Lab 7: Let's play GAN

310551083 許嘉倫

1 Introduction

在這項作業中我們需要實作 conditional GAN (cGAN)，根據給定的 multi-label conditions 我們需要合成出相對應的圖片。而在本次作業中，會給定一個 z vector 和 condition vector，將這兩個 vector 輸入 generator model 產生相對應的圖片。整體架構如 figure 1 所示。接著將產生出來的圖片輸入至助教提供的 pre-trained classifier 計算 accuracy。

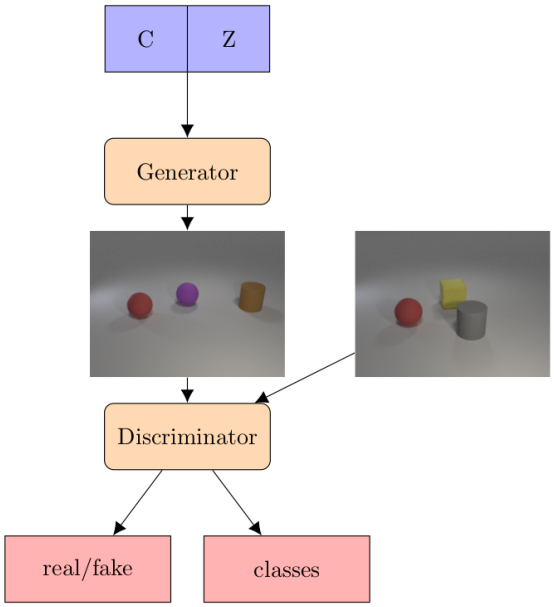


Figure 1: The illustration of cGAN

2 Implementation details

2.1 GAN achitecture

這次作業我選擇的架構是 auxiliary GAN，generator 的部份和 DCGAN 一樣，唯一的差別在於 discriminator。原本 discriminator 的 output 是一個分數用來判斷圖片的真實性，越像真實圖片 output 會越接近1，反之則越接近0。而 auxiliary GAN 則是在 discriminator 多了一個 classifier 的部份，來計算 classification loss。Figure 2 是auxiliary GAN 的 discriminator 架構。

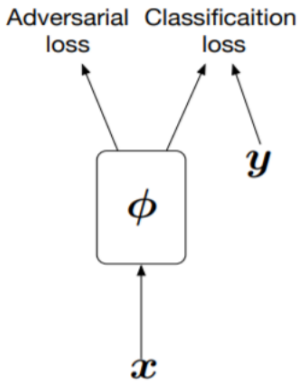


Figure 2: The architecture of auxiliary GAN

2.2 Generator

首先將 condition vector embedding 再與 noise concatenate 在一起，接著輸入 generator 裡產生圖片，而 generator 的架構和 DCGAN 一樣，有五層的 deconvolution layer。並在最後利用 Tanh activation function 將圖片 normalize 到-1 和1 之間。

```
class Generator(nn.Module):
    def __init__(self,args):
        super(Generator, self).__init__()
        self.ngf, self.nc, self.nz = args.ngf, args.nc, args.latent_dim
        self.n_classes = args.n_classes

        # condition embedding
        self.label_emb = nn.Sequential(
            nn.Linear(self.n_classes, self.nc),
            nn.LeakyReLU(0.2, True)
        )
        self.main = nn.Sequential(
            # input is Z, going into a convolution
            nn.ConvTranspose2d(self.nz + self.nc, self.ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(self.ngf * 8),
            nn.ReLU(True),
            # state size. (ngf*8) x 4 x 4
            nn.ConvTranspose2d(self.ngf * 8, self.ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ngf * 4),
            nn.ReLU(True),
            # state size. (ngf*4) x 8 x 8
            nn.ConvTranspose2d(self.ngf * 4, self.ngf * 2, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ngf * 2),
            nn.ReLU(True),
            # state size. (ngf*2) x 16 x 16
            nn.ConvTranspose2d(self.ngf * 2, self.ngf, 4, 2, 1, bias=False),
            nn.BatchNorm2d(self.ngf),
            nn.ReLU(True),
            # state size. (ngf) x 32 x 32
            nn.ConvTranspose2d(self.ngf, 3, 4, 2, 1, bias=False),
```

```
            nn.Tanh()
            # state size. (rgb channel = 3) x 64 x 64
        )

    def forward(self, noise, labels):
        label_emb = self.label_emb(labels).view(-1, self.nc, 1, 1)
        gen_input = torch.cat((label_emb, noise), 1)
        out = self.main(gen_input)
        return out
```

2.3 Discriminator

Discriminator 的部份也很像 DCGAN 的 discriminator，不同的地方在於最後會有兩個 fully connected layer 的分支，一個是判段圖片真實性，一個是 multi-label classifier。

```
class Discriminator(nn.Module):
    def __init__(self, args):
        super(Discriminator, self).__init__()
        self.ndf, self.nc = args.ndf, args.nc
        self.n_classes = args.n_classes
        self.img_size = args.img_size
        self.main = nn.Sequential(
            # input is (rgb chnannel = 3) x 64 x 64
            nn.Conv2d(3, self.ndf, 3, 2, 1, bias=False),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
            # state size. (ndf) x 32 x 32
            nn.Conv2d(self.ndf, self.ndf * 2, 3, 1, 0, bias=False),
            nn.BatchNorm2d(self.ndf * 2),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
            # state size. (ndf*2) x 30 x 30
            nn.Conv2d(self.ndf * 2, self.ndf * 4, 3, 2, 1, bias=False),
            nn.BatchNorm2d(self.ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
            # state size. (ndf*4) x 16 x 16
            nn.Conv2d(self.ndf * 4, self.ndf * 8, 3, 1, 0, bias=False),
            nn.BatchNorm2d(self.ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
            # state size. (ndf*8) x 14 x 14
            nn.Conv2d(self.ndf * 8, self.ndf * 16, 3, 2, 1, bias=False),
            nn.BatchNorm2d(self.ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout(0.5, inplace=False),
            # state size (ndf*16) x 8 x 8
            nn.Conv2d(self.ndf * 16, self.ndf * 32, 3, 1, 0, bias=False),
            nn.BatchNorm2d(self.ndf * 32),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            nn.Dropout(0.5, inplace=False),

        )

        # discriminator fc
        self.fc_dis = nn.Sequential(
            nn.Linear(5*5*self.ndf*32, 1),
            nn.Sigmoid()
        )
        # aux-classifier fc
        self.fc_aux = nn.Sequential(
            nn.Linear(5*5*self.ndf*32, self.n_classes),
            nn.Sigmoid()
        )

    def forward(self, input):
        conv = self.main(input)
        flat = conv.view(-1, 5*5*self.ndf*32)
        fc_dis = self.fc_dis(flat).view(-1, 1).squeeze(1)
        fc_aux = self.fc_aux(flat)
        return fc_dis, fc_aux
```

2.4 Loss function

在 loss function 的部份除了原本的 adversarial loos外還需要加上 multi-label classification loss。所以實作上會利用兩個 binary cross entropy 來分別計算 adversarial loss 和 classification loss。其中為了讓分類結果較好，我在 classification loss 前乘上 aux weight 讓 accuracy 更高。

```
for i, (image, cond) in enumerate(train_dataloader):
    ############################
    # (1) Update Discriminator: maximize log(D(x)) + log(1 - D(G(z)))
    ###########################
    optimizer_D.zero_grad()
    # train with real
    real_image = image.to(device)
    cond = cond.to(device)
    batch_size = image.size(0)
    # Use soft and noisy labels [0.7, 1.0]. Salimans et. al. 2016
    real_label = ((1.0 - 0.7) * torch.rand(batch_size) + 0.7).to(device)
    aux_label = cond

    # train with fake
    noise = torch.randn(batch_size, args.latent_dim, 1, 1, device=device)
    fake_image = generator(noise, aux_label)
    # Use soft and noisy labels [0.0, 0.3]. Salimans et. al. 2016
    fake_label = ((0.3 - 0.0) * torch.rand(batch_size) + 0.0).to(device)

    # occasionally flip the labels when training the discriminator
    if random.random() < 0.1:
        real_label, fake_label = fake_label, real_label
```

```
    dis_output, aux_output = discriminator(real_image)
    dis_errD_real = dis_criterion(dis_output, real_label)
    aux_errD_real = aux_criterion(aux_output, aux_label)
    errD_real = dis_errD_real + args.aux_weight * aux_errD_real
    errD_real.backward()
    D_x = dis_output.mean().item()
    # compute the current classification accuracy
    accuracy = compute_acc(aux_output, aux_label)

    dis_output, aux_output = discriminator(fake_image.detach())
    dis_errD_fake = dis_criterion(dis_output, fake_label)
    aux_errD_fake = aux_criterion(aux_output, aux_label)
    errD_fake = dis_errD_fake + args.aux_weight * aux_errD_fake
    errD_fake.backward()
    D_G_z1 = dis_output.mean().item()

    errD = errD_real + errD_fake

    optimizer_D.step()

    ############################
    # (2) Update Generator: maximize log(D(G(z)))
    ###########################
    for _ in range(args.dis_iters):
        optimizer_G.zero_grad()
        noise = torch.randn(batch_size, args.latent_dim, 1, 1, device=device)
        fake_image = generator(noise, aux_label)
        generator_label = torch.ones(batch_size).to(device)  # fake labels are real for generato
        dis_output, aux_output = discriminator(fake_image)
        dis_errG = dis_criterion(dis_output, generator_label)
        aux_errG = aux_criterion(aux_output, aux_label)
        errG = dis_errG + args.aux_weight * aux_errG
        errG.backward()
        D_G_z2 = dis_output.mean().item()
        optimizer_G.step()
```

2.5 Hyperparameters

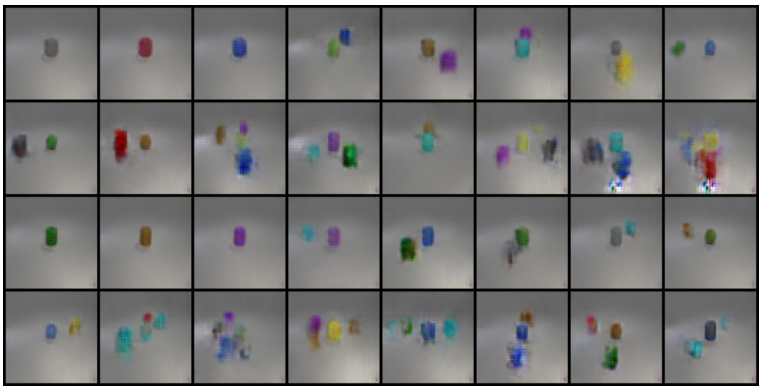
以下是 ACGAN 的 hyperparameters:

- epochs: 300
- learning rate: 2e-4
- aux weight: 100
- adversarial loss: binary cross entropy
- classification loss: binary cross entropy

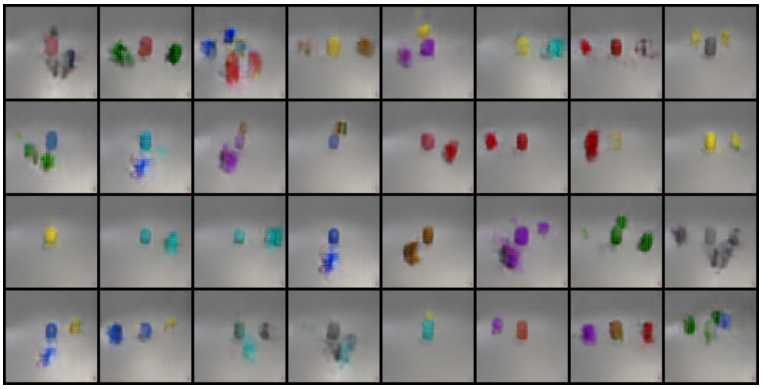
3 Results and discussion

3.1 Show your results based on the testing data

	test.json	new_test.json
accuracy	84.31%	78.57%



(a) ACGAN test.json



(b) ACGAN new_test.json

Figure 3: ACGAN test.json and new_test.json

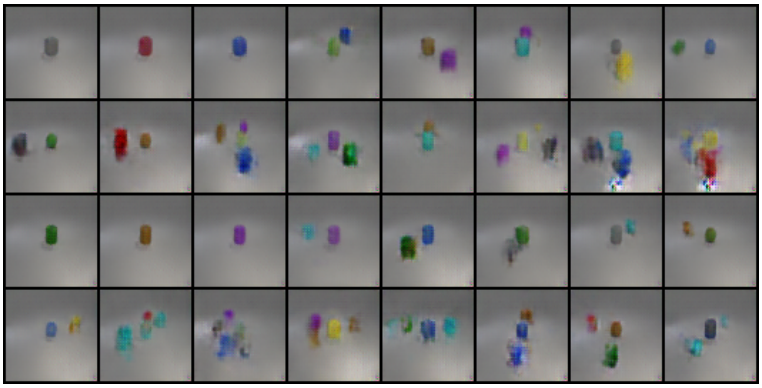
3.2 Discuss the results of different models architectures

此外我也有比較 DCGAN 和 ACGAN 的差異。如 figure 4 所示，我們可以清楚的看到 DCGAN 的圖片效果看起來比 ACGAN 好，但實際在計算 accuracy 時分數卻比較低。那會有這樣的現象可能是因為我在 classification loss 前乘上一個 aux weight，讓 model 專注學習分類的效果，而導致圖片比較不真實。不過這樣的方法卻可以讓我在 multi-label accuracy 得到更好的表現。

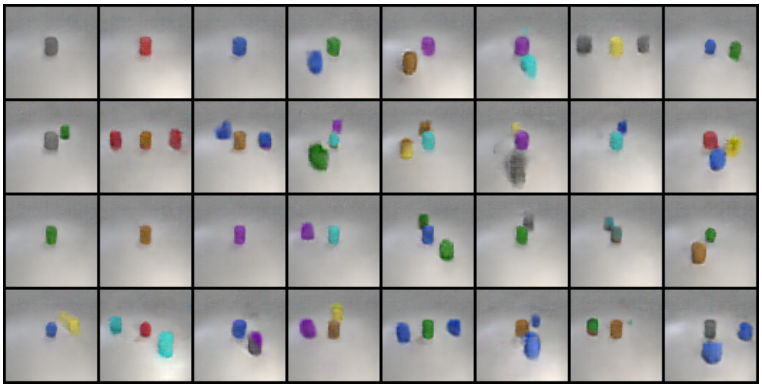
	ACGAN	DCGAN
accuracy	84.31%	69.44%

另外我也有比較 aux weight 不同時也有不同的結果，如 figure 5所示，其中 aux weight 100 可以得到最高 accuracy。

aux weight	1	24	100
accuracy	37.5%	63.89%	84.31%

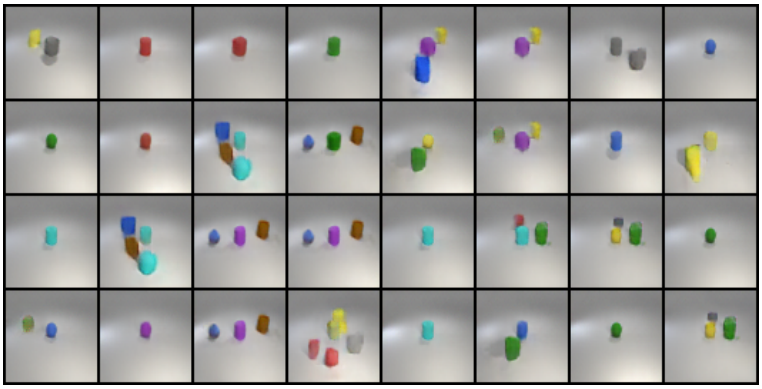


(a) ACGAN test.json

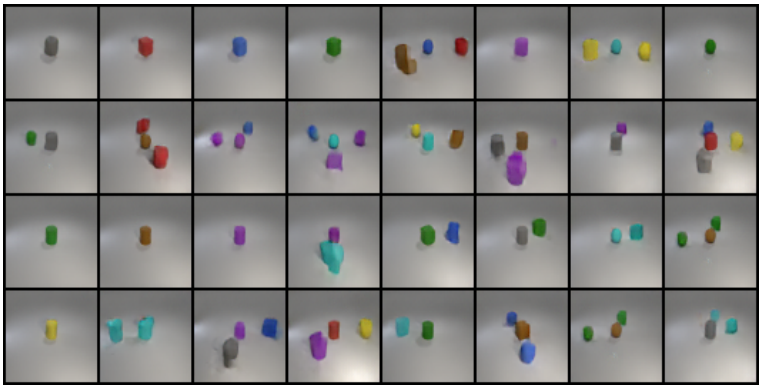


(b) DCGAN test.json

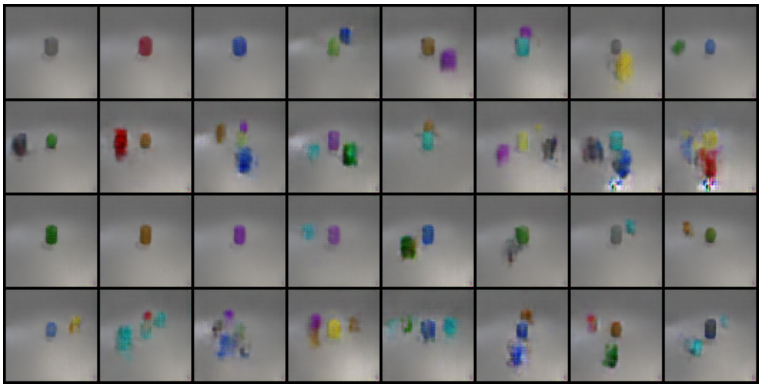
Figure 4: ACGAN test.json and DCGAN test.json



(a) aux weight = 1, accuracy = 37.5%



(b) aux weight = 24, accuracy = 63.89%



(c) aux weight = 100, accuracy = 84.31%

Figure 5: Different aux weight of classification loss