

# 309554042 數據所 李政毅

## Report

- Introduction (10%)

### Task1

1.利用conditonal GAN訓練ICLEVER data, 能夠給定condition後生成該condition對應的圖片，總共有24種不同的幾何物體，condition用一個dimension=24的vector去控制，每張圖片裡面最多只有三個物體，也就是說每個vector裡面最多只會有3個1，並透過pretrain的classifier來做評。

2.利用conditional NF訓練ICLEVER data, 先透過flow model把input 跟condition丟進去，訓練latent 並使它跟 $N(0,1)$ 靠近，訓練完後再透過從 $z \sim N(0,1)$ 跟condition來逆生成回我們想要的圖片，並透過pretrain的classifier來做評分

### Task2

透過condition NF訓練Celeb A data, 把flow model 訓練好後，要能夠產生有特定特徵的人臉，以及做linear interpolation，也就是給定兩個人臉，能夠依據兩個人臉的幾何距離從A臉 不斷靠近到B臉的效果，最後一件事是把一項attribute的特徵給數值化，透過有attribute跟沒有的照片作為對照，來透過調整這項attribute數值就能夠控制某張照片的效果。

- Implementation details (15%)

不好意思，由於期末時間沒有分配好的關係這次只完成了Task1的CGAN部份，故只對這部份實做細節做說明～

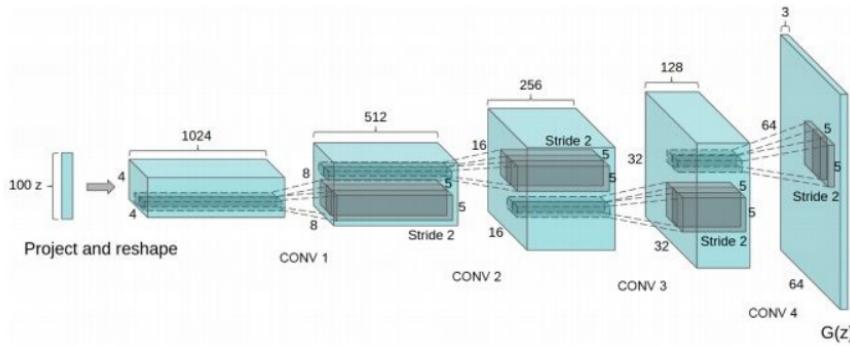
首先在先用dataloader的部份由於train.json是對應的圖片，而text.json指的則是不同的condition label，所以一開始先針對train跟text做不同的資料讀取，並且對每張照片做resize及normalize

```
if mode == 'train':
    data = json.load(open(os.path.join(root_folder, 'train.json')))
    obj = json.load(open(os.path.join(root_folder, 'objects.json')))
    img = list(data.keys())
    label = list(data.values())
    for i in range(len(label)):
        for j in range(len(label[i])):
            label[i][j] = obj[label[i][j]]
    tmp = np.zeros(len(obj))
    tmp[label[0]] = 1
    label[0] = tmp
    return np.squeeze(img), np.squeeze(label)

if mode == 'test':
    data = json.load(open(os.path.join(root_folder, 'test.json')))
    obj = json.load(open(os.path.join(root_folder, 'objects.json')))
    label = data
    for i in range(len(label)):
        for j in range(len(label[i])):
            label[i][j] = obj[label[i][j]]
    tmp = np.zeros(len(obj))
    tmp[label[0]] = 1
    label[0] = tmp
    return None, label

self.transformations=transforms.Compose([
    transforms.Resize((64,64)),
    transforms.ToTensor(),
    transforms.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5))
])
```

處理完照片後，使用DCGAN的架構作為model架構，並依據原始paper的各層設計來實做



架構如下，參考：<https://zhuanlan.zhihu.com/p/83630387> 設計網路，下方虛線處為截圖

1. DCGAN的生成器和判別器都捨棄了CNN的池化層，判別器保留CNN的整體架構，生成器則是將捲積層替換成了反捲積層（**fractional-strided convolution**）或者叫轉置卷積層（**Convolution Transpose**）。
2. 在判別器和生成器中在每一層之後都是用了Batch Normalization（BN）層，有助於處理初始化不良導致的訓練問題，加速模型訓練，提升了訓練的穩定性。
3. 利用1\*1卷積層替換到所有的全連接層。
4. 在生成器中除輸出層使用Tanh（Sigmoid）激活函數，其餘層全部使用ReLU激活函數。
5. 在判別器所有層都使用LeakyReLU激活函數，防止梯度稀。

Generator:

一開始先把dim=24的condition利用fc擴展成dim=200的vector，接著再把擴展過後的vector跟dim=100的noise vector給concatenate在一起後，再丟入五層含有BN以及ReLU的轉置捲積層後經過tanh輸出結果

```
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)]
        # In G use ConvTranspose2d and BN & ReLU
        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()
            ))
        # Last layer use Tanh as activation
        self.convT5=nn.ConvTranspose2d(64,3,kernel_size,stride=(2,2),padding=(1,1))
        self.tanh=nn.Tanh()

    def forward(self,z,c):
        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則是把dim=24 的condition 經由fc跟reshape轉成1\*64\*64的size來擴充，接著再把擴充後的matrix跟training data concate在一起變成(1+3)\*64\*64的size，再把這個matrix放到五層包含BN以及LeakyRelu的捲積層後，再把最後的結果從512resize成1，並經由sigmoid來判斷真假。

```
class Discriminator(nn.Module):
    def __init__(self,img_shape,c_dim):
        super(Discriminator, self).__init__()
        self.C,self.H,self.W=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]
        # In D use Conv2d & BN & LeakyRelu
        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()
            ))
        # the last layer use sigmoid to classify true or false
        self.conv5=nn.Conv2d(512,1,kernel_size,stride=(1,1))
        self.sigmoid=nn.Sigmoid()

    def forward(self,X,c):
        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
```

訓練時還發現若僅把Generator跟Discriminator各練一次，會因為Discriminator太強而導致訓練出來的結果很差，因此調整成Generator跟Discriminator訓練的次數是3:1或是4:1來增加Generator的強度，讓G跟D更接近、score表現更好。

```
"""
train discriminator
"""

optimizer_d.zero_grad()

#real images
predicts = discrimiator(images, conditions)
real = torch.ones(len(predicts)).to(device)
loss_real = Criterion(predicts, real)
#fake images
z = torch.randn(len(predicts), z_dim).to(device)
fake = torch.zeros(len(predicts)).to(device)

gen_imgs = generator(z, conditions)
predicts = discrimiator(gen_imgs, conditions)
loss_fake = Criterion(predicts, fake)
#bp
loss_d = loss_real + loss_fake
loss_d.backward()
optimizer_d.step()

"""
train generator
"""
for _ in range(3): # improve strength of generator

    optimizer_g.zero_grad()
    z = torch.randn(len(predicts), z_dim).to(device)
    gen_imgs = generator(z, conditions)
    predicts = discrimiator(gen_imgs, conditions)
    loss_g = Criterion(predicts, real)
    #bp
    loss_g.backward()
    optimizer_g.step()
```

Hyper parameters:

epochs=200

lr=0.001 or 0.0002

epochs=200

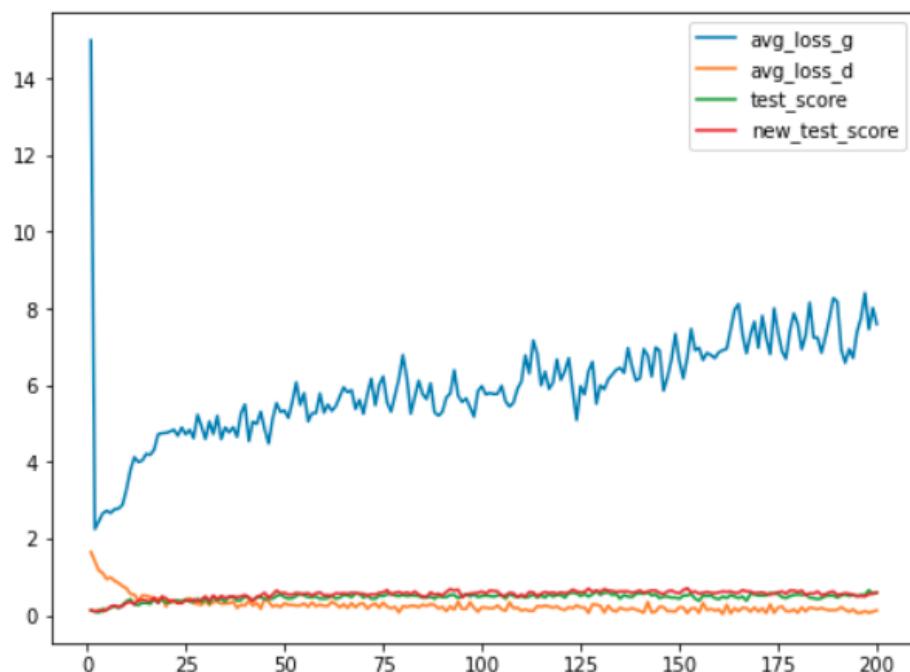
batch\_size=128

z\_dim= 100

c\_dim= 200

- Results and discussion (75%)

即便我們試圖增加generator的訓練次數來達到減少G跟D強度的差距，但嘗試了後從實驗結果來看可以發現G跟D還是很難在實物上達到平衡的，可以看到D還是很明顯比G強。



cGAN:

在訓練的過程中分別把testing score跟new\_testing score最好的ckpt記錄下來

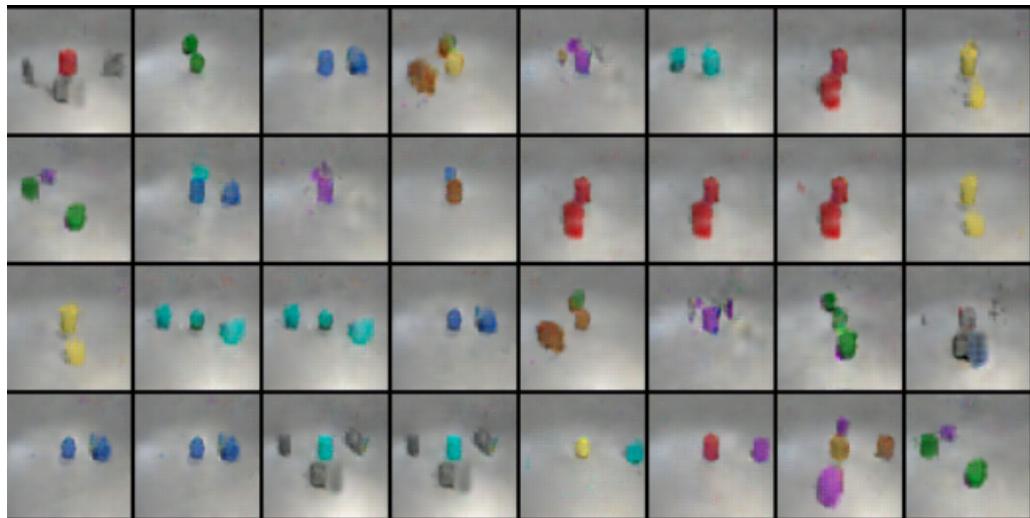
在lr=0.0002,epochs=200,G:D訓練數為4:1的情況練到最好的testing score = 0.76

在lr=0.001 ,epochs=200,G:D訓練數為3:1的情況練到new\_testing score =0.70

testing score : 0.76



new\_testing score : 0.70



### 小結論:

1. 在cGAN中發現Discriminator比Generator來的強，而若D太強勢會導致D並沒有給G夠多的資訊來學習，可能會導致gradient vanishing而讓結果變得很差，因此在訓練的過程中提昇G的訓練次數，訓練4次G才訓練一次D，來試圖縮減G和D的差距
2. 在discriminator中加入leaky relu以及generator中加入relu都能夠有效避免gradient vanishing
3. noise z用 $N(0,1)$ 去採樣的結果會比uniform採樣的結果來的好，且更合理
4. 雖然Generator是由convTranspose2d構成、Discriminator是用conv2d構成，但在層跟層中間加入Batch normalization對G跟D的訓練速度跟穩定度都有幫助
5. 若把照片在normalize時把normalize的範圍限縮在在[-1,1]的效果可能會更好，原因是因為generator最後一層的output tanh範圍也是在[-1,1]的。
6. follow DCGAN原paper指出中learning rate用0.0002而optimizer的adam beta用[0.5,0.99]，確實讓結果更好
7. 而cNF的優勢則是生成結果較具多樣性，且需要調整的參數較少，所以模型較穩定

### Reference:

- <https://arxiv.org/pdf/1511.06434.pdf>
- <https://zhuanlan.zhihu.com/p/83630387>