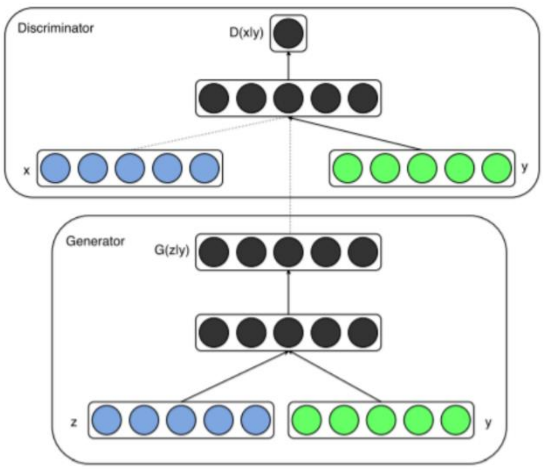
Let’s Play GANs

309552042 網工所 黃敏涓

I. Introduction (5%)



在生成模型（D）和判別模型（G）的建模中均引入條件變量 y（conditional variable y），使用額外信息 y 對模型增加條件，引導數據生成過程。
當條件變量 y 是類別標籤，可以看做 CGAN 是把純無監督的 GAN 變成有監督的模型。

GAN 之 objective function

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

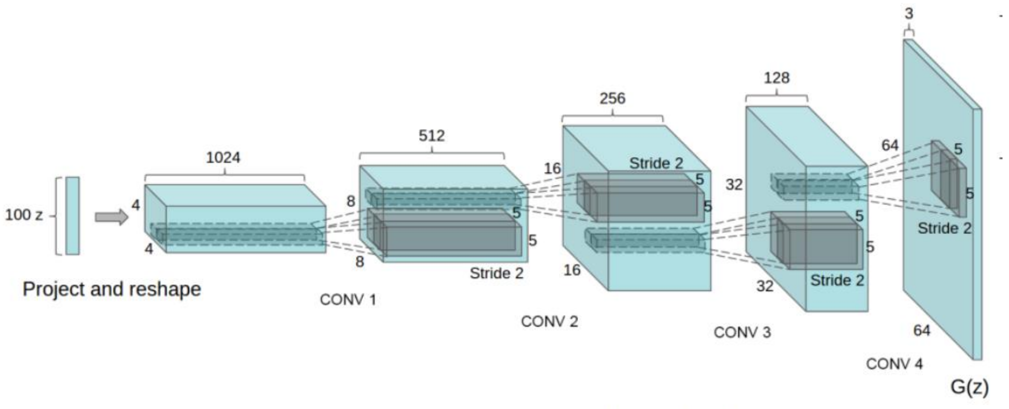
CGAN 之 objective function

$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x}|\boldsymbol{y})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z}|\boldsymbol{y})))].$$

加入 condition 後的 min-max player 問題，Discriminator 希望 maximize logD(x|y)，Generator 希望 maximize D(G(z|y))也就是 minimize log(1-D(G(z|y)))，其中 z 為從 gaussian distribution N(0,1) 中隨機抽取的噪聲。

II. Implementation details (15%)

a. Model Architecture: **DCGAN**

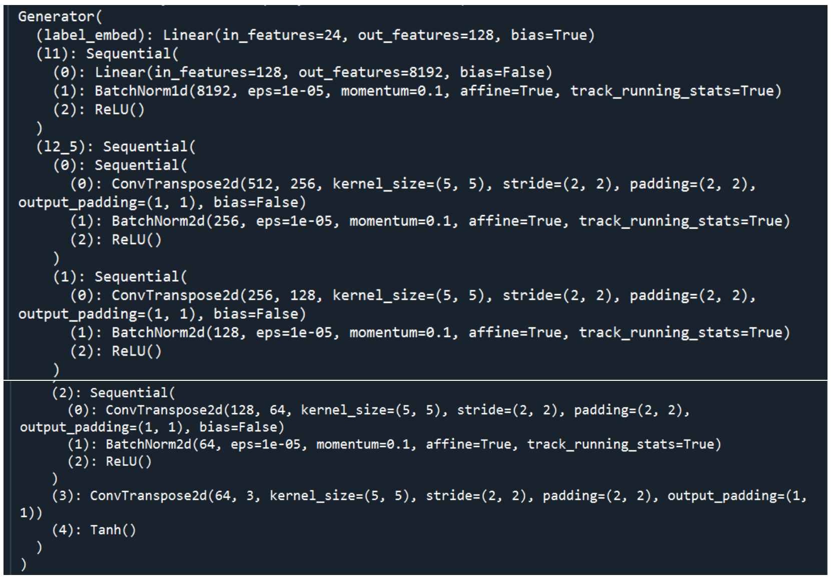


DCGAN 在 Unsupervised Representation Learning with Deep Convolutional 論文中被提出，將 convolution 引進網路結構中，在 discriminator 中輸入的圖像會經

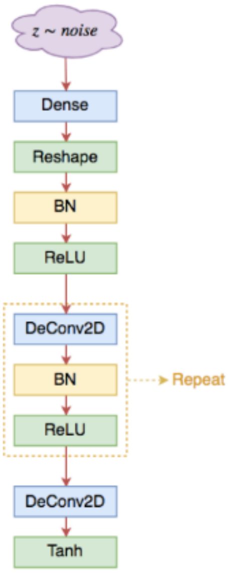
過層層 convolution 之後變成一個預測是否為真實圖片的機率；在 generator 中會把輸入的 z 向量經過層層 deconvolution 並輸出生成的圖。

Generator:

DCGAN 論文中的 Generator 架構，使用 ConvTranspose2d(deep convolution layer)，BatchNorm 和 ReLu 來建構大部分網路架構

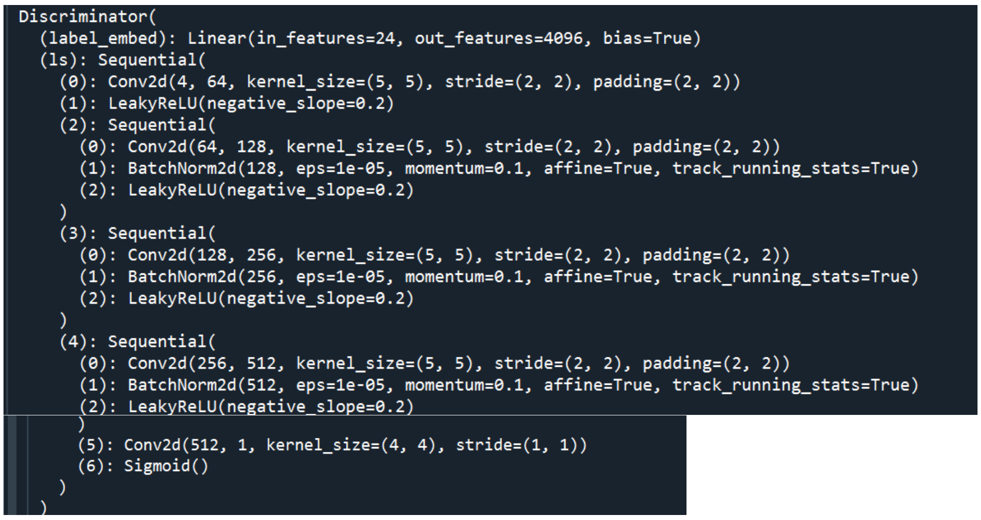


```
Generator(
  (label_embed): Linear(in_features=24, out_features=128, bias=True)
  (l1): Sequential(
    (0): Linear(in_features=128, out_features=8192, bias=False)
    (1): BatchNorm1d(8192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): ReLU()
  )
  (l2_5): Sequential(
    (0): Sequential(
      (0): ConvTranspose2d(512, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2),
output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
    )
    (1): Sequential(
      (0): ConvTranspose2d(256, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2),
output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
    )
    (2): Sequential(
      (0): ConvTranspose2d(128, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2),
output_padding=(1, 1), bias=False)
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): ReLU()
    )
    (3): ConvTranspose2d(64, 3, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2), output_padding=(1,
1))
    (4): Tanh()
  )
)
```

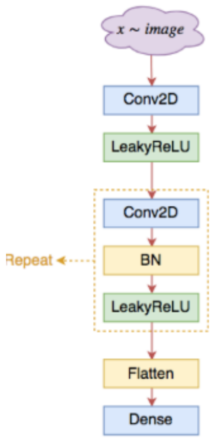


Discriminator:

DCGAN 論文中的 Discriminator 架構，使用 Convolutional layer，BatchNorm 和 LeakyReLu 來建構大部分網路架構

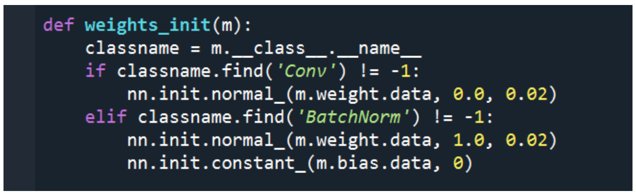


```
Discriminator(
  (label_embed): Linear(in_features=24, out_features=4096, bias=True)
  (ls): Sequential(
    (0): Conv2d(4, 64, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
    (1): LeakyReLU(negative_slope=0.2)
    (2): Sequential(
      (0): Conv2d(64, 128, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (3): Sequential(
      (0): Conv2d(128, 256, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (4): Sequential(
      (0): Conv2d(256, 512, kernel_size=(5, 5), stride=(2, 2), padding=(2, 2))
      (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (5): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1))
    (6): Sigmoid()
  )
)
```



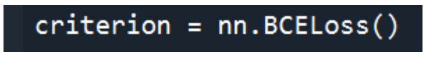
Initialization:

在 DCGAN 原始論文中，網路參數從 N(0,0.02)初始化



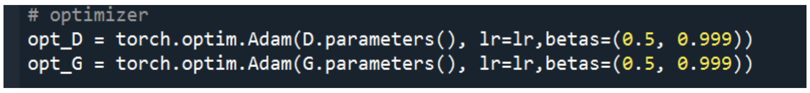
```
def weights_init(m):
    classname = m.__class__.__name__
    if classname.find('Conv') != -1:
        nn.init.normal_(m.weight.data, 0.0, 0.02)
    elif classname.find('BatchNorm') != -1:
        nn.init.normal_(m.weight.data, 1.0, 0.02)
        nn.init.constant_(m.bias.data, 0)
```

Loss Function:



```
criterion = nn.BCELoss()
```

Optimizer:



```
# optimizer
opt_D = torch.optim.Adam(D.parameters(), lr=lr,betas=(0.5, 0.999))
opt_G = torch.optim.Adam(G.parameters(), lr=lr,betas=(0.5, 0.999))
```

b. hyper parameter

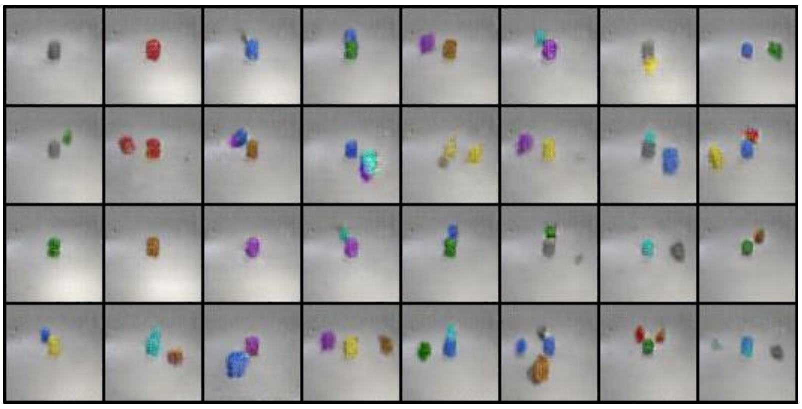
Learning Rate = 1e-4

Epoch = 1000

Batch_Size = 32

III. Results and discussion (20%)

Acc: 0.61



使用 DCGAN 網路，確保生成模型的穩定性，但要成功收斂，需要超參數適當的設定，當學習率太大或 epoch 太少時，模型的 performance 都十分差。