

# ADL 2019 HW4

R07922001 陳佳佑

## Generator Network Structure

```
class Generator(nn.Module):
    def __init__(self, featureDim, noiseDim, hidden_size=64):
        super(Generator, self).__init__()
        self.num_channels = 3
        self.noise_dim = noiseDim
        self.ngf = hidden_size

        self.noise_embed = nn.Sequential(
            nn.Linear(self.noise_dim, self.ngf*4*4),
        )

        self.label_embed = nn.Linear(15, self.ngf*8)

        self.noise_conv = nn.Sequential(
            nn.ConvTranspose2d(self.ngf, self.ngf * 8, 4, 2, 1),
            nn.BatchNorm2d(self.ngf * 8),
            nn.ReLU(True),
        )

        self.netG = nn.Sequential(
            nn.ConvTranspose2d(self.ngf * 16, self.ngf * 4, 4, 2, 1),
            nn.BatchNorm2d(self.ngf * 4),
            nn.ReLU(True),

            nn.ConvTranspose2d(self.ngf * 4, self.ngf * 2, 4, 2, 1),
            nn.BatchNorm2d(self.ngf * 2),
            nn.ReLU(True),

            nn.ConvTranspose2d(self.ngf * 2, self.ngf * 1, 4, 2, 1),
            nn.BatchNorm2d(self.ngf * 1),
            nn.ReLU(True),

            nn.ConvTranspose2d(self.ngf * 1, self.num_channels, 4, 2, 1),
            nn.Tanh()
        )
        self.apply(weights_init)

    def forward(self, embed_vector, noise):
        noise = self.noise_embed(noise).view(noise.size(0), -1, 4, 4)
        embed_vector = self.label_embed(embed_vector)
        embed_vector = embed_vector.unsqueeze(2).expand(
            noise.size(0),
            self.ngf*8,
            64).view(-1, self.ngf*8, 8, 8)

        noise = self.noise_conv(noise)
        latent_vector = torch.cat([embed_vector, noise], dim=1)
        output = self.netG(latent_vector)
        return output
```

The noise vector would first be projected to a higher space and reshape to a graph structure G1. The embed better would be projected to a higher space and do the expansion to a graph structure G2. Then G1 would be convoluted to G3 and G2 would concat with G3 and becomes G4. Finally G4 would pass to the Deconv network netG and pass Tanh to be a 3 \* 128 \* 128 size stuff.

## Discriminator Network Structure

```
class Discriminator(nn.Module):
    def __init__(self, FeatureDim, adv_loss, hidden_size=64):
        super(Discriminator, self).__init__()

        self.adv_loss = adv_loss
        self.ndf = hidden_size
        self.label_embed = nn.Linear(15, self.ndf*8)

        self.conv1 = nn.Conv2d(3, self.ndf, 4, 2, 1, bias=False)
        self.conv2 = nn.Conv2d(self.ndf, self.ndf*2, 4, 2, 1, bias=False)
        self.conv3 = nn.Conv2d(self.ndf*2, self.ndf*4, 4, 2, 1, bias=False)
        self.conv4 = nn.Conv2d(self.ndf*4, self.ndf*8, 4, 2, 1, bias=False)
        self.conv5 = nn.Sequential(
            nn.Conv2d(self.ndf*8*2, self.ndf*1, 4, 2, 1, bias=False),
            nn.LeakyReLU(0.2, True)
        )

        self.netD = nn.Sequential(
            self.conv1,
            nn.BatchNorm2d(self.ndf),
            nn.LeakyReLU(0.2, True),
            self.conv2,
            nn.BatchNorm2d(self.ndf*2),
            nn.LeakyReLU(0.2, True),
            self.conv3,
            nn.BatchNorm2d(self.ndf*4),
            nn.LeakyReLU(0.2, True),
            self.conv4,
            nn.BatchNorm2d(self.ndf*8),
            nn.LeakyReLU(0.2, True),
        )

        self.proj = nn.Linear(self.ndf*4*4, self.ndf, bias=False)
        self.netD2 = nn.Linear(self.ndf, 1, bias=False)
        self.netAC = nn.Linear(self.ndf, 15, bias=False)
        self.apply(weights_init)

    def forward(self, inputs, label):
        label = self.label_embed(label)
        label = label.unsqueeze(2).expand(
            label.size(0), -1, 64).view(
            label.size(0), -1, 8, 8)

        imageC = self.netD(inputs)
        x = torch.cat([imageC, label], dim=1)
        x = self.conv5(x).view(label.size(0), -1)
        x = self.proj(x)
        prob = self.netD2(x)
        cls = self.netAC(x)
        return prob.squeeze(), cls.squeeze()
```

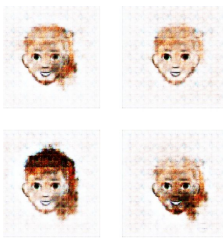


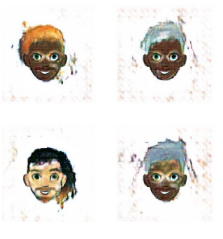
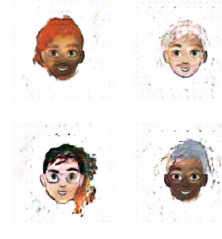





Similar to the generator, the discrete part would be embed and scale to a image G1. After convoluting the input image, it would concat with G1 and then a simple conv5. Finally, a DNN to label and real/fake score.

## Loss Terms

In this case, I adopt the WGAN-GP loss term and the ACGAN loss. However, my discriminator would have the input of origin label, which is different to the original ACGAN loss.

- Notation
  - $D$  is the discriminator
  - $G$  is the generator
  - $L$  is the binary cross-entropy function
  - $\theta$  is the parameter of discriminator
  - $l$  is the label
- Generative Loss
  - $G_{loss} = -E_{x_{fake} \sim p_g} D(x_{fake}, l_{real}, \theta)_{real\ prob} + 10 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{real}, \theta)_{label}, l_{real})$
- Discriminator Loss
  - $D_{loss} = E_{x_{fake} \sim p_g} D(x_{fake}, l_{real}, \theta)_{real\ prob} - E_{x_{real} \sim p_{real}} D(x_{real}, l_{real}, \theta) +$   
 $10 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{real}, \theta)_{label}, l_{real}) +$   
 $10 * E_{x_{real} \sim p_{real}} L(D(x_{real}, l_{real}, \theta)_{label}, l_{real})$

## Result Evolution

Epoch 1	Epoch 2	Epoch 3	Epoch 4	Epoch 5
				
Epoch 6	Epoch 7	Epoch 8	Epoch 9	Epoch 10
				

## Final Result



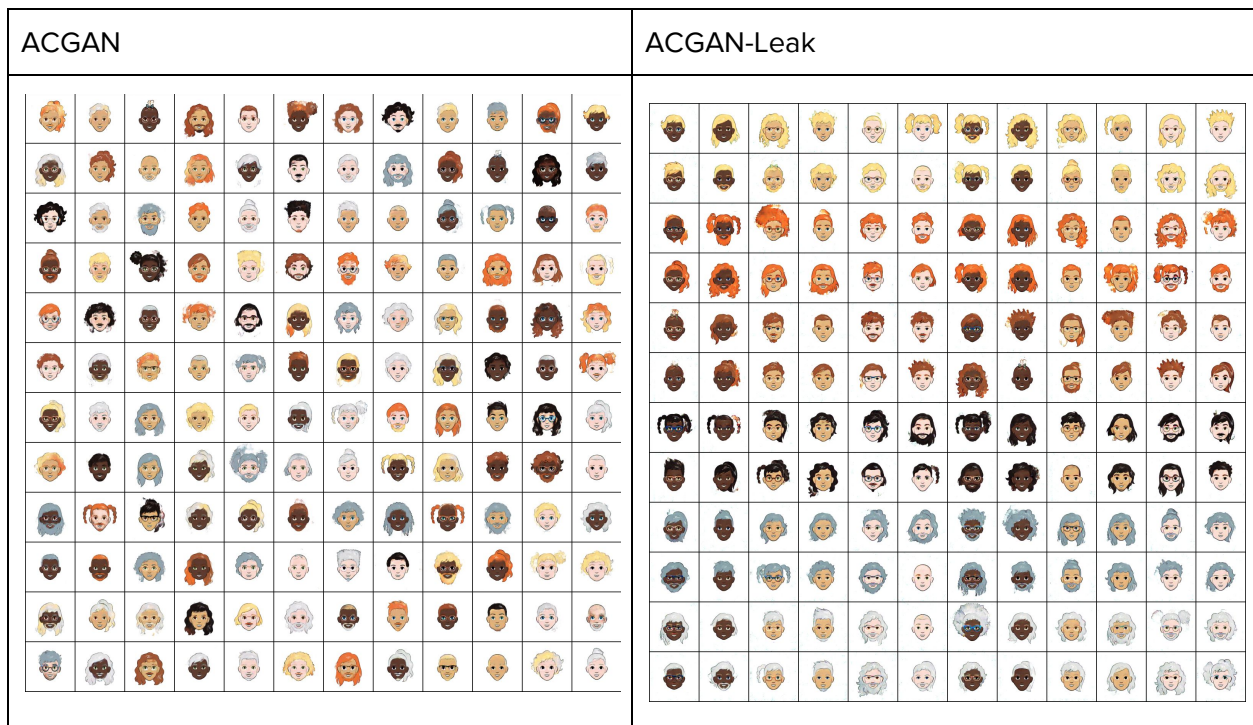
## Hyperparameters

- Feature-map size: 64
- Generator Discriminator update rate : 1 vs 1
- Batch\_size: 4
- Learning rate: 2e-4
- Optimizer: Adam
  - Beta: (0.5, 0.999)



## Experiment 1: ACGAN vs ACGAN-Leak

In the original ACGAN paper, the input of the discriminator has only an image instead of the real label. I have tried to use the loss. The image quality is quite great, however the condition is almost broken. After looking at the slide, I try to add the label into the input of discriminator, and it really helps the conditional. In general, we could think that the input of the label gives some sense of autoencoder loss. It could increase the stability of the conditional function.



## Experiment 2: Another Auxiliary Loss

The loss above only considered the real exist label stuff. However, it's quite strange that isn't it enough for the model to learn the condition. Since, some combination may occur rarely and it may cause some label imbalance issue. As a result, I add a new term for the loss.

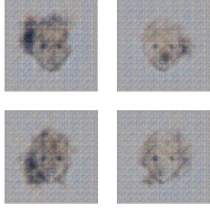
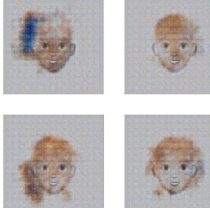

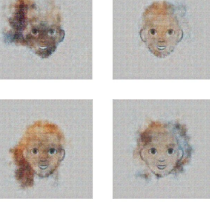
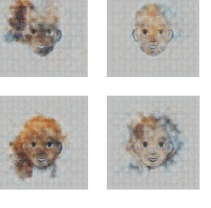
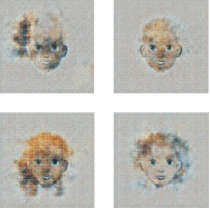

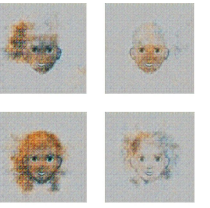
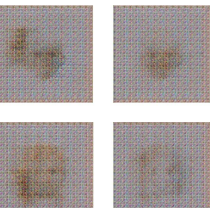
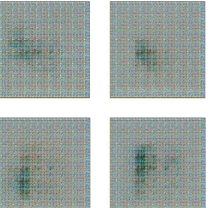
- Generative Loss

$$G_{loss} = -0.5 * E_{x_{fake} \sim p_g} D(x_{fake}, l_{real}, \theta)_{real\ prob} + 5 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{real}, \theta)_{label}, l_{real}) + \\ -0.5 * E_{x_{fake} \sim p_g} D(x_{fake}, l_{fake}, \theta)_{real\ prob} + 5 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{fake}, \theta)_{label}, l_{fake})$$

- Discriminator Loss

$$D_{loss} = 0.5 * E_{x_{fake} \sim p_g} D(x_{fake}, l_{real}, \theta)_{real\ prob} - E_{x_{real} \sim p_{real}} D(x_{real}, l_{real}, \theta) + \\ 0.5 * E_{x_{fake} \sim p_g} D(x_{fake}, l_{fake}, \theta)_{real\ prob} + 10 * E_{x_{real} \sim p_{real}} L(D(x_{real}, l_{real}, \theta)_{label}, l_{real}) + \\ 5 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{real}, \theta)_{label}, l_{real}) + 5 * E_{x_{fake} \sim p_g} L(D(x_{fake}, l_{fake}, \theta)_{label}, l_{fake})$$

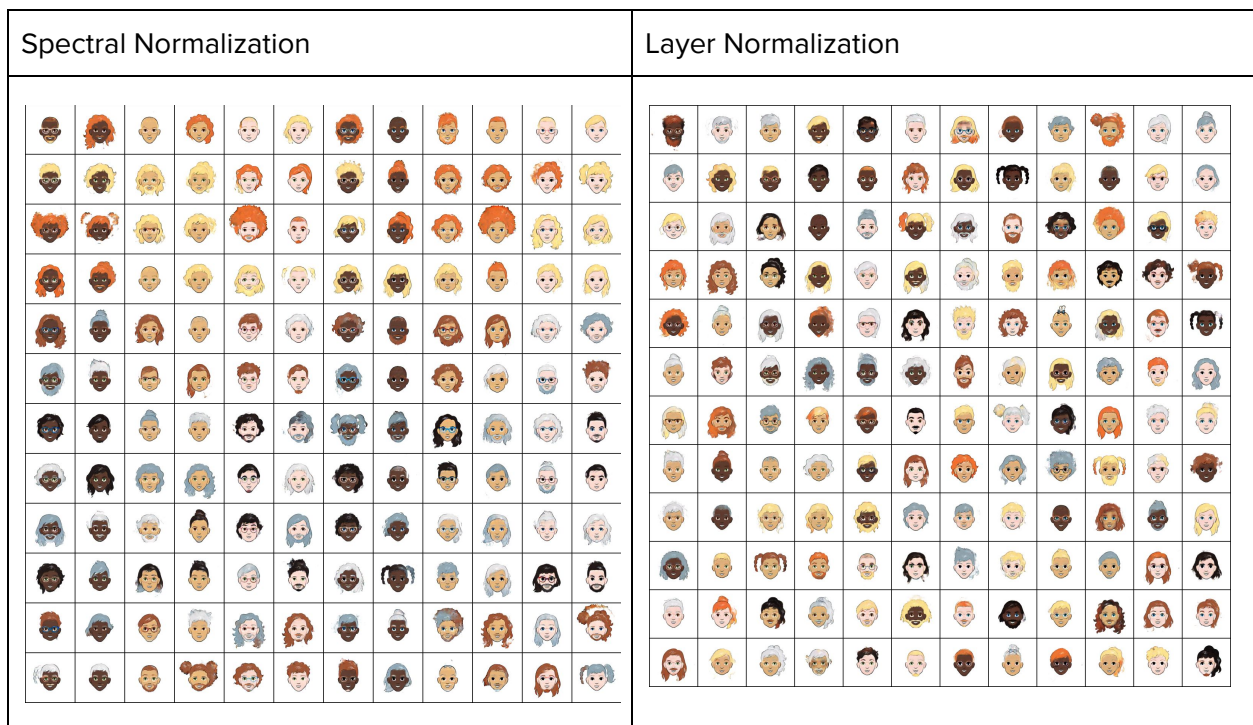
Performance

Epoch 2	Epoch 4	Epoch 6	Epoch 8	Epoch 10
				
Epoch 12	Epoch 14	Epoch 16	Epoch 18	Epoch 20
				

The results show that the equilibrium breaks at epoch 18 and didn't come back. From the loss history, I observe that the discriminator becomes too strong and the generator don't have the ability to attack it after epoch 18.

### Experiment 3: Layer normalization and Spectral normalization

Since batch normalization would do normalization across each instance, it would perturb the calculation of the norm of gradient penalty. Some people suggests that layer normalization could replace the character of batch normalization in WGAN-GP, since layer normalization only do normalize in a instance. The spectral normalization proposes a new loss that regularizes the Lipschitz constant of discriminator network to help the convergence. In this experiment, I just replace the batch normalization term in discriminator with layer normalization and spectral normalization.



The image uses the same label in corresponding places as the image in the first model. We could see that both the normalization has great performance of plotting. However, the conditional attribute is totally wrong. We could see that spectral normalization has performed a little conditional factor, the face color and the hair color. However, the case in Layer normalization is a hazard. It's hard to observe any layer normalization in it.