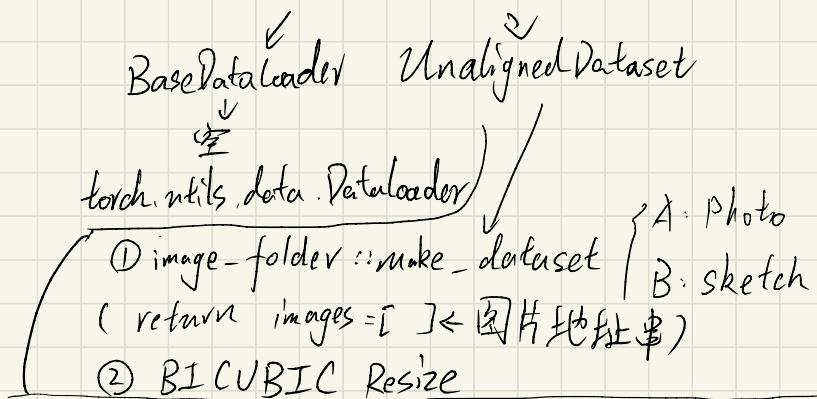


Photo Sketch MAN

1 data-loader \rightarrow Create Dataloader \rightarrow Custom Dataset Data Loader



2 dataset = dataloader.load_data() ($\text{torch.utils.data.DataLoader}$)

3 val_dataset = - - -

4 $G_A = \text{nets.define_G}$ resnet 9 blocks.

5 $G_B =$ - - -

6 $D_A1 \ D_A2 \ D_A3$, N Layer Discriminator

7 $D_B1 \ D_B2 \ D_B3$ - - -

8 resume - - -

9 optimizer_G Adam $\text{itertools.chain}(G_A.parameters(), G_B.parameters())$

opt_D_A1 D_A2 D_A3 DB1 DB2 DB3.

10 fake - A - pool = ImagePool
fake A64 Pool
fake A 128 Pool.
fake B - Pool
B64
B 128 Pool

↑
30 张图片 4维 图片. 长宽. channel
小于30 : 直接加;
大于30 50% 概率直接加
50%, 概率抽出 Pool 中 - 张图

networks ..

11 criterion GAN : GANLoss { MSE Loss
BCE Loss

criter_cycle : L1 loss

criter_Idt : L1 Loss

criter_Res : L1 Loss

- Patch : patch loss.

12 transform

13 update_d (net_D, real, fake):

pred_fake = net_D (fake.detach())

loss_D-fake = criter_GAN (pred_fake, False)

↳ 创造一个全1/全0的 tensor

pred_real = net_D (real.detach())

loss_D-real = criter_GAN (pred_real, True)

loss_D = $\frac{1}{2} \cdot (loss_D-fake + loss_D-real)$

loss_D.backward()

return 3 loss

14. train()

建立所有 loss 的 record = AugMeter() 用作之後打印显示
for epoch in range(1, opt.niter + opt.niter_decay + 1):
 for i, data in enumerate(dataset):

$$\text{input-A} = \underset{\text{photo}}{\text{data['A'].float()}} \quad \text{input-B} = \underset{\text{sketch}}{\text{data['B'].float()}}$$

$$\text{input-A}_\text{img} = \text{tensor2im}(\text{inputA}_\text{img})$$

256

$$\text{input-B}_\text{img} = \text{tensor2im}(\text{inputB}_\text{img})$$

BICUBIC, resqueeze(0)

$$\Rightarrow \text{input-A} \sim 28 (\text{num}, \text{H}, \text{W}, \text{C}) \quad \text{input-B} \sim 28$$

input-A 64

input-B 64

$$\Rightarrow \text{real-A} \sim 128 \cdot 64 \quad \text{real-B} \sim 128 \cdot 64.$$

PHOTO \rightarrow SKETCH \rightarrow PHOTO

fake-B 64, fake-B 128, fake-B = G_A(real-A)

rec-A 64, rec-A 128, rec-A = G_B(fake-B)

SKETCH \rightarrow PHOTO \rightarrow SKETCH

↑↑↑

$$\begin{array}{lll} DA1 & DA2 & DA3 - \text{zero_grad}() \\ DB1 & DB2 & DB3 - \text{zero_grad}() \end{array}$$

fakeA = fake-A.pool.query(torch.cat([realB, fakeA], 1).data)

realA = torch.cat([realB, realA], 1)

loss-DA256-real, loss-DA256-fake, loss-DA256 = update(DA1, realA, fakeA)

optimizer-D_A1.step()

128 ...

64 ...

B1 ...

B2 ...

B3 ...

update
D

G(A).no_grad() G(B).no_grad()

original GAN
Loss

G(A) should fool the discriminator.

pred_fakeB = DA₁(fakeB)

loss_GA_GAN = criterion_GAN(pred_fakeB, True)

pred_fakeB128 = DA₂(fakeB128) - - -

G(B1 G(B2 G(B3

{ G(A) = B

syn - A256 = criterionRec(fakeA, realA)

A 128

A 64

B 256

B 128

B 64

cyc - A256 = criterionRec(RecA, realA)

1

1

1

1

then min lambda.

loss_G = loss_... + cyc_... + syn_...

loss_G.backward()

optimizer_G.step()

record.update() - - -

display

save