<https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix>

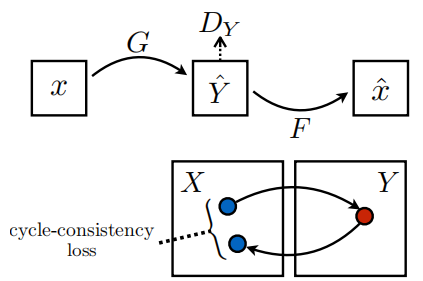
### START

Here are how all the loss works in this model.

We have two data set A and B.

|  |
| --- |
| Generators: G\_A: A -> B; G\_B: B -> A. |
| Discriminators: D\_A: G\_A(A) vs. B; D\_B: G\_B(B) vs. A. |
| Forward cycle loss: lambda\_A \* ||G\_B(G\_A(A)) - A|| (Eqn. (2) in the paper) |
| Backward cycle loss: lambda\_B \* ||G\_A(G\_B(B)) - B|| (Eqn. (2) in the paper) |
| Identity loss (optional): lambda\_identity \* (||G\_A(B) - B|| \* lambda\_B + ||G\_B(A) - A|| \* lambda\_A) |

|  |  |  |
| --- | --- | --- |
| fake\_B = G\_A(real\_A) | rec\_A = G\_B(fake\_B) | idt\_A = G\_A(real\_B) |
| fake\_A = G\_A(real\_B) | rec\_B = G\_A(fake\_A) | idt\_B = G\_B(real\_A) |



Idt\_loss:

If our generator G\_A is good enough to produce picture real\_A -> fake\_B. Then when we put real\_B into G\_A, it should return the identity because G\_A is used to transfer input into B,

and if we already have real\_B as input, G\_A should change nothing.

|  |  |  |  |
| --- | --- | --- | --- |
| Loss\_G\_A | Loss\_D\_A | Loss\_cycle\_A | Loss\_idt\_A |
| Loss\_G\_B | Loss\_D\_B | Loss\_cycle\_B | Loss\_idt\_B |

Loss\_G and Loss\_D are calculated in GAN\_Loss class. This is normal GAN loss function.

I will explain with code in GANLoss class.

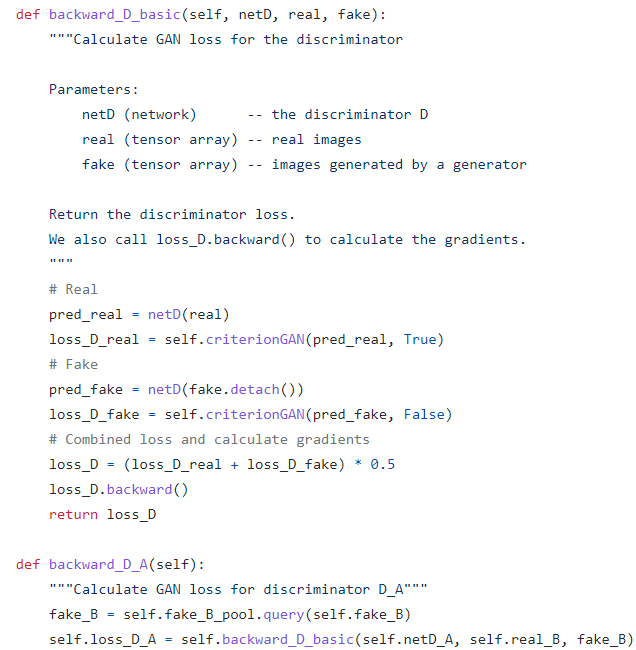
self.loss\_G\_A = self.criterionGAN(self.netD\_A(self.fake\_B), True) --In cyclegan\_model.py

Code below is from GANLoss. First, they build a ground truth map(is target is real than all elements are 1 otherwise 0). And calculate the loss with ground truth and prediction. The size of this map is the same as prediction.

target\_tensor = self.get\_target\_tensor(prediction, target\_is\_real)  
loss = self.loss(prediction, target\_tensor) --In networks.py line 264

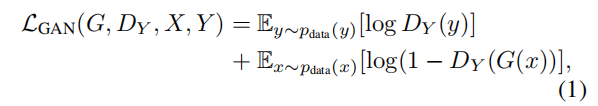
So, when calculating Loss\_G\_A, we put the fake\_B into D\_A, since we want G to be good, so we set ‘target\_is\_real’ to be true, so the BP can help us make progress on G.

Loss\_D



When it comes to D, we want D to be more smart so that it can distinguish real and fake images. So **the input is D net real and fake images**. In the code, the label of **real** image is **true**,the label of **fake** image is **false,** that’s the direction of training.

There is one thing might cause ambiguity.

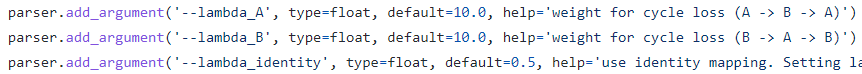


In the paper, **DY** is associated with **real Y** and **fake\_Y** (‘G(x)’).But here **D\_A** is associated with **real B** and **fake B**. But it does not matter since it’s only naming problems.

In the code notes we can also see:



Idt\_loss and cycle\_loss are just **L1 distance** between each corresponding pixel since in the ideal situation,the input should be identical with output .What’s more, idt loss have extra two weights to control.



So,

**idt\_loss\_A = L1loss\*lambda\_A\*lambda\_identity;**

**idt\_loss\_B = L1loss\*lambda\_B\*lambda\_identity;**

By default, all the loss functions are equally important, which means

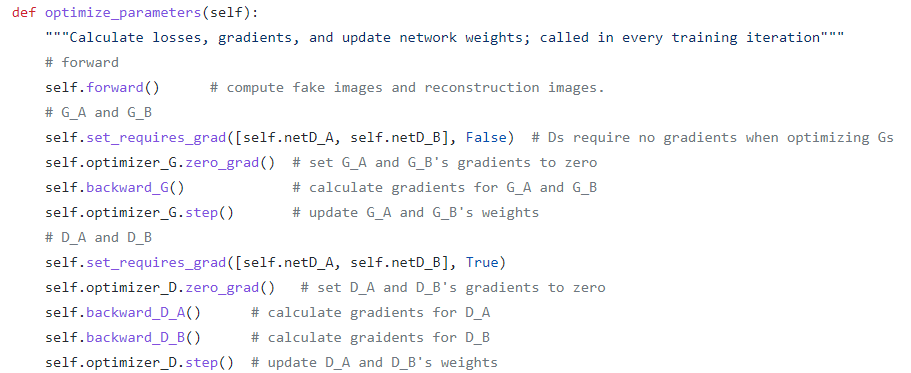
self.loss\_G = self.loss\_G\_A + self.loss\_G\_B + self.loss\_cycle\_A + self.loss\_cycle\_B + self.loss\_idt\_A + self.loss\_idt\_B.

### TRAINING

There are some tricks regarding training.

#### 1 optimize more frequently on generators.

I use unet\_128 and N\_layers discriminators. During the training, the loss of discriminator close to 0 while generator loss is still high. This function will be called in the train.py. We Can set a count flag there and modify this function to accept.



#### 2 put more weights on loss\_G\_A/B.

In the loss\_G(loss of generator), there are three components generator loss, cycle loss and identity loss. Cycle loss and identity loss are used to help us keep new image has the same boundaries. So the main body should be loss\_G\_A/B, that’s where style transfer happens. So we can put more weights on them.

self.loss\_G = self.loss\_G\_A + self.loss\_G\_B + self.loss\_cycle\_A + self.loss\_cycle\_B + self.loss\_idt\_A + self.loss\_idt\_B

This code is in line 177 in cycle\_gan\_model.py.

But once you find new image is not so clear(maybe the boundaries are not so clear), we can reduce the weights.

#### 3 checkerboard effect.

One annoying thing here is checkerboard effect. Just like its name, if you dive into every pixel, you may find pixels keep changing from dark-light-dark......

The problems happens in upsampling stage.

The generator in this github can be resnet and U-net. In the resnet, they use transposedconv with

**Kernel\_size =3** and **stripe = 2.** In the transposed convolution, once kernel size can not be divided by stripe , the feature map will suffer pixel overlap. So, it’s better to use U-net(kernel\_size = 4 and stripe = 2) or we use **interpolation+conv** instead of transposedconv.

#### 4 add more loss function

When it comes to some specific tasks, we can add more loss function e.g. landmark , segmentation mask,semantic,feature.

Here is a reference about CYCADA.

<https://arxiv.org/pdf/1711.03213.pdf>

This is the result (photo->painting) in epoch 15, that’s already quite good isn’t it?

