利用深度卷積對抗生成網路(DCGAN)
生成手寫數字

1.資料處理:

```
[3]  (train_images, train_labels), (_, _) = tf.keras.datasets.mnist.load_data()

[4]  train_images = train_images.reshape(train_images.shape[0], 28, 28, 1).astype('float32')
     train_images = (train_images - 127.5) / 127.5
     BUFFER_SIZE = 60000
     BATCH_SIZE = 256

[5]  train_dataset = tf.data.Dataset.from_tensor_slices(train_images).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
```

資料前處理:
將 train_image 的格式設定為 28*28*1 資料型態為 float
BUFFER_size 為 60000
BATCH_size 為 256

2.網路模型:

```
[6]  def generator_model():
         model = tf.keras.Sequential()
         model.add(layers.Dense(7*7*256, use_bias=False, input_shape=(100,)))
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Reshape((7, 7, 256)))
         assert model.output_shape == (None, 7, 7, 256)

         model.add(layers.Conv2DTranspose(128, (5, 5), strides=(1, 1), padding='same', use_bias=False))
         assert model.output_shape == (None, 7, 7, 128)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(64, (5, 5), strides=(2, 2), padding='same', use_bias=False))
         assert model.output_shape == (None, 14, 14, 64)
         model.add(layers.BatchNormalization())
         model.add(layers.LeakyReLU())

         model.add(layers.Conv2DTranspose(1, (5, 5), strides=(2, 2), padding='same', use_bias=False, activation='tanh'))
         assert model.output_shape == (None, 28, 28, 1)

         return model
```

```
[7]  M↓
def discriminator_model():
    model = tf.keras.Sequential()
    model.add(layers.Conv2D(64, (5, 5), strides=(2, 2), padding='same',
                                     input_shape=[28, 28, 1]))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Conv2D(128, (5, 5), strides=(2, 2), padding='same'))
    model.add(layers.LeakyReLU())
    model.add(layers.Dropout(0.3))

    model.add(layers.Flatten())
    model.add(layers.Dense(1))

    return model
```

網路模型:
分為生成模型與判別模型(generator_model and discriminator_model)
使用生成器模型產生隨機 pixel 組合後在利用判別器模型檢查真實度
如果是一張越不符合真實的圖片就將 label 標註數字趨近於 0，
反之，越接近真實則 label 數字趨近於 1。

在此使用的模型由捲積及反捲積組成的，統稱 DCGAN(Deep convolution GAN)

1. 生成器和判別器均不採用池化層，而採用（帶步長的）的捲積層；其中判別器採用普通捲積（Conv2D），而生成器採用反捲積（DeConv2D）
2. 在生成器和判別器上均使用 Batch Normalization
3. 在生成器除輸出層外的所有層上使用 RelU 激活函式，而輸出層使用 Tanh 激活函式
4. 在判別器的所有層上使用 LeakyReLU 激活函式
5. 捲積層之後不使用全連接層
6. 判別器的最後一個捲積層之後也不用 Global Pooling，而是直接 Flatten

3.產生雜訊當輸入:

```
In [8]: generator = generator_model()

        noise = tf.random.normal([1, 100])
        generated_image = generator(noise, training=False)

        plt.imshow(generated_image[0, :, :, 0], cmap='gray')
```

產生一張雜訊並丟進模型內

4.LOSS FUNCTION:

```
[10]  ▷ M↓
cross_entropy = tf.keras.losses.BinaryCrossentropy(from_logits=True)

[11]  ▷ M↓
def discriminator_loss(real_output, fake_output):
    real_loss = cross_entropy(tf.ones_like(real_output), real_output)
    fake_loss = cross_entropy(tf.zeros_like(fake_output), fake_output)
    total_loss = real_loss + fake_loss
    return total_loss

[12]  ▷ M↓
def generator_loss(fake_output):
    return cross_entropy(tf.ones_like(fake_output), fake_output)
```

Discriminator 的 loss 為判定為真實(label=1)的 loss
+判定為假(label=0)的 loss
Generator 的 loss 代表圖片被判別有多真實的 loss

5.Training:

```
[13]  ▷ M↓
generator_optimizer = tf.keras.optimizers.Adam(1e-4)
discriminator_optimizer = tf.keras.optimizers.Adam(1e-4)

[14]  ▷ M↓
EPOCHS = 101
noise_dim = 100
num_examples_to_generate = 16
seed = tf.random.normal([num_examples_to_generate, noise_dim])

[15]  ▷ M↓
def generate_plot_image(gen_model, test_noise):
    pre_images = gen_model(test_noise, training=False)
    fig = plt.figure(figsize=(4, 4))
    for i in range(pre_images.shape[0]):
        plt.subplot(4, 4, i+1)
        plt.imshow((pre_images[i, :, :, 0] + 1)/2, cmap='gray')
        plt.axis('off')
    plt.show()
```

```
[16]
def train_step(input_dim, images):
    noise = tf.random.normal([BATCH_SIZE, input_dim])

    with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
        real_out = discriminator(images, training=True)

        gen_image = generator(noise, training=True)
        fake_out = discriminator(gen_image, training=True)

        gen_loss = generator_loss(fake_out)
        disc_loss = discriminator_loss(real_out, fake_out)
    gradient_gen = gen_tape.gradient(gen_loss, generator.trainable_variables)
    gradient_disc = disc_tape.gradient(disc_loss, discriminator.trainable_variables)
    generator_optimizer.apply_gradients(zip(gradient_gen, generator.trainable_variables))
    discriminator_optimizer.apply_gradients(zip(gradient_disc, discriminator.trainable_variables))
```

設定訓練次數為 101(會設 101 是因為 result 結果要每 10 次的倍數 display 一次)
雜訊的訓練次數設為 100
訓練循環從生成器接收隨機 seed 作為輸入開始。
Seed 用於產生圖像。

6.總結:

1.由於 DCGAN 的生成器中使用了反卷積，而反卷積存在"棋盤效應"，這個棋盤效應約束了 DCGAN 的生成能力上限。
2.DCGAN 的非線性能力不足:輸入圖片大小固定後，整個 DCGAN 架構基本都固定的，包括模型的層數，唯一可以變化的似乎只有卷積核大小，這變成只能改變模型寬度，沒有一種自然而直接的方法增加深度。

3.如果使用 resnet 來當作 gan 的變形來使用的話，效果會更好 GAN 上主流的生成器和判別器架構確實已經變成了 ResNet，但 ResNet 比 DCGAN 要慢得多，所需要的顯存要多得多。基於本人硬體上的不足，沒能使用 resnet 來試，非常可惜。