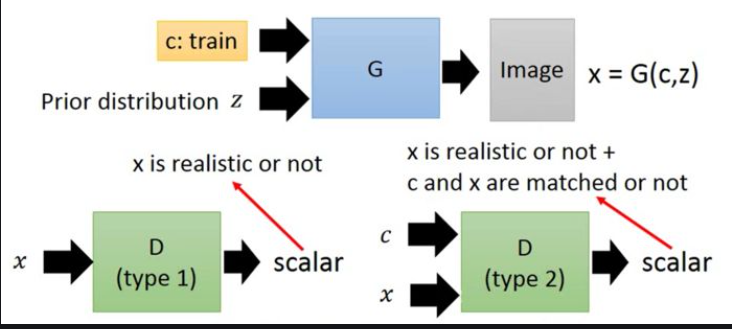
LAB 07 cGAN and cNF

Introduction

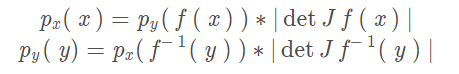
cGAN :

The ideal of cGAN is let two model improve by each other by training , one is generator which purpose is generating images , another is discriminator which purpose is judging the images created by generator .



NF :

The ideal of NF is to find an invertible transform ( f ) between the data distribution ( Px ) and latent ( Py ) .



Implement

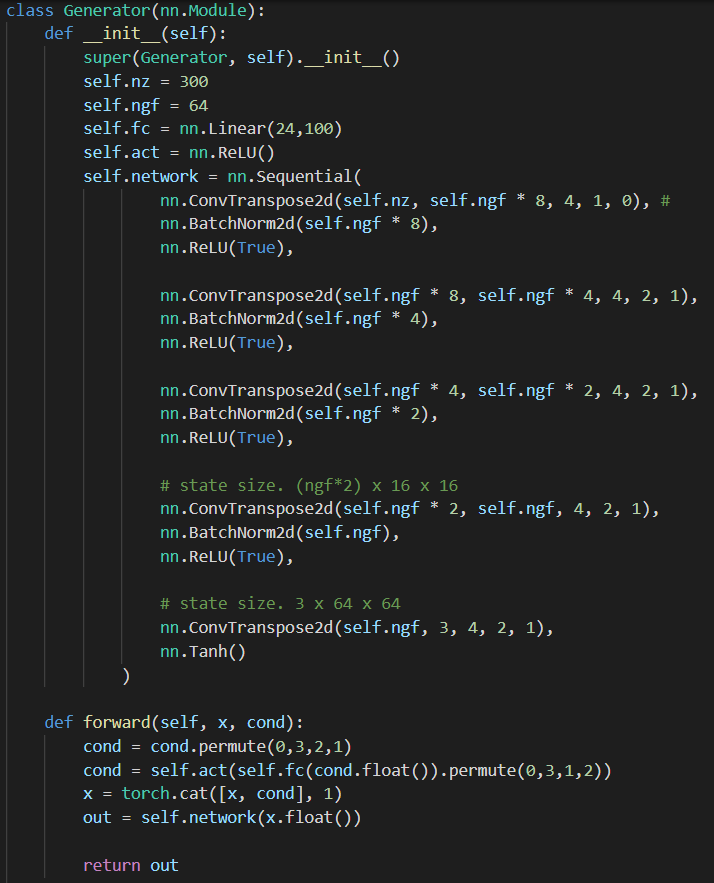
Task1

1. cGAN
   1. Architecture

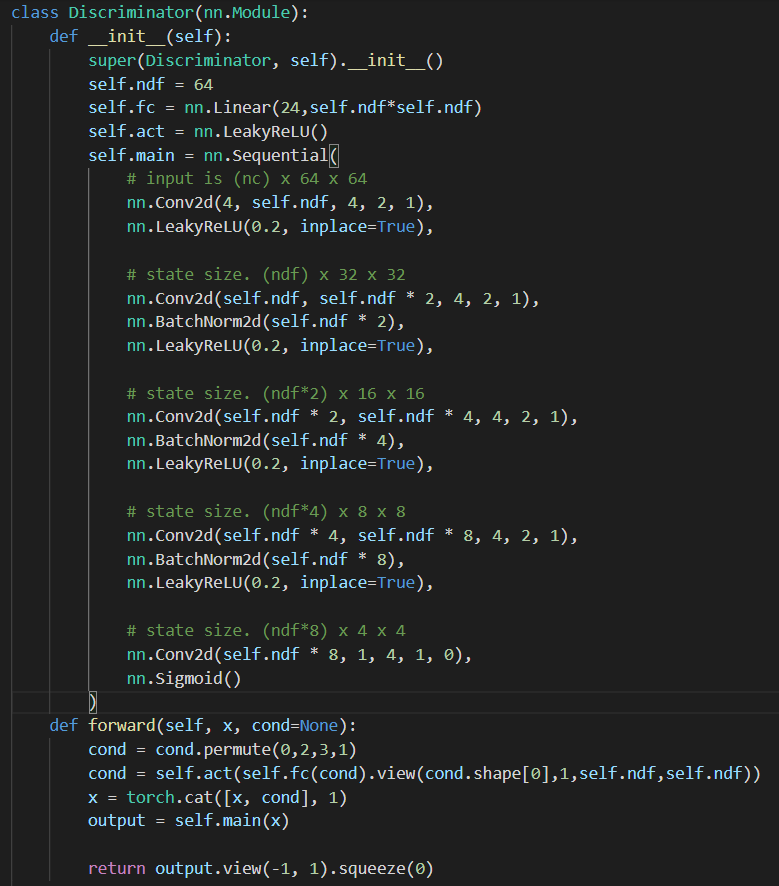
My architecture is DCGANs .

* 1. Generator and Discriminator

To implement conditional GAN , I use the fully connective layer to convert the length of the condition from 24 to 100 at the generator , and the length of latent input is 200 , so the total is 300 , and output size is 64x64 .

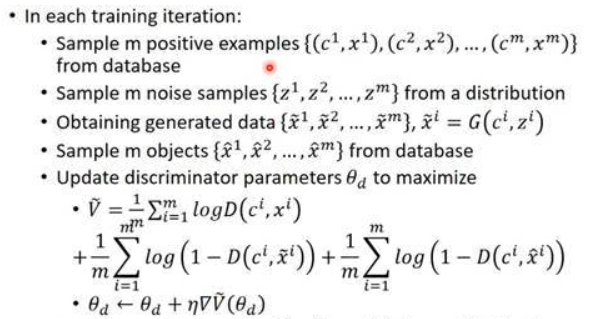


At the discriminator , I use the fully connective layer to convert the size of the condition from 24 to 1x64x64 ( one channel ) and connect it to the input image (three channels) , so the size of input is batch size x 4 x 64 x 64 , and the output is a scalar .

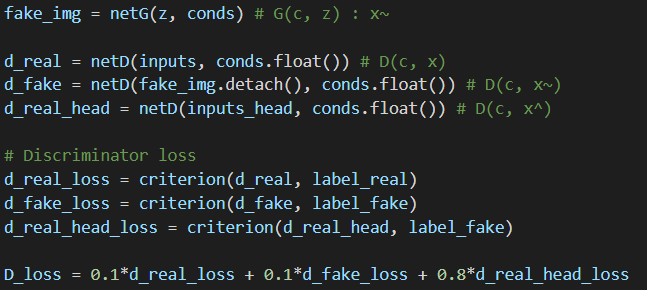


* 1. Loss Function

First , The loss function of the discriminator is designed according to the algorithm of the cGANs and I add the weight to each sub loss to improve the performance of the model .



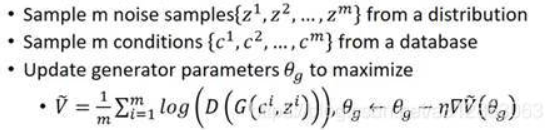
There has three sub loss .

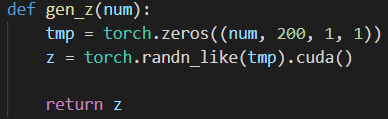


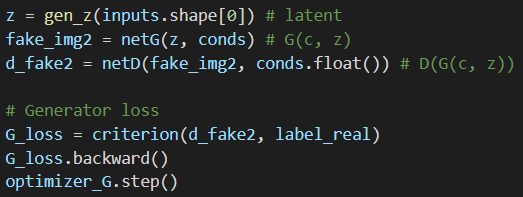




Second , The loss function of the generator is also designed according to the algorithm of the cGANs .







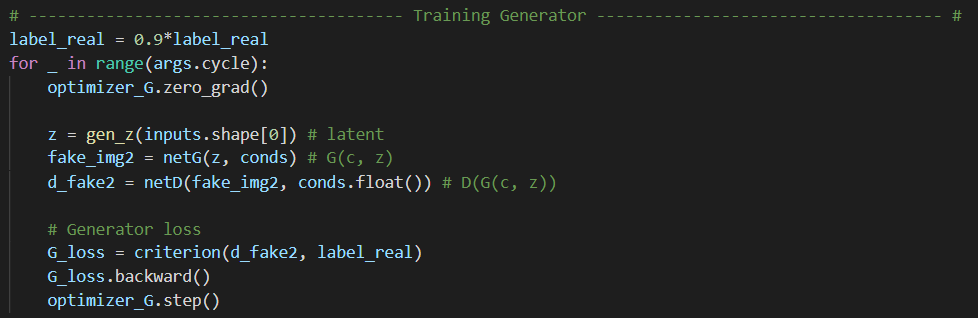
* 1. Implement

Training Function :

My learning rate is 2e-4 , batch size is 64, epochs 200 and for each iteration , generator will be trained 5 times more than the discriminator to make generator powerful enough . This is what one epoch do .

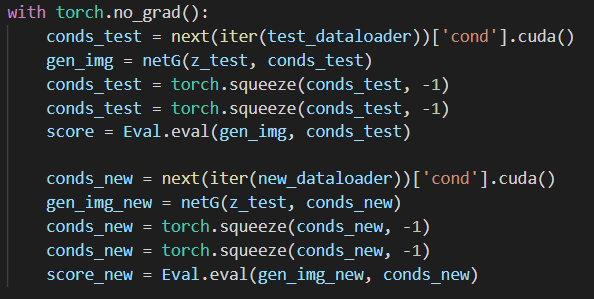






Testing Function :



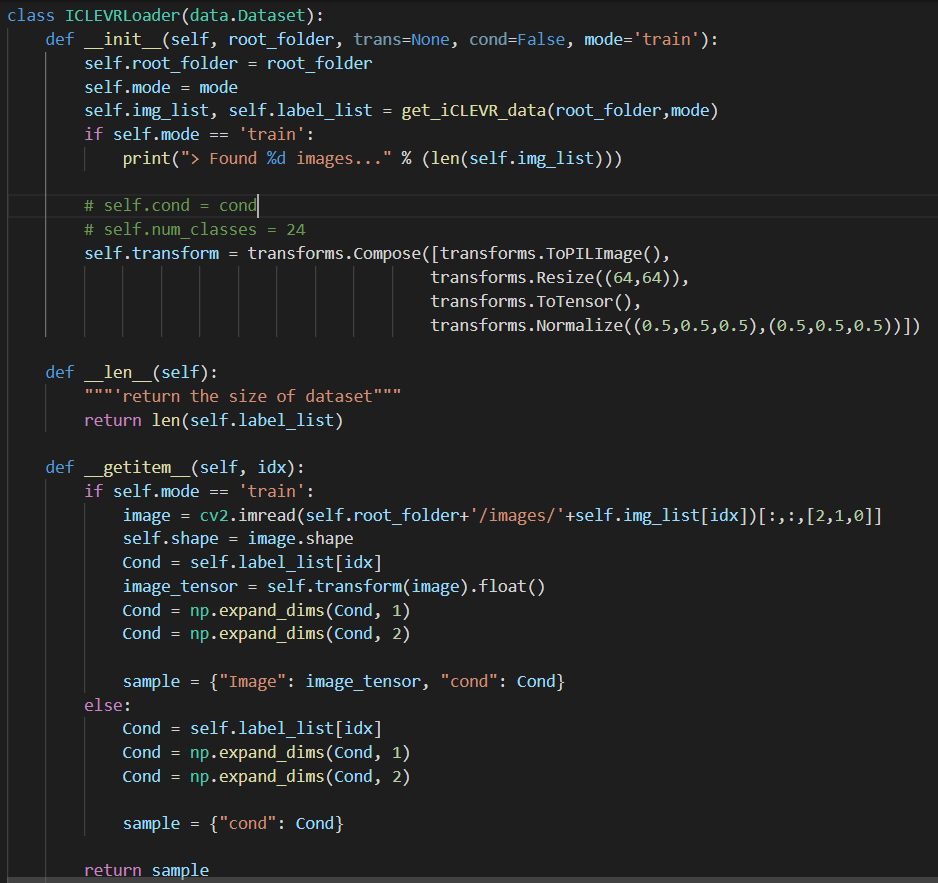


The size of the z\_test is 32x200x1x1 , and test the test.json and new.json .

Dataloader :

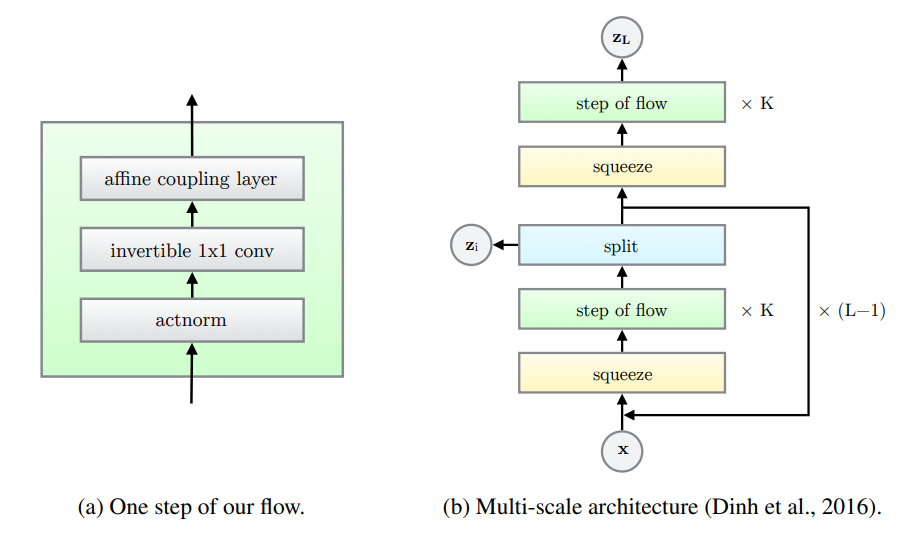
The dataloader has two mode , one is training : it will return both images and conditions , another is testing : it will only return conditions and you can select test.json or new.json .

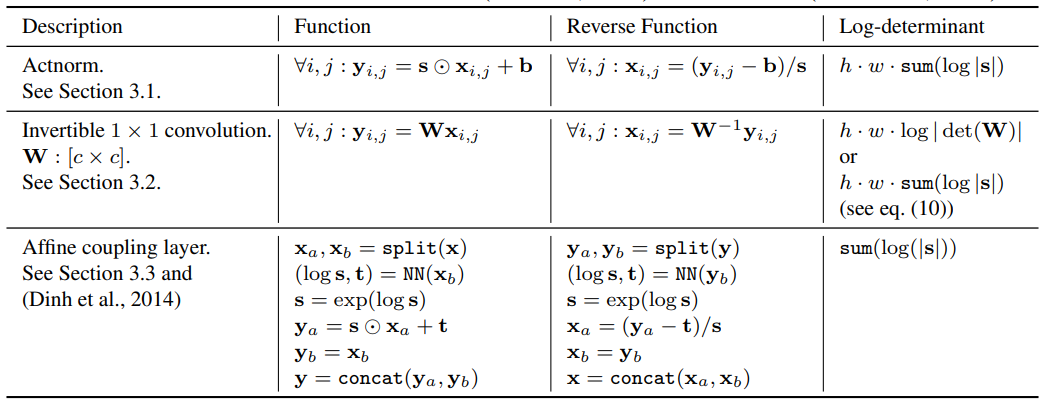




1. cNF
   1. Architecture

I choose the Glow . : <https://github.com/5yearsKim/Conditional-Normalizing-Flow>

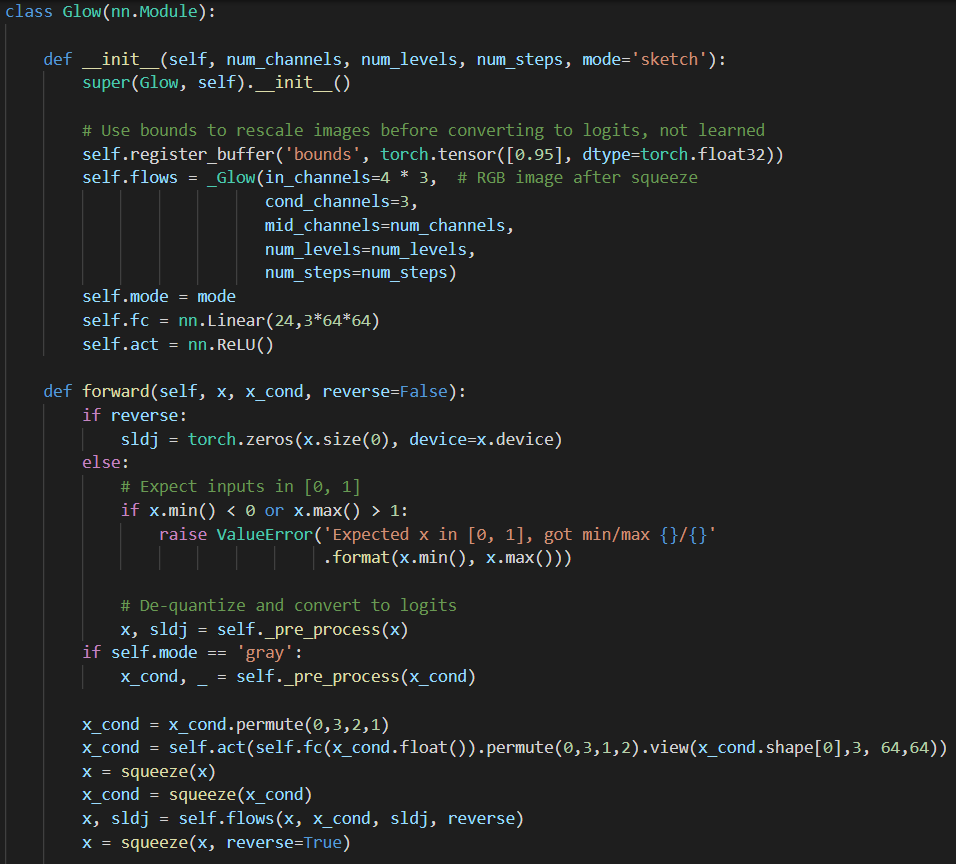




Paper : <https://arxiv.org/pdf/1807.03039.pdf>

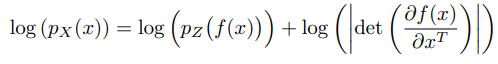
* 1. Model

The source code is a conditional NF , but its origin condition size looks is same as the images ( channels x64x64 ) , so I use the fully connective layer to change my conditions from 24 to 3x64x64 ( the num of channel doesn’t have to be 3 , it can be any number ) .

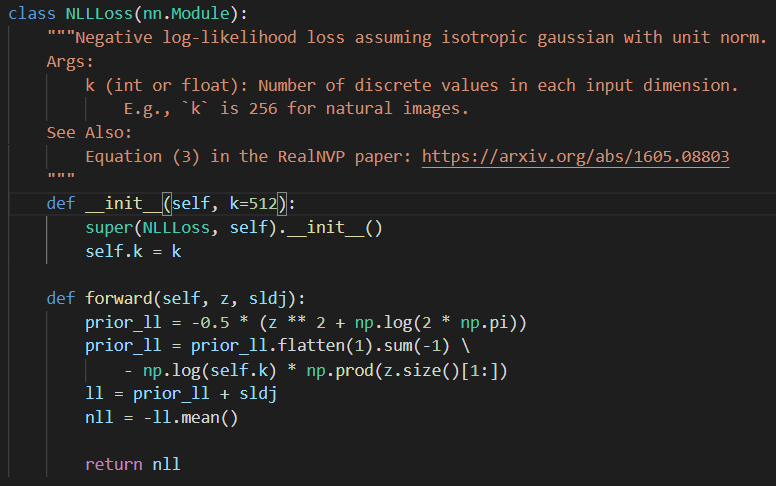


The red boxes are my implementation .

Algorithm of loss function :



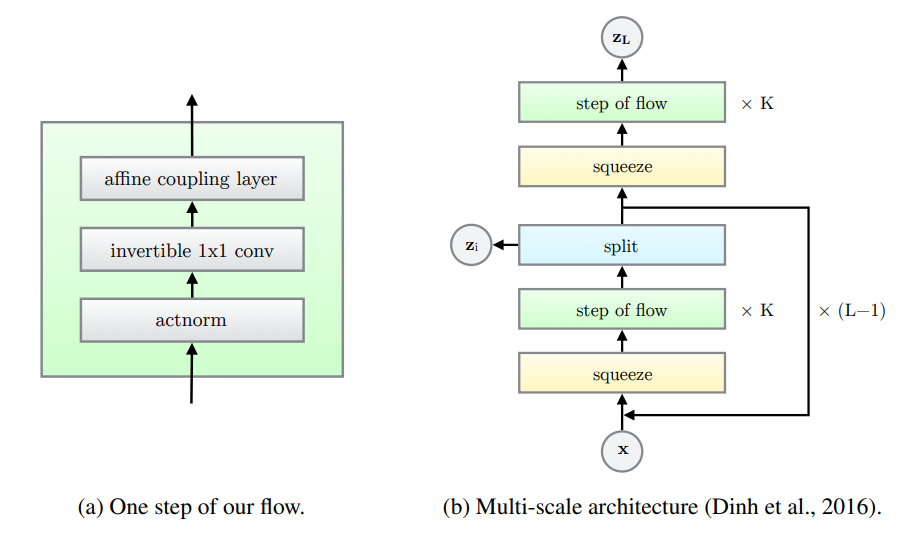




* 1. Implement

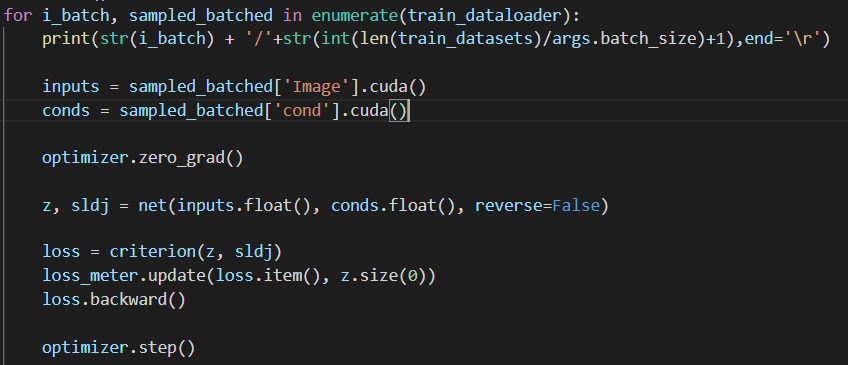
Training Function :

My learning rate is 2e-4 , batch size is 16, epochs 150 , num\_channel is 512 ( the channel in coupling , invertible conv, actnorm ) , num\_level is 4 ( L ), num\_steps is 6 ( K ) .( The source code using 1e-3, 4, 128, 3, 8 )



This is what one epoch do . ( reverse : False : data to latent , True : latent to data )

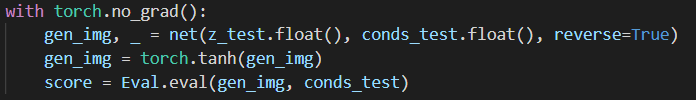




Testing Function :

The test\_dataloader can be test.json or new.json .





Dataloader :

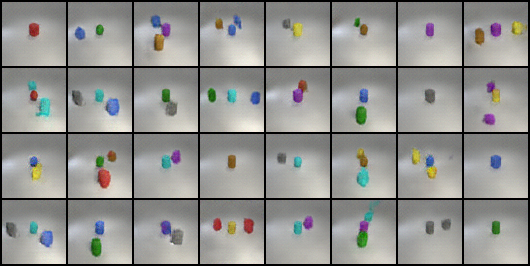
The dataloader is same as cGANs .

Result :

cGAN :

1. Test





1. New

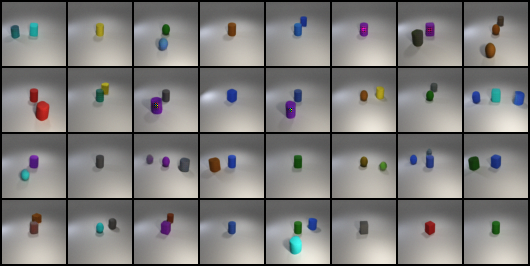




cNF :

1. Test





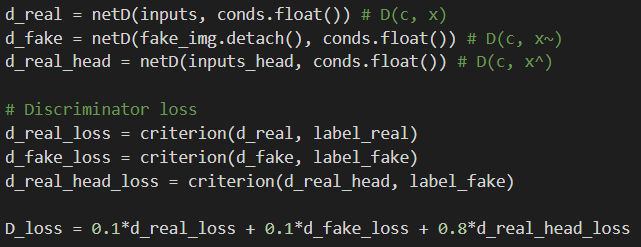
1. New





Discussion :

In the case of cGAN , It is hard to balance the abilities of generator and discriminator . If one of them is too powerful than another one , the training may fail . During this lab , I tried different learning rate , weight of sub loss in discriminator and the num of training times for generator in one iteration . I found that adjusting the weight of loss is the most efficient method to find good enough result . If using too big learning rate (like 0.001) , the loss can’t smoothly reduce , if too small , it takes too much time for training ( It took me 1.5 days to get my final result . ) . If using more training times for generator also takes too much time . After set the appropriate parameters , I only modified the weight loss and finally I use 0.1, 0.1, 0.8 for three sub loss :



For adding conditions , I just make its size close to image (1x64x64) and use the activation function .

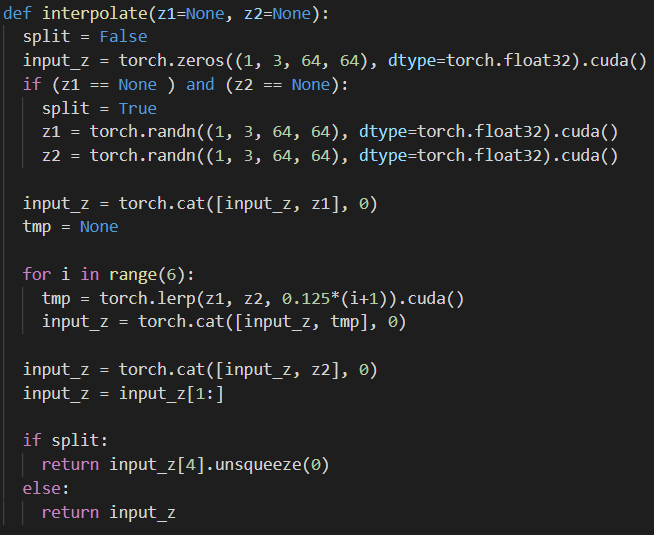
In the case of NF , because I used Glow , it has some parameters for model structure , like num\_channel , num\_level and num\_steps . Although using bigger parameters , the model will be better , but it also takes more vram and time .(I use 512/4/6 , batch size 16 , epoch 150 , it needs 6G vram and 1.5 days to train .) I found that make conditions have more channels will have more chance to get better result , so I use fully connective layer to change size from 24 to 3x64x64 . For the learning rate , I just using 2e-4 , same as cGAN case . I also found that appropriate small batch size will get better result , as my testing , 16 is the best .

The different between the two model , is the target of learning . NF is to find an invertible transform for data distribution and latent , but GAN is like teaching it to generate a data like image . In theory , NF is easier to training than GAN .

Task2

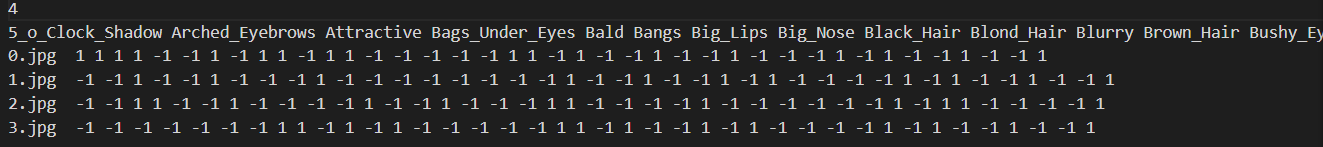
The NF model I used in this part is totally same as NF model in task1 , the only different is that origin conditions size is 40 and epochs is 50 .

For three tasks , the latent I used is first using two latent to interpolate and select the middle one as input latent , that will make higher quality result image .



**Conditional face generation :**

I choose 4 data in training datasets and change the condition label .







Randomly initial 4 latent and add the conditions .

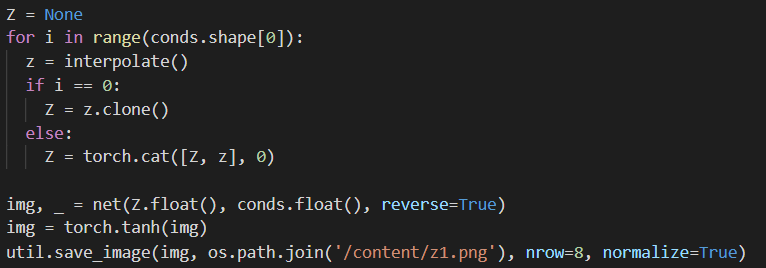


Image :

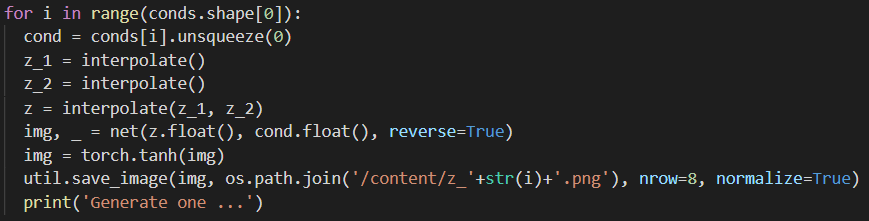


**Linear interpolation :**

I randomly generate two latent and interpolate 6 images and randomly choose the conditions in datasets .







Images :







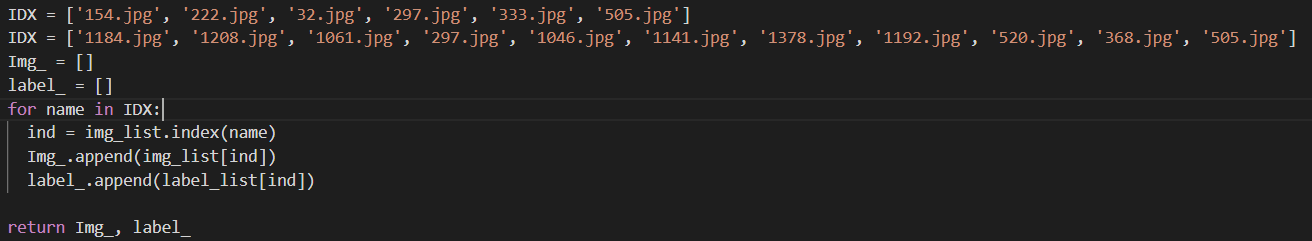


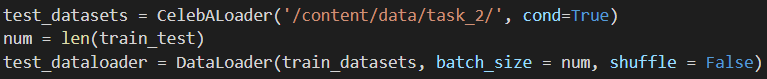


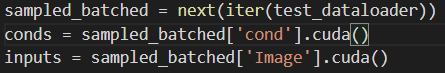
**Attribute manipulation :**

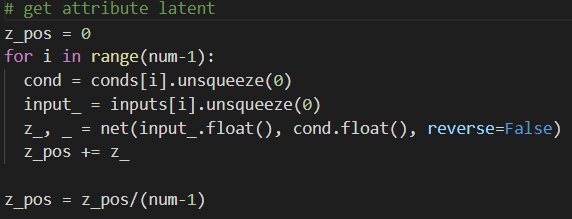
I select two attributes : smiling and gold color hair . I choose five to ten images that contain the attribute from datasets , get the average of the sum of those latent generated by model and use different scalar when add it to the target image latent . Randomly choose the conditions in datasets .

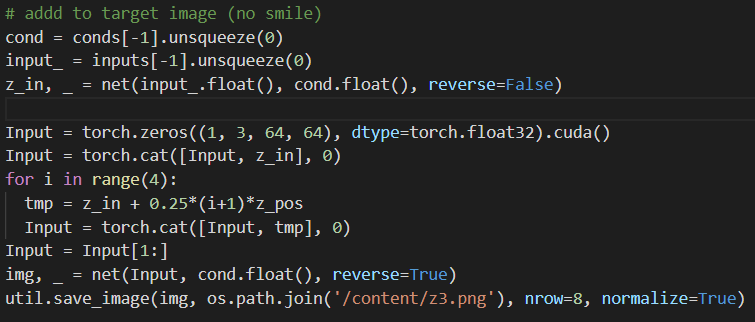
In dataloader : select attribute images . ( 505 is target image )











Images :

Smiling :



Gold color hair :

