Photo Sketch MAN

1 data_loader → CreateDataloader → CustomDatasetDataLoader

BaseDataLoader ↙ UnalignedDataset ↘

空

torch.utils.data.DataLoader

① image_folder::make_dataset { A: photo B: sketch

(return images=[ ] ← 图片地址集)

② BICUBIC Resize

2 dataset = dataloader.load_data() (torch.utils.data.DataLoader)

3 val_dataset = - - - -

4 GA = nets.define_G resnet 9 blocks

5 GB = - - - -

6 DA1 DA2 DA3 . NLayerDiscriminator

7 DB1 DB2 DB3 - - - .

8 resume - - -

9 optimizer_G Adam itertools.chain(GA.parameters(), GB.parameters())

opt_D_A1 DA2 DA3 DB1 DB2 DB3.

10 fake_A_pool = ImagePool fake A64 Pool fake A128 Pool.
(256) B64 B128 Pool
fake_B_Pool
↑
50张图片 4维 图片、长宽、channel
小于50：直接加；
大于50 50%的率直接加
50%的率换出Pool中一张图

networks::

11 criterionGAN : GANLoss { MSELoss / BCELoss
criter_cyde : L1loss
criter_Idt : L1Loss
criter_Res : L1Loss
- Patch : patch loss.

12 transform

13 update_d(net_D, real, fake):
pred_fake = netD(fake.detach())
loss_D_fake = criter_GAN(pred_fake, False)
↳ 创造一个全1/全0的tensor，
pred_real = netD(real.detach())
loss_D_real = criter_GAN(pred_real, True)
loss_D = $\frac{1}{2}$·(loss_D_fake + loss_D_real)
loss_D.backward()
return 3 Loss

14. train()

建立所有 loss 的 record = AvgMeter() 用作之后打印显示

for epoch in range(1, opt.niter + opt.niter_decay + 1):

for i, data in enumerate(dataset):

input_A = data['A'].float()　input_B = data['B'].float()

photo　sketch

input_A_img = tensor2im(input_A_img)

input_B_img = tensor2im(input_B_img)

256

BICUBIC, unsqueeze(0)

⇒ input_A128 (num, H, W, C)　input_B128

input_A64　input_B64

⇒ real_A　real_A128　A64　real_B　real_B128　B64

PHOTO → SKETCH → PHOTO

fake_B64, fake_B128, fake_B = GA(real_A)

rec_A64, rec_A128, rec_A = GB(fake_B)

SKETCH → PHOTO → SKETCH

同上.

update D

DA1 DA2 DA3 . zero_grad()

DB1 DB2 DB3 . zero_grad()

fake_A = fake_A_pool.query(torch.cat(real_B, fake_A), 1).data)

real_A = torch.cat((real_B, real_A), 1)

loss_DA256_real, loss_DA_256_fake, loss_DA256 = update(DA1, real_A, fake_A)

optimizer_D_A1.step()

128 ...

64 ...

B1 ...

B2 ...

B3 ...

GA.nograd() GB.no_grad()

original GAN Loss

G(A) should fool the discriminator.

pred_fakeB = DA(fakeB)

loss_GA_GAN = criterion GAN (pred_fakeB, True)

pred_fakeB128 = DA2(fakeB(128)) ...

GB1 GB2 GB3

G(A)=B

syn_A256 = criterionRec(fakeA, realA)

A 128

A 64

B 256

B 128

B 64

GB(GA(A))≈A 两次转换结果应相似

cyc_A256 = creterionRec(RecA, realA)

|

|

|

|

eta mu Lumbda.

loss_G = loss_... + cyc... + syn...

loss_G.backward()

optimizer_G.step()

record.update() ....

display

save