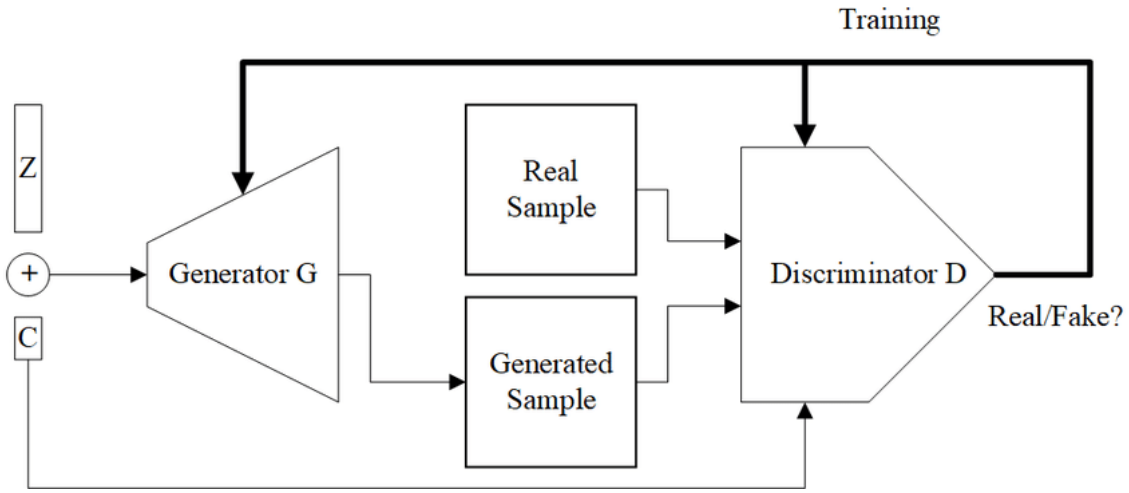
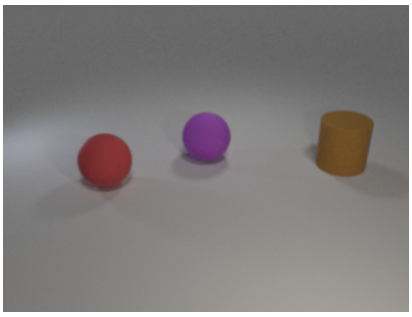
1. Introduction

此次 Lab 的目標是訓練 Condition GAN，利用訓練好的 Generator 加上特定的 condition 以產生我們想要的圖片。Training data 是名為 ICLEVR 的幾何物體圖片，一共有 24 種不同的幾何物體，因此我們的 condition 會是一個 24-dim 的 one hot vector，且一次最多不會超過三種物體，也就是說 condition 的 24-dim 中只會有 1~3 個 1 出現。





2. Implementation details

Generator

因為 GAN 是以難訓練而聞名，所以選擇最簡單的 cDCGAN 來實作此次 Lab 的模型架構。（模型架構是參考 PyTorch 官網來實作的）

首先 Generator 在產生圖片時，會先將 condition vector 先經過一個 Fully Connected 從 24-dim mapping 到 c_dim 的維度，接著將 sample 出來的 latent vector 與 condition vector 串接在一起，變成一個 c_dim + z_dim 維度大小的 vector，然後才會將此 vector 當成 Generator 的輸入，經過五層的 Convolution（BatchNorm and ReLU fucntion）去產生 Fake images。

```
def __init__(self,z_dim,c_dim):
    super(Generator,self).__init__()
    self.z_dim=z_dim
    self.c_dim=c_dim
    self.conditionExpand=nn.Sequential(
        nn.Linear(24,c_dim),
        nn.ReLU()
    )
    kernel_size=(4,4)
    channels=[z_dim+c_dim,512,256,128,64]
    paddings=[(0,0),(1,1),(1,1),(1,1)]
    for i in range(1,len(channels)):
        setattr(self,'convT'+str(i),nn.Sequential(
            nn.ConvTranspose2d(channels[i-1],channels[i],kernel_size, \
                               tride=(2,2),padding=paddings[i-1]),
            nn.BatchNorm2d(channels[i]),
            nn.ReLU()
        ))
    self.convT5=nn.ConvTranspose2d(64,3,kernel_size,stride=(2,2),padding=(1,1))
    self.tanh=nn.Tanh()

def forward(self,z,c):
    """
    :param z: (batch_size,100) tensor
    :param c: (batch_size,24) tensor
    :return: (batch_size,3,64,64) tensor
    """
    z=z.view(-1,self.z_dim,1,1)
    c=self.conditionExpand(c).view(-1,self.c_dim,1,1)
    out=torch.cat((z,c),dim=1)  # become(N,z_dim+c_dim,1,1)
    out=self.convT1(out)  # become(N,512,4,4)
    out=self.convT2(out)  # become(N,256,8,8)
    out=self.convT3(out)  # become(N,128,16,16)
    out=self.convT4(out)  # become(N,64,32,32)
    out=self.convT5(out)  # become(N,3,64,64)
    out=self.tanh(out)    # output value between [-1,+1]
    return out
```

Discriminator

Discriminator 則會將 condition 經過一層 Fully Connected 從 24-dim mapping 到 1*64*64，然後與圖片（Generator 產生或真實的圖片）進行 concatenate 變成 4*64*64 的 vector，一樣在經過五層的 Convolution（BatchNorm and LeakyReLU function）之後，Discriminator 會輸出 0 or 1 分別代表輸入進來的圖片是假的或是真實的。

```python
def __init__(self,img_shape,c_dim):
    super(Discriminator, self).__init__()
    self.H,self.W,self.C=img_shape
    self.conditionExpand=nn.Sequential(
        nn.Linear(24,self.H*self.W*1),
        nn.LeakyReLU()
    )
    kernel_size=(4,4)
    channels=[4,64,128,256,512]
    for i in range(1,len(channels)):
        setattr(self,'conv'+str(i),nn.Sequential(
            nn.Conv2d(channels[i-1],channels[i],\
                      kernel_size,stride=(2,2),padding=(1,1)),
            nn.BatchNorm2d(channels[i]),
            nn.LeakyReLU()
        ))
    self.conv5=nn.Conv2d(512,1,kernel_size,stride=(1,1))
    self.sigmoid=nn.Sigmoid()

def forward(self,X,c):
    """
    :param X: (batch_size,3,64,64) tensor
    :param c: (batch_size,24) tensor
    :return: (batch_size) tensor
    """
    c=self.conditionExpand(c).view(-1,1,self.H,self.W)
    out=torch.cat((X,c),dim=1)  # become(N,4,64,64)
    out=self.conv1(out)  # become(N,64,32,32)
    out=self.conv2(out)  # become(N,128,16,16)
    out=self.conv3(out)  # become(N,256,8,8)
    out=self.conv4(out)  # become(N,512,4,4)
    out=self.conv5(out)  # become(N,1,1,1)
    out=self.sigmoid(out)  # output value between [0,1]
    out=out.view(-1)
    return out
```

Training

訓練時會區分成兩個區塊，先訓練 Discriminator 再來訓練 Generator，而根據助教提供的 Training tips，在訓練 Discriminator 時要將 Real images 跟 Fake images 分開來才會訓練得比較好，所以我們會先將 Real images 與 condition 一起送進 Discriminator，並將預測出來的二分類結果使用 Binary Cross Entropy 計算 loss_real，然後才會使用 Generator 將隨機 sample 的 latent vector 以及 condition 送進去以產生 Fake images，再將 Fake images 與 condition 一起送進 Discriminator，再一次將預測出來的二分類結果使用 Binary Cross Entropy 計算

loss_fake，將 loss_real + loss_fake 當成我們最後要進行 Backpropagation 的目標，但這時要注意，我們要在產生的 Fake images 加上.detach() 避免 BP 時跟新到 Generator 內的參數。

訓練 Generator 時，會將 sample 出的 latent vector 以及 condition 送進去 Generator 以產生出 Fake images，但不同的是，我們會希望 Generator 產生出的照片能夠與真實的照片越接近越好，所以我們會讓 Generator 產生出的二分類結果與真實照片去計算 Binary Cross Entropy，並且希望此 loss 能夠越小越好。

這邊還有一個特別的點是，在正確的演算法中，Generator 應該要跟新一直到 merge 才會再繼續跟新 Discriminator，但是這樣做顯然會花非常多時間，所以我們只選擇讓 Generator 做四次跟新，期盼這樣做能夠逼近真正收斂的參數。

```
for i,(images,conditions) in enumerate(dataloader):
    g_model.train()
    d_model.train()
    batch_size=len(images)
    images = images.to(device)
    conditions = conditions.to(device)

    real = torch.ones(batch_size).to(device)
    fake = torch.zeros(batch_size).to(device)
    """
    train discriminator
    """
    optimizer_d.zero_grad()

    # for real images
    predicts = d_model(images, conditions)
    loss_real = Criterion(predicts, real)
    # for fake images
    z = torch.randn(batch_size, z_dim).to(device)
    # 產生fake imgae時，隨機生成latent vector但condition不是隨機生成，
    # 而是拿training data中已經有的condition，
    # 自己隨機生成的condition vector(24dim中會有1~3個1)反而會train壞掉。
    gen_imgs = g_model(z,conditions)
    # gen_imgs.detach() 是為了不要讓BP去跟新到Generator的參數
    predicts = d_model(gen_imgs.detach(), conditions)
    loss_fake = Criterion(predicts, fake)
    # bp
    loss_d = loss_real + loss_fake
    loss_d.backward()
    optimizer_d.step()
```

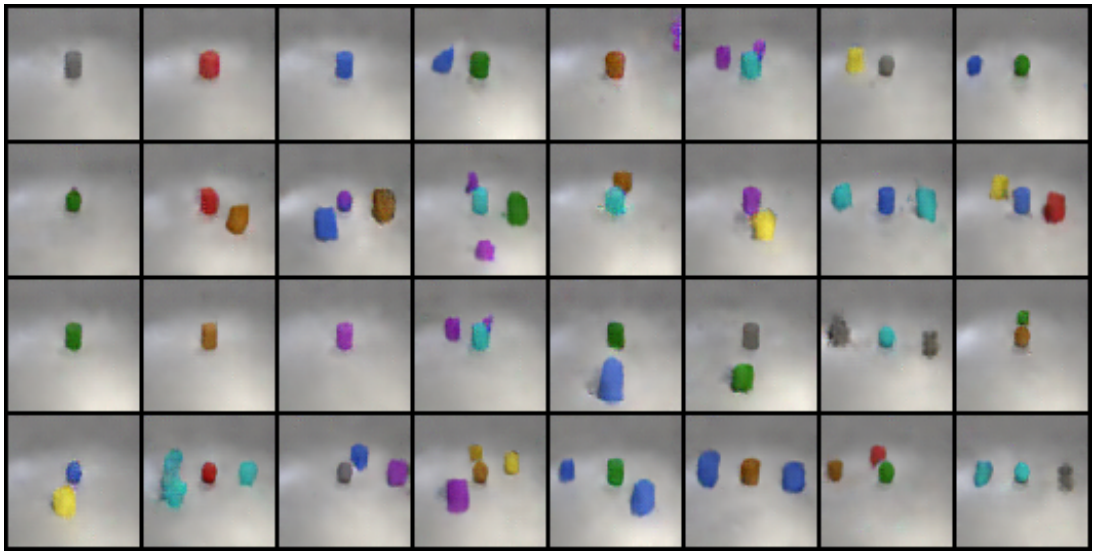
```
"""
train generator
"""
for _ in range(4):
    optimizer_g.zero_grad()

    z = torch.randn(batch_size,z_dim).to(device)
    gen_imgs = g_model(z, conditions)
    predicts = d_model(gen_imgs,conditions)
    loss_g = Criterion(predicts,real)
    # bp
    loss_g.backward()
    optimizer_g.step()
```

3. Results and Discussion

Results

底下分別為 test.json 以及 new_test.json 的結果，以及產生出來的圖片，在 test.json 上可以達到 0.71，但是在 new_test.json 上平均來說只有 0.64 的分數。

testing score: 0.71



```
testing score: 0.62
testing score: 0.64
testing score: 0.64
testing score: 0.62
testing score: 0.63
testing score: 0.65
testing score: 0.65
testing score: 0.64
testing score: 0.65
testing score: 0.63
testing score: 0.62
```

Discussion

1. 雖然助教提供的 tips 中提到應該要用 LeakyReLU 代替 ReLU，但是經過測試，在 Generator 中使用 ReLU，Discriminator 中使用 LeakyReLU 能夠獲得比較好的成績。
2. 有關將 condition 加進去與 input concatenate 的策略，我是讓 Generator 以及 Discriminator 在訓練時兩個網路都餵 condition，可以理解成 Discriminator 的工作除了在判斷圖片是真是假，同時也兼顧了一項任務，就是判斷這張圖片和送進來的 condition 是否有對應。
3. 生成 Fake images 時，雖然 latent vector 是隨機 sample 出來的，但 condition 如果同樣使用隨機 sample，此時會訓練不起來，應該要使用 training data 中有的 condition 比較能訓練起來，我在想是因為如果 Real images 與 Fake

images 使用不同的 condition 或許會導致 Discriminator 非常容易看出哪些是 Real 哪些是 Fake 的 images，導致 Generator 不曉得應該要先讓產生出的照片越真實越好，還是要越符合 condition 的條件越好，此時 Generator 就會一直訓練不起來，Discriminator 也就沒辦法跟著變好。

4. 目前的策略是 Discriminator 訓練一次，Generator 訓練四次，有試過將 Generator 訓練多次一點（五次、六次），除了時間會變久之外，似乎對於 performance 沒有明顯的提升。