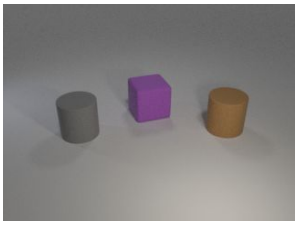
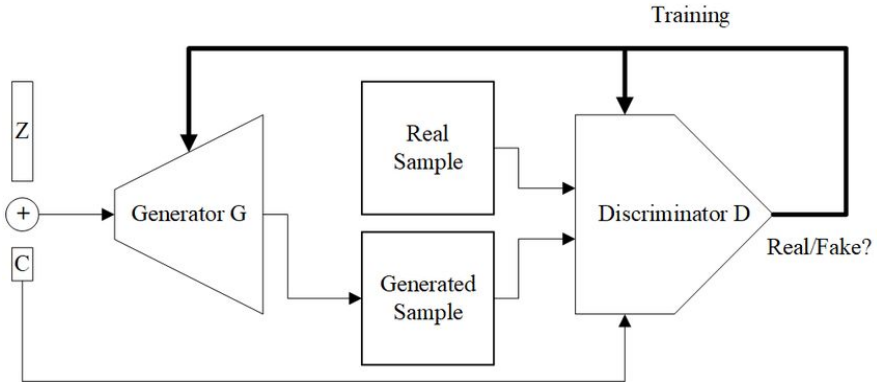
lab6

資工碩一 吳承翰 0856105

1.Introduction

利用conditional GAN訓練一個可以生成指定條件的圖片。
training data為ICLEVR的幾何物體圖片，總共有24種不同的幾何物體，因此condition為一個24-dim 的 one-hot vector。如: [0,0,0,1,0,0,1,.....0,0,0]





2.Implement details

我使用cDCGAN作為model的架構。
Generator的部份會把condition vector與雜訊z(100-dim) concat起來，但condition vector會先經過一個fully connected layer把24-dim變為200-dim來擴充資訊，因此成為一個300-dim的vector後再連續做5次ConvTranspose變成fake image。
ConvTranspose時也一併使用BatchNormalized與ReLU function。

```
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
```

```python
class Generator(nn.Module):
    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,stride=(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()
```

Discriminator會把24-dim的condition vector經由fully connected layer&reshape變成一張(1*64*64)的圖，同樣也是為了擴充資訊，之後再與training data或是Generator的圖片做concat變成(3+1*64*64)的圖片，在連續做5次Conv就可以得到一個scalar了。
Conv時也一併使用BatchNormalized與LeakyReLU function。

由於output為一個代表是否為真照片的scalar，因此loss function使用binary cross entropy。

```python
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
```

```python
class Discriminator(nn.Module):
    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()
```

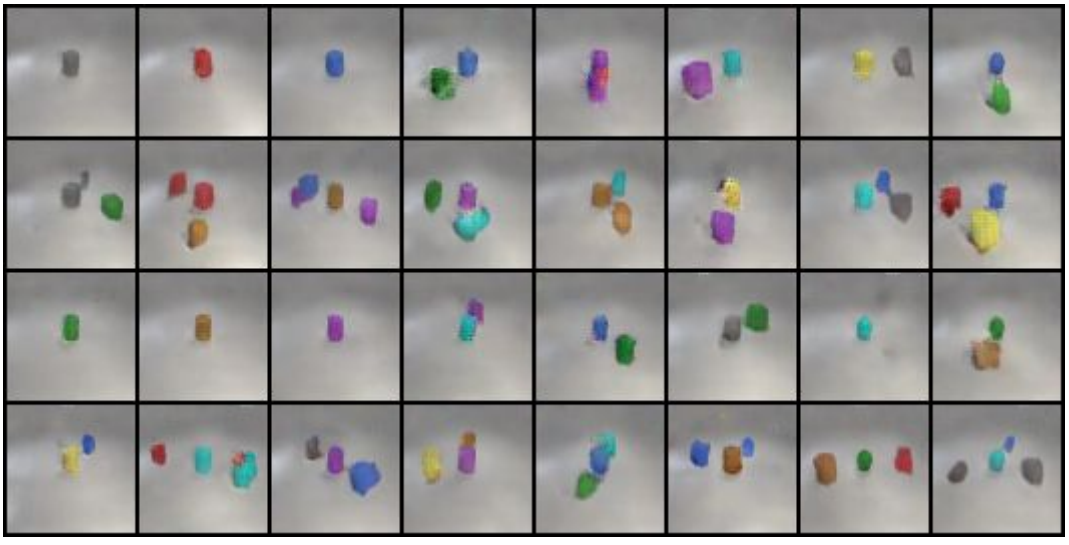
總共訓練200個epoch，learning rate為0.0002，batch_size為64
雜訊z_dim=100，條件c_dim=200

3.results and discussion

average score: 0.71左右。

```
score: 0.74
score: 0.69
score: 0.69
score: 0.69
score: 0.71
score: 0.72

avg score: 0.71
```



我也有試著在同樣model structure的情況下使用WGAN-GP loss function但結果score只有0.40，不知道哪裡寫錯了。
訓練時，generator loss要與discriminator loss越相近越好，於是我generator與discriminator訓練的次數比例調為4：1，這使他們在訓練前期loss相近。

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

    z = random_z(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()
```

結論：

1. train Generator 4times 跟 train Generator 5times 結果差不多
2. Generator要用RELU + Discriminator要用LeakyRelu
3. 加入BatchNormalized 可以提高score
4. 生成fake images時所用的condition vector用training data已有的condition就好了，自己隨機random的condition vector(24-dim中有1~3個1)反而會train壞掉