Problem 1. VAE (6%)

1) The architecture of our VAE network is shown in fig1_1. batch size=32, optimizer uses Adam with learning rate lr=2e-4, betas=(0.5,0.999), KL lambda=1e-5, Dim of z selected at 512.

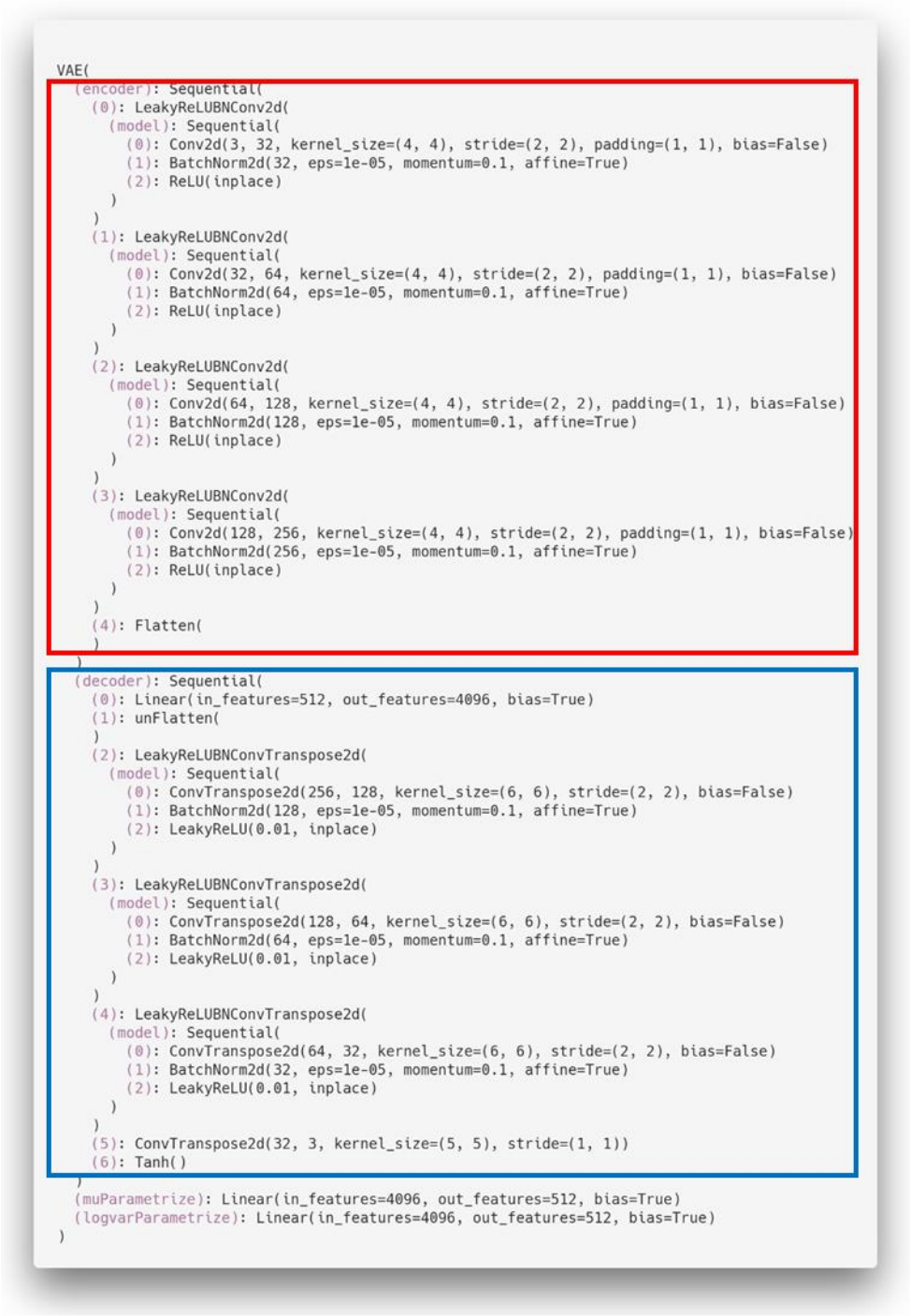

fig1_1

2) The learning curve of training our model is shown in fig1_2, the left curve is KLD against training steps and the right curve is the MSE against training steps.

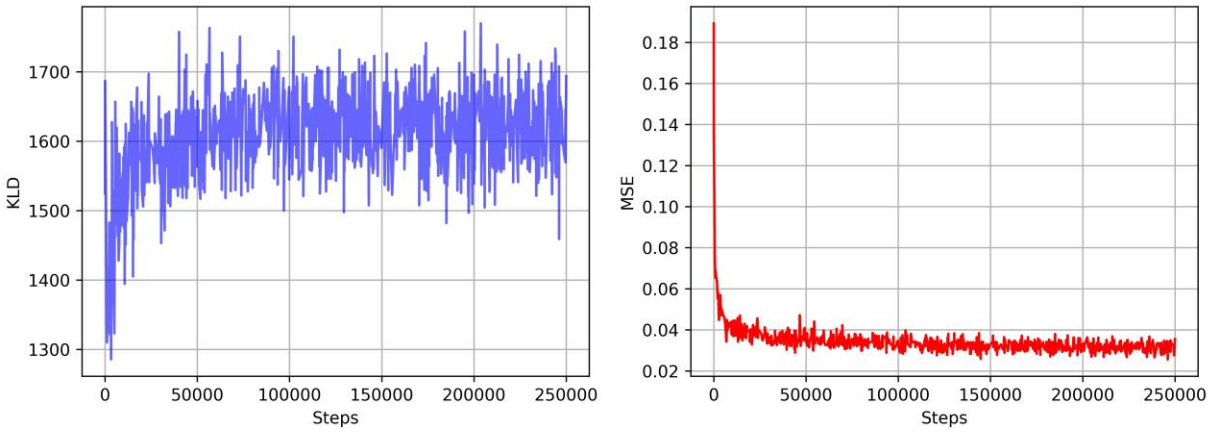

fig1_2

3) The selected test images are shown in the first row of fig1_3 and the reconstructed results are shown in the second row of fig1_3. Besides, the MSE of entire test images is approximately **0.0291**.

fig1_3

4) 32 random generated images of our model are plotted in fig1_4.

fig1_4

5) We visualize the latent space in fig1_5 (map it from high dimensional to 2D using tSNE from sklearn), where the blue dots are faces which have the *bangs* attribute, the red dots are faces don't have this attribute. We use 1000 test images

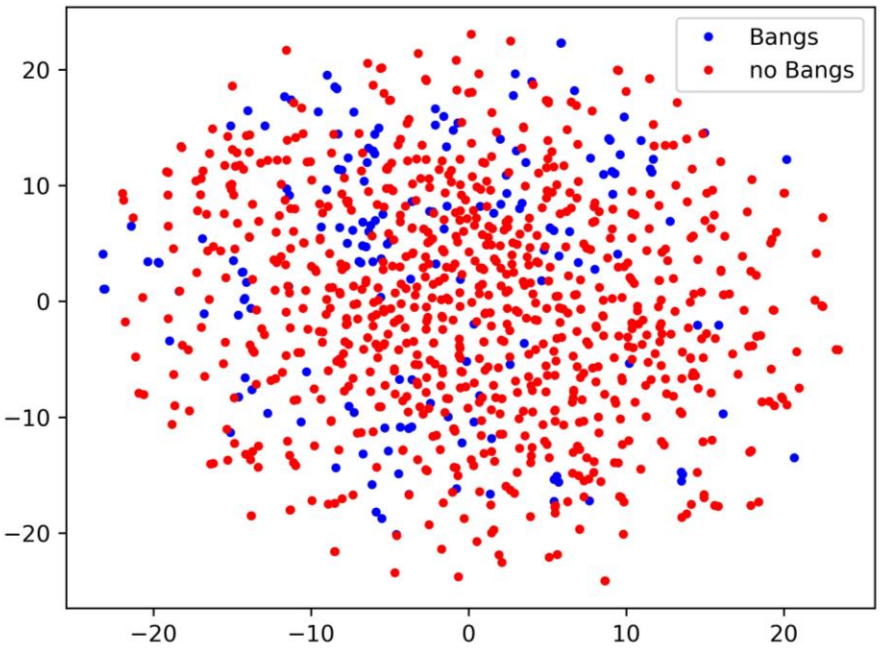

fig1_5

6) What I learned from implement VAE:
   a. We learned how to construct the AE/VAE model and tune hyper-parameters. We start from Auto Encoder model, which is relatively easy to realize, then modified it to VAE.
   b. We understand the difference between VAE and AE. Using AE, we have no idea about how AE access the latent space. But using VAE, we can not only reconstruct image from image but also from noise, which is advanced than AE. It helps us to understand.
   c. In the beginning, we didn't realize to normalize the image to -1~1, the results is bad. After fixed this bug, we realized the significance of data normalization.
   d. A bug that we use rand but not normal to generate noise, which can affect the results.
   e. In the beginning, we set the batch size to >512, with the hope that the training speed becomes quickly. However, the results become very bad. But the MSE will convergetes at 0.06 more than 0.5. Therefore, we change the batch size to 80, and found that it can reach to 0.035. Besides, generated faces are not good due to the large batch size.
   f. Besides, we also learned how to visualize the latent space using tSNE.
   g. KL lambda finds a trade-off between the reconstruction quality and the generated image quality.

Problem 2. GAN (5%)

1) The architecture of our GAN network (DCGAN) is shown in fig2_1, where the red box is the generator and the blue box is the discriminator. For generator, the optimizer uses Adam with learning rate G_lr=2e-4, betas=(0.5,0.999). For discriminator, the optimizer uses Adam with learning rate D_lr=5e-5, betas=(0.5,0.999). batch_size=128 and z_dim=100. The criterion we employed BCE loss:

$$loss(o, t) = -1/n \sum_i (t[i] * log(o[i]) + (1 - t[i]) * log(1 - o[i]))$$

In the process of training, we train the discriminator first, then the generator.

```
generator(
  (gen): Sequential(
    (0): ConvTranspose2d(100, 512, kernel_size=(4, 4), stride=(1, 1), bias=False)
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True)
    (2): LeakyReLU(0.1, 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): LeakyReLU(0.1, 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): LeakyReLU(0.1, 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): LeakyReLU(0.1, inplace)
    (12): ConvTranspose2d(64, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    (13): Tanh()
  )
)
discriminator(
  (dis): 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()
  )
)
```

fig2_1

2) The learning curve is shown in fig2_2. (left) shows the discriminator loss against steps. We can observe that the discriminator loss converges to approximately 0.5, which indicates that the discriminator cannot distinguish the fake images and the real images. Besides, the generator loss is increasing and converges after about 70k steps. (right) shows the discriminator accuracy against steps. We can observe that all the two accuracy curves converge after about 20k. The blue and the red curve results from feeding the real and generated images into the discriminator, respectively. We can find that the discriminator is trained well.

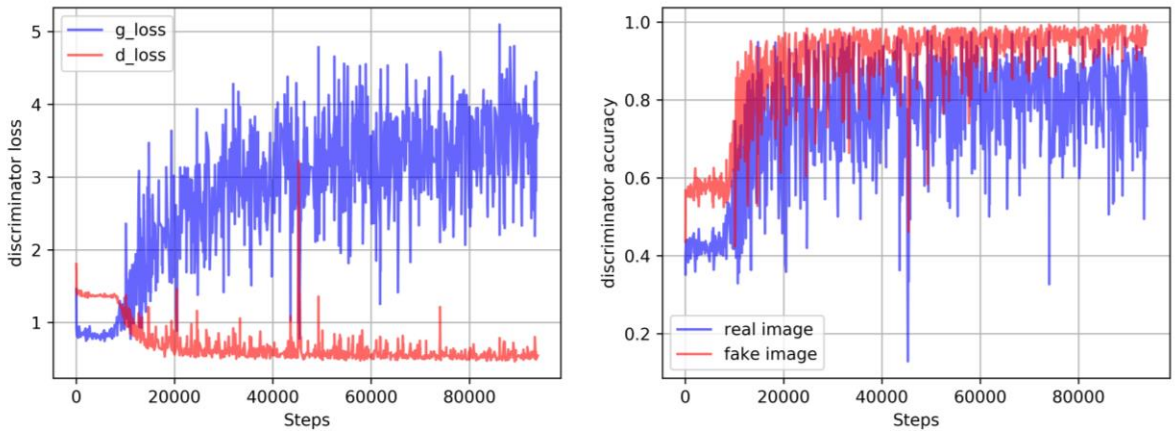

fig2_2

3) 32 random generated images of our model is plotted in fig2_3.

fig2_3

4) After this homework, I agree that GAN is quite hard to train.
   - We tried a lot of tricks to balance GAN that is to avoid the fast convergence of discriminator network. Including, a) add noise to the output/real image (not work). b) add noise to the label (i.e., soft label), it can work well. c) flip the image, but the results are not so good if consist up-down flip. d) Generator network is updated twice for each D network update, which differs from the original paper. (bad). e) Maximum D(G(z)) instead of minimum 1-D(G(z)) to avoid vanish gradient issue. (well)
   - Discriminator Loss 0 is a failure mode which indicates the discriminator is too strong, therefore we need to consider how to balance the training process.
   - We also learned how to make a GAN experiment reproducible.
   - With the increase of Epoch, the training results could be much worse. As shown in fig2_4, the images boxed up using yellow box are the bad results.
5) The differences between images generated by VAE and GAN are that: a) VAE generated images are much more smooth than GAN, because VAE using MSE as the reconstruction loss which will result in the loss of high-frequency signals; b) Compare with VAE, GAN generated images are much sharper (contain a lot of high-frequency signals) but they have more artifacts (as shown in fig2_4), which we don't want.

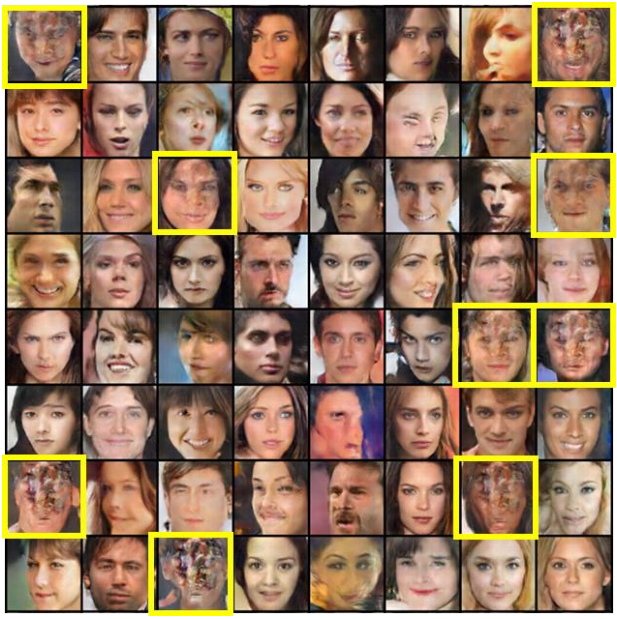

fig2_4

Problem 3. ACGAN (4%)

1) The architecture of our VAE network is shown in fig3_1, where the red box is the generator and the blue box is the discriminator. For generator, the optimizer uses Adam with learning rate G_lr=2e-4, betas=(0.5,0.999). For discriminator, the optimizer uses Adam with learning rate D_lr=2e-4, betas=(0.5,0.999). batch_size=128 and z_dim=100. We use BCE loss as in Problem 2 for discriminator and CrossEntropyLoss for classification.

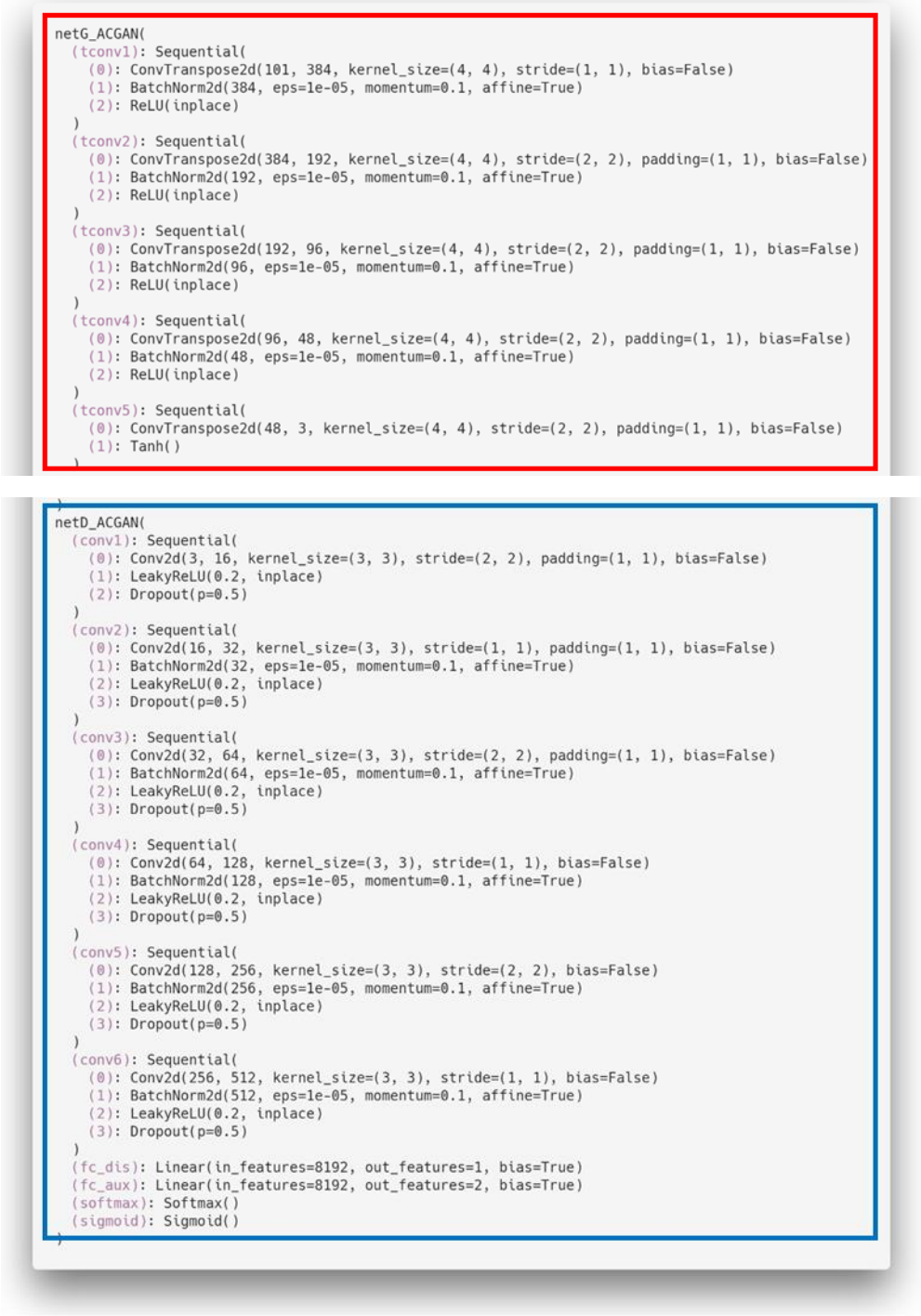

fig3_1

2) The learning curve is shown in fig3_2. (left) shows the discriminator loss against steps. We can observe that the generator loss and the discriminator reach to balance after about 10k steps. (right) shows the classify accuracy against steps. We can observe that the classifier performances better on fake images than real images.

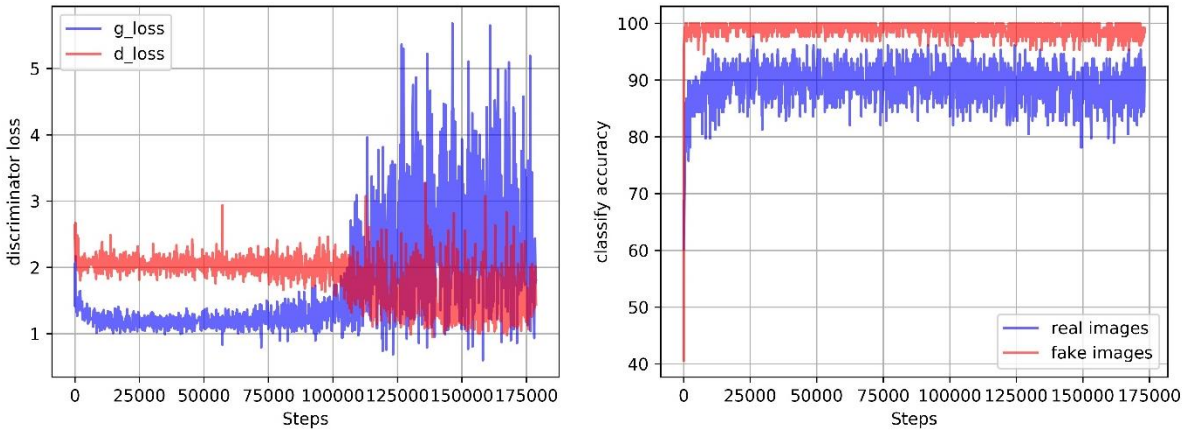

fig3_2

3) 10 pairs of randomly generated images are shown in fig3_3. The first row has the attribute “smile” and the second row does not have this attribute.

fig3_3