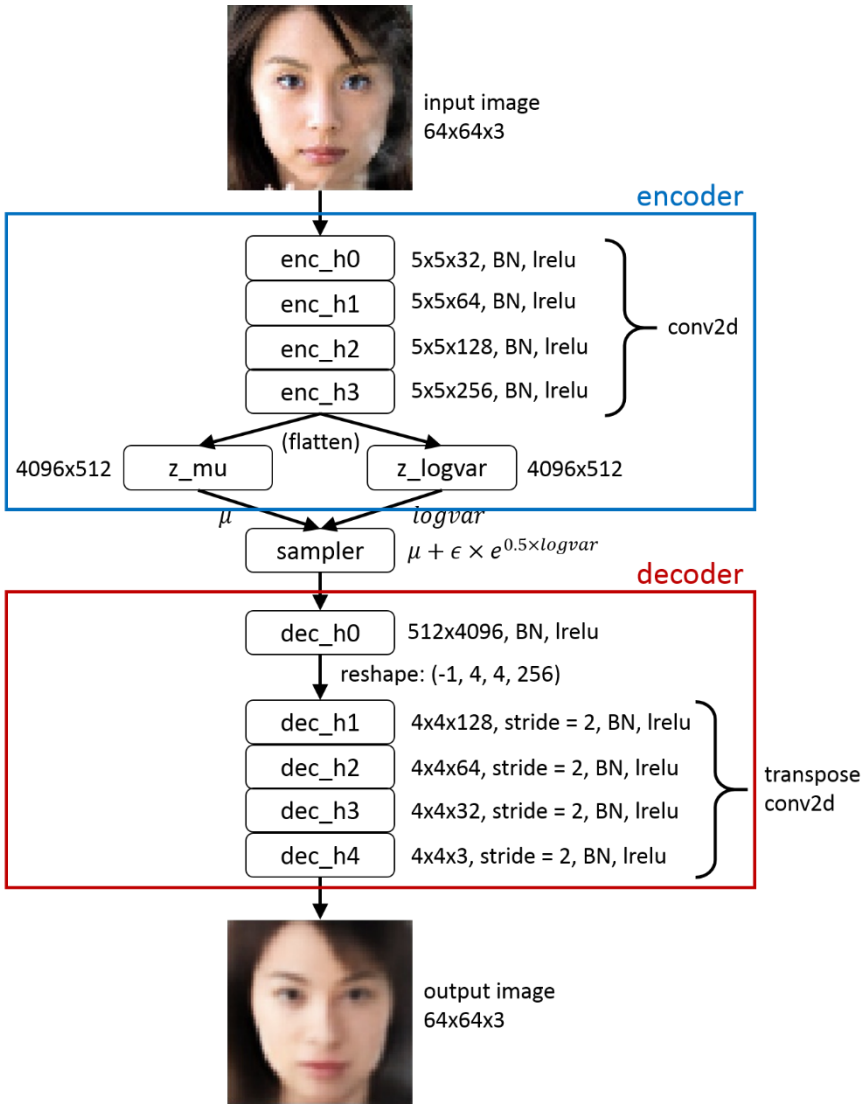
DLCV Spring 2018 HW4

d05921018 林家慶

Problem 1. VAE

1. Describe the architecture & implementation details of your model



As described in the above figure, the **encoder** contains four conv2d layers followed by two separate dense layers to produce mean and log-variance, each is a 512-dimensional vector. For each input image, the **sampler** generate 512 samples from the standard normal distribution ($N(0,1)$, implemented by *tf.random_normal()*) and outputs the reparametrized 512-dimensional normal samples, which is used as the input of the **decoder** below. The **decoder** contains a dense layer followed by four transposed conv2d layers that gradually enlarge the size of the feature maps, and finally outputs the desired reconstructed images or random-generated images.

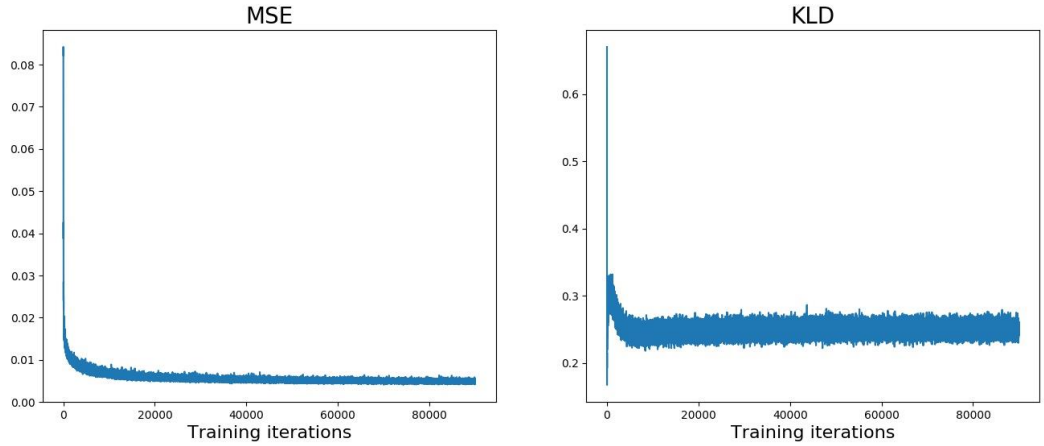
All Batch Normalization (BN) layers used *epsilon*=1e-5 and *momentum* = 0.9. All leaky-RELU activation functions (lrelu) used *alpha*=0.1.

I used simple MSE (take mean over the entire image for H, W, and C directions) to compute the reconstruction losses. As for the KL divergences, I used the formula:

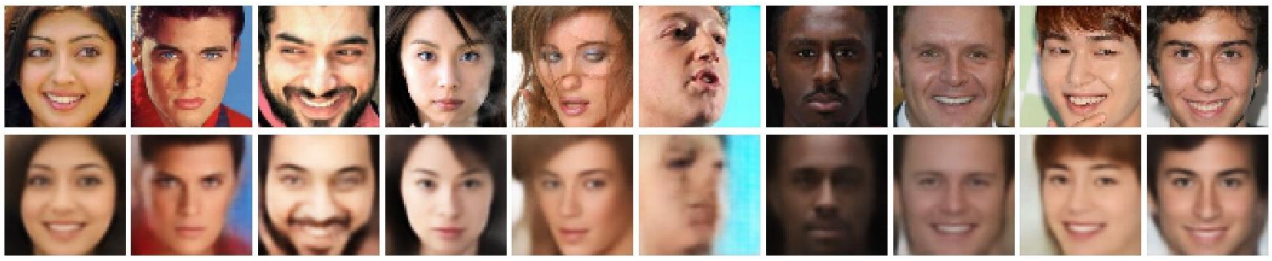
$$L_{KL} = 0.5 \times mean(e^{logvar} + \mu - 1 - logvar),$$

where the *mean* are taken over the latent (512-dim) space. After some experiments, I found that a ratio between MSE and KL divergence (λ_{KL}) set to 1e-2 leads to acceptable performance.

2. Plot the learning curve (reconstruction loss and KL divergence) of your model [fig1_2.jpg]

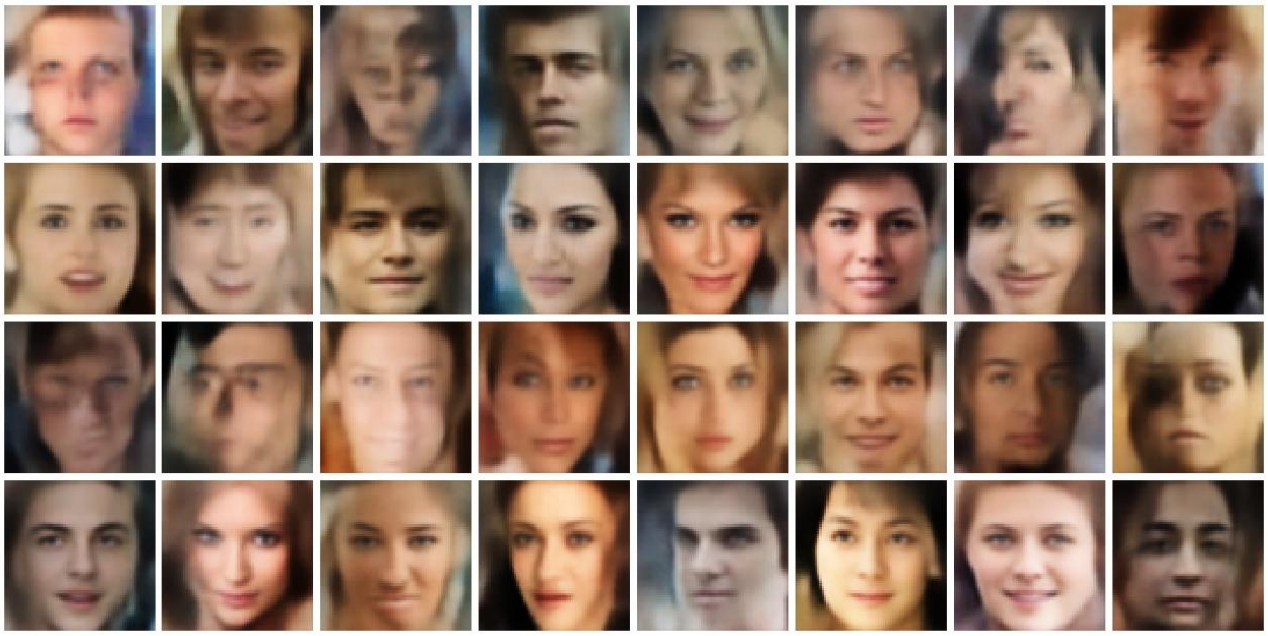


3. Plot 10 testing images and their reconstructed result of your model [fig1_3.jpg] and report your testing MSE of the entire test set

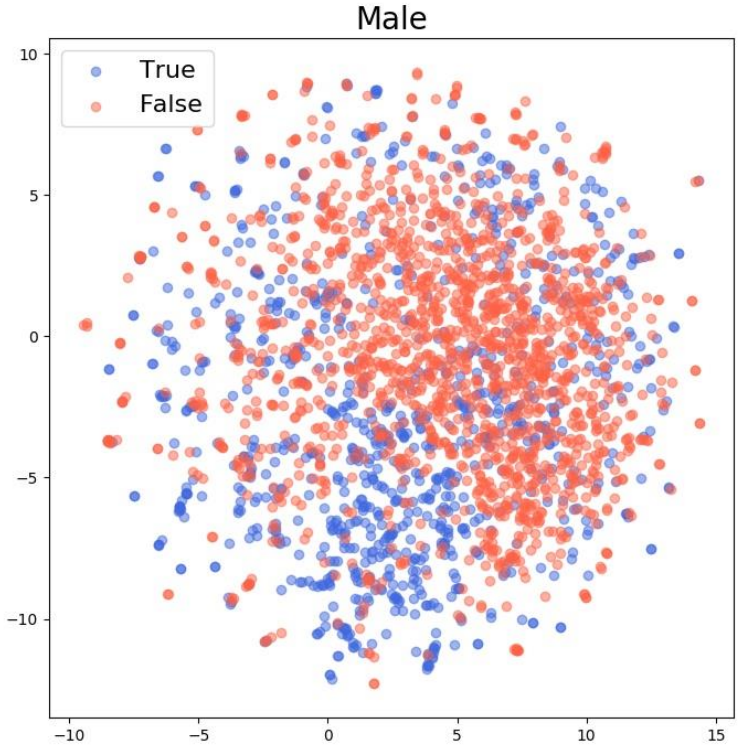


Testing MSE of the entire test set: 0.005205

4. Plot 32 random generated images of your model [fig1_4.jpg]



5. Visualize the latent space by mapping test images to 2D space (with tSNE) and color them with respect to an attribute of your choice [fig1_5.jpg]



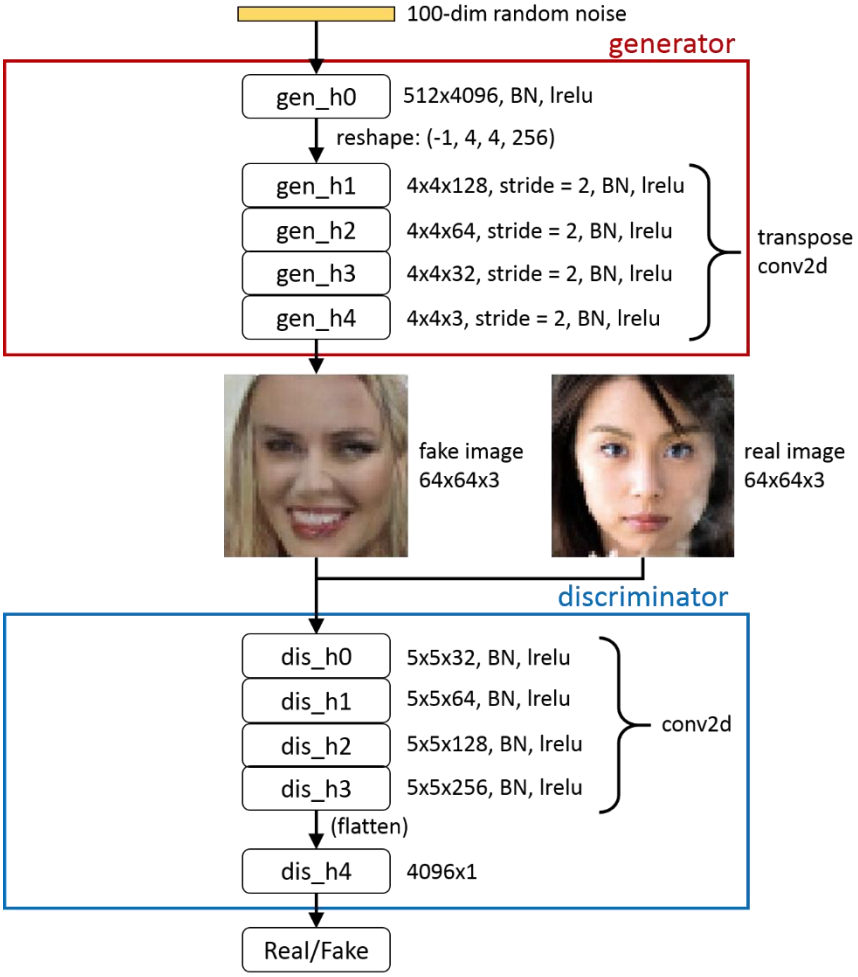
According to the above figure, the latent space might somehow catch difference between male and female.

6. Discuss what you've observed and learned from implementing VAE

(1) λ_{KL} should be set carefully to balance the quality of reconstructed images and random-generated images. A lower λ_{KL} (e.g., 1e-5 as first suggested by TA) emphasizes the importance of the reconstructed images and hence leads to extremely bad random-generated images. I think the main reason is the scale of KL divergence, which is below 1 in my implementation since I used *tf.reduce_mean()* to compute it over the latent space, whereas TA's might use *tf.reduce_sum()*.
(2) It makes no significant difference when I change the dimension of the latent space (512 or 1024) or batch size (32 or 64).

Problem 2. GAN

1. Describe the architecture & implementation details of your model



As described in the above figure, the **generator** takes a 100-dim random vector as its input, which will go through a dense layer followed by four transposed conv2d layers that gradually enlarge the size of the feature maps, and finally outputs the desired random-generated images. The **discriminator**, on the other hand, takes a 64x64x3 image (either from the real image dataset or from the output of the **generator**) as its input, which will go through four conv2d layers followed

by a dense layer to output the logit representing the log probability its input being a real image.

All Batch Normalization (BN) layers used *epsilon*=1e-5 and *momentum* = 0.9. All leaky-RELU activation functions (lrelu) used *alpha*=0.1.

Note that the output of **discriminator** is logit, so I used:

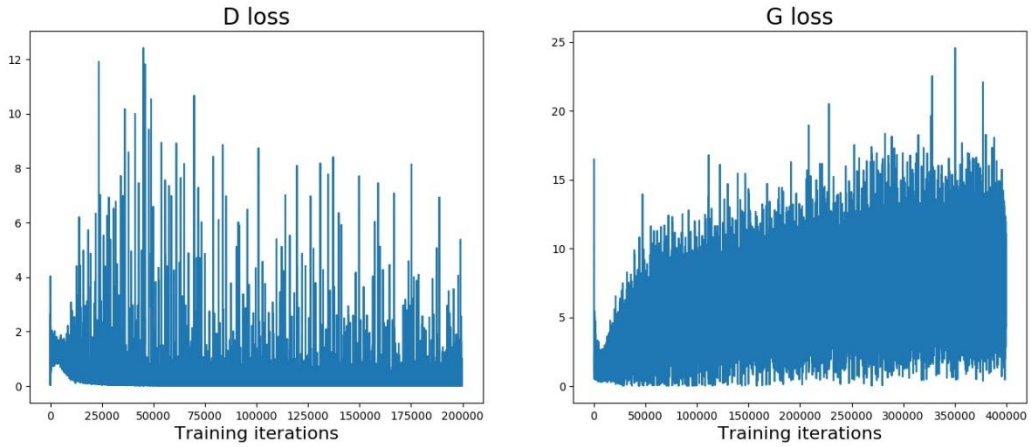
tf.nn.sigmoid_cross_entropy_with_logits()

to compute the losses resulted by both real and fake images. I also used

tf.trainable_variables()

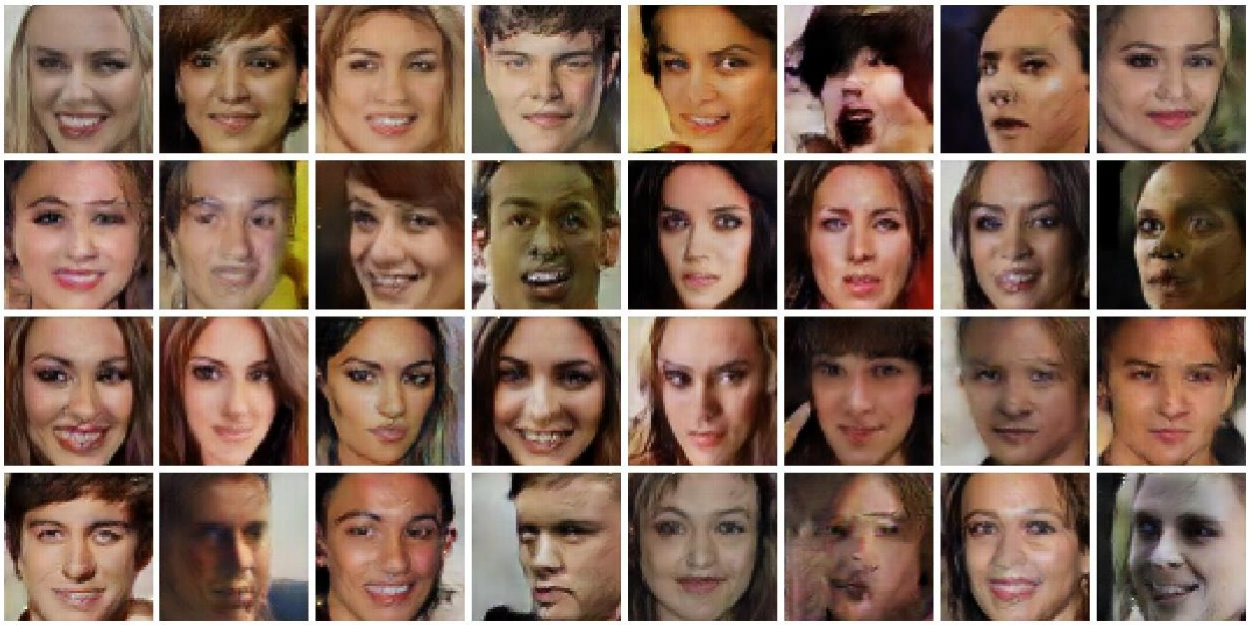
to extract trainable variables of **generator** and **discriminator**, separately, and put them into the tensorflow operations of optimization for **generator** and **discriminator**, respectively.

2. Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represent [fig2_2.jpg]



Since the updating ratio of **discriminator** and **generator** is 1:2, I got two time more G loss than D loss. From the figure above, it can be seen that the minimax competition between **discriminator** and **generator** reaches Nash equilibrium after somewhere around 25000 iterations for **discriminator** (corresponding to ~50000 iterations for **generator**), which suggests that the GAN training has succeed.

3. Plot 32 random generated images of your model [fig2_3.jpg]



4. Discuss what you've observed and learned from implementing GAN

(a) The BN layers and transpose conv2d (rather than upsamping) are keys to the successful training of GAN.

(b) I also implemented WGAN-GP, but the image qualities are not very good compared to those from GAN, as can be seen from the following example:

I think the main reason may be the lack of careful tuning of the scaling factor of the gradient penalty. Also, according to the WGAN-GP paper, the difference between GAN and WGAN-GP is

mainly on the "bad-constructed" networks, i.e., the authors only claims that WGAN-GP can still generate images with moderate qualities when the neural network is not well-constructed (e.g., use dense layers rather than CNN). If the network architecture is well-designed, such as following the guidelines given by DCGAN, then GAN and WGAN should have similar performance.

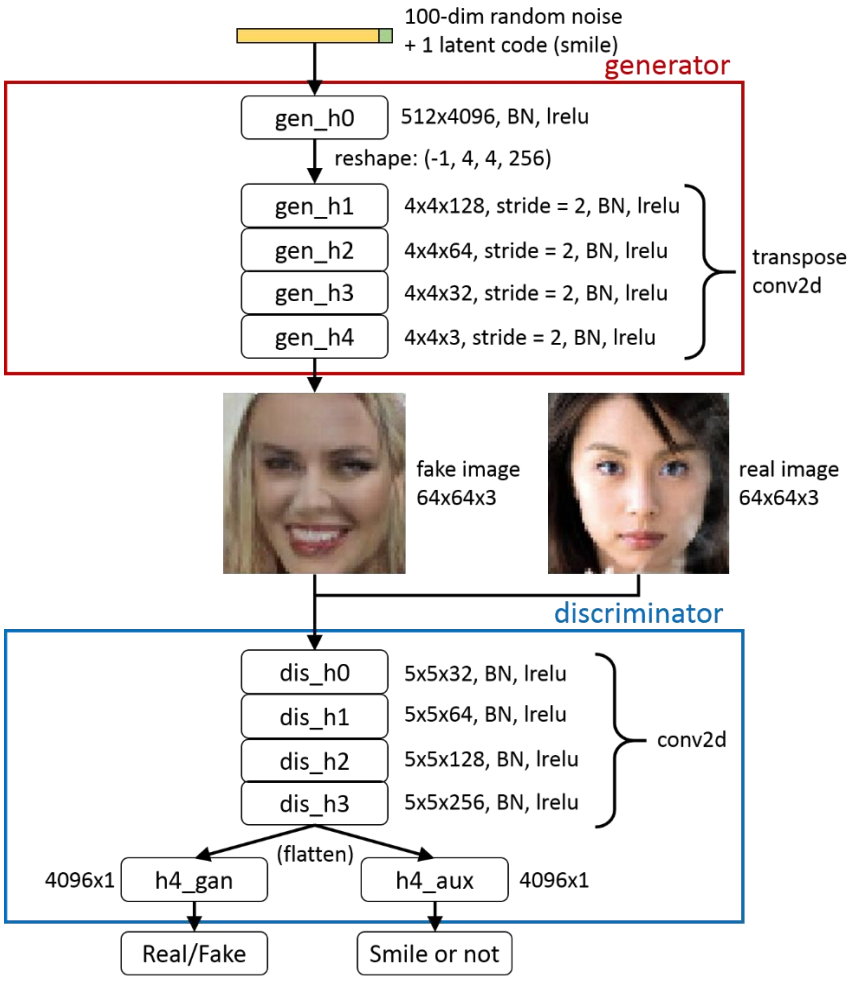
5. Compare the difference between image generated by VAE and GAN, discuss what you've observed

(a) VAE produces more blurry images than GAN

(b) VAE produces images with similar qualities, whereas GAN sometimes produces extremely bad images, like the third to the right in the first row in fig2_3.jpg above.

Problem 3. ACGAN

1. Describe the architecture & implementation details of your model



As described in the above figure, the architecture of ACGAN is very similar as the architecture of GAN in Problem 2. The only two exceptions are: (1) There is one more dimension representing

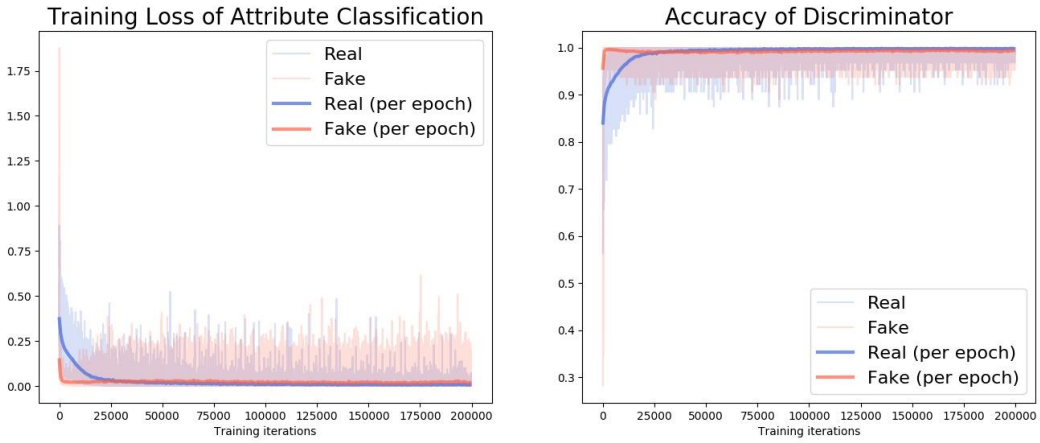
the latent code (smile or not in my implementation) appended to the random noise to make the input of the **generator** 101 dimension; (2) The **discriminator** has one more output (dense) layer representing the classification results (smile or not in my implementation) of its input images.

The definitions of loss related to GAN are the same as those defined in GAN in Problem 2. As for the auxiliary loss, I used

tf.nn.sigmoid_cross_entropy_with_logits()

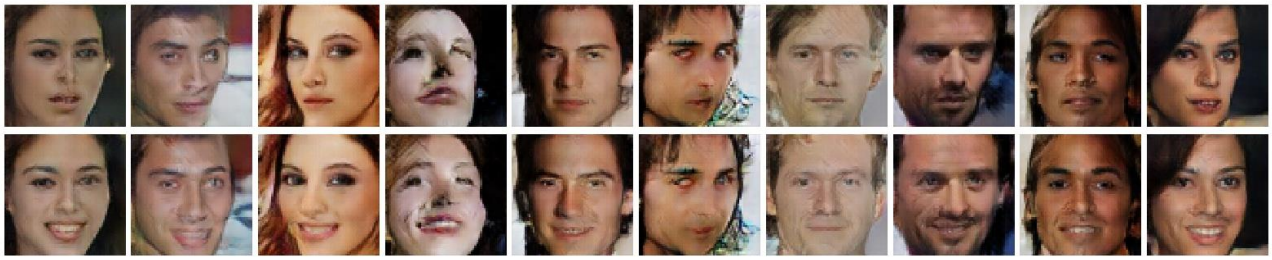
since my **discriminator** outputs logits. The loss for both **generator** and **discriminator** will be added by this auxiliary loss, and in either cases the training process should try to minimize it.

2. Plot the learning curve (in the way you prefer) of your model and briefly explain what you think it represent [fig3_2.jpg]



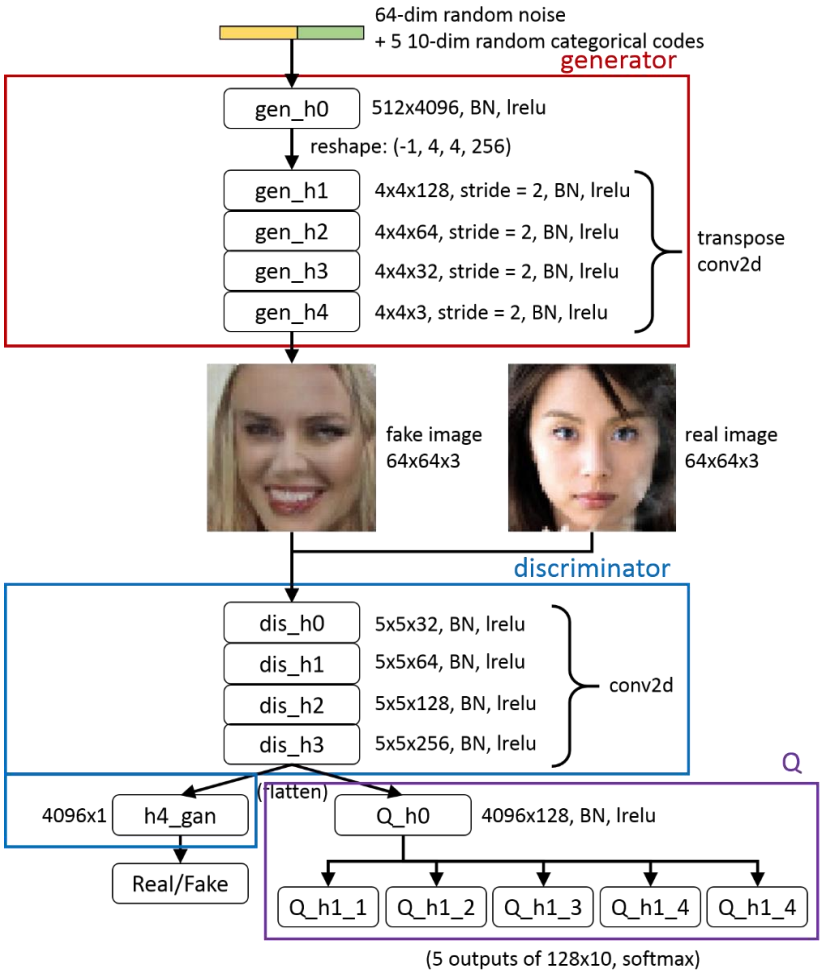
Again, it can be seen that the minimax competition between **discriminator** and **generator** reaches Nash equilibrium after somewhere around 25000 iterations, which suggests that the GAN training has succeed.

3. Plot 10 pair of random generated images of your model, each pair generated from the same random vector input but with different attribute. This is to demonstrate your model's ability to disentangle feature of interest [fig3_3.jpg]



Bonus. InfoGAN

1. The network architecture



As described in the above figure, the architecture of INFOGAN is very similar as the architecture of ACGAN in Problem 3. The only two exceptions are: (1) I used five 10-dim categorical latent codes, which is appended to the 64-dim random noise to make the input of the **generator** 114 dimension; (2) The auxiliary output of **discriminator** from ACGAN is changed to the **Q network**, which has one more dense layer followed by five dense components with softmax activation functions representing the logits of the five 10-dim categorical labeling. During the training process, the five 10-dimension categorical latent codes are randomly generated.

The loss function of the **Q network** is the sum of the conditional entropy (computed as the binary cross entropy between the output of **Q network** and the random input latent codes) and the input entropy (computed as the binary cross entropy between the random input latent codes to themselves), as shown in the following formulae:

cond_entropy = mean(-sum(log(Q_out) * input_labels))

entropy = mean(-sum(log(input_labels) * input_labels))

loss_q = cond_entropy + entropy

During training, the **generator** and **discriminator** are trained as usual, while the **Q network** is

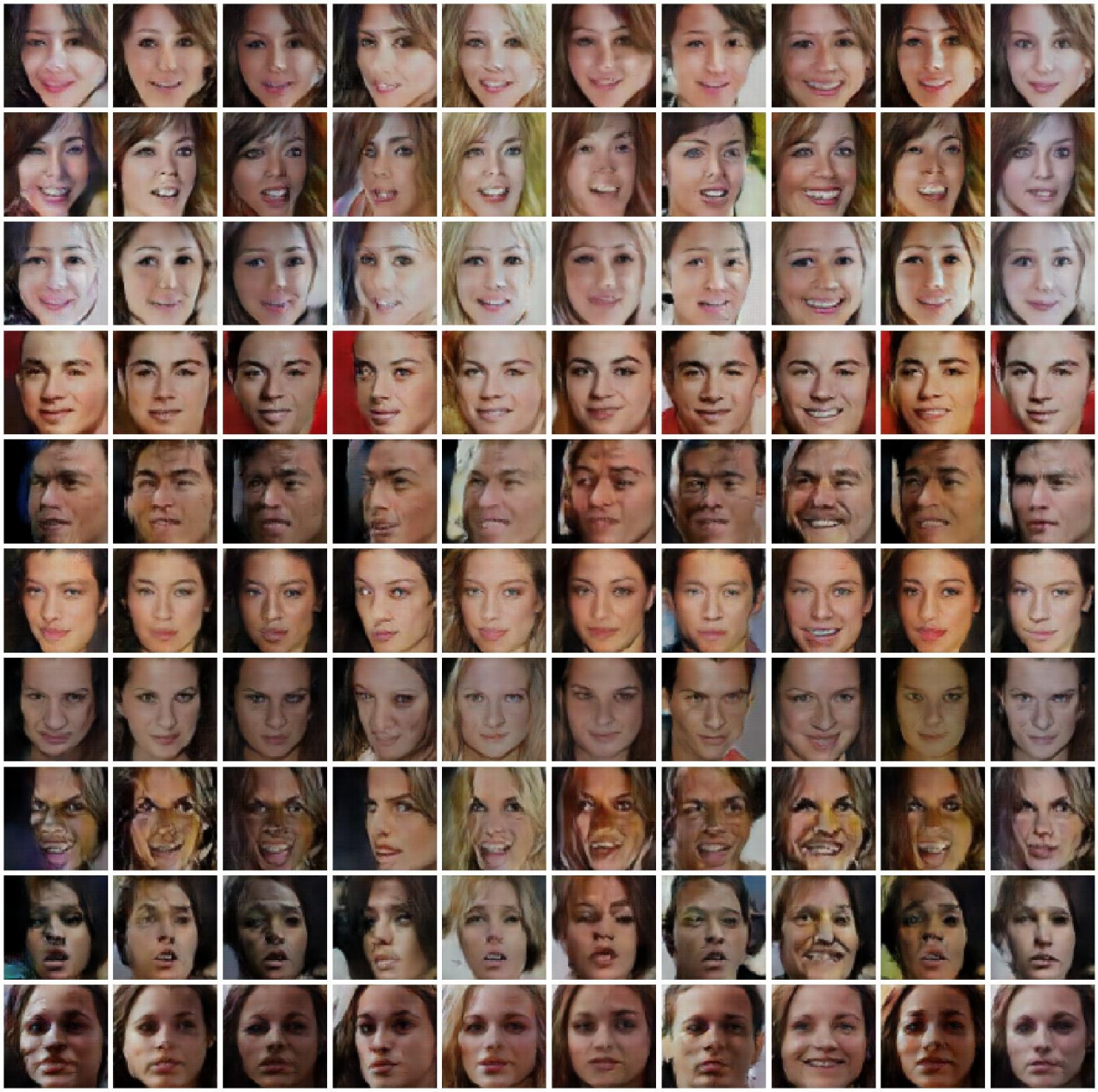
trained by minimizing the above loss_q via adjusting both parameters from the **generator** and **Q network**.

2. Manipulation of categorical latent codes

The first 10-dim categorical code:

It can be seen that this categorical code might catch the latent information of the hair styles, such as the "Bang" effects of the 3rd bit.

The second 10-dim categorical code:



It can be seen that this categorical code might catch the latent information of the hair colors, such as the "Blonde" effects of the 5th bit.

3. Discussion

(1) From the InfoGAN paper, it is hard to tell how the authors manipulated the categorical latent codes of CelebA. So I guess they just consider one categorical code at a time and enumerate each bit to see the effects catched by this code.

(2) The number of training data in this homework is five time less than the number used in the InfoGAN paper, and that's may be the main reason that I cannot see the emotion effects or glasses

effects.