6.869 Pset 7
Michael Everett
Nov 19, 2018

Generator

```
def __init__(self, conv_dim=64, init_zero_weights=False):
    super(CycleGenerator, self).__init__()

    ############################################
    ##   FILL THIS IN: CREATE ARCHITECTURE   ##
    ############################################

    # 1. Define the encoder part of the generator (that extracts features from the inp
    self.conv1 = conv(in_channels=3, out_channels=conv_dim, kernel_size=4)
    self.conv2 = conv(in_channels=conv_dim, out_channels=conv_dim*2, kernel_size=4)

    # 2. Define the transformation part of the generator
    self.resnet_block = ResnetBlock(conv_dim=conv_dim*2)

    # 3. Define the decoder part of the generator (that builds up the output image frc
    self.deconv1 = deconv(in_channels=conv_dim*2, out_channels=conv_dim, kernel_size=4
    self.deconv2 = deconv(in_channels=conv_dim, out_channels=3, kernel_size=4, batch_r
```

CycleGAN Training Loop

```
# Train with real images
d_optimizer.zero_grad()

# 1. Compute the discriminator losses on real images
D_X_loss = (1.0/images_X.size()[0])*sum((D_X(images_X) - 1)**2)
D_Y_loss = (1.0/images_Y.size()[0])*sum((D_Y(images_Y) - 1)**2)

d_real_loss = D_X_loss + D_Y_loss
d_real_loss.backward()
d_optimizer.step()

# Train with fake images
d_optimizer.zero_grad()

# 2. Generate fake images that look like domain X based on real imag
fake_X = G_YtoX(images_Y)

# 3. Compute the loss for D_X
D_X_loss = (1.0/images_Y.size()[0])*sum(D_X(fake_X)**2)

# 4. Generate fake images that look like domain Y based on real imag
fake_Y = G_XtoY(images_X)


# 5. Compute the loss for D_Y
D_Y_loss = (1.0/images_X.size()[0])*sum(D_Y(fake_Y)**2)

d_fake_loss = D_X_loss + D_Y_loss
d_fake_loss.backward()
d_optimizer.step()
```

```
########################################
##    FILL THIS IN: Y--X-->Y CYCLE     ##
########################################
g_optimizer.zero_grad()

# 1. Generate fake images that look like domain X based on real images in domain
fake_X = G_YtoX(images_Y)

# 2. Compute the generator loss based on domain X
g_loss = (1.0/images_Y.size()[0])*sum((D_X(fake_X)-1)**2)

if opts.use_cycle_consistency_loss:
    reconstructed_Y = G_XtoY(fake_X)
    # 3. Compute the cycle consistency loss (the reconstruction loss)
    cycle_consistency_loss = (1.0/images_Y.size()[0])*sum(sum(sum(sum((images_Y
    g_loss += cycle_consistency_loss

g_loss.backward()
g_optimizer.step()



########################################
##    FILL THIS IN: X--Y-->X CYCLE     ##
########################################

g_optimizer.zero_grad()

# 1. Generate fake images that look like domain Y based on real images in domain
fake_Y = G_XtoY(images_X)

# 2. Compute the generator loss based on domain Y
g_loss = (1.0/images_X.size()[0])*sum((D_Y(fake_Y)-1)**2)

if opts.use_cycle_consistency_loss:
    reconstructed_X = G_YtoX(fake_Y)
    # 3. Compute the cycle consistency loss (the reconstruction loss)
    cycle_consistency_loss = (1.0/images_X.size()[0])*sum(sum(sum(sum((images_X
    g_loss += cycle_consistency_loss

g_loss.backward()
g_optimizer.step()
```

CycleGan Experiments

Problem	sample-000400-x-y.png	sample-000400-y-x.png
1		
2		
	sample-000100-x-y.png	sample-000100-y-x.png
3		
4		

5.

The cycle loss term seems to help. The generated emojis look brighter in 2 than 1 -- they look very dull in 1. However, the strange grid-like pattern seems amplified with the cycle loss, which is undesirable.

The similarity is because the regular loss terms cause an averaging-type effect, which appears to roughly get the emoji shape correctly, whereas the cycle loss term causes the generators to produce something that actually seems like the original emoji.

The pre-trained model helps in particular to get the background color correct.