DLCV HW4

電機四 廖宜倫 **B03901001**

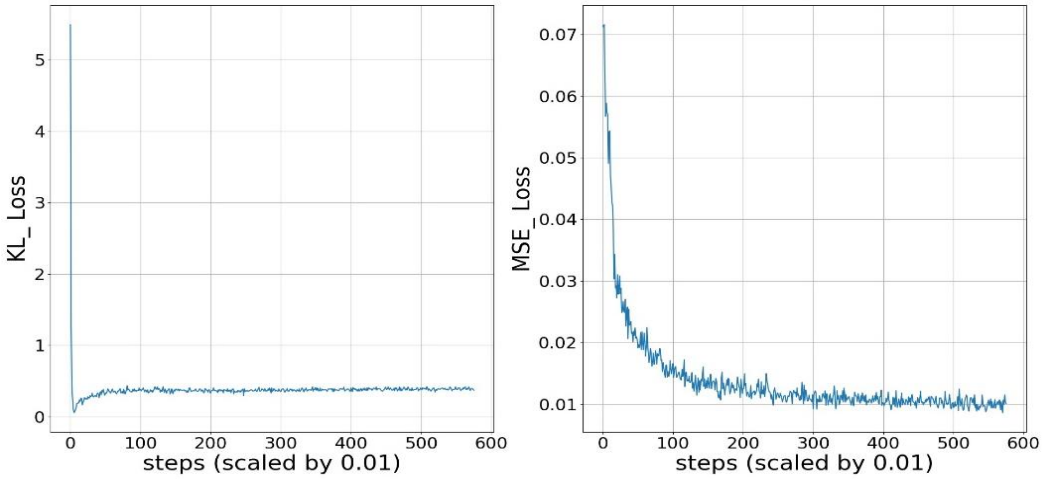
Problem 1. VAE

1. Architecture and implementation

```
VAE(
  (conv1): Sequential(
    (0): Conv2d(3, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU() )
  (conv2): Sequential(
    (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU() )
  (maxpooling1): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (conv3): Sequential(
    (0): Conv2d(128, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU() )
  (conv4): Sequential(
    (0): Conv2d(256, 256, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU() )
  (maxpooling2): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (conv5): Sequential(
    (0): Conv2d(256, 512, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU() )
  (conv6): Sequential(
    (0): Conv2d(512, 512, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (1): ReLU() )
  (maxpooling3): MaxPool2d(kernel_size=(2, 2), stride=(2, 2), dilation=(1, 1), ceil_mode=False)
  (fc_encode1): Linear(in_features=32768, out_features=128, bias=True)
  (fc_encode2): Linear(in_features=32768, out_features=128, bias=True)
  (fc_decode): Linear(in_features=128, out_features=32768, bias=True)
  (deconv1): Sequential(
    (0): ConvTranspose2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): ReLU() )
  (deconv2): Sequential(
    (0): ConvTranspose2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): ReLU() )
  (deconv3): Sequential(
    (0): ConvTranspose2d(512, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): Sigmoid() )
)
```

The architecture of my VAE model is shown above. I use MSE for reconstruction loss and 10^{-3} for KL loss weights. I use Adamax as the optimizer with learning rate equal to $5\text{x}10^{-4}$ to train the network.

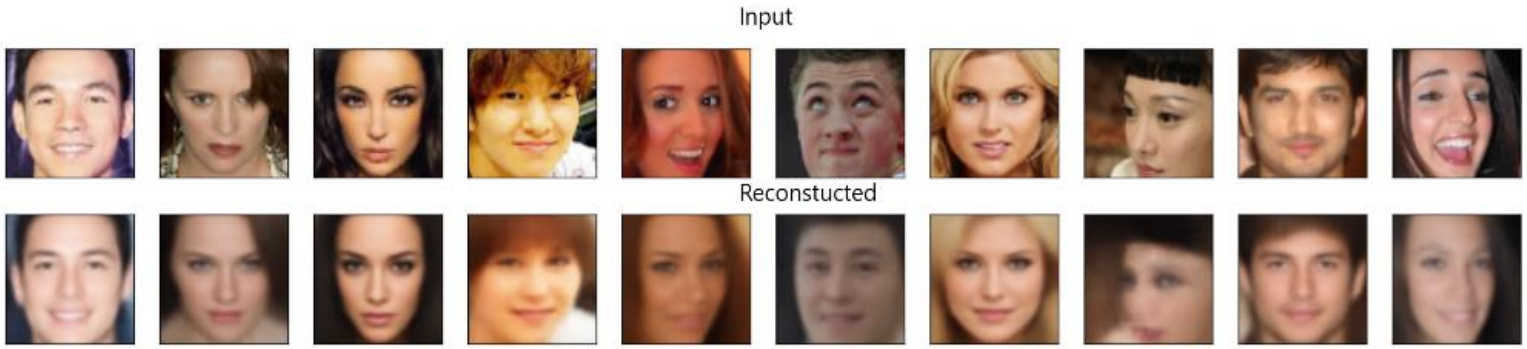
2. Learning curve



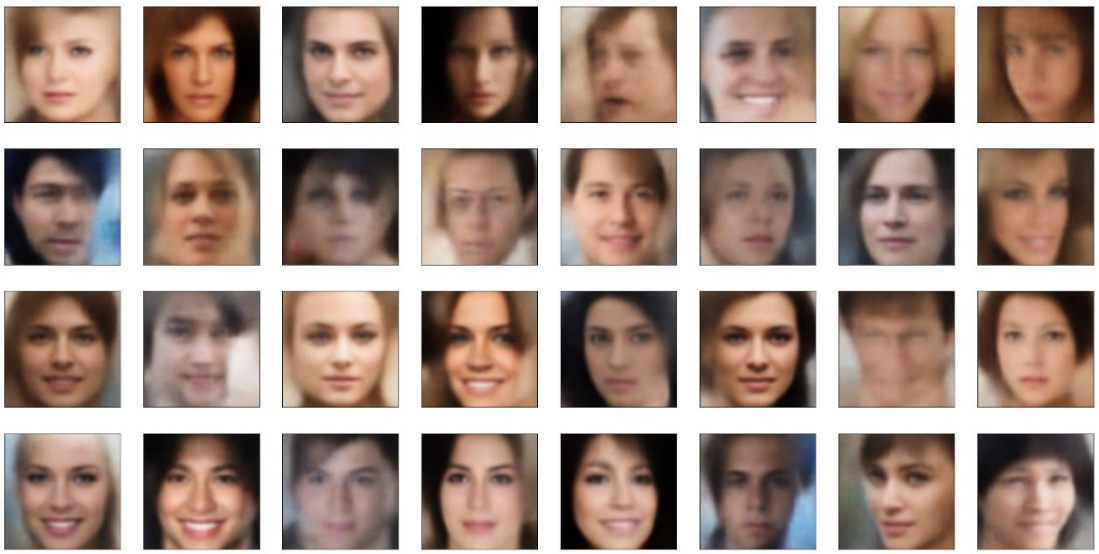
The horizontal axis is for steps, which is scaled by 0.01. That is, the rightmost value is 600*100 = 60000.

3. MSE

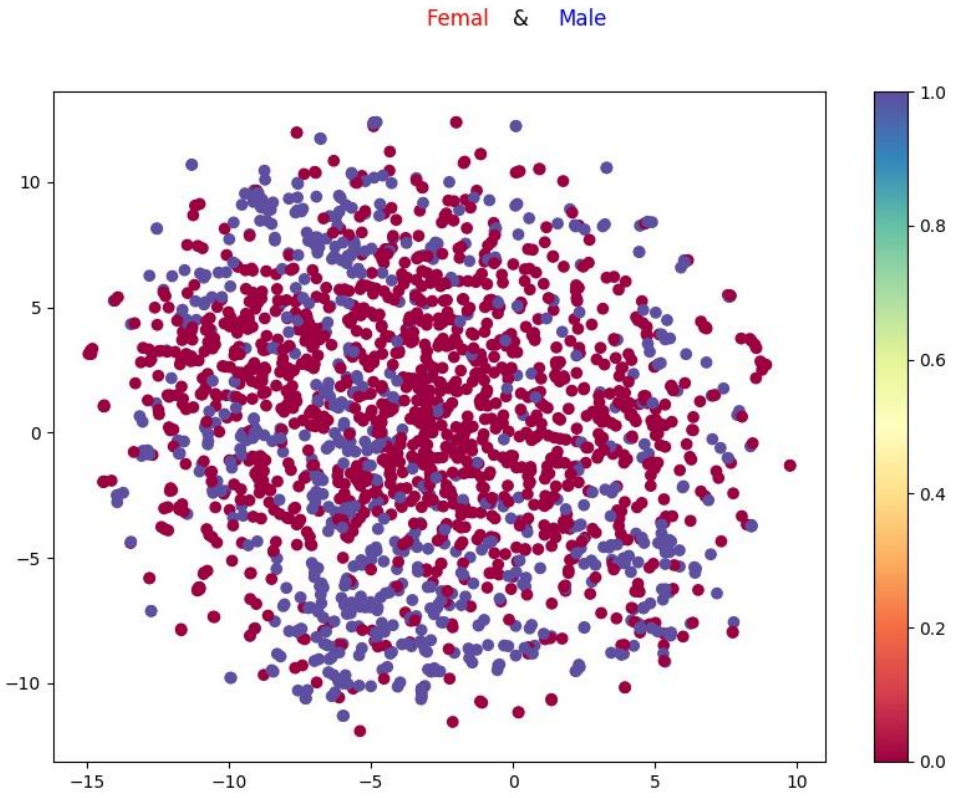
I use torch.nn.functional.mse_loss() to evaluate MSE error with the value of each pixel in the range of [0, 1]. The MSE of the testing set is 0.007088815362658351.



4. Randomly generated images of VAE



5. tSNE



6. Discussion

First, VAE has the ability to generate new images. As shown in 4., VAE can generate (decode) images from random noises yet the resolution seems somewhat low and images are blurred. This corresponds to the fact that we train the autoencoder with MSE loss. To reduce the mean loss, the decoder tends to predict blurred images. Second, the trend of the growth in the loss of VAE during training is roughly the same. That is, the KL loss keeps growing and MSE loss keeps decreasing.

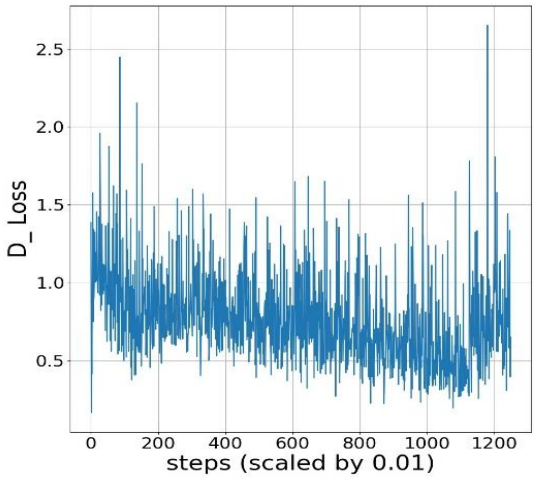
Problem 2. GAN

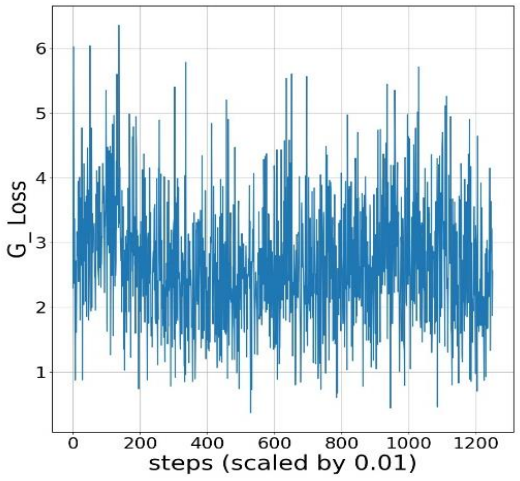
1. I use the architecture of DCGAN for this task.

```
generator(
  (main): Sequential(
    (0): ConvTranspose2d(128, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh() )
)
discriminator(
  (main): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (4): LeakyReLU(0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (7): LeakyReLU(0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (10): LeakyReLU(0.2, inplace)
    (11): Conv2d(512, 1, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): Sigmoid()
  )
)
```

I use Adam optimizer to train generators and discriminators with learning rate equal to $5x10^{-4}$.

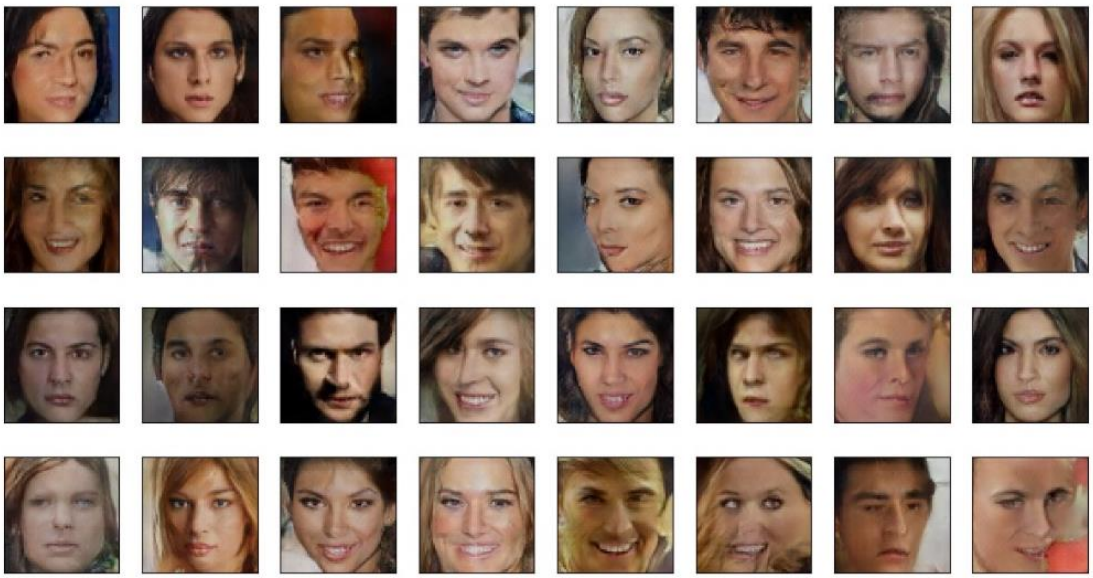
2.





As we can observe from the figure, when the loss of the discriminator, D_Loss, decreases, that of the generator, G_loss, increases, and vice versa. This corresponds to adversarial loss where loss functions are defined by other networks.

3.



4. Compared to VAE, GAN is more difficult to train. It is easy for discriminators to become stronger than generators, resulting in random noise images generated. As for my implementation, I notice that training more steps does not necessarily result in better image quality in 3. This may be due to that the discriminator is gradually outperforming the generator yet the images generated seem still like human faces. I also find that for this task, discriminators become strong faster than generators do. Thus, parameters of generators should be updated more frequently.

5. The images produced by VAE are more blurred than those produced by GAN. This is mainly due to that while training, the two methods aim at reducing different loss functions. The former tries to reduce MSE, resulting in blurred images. The latter tries to reduce adversarial loss. Yet the latter produces images of higher quality, it is more unstable during training.

Problem 3. ACGAN

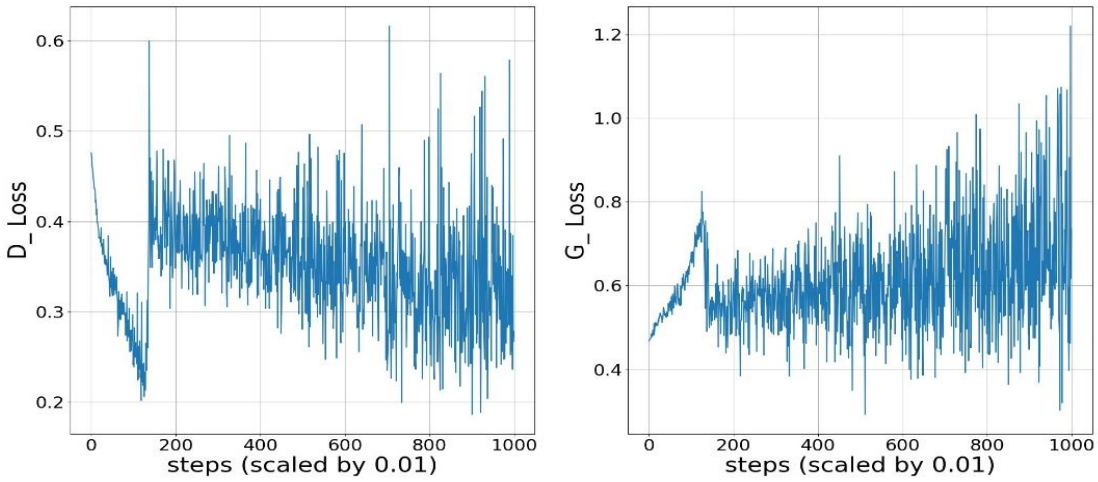
1.

```
generator(
  (fc): Sequential(
    (0): Linear(in_features=141, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
    (3): Linear(in_features=256, out_features=256, bias=True)
    (4): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU(inplace)
  )
  (deconv): Sequential(
    (0): ConvTranspose2d(256, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (2): ReLU(inplace)
    (3): ConvTranspose2d(512, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (5): ReLU(inplace)
    (6): ConvTranspose2d(256, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (7): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (8): ReLU(inplace)
    (9): ConvTranspose2d(128, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (10): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True)
    (11): ReLU(inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
discriminator(
  (conv_n): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (1): LeakyReLU(0.2, inplace)
    (2): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (3): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True)
    (4): LeakyReLU(0.2, inplace)
    (5): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (6): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True)
    (7): LeakyReLU(0.2, inplace)
    (8): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (9): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (10): LeakyReLU(0.2, inplace)
    (11): Conv2d(512, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (12): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (13): LeakyReLU(0.2, inplace)
  )
  (fc): Sequential(
    (0): Linear(in_features=512, out_features=256, bias=True)
    (1): BatchNorm1d(256, eps=1e-05, momentum=0.1, affine=True)
    (2): LeakyReLU(0.2)
  )
  (dc): Sequential(
    (0): Linear(in_features=256, out_features=1, bias=True)
    (1): Sigmoid()
  )
  (cl): Sequential(
    (0): Linear(in_features=256, out_features=13, bias=True)
    (1): Sigmoid()
  )
)
```

I use Adam optimizer to train networks with learning rate equal to 2×10^{-4}. The loss of auxiliary classifier is treated as MSE loss in my implementation. This is because the auxiliary label is not one-hot. The MSE loss and adversarial loss are weighed the same. In addition, I find from GAN that discriminators usually learn faster. Thus, I

adjust the times of update of generators per update of discriminators according to their learning progress. That is, if the loss of generators is larger than that of discriminators by some pre-defined thresholds, generators should be updated more frequently.

2.



The losses of the discriminator and the generator are the averages of adversarial loss and classification loss, which is regarded as MSE loss here. Roughly the mean of loss of the discriminator (D_Loss) is decreasing. This results from that the discriminator is obtaining the ability to classify generated images and real images. As for oscillation of losses, this corresponds to adversarial loss as discussed in Problem 2. 2.

3. I train ACGAN with all attributes but only change the gender attribute here. The upper row shows images generated with the gender attribute set to 0 (female) and the lower shows those with that attribute set to 1 (male).