches 的 loss 。以下我們實驗五種不同 real batches 的 loss 和 fake batches 的 loss 佔總 loss 的比例，並觀察模型表現的好壞。

1. Real : Fake = 1 : 1 （原先比例）
2. Real : Fake = 3 : 7
3. Real : Fake = 0 : 1
4. Real : Fake = 7 : 3
5. Real : Fake = 1 : 0

實驗結果：



觀察實驗結果可以發現，只要任何一個 Real 或 Fake 的 loss 的比例為 0（圖中的紫線跟綠線），模型的效能就無法提高，因為他無法學習辨別「什麼是假資料」和「什麼是真資料」。

另外，比較值得一提的是，我們發現若將真資料的 loss 比例調得比 Generator 生成的假資料的 loss 的比例來得高（對比圖中的紅線跟侷限），則可以稍微提高模型的 performance。這可能意味者，讓模型能夠學習何謂真實的資料，比辨別 Generator 產生的假資料來的稍微重要一些。