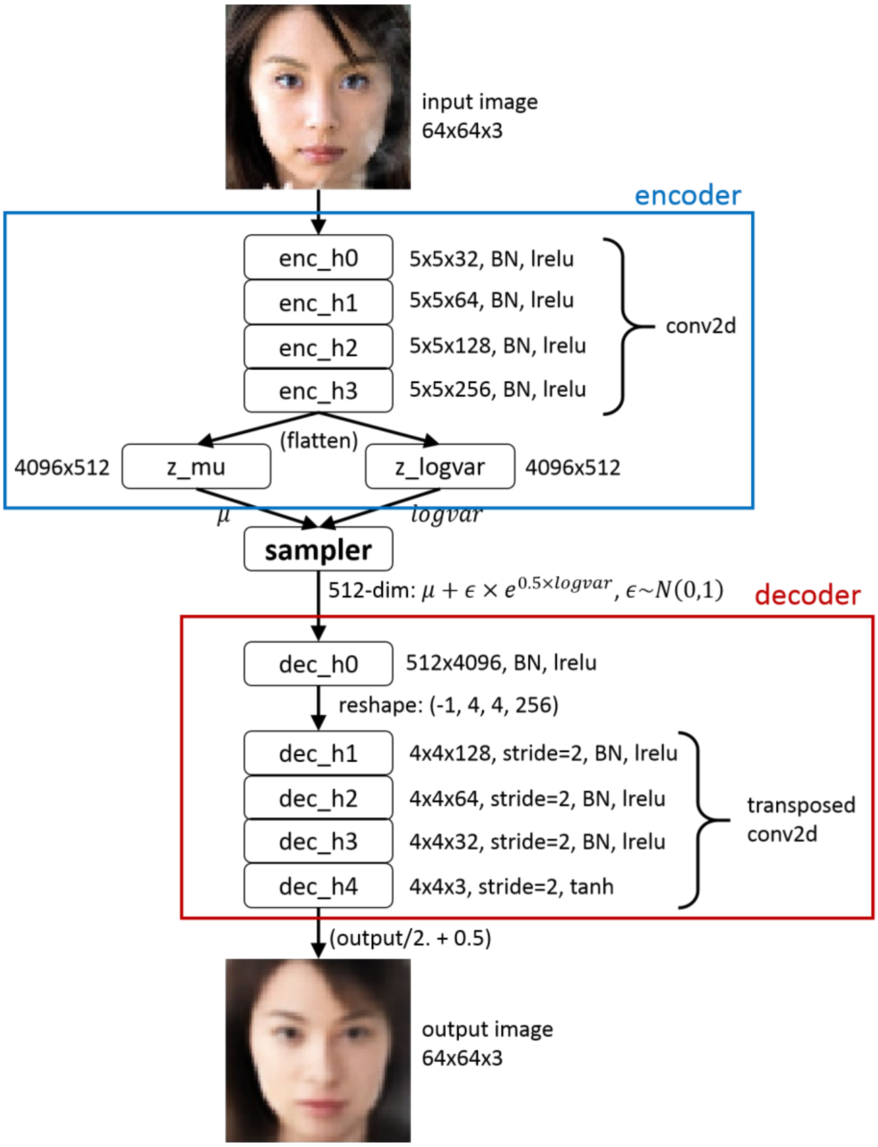
DLCV Spring 2018 HW4

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 latent vector, 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 in the end outputs the desired reconstructed images or random-generated images.

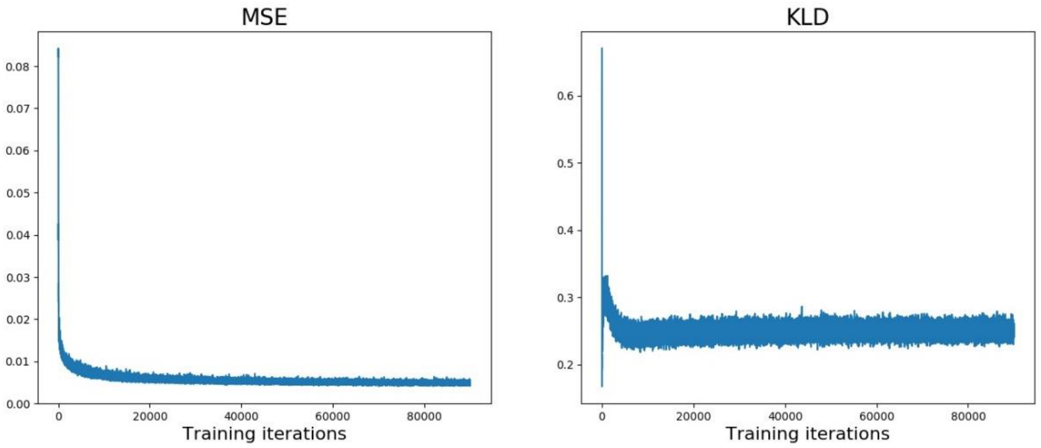
All Batch Normalization (BN) layers used *epsilon*=1e-5 and *momentum*=0.9. All leaky-RELU activation functions (lrelu) used *alpha*=0.01.

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:

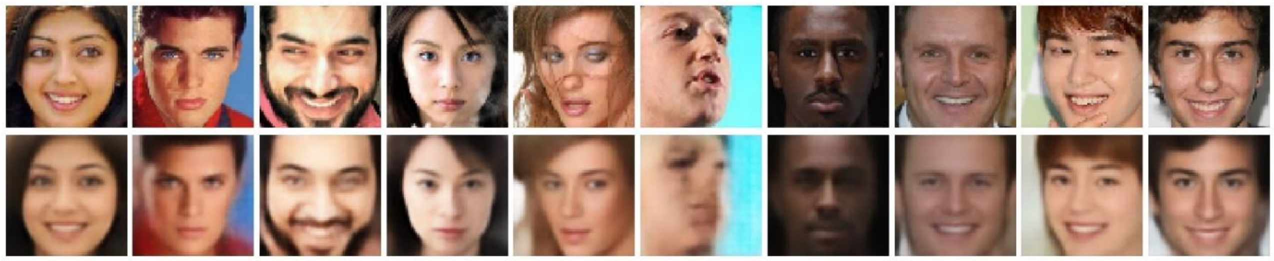
$$L_{KL} = 0.5 \times \text{mean}(e^{logvar} + \mu - 1 - logvar),$$

where the "mean" are taken over the latent (512-dim) space. I used Adam optimizer with *momentum*=0.5 and initial *learning_rate*=2e-4 to minimize the total loss: MSE + $\lambda_{KL} \times L_{KL}$. After some experiments, I found that a ratio λ_{KL} set to 1e-2 leads to acceptable performance. I used batch size 64 to train 200 epochs and picked the model with the smallest MSE of the entire test set (after training).

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

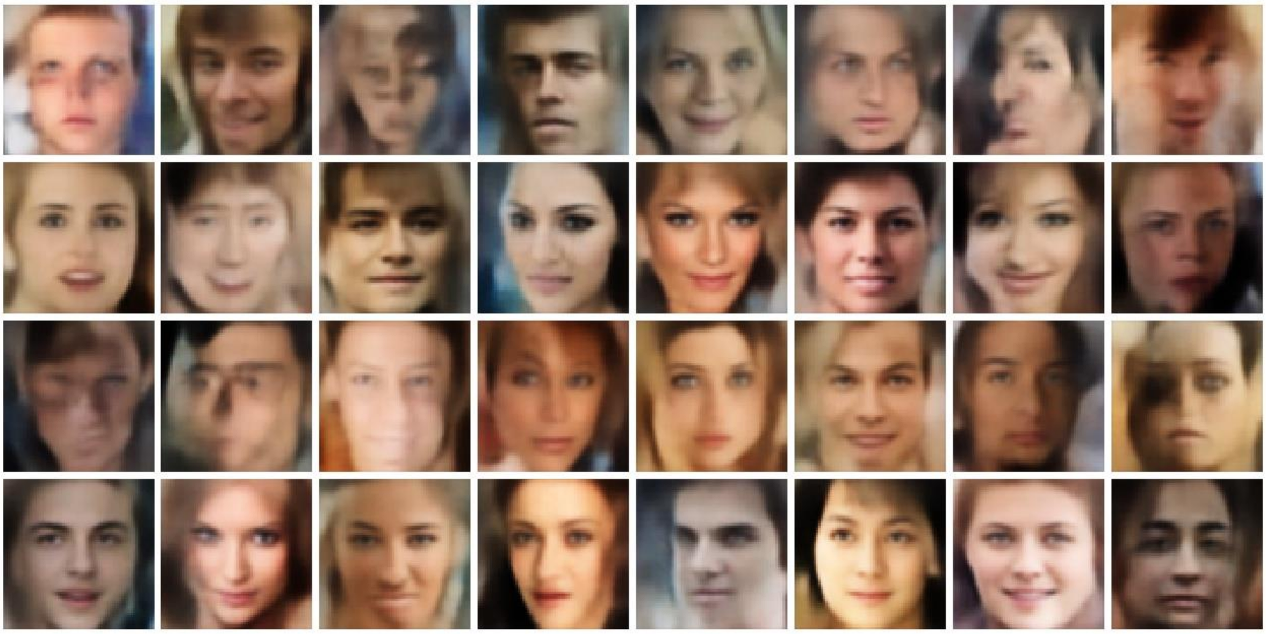


3. Plot 10 testing images and their reconstructed result of your model [fig1_3.jpg] and report your testing MSE of the entire test set

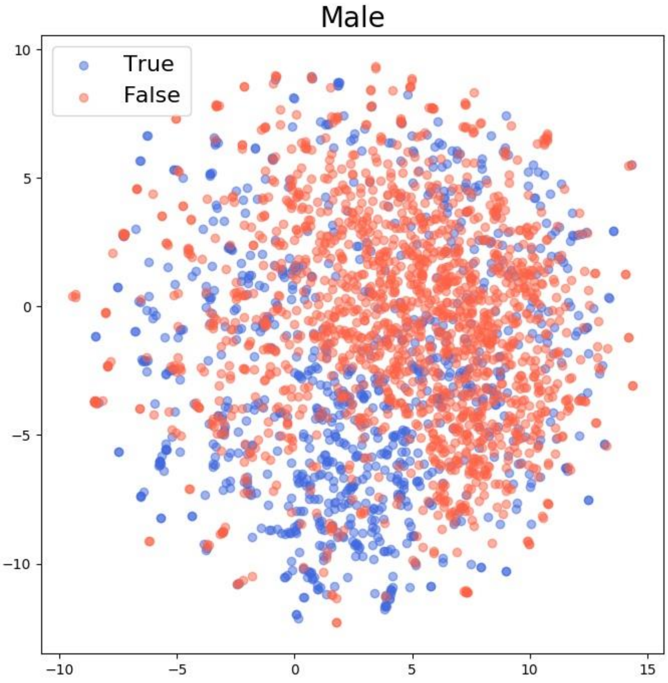


Testing MSE of the entire test set: 0.005205

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]



I fed all 2621 testing images into the trained encoder and used the TSNE function imported from *sklearn.manifold* to project all their 512-dim latent vectors into 2-dim plane. According to the above figure, the latent space might somehow catch the gender information, since most of the females have a trend toward the upper-right corner in the 2-dim space, whereas the males are distributed more evenly over the entire plane.

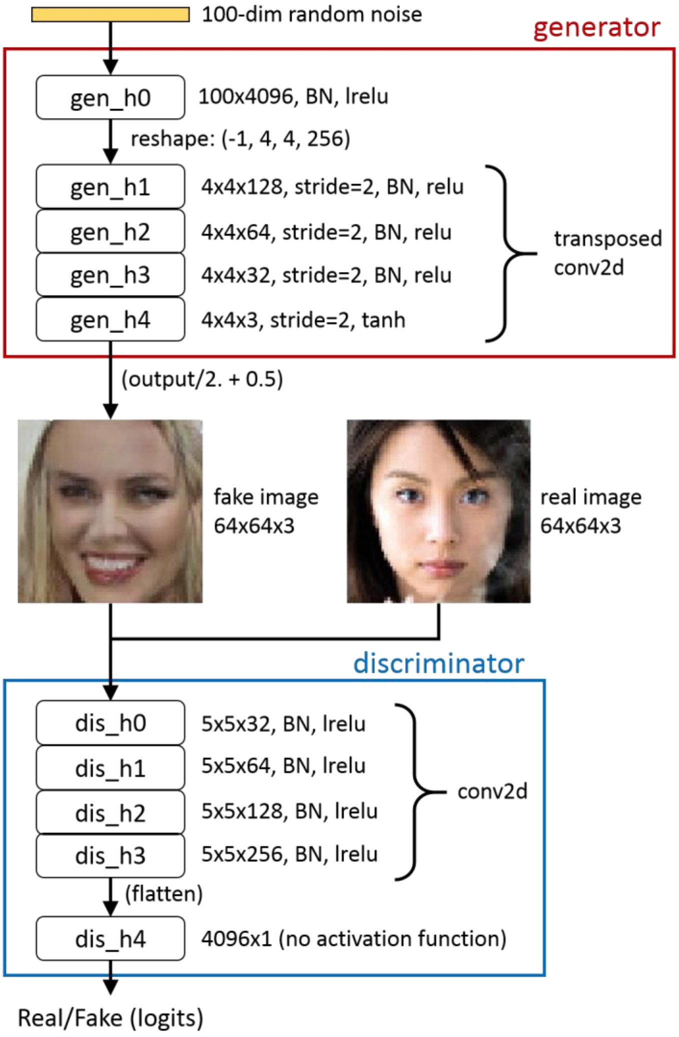
6. Discuss what you've observed and learned from implementing VAE

(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), batch size (32 or 64), or learning rate (1e-4, 2e-4, or 1e-5).

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 in the end 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 of 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 in the **discriminator** have *alpha*=0.1.

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

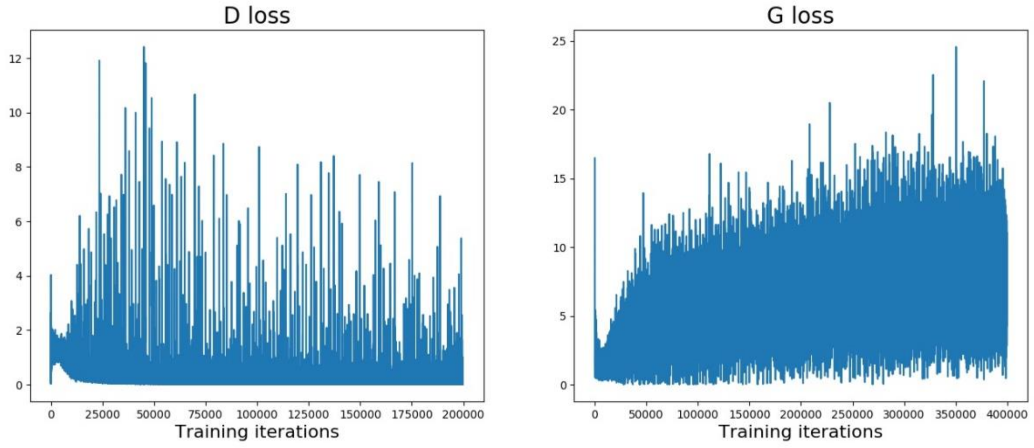
tf.nn.sigmoid_cross_entropy_with_logits()

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

tf.trainable_variables()

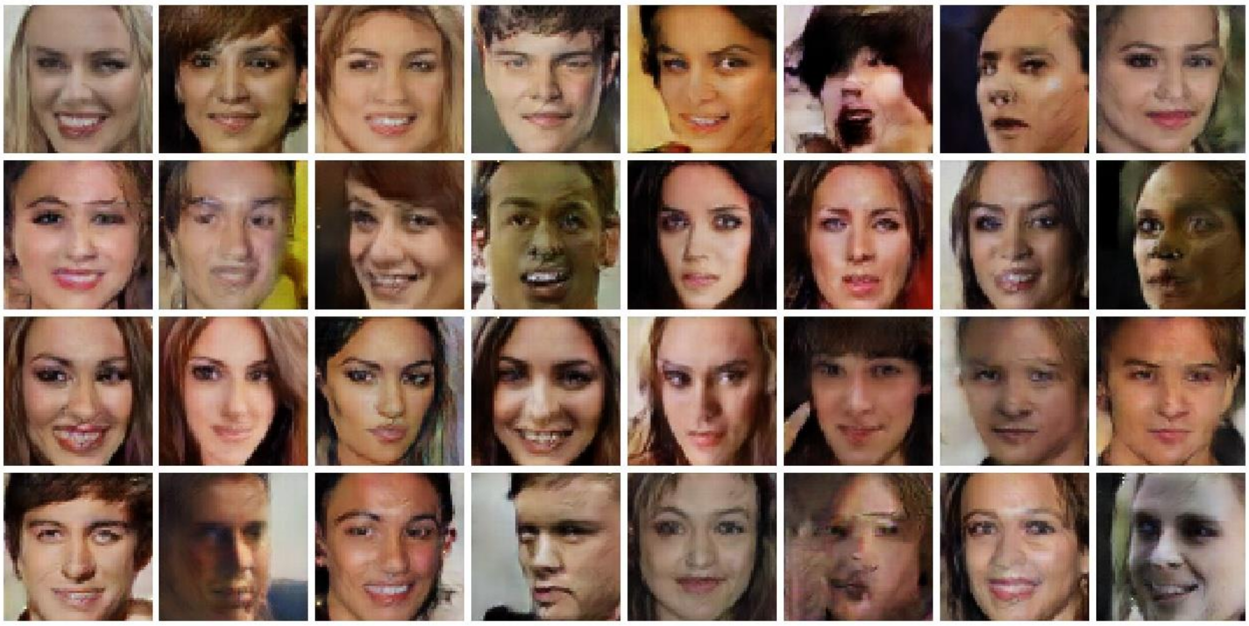
to extract trainable variables of **generator** and **discriminator**, separately, and put them into the minimizers of Adam optimizers (with *momentum*=0.5 and initial *learning_rate*=2e-4) for **generator** and **discriminator**, respectively. The updating ratio between **discriminator** and **generator** is set to 1:2. I used batch size 64 to train 300 epochs.

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 between **discriminator** and **generator** is 1:2, I got two times 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.

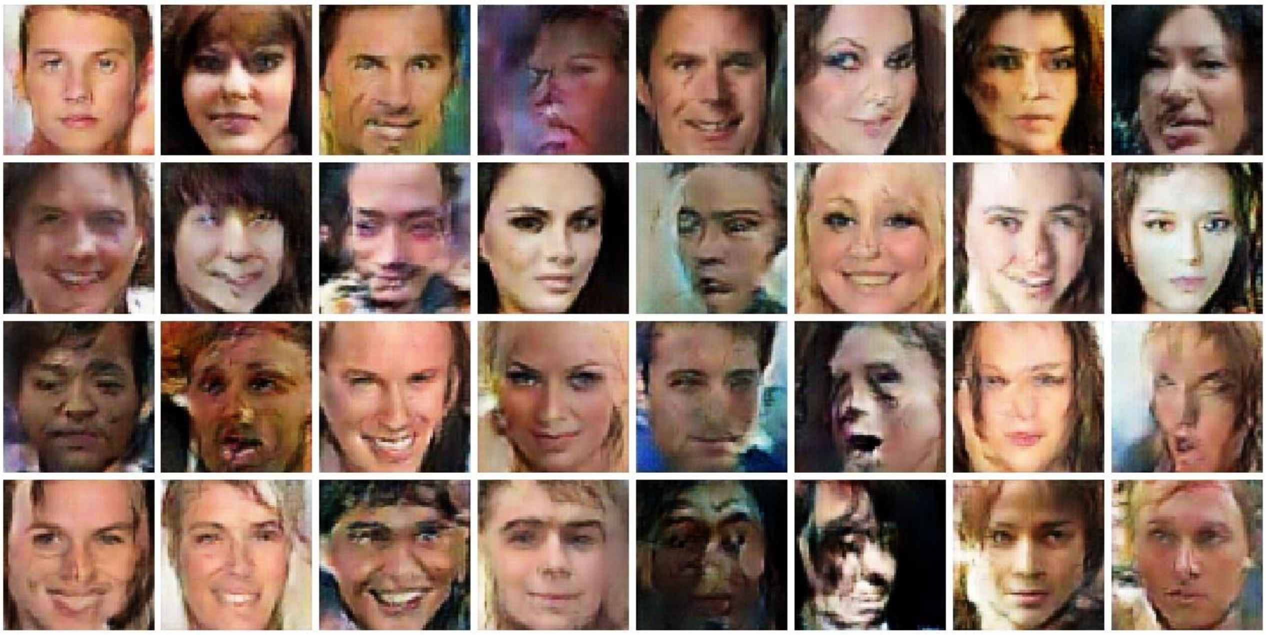
3. Plot 32 random generated images of your model [fig2_3.jpg]



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

(1) Following the architecture guideline provided by Radford et al. in DCGAN [1], the BN layers and transposed conv2d (rather than upsamping) are important keys to the successful training of GAN.

(2) 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:

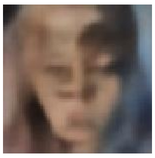


I think the main reason may be the lack of careful tuning of the scaling factor of the gradient penalty. Also, according to the WGAN-GP paper [2], the difference between GAN and WGAN-GP is mainly on the "difficult GAN architectures", i.e., the authors only claims that WGAN-GP can still generate images with moderate qualities when the neural network is not well-constructed (e.g., no BN and a constant number of filters for the **generator**, use 101-layer ResNet for both **generator** and **discriminator**, etc.). If the network architecture is well-designed, such as my implementation that follows the guidelines given in [1], then GAN and WGAN-GP should have similar performance.

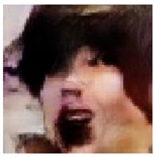
5. Compare the difference between image generated by VAE and GAN, discuss what you've observed

(1) VAE produces more blurry images than GAN, especially for the background and hair. Since one of the main goal of VAE is to minimize the reconstruction losses, it may tend to produce the "average" of all its training images, and the boundary areas (background and hair) may be much more diverse than the central areas (human faces) of the images.

(2) Although GAN has less number of latent dimensions (100) than VAE (512), it seems that both of them do not suffer from the "mode collapse" effect, i.e., they both produce images with good diversities. However, both of them sometimes produce extremely bad images, like the third to the left in the first row of fig1_4.jpg (VAE):

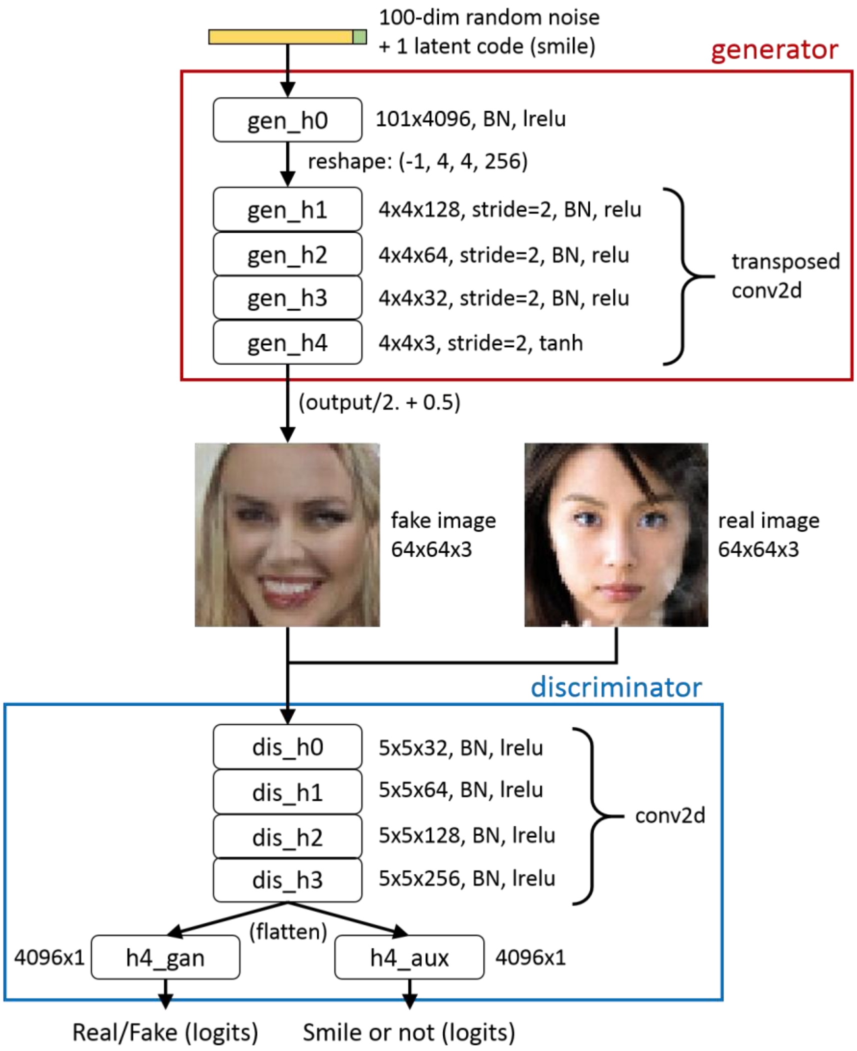


and the third to the right in the first row of fig2_3.jpg (GAN):

.

Problem 3. ACGAN

1. Describe the architecture & implementation details of your model



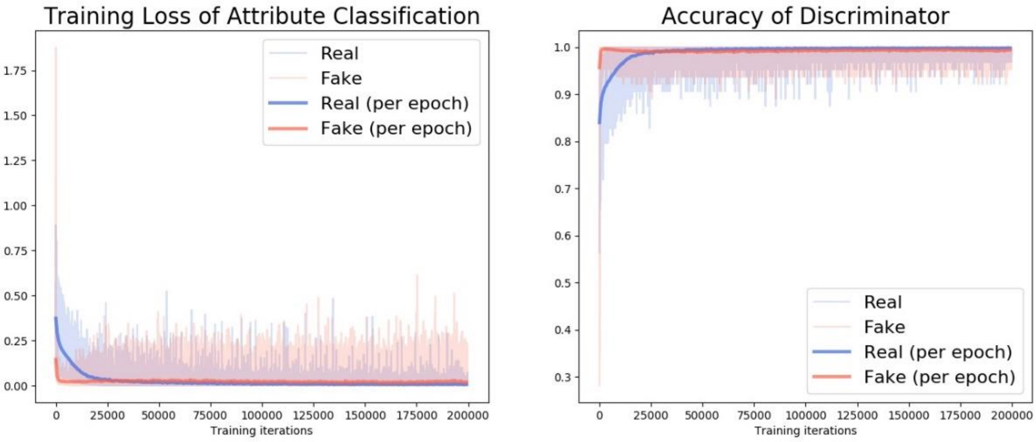
As described in the above figure, the architecture of ACGAN is very similar as the architecture of GAN in **Problem 2**. The only two exceptions are: (1) There is one more dimension representing the latent code (smile or not in my implementation) appended to the random noise to make the input of the **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()

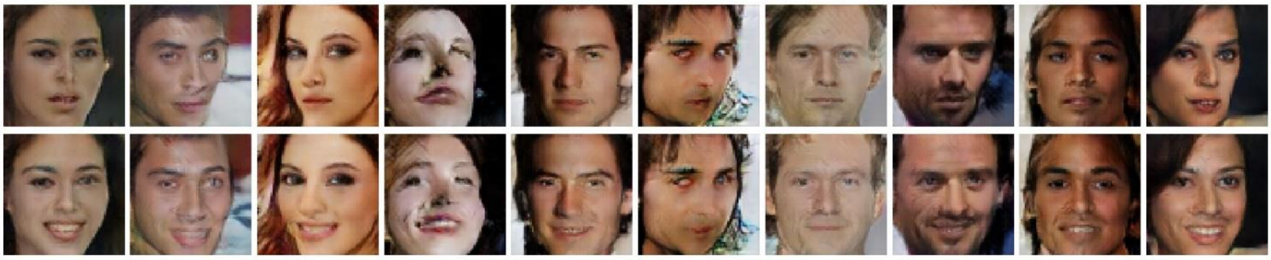
since my **discriminator** outputs logits. The loss for both **generator** and **discriminator** will be added by this auxiliary loss, and in either cases the Adam optimizers (with *momentum*=0.5 and initial *learning_rate*=2e-4) should try to minimize it. The updating ratio between **discriminator** and **generator** is set to 1:1. I used batch size 64 to train 300 epochs.

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.

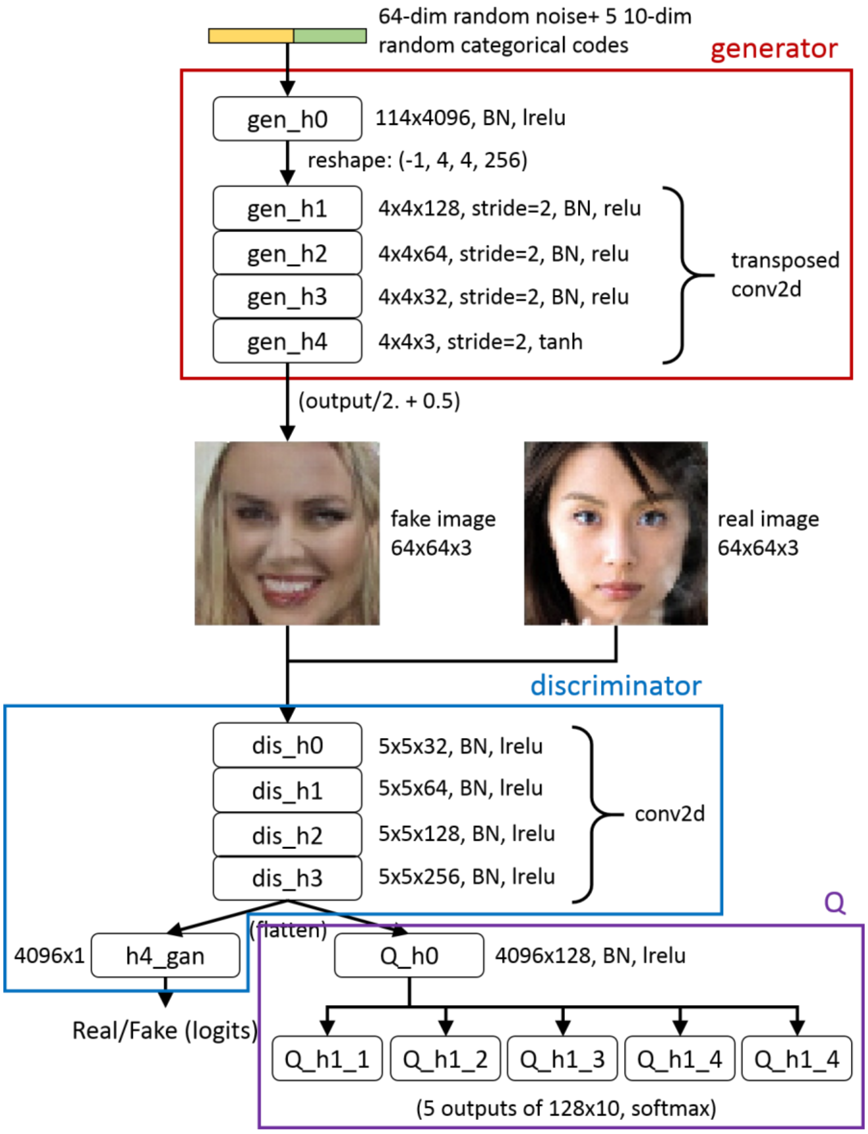
3. Plot 10 pair of random generated images of your model, each pair generated from the same random vector input but with different attribute. This is to demonstrate your model's ability to disentangle feature of interest [fig3_3.jpg]



The first row has latent code 0 (not smile) and the second row has 1 (smile).

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 probabilities of all 10 labels for each of the five categorical codes. 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 process, there are three Adam optimizers (all with *momentum*=0.5 and initial *learning_rate*=2e-4), with two of them optimize **generator** and **discriminator** as usual, and one of them optimizes the **Q network** by minimizing the above **loss_q** via adjusting both parameters from the **generator** and **Q network**. The updating ratio between **discriminator**, **generator**, and **Q network** is set to 1:1:1. I used batch size 64 to train 300 epochs.

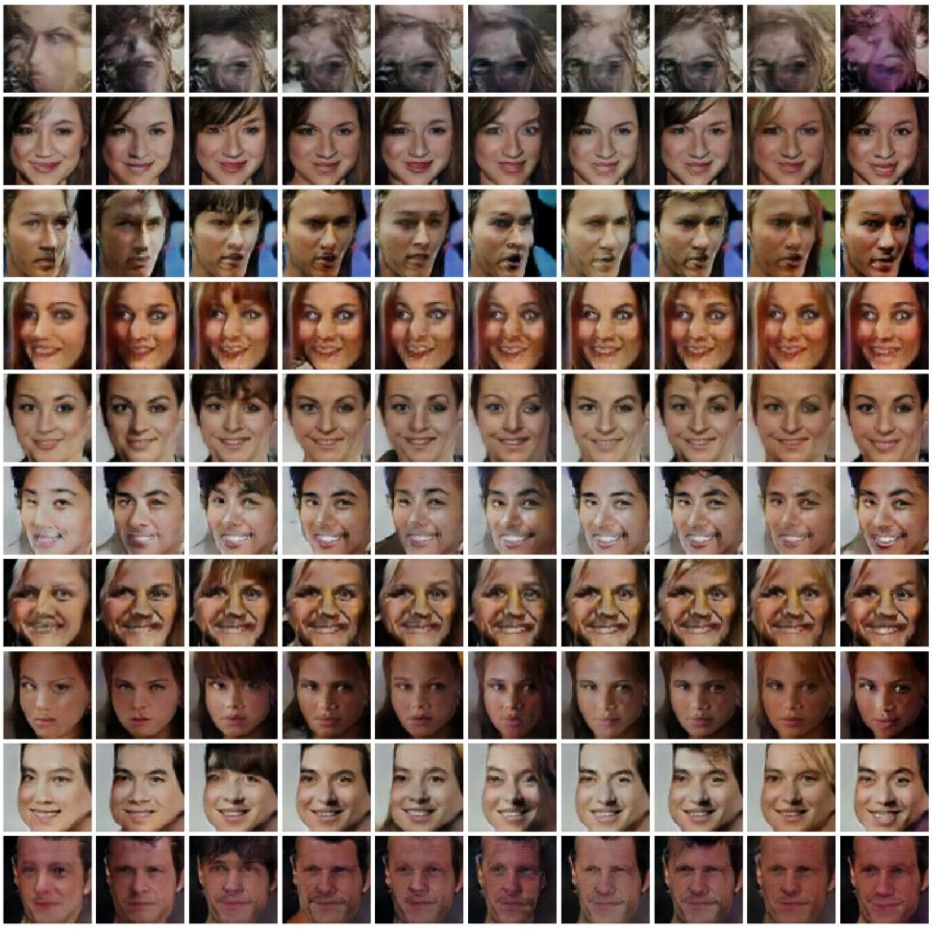
2. Manipulation of categorical latent codes [fig4_01.jpg and fig4_02.jpg]

In each of the following 2 examples, only one 10-dim latent code is enumerated as:

[1,0,0,0,0,0,0,0,0,0], [0,1,0,0,0,0,0,0,0,0], …, [0,0,0,0,0,0,0,0,0,1],

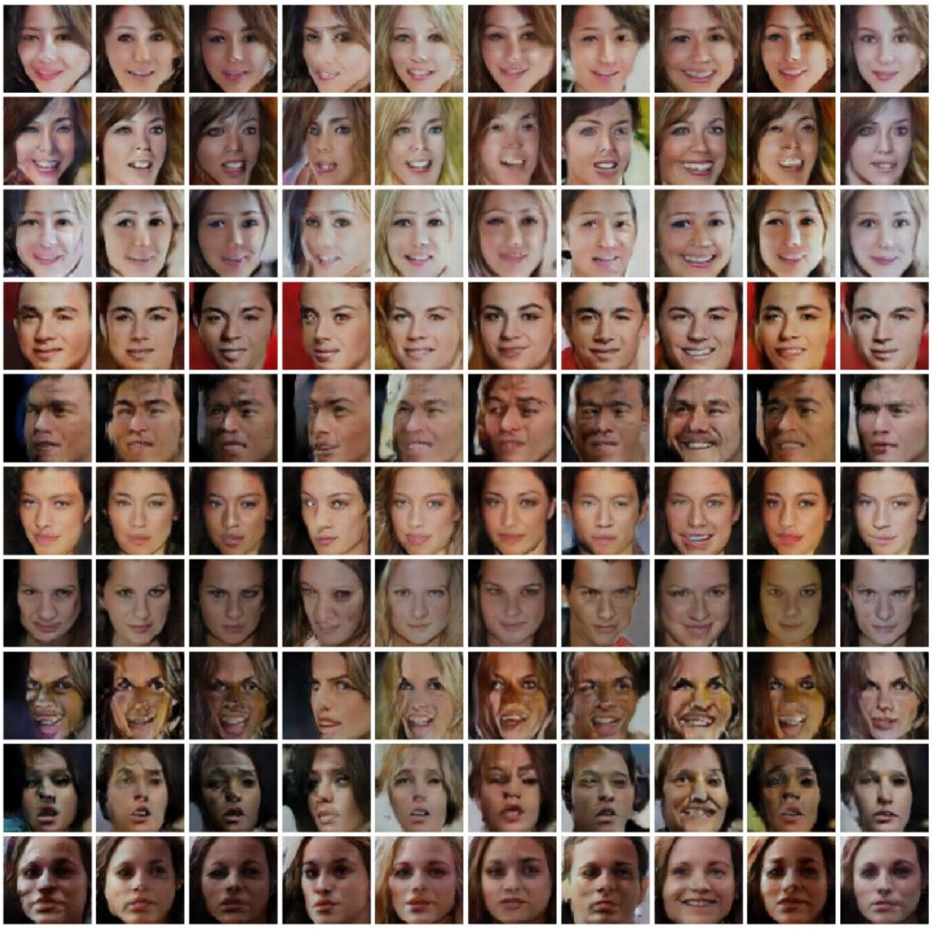
one for each column, while the 64 random noise and the rest 4 10-dim latent codes are kept the same within each row.

(1) The 1st code:



It can be seen that the 1st categorical code might catch the latent information of the hair styles, such as the "Bang" effect of the 3rd bit, and the "Bald" effect of the 10th bit.

(2) The 2nd code:



It can be seen that the 2nd categorical code might catch the latent information of the hair colors, such as the "Blonde" effect of the 5th bit, and the "Dark hair" effect of the 7th bit.

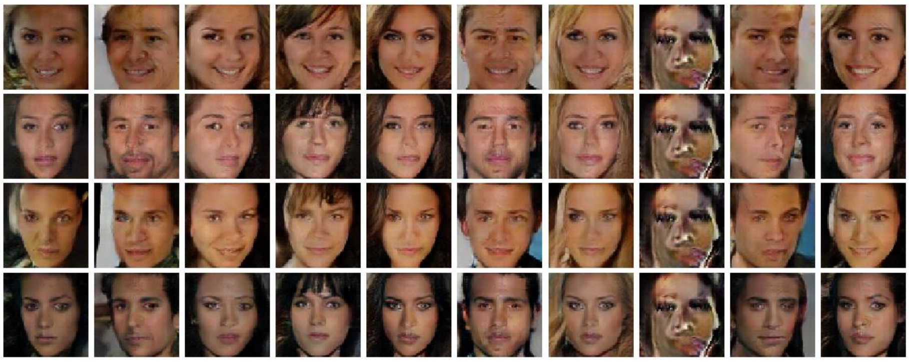
3. Discussion

(1) From the InfoGAN paper [3], 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 latent information catched by each code. Otherwise, it would be too inefficient to consider all c^d different bit patterns, where c is the number of categorical codes, and d is the dimension in each code.

(2) The number of training data in this homework is five times less than the number used in the InfoGAN paper (40,000+ vs. 200,000), so I halved both the dimension of the random noise (128 to 64) and the number of categorical codes (10 to 5). Also, the less number of training images may be the main reason of the slightly worse performance of my implementation. For example, I cannot see the emotion effects or the glasses effects illustrated in the InfoGAN paper.

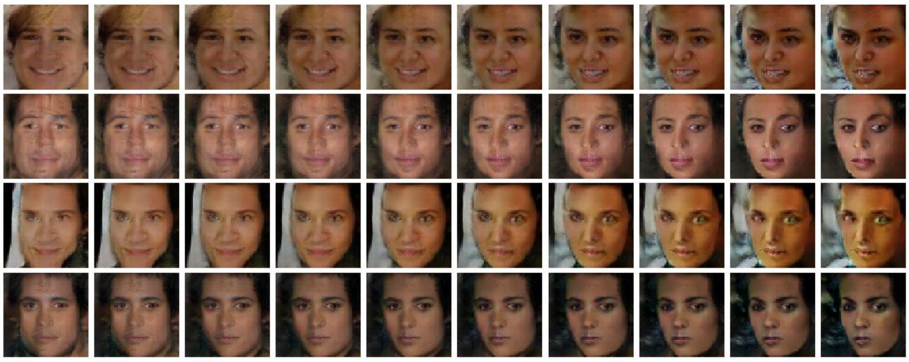
(3) I also tried the same settings of the MNIST experiment illustrated in the InfoGAN paper: 62-dim random noise + 1 10-dim categorical code + 2 continuous codes. The results are also interpretable:

(a) Manipulation of the only categorical code:



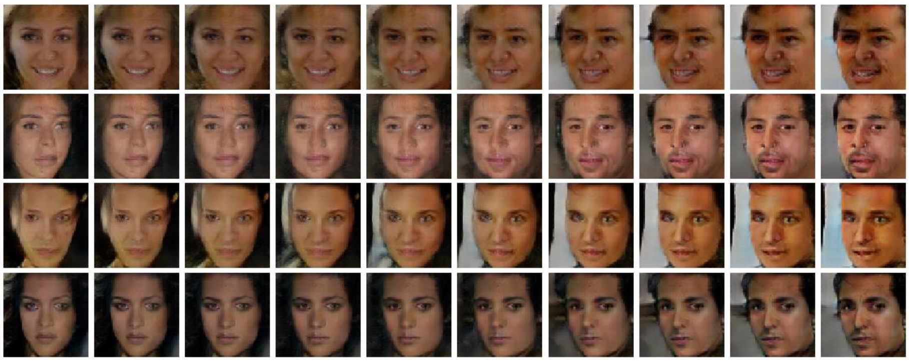
It can be seen that the only categorical code might catch the latent information of "different kinds" of faces, just like different digits in the MNIST experiment.

(b) Manipulation of the 1st continuous code:



It can be seen that the 1st continuous code might catch the gender information, from left to right, we can see a man gradually becomes a woman.

(c) Manipulation of the 2nd continuous code:



It can be seen that the 2nd continuous code might also catch the gender information, as well as the pose information: from left to right, we can see a woman gradually becomes a man, with head slightly turning to the right.

Reference

[1] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks", arXiv:1511.06434, 2015.
[2] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans", arXiv:1704.00028, 2017.
[3] A. Odena, C. Olah, and J. Shlens, "Conditional image synthesis with auxiliary classifier gans", arXiv:1610.09585, 2016.