T-SUM

2021271314 김건희

01 코드 설명

```
OUTPUT_CHANNELS = 3

generator_g = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')
generator_f = pix2pix.unet_generator(OUTPUT_CHANNELS, norm_type='instancenorm')

discriminator_x = pix2pix.discriminator(norm_type='instancenorm', target=False)
discriminator_y = pix2pix.discriminator(norm_type='instancenorm', target=False)
```

분별망, 생성망 네트워크

- 분별망은 이미지를 입력으로 받아들여 입력 이미지가 진짜인지 가짜인지를 판별하는 역할
- 생성망은 입력 이미지를 받아들여 실제 이미지와 유사한 가짜 이미지를 생성하는 역할

01 코드 설명

```python
LAMBDA = 10

loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real, generated):
  real_loss = loss_obj(tf.ones_like(real), real)

  generated_loss = loss_obj(tf.zeros_like(generated), generated)

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss * 0.5

def generator_loss(generated):
  return loss_obj(tf.ones_like(generated), generated)

def calc_cycle_loss(real_image, cycled_image):
  loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

  return LAMBDA * loss1

def identity_loss(real_image, same_image):
  loss = tf.reduce_mean(tf.abs(real_image - same_image))
  return LAMBDA * 0.5 * loss
```

손실함수 소개

- discriminator_loss(real, generated)

: 분별망의 손실 함수, 이진 교차 엔트로피 손실 함수를 사용하여 실제 이미지와 생성된 이미지의 분별망 출력 사이의 손실을 계산 한다.

- generator_loss(generated)

: 생성망의 손실 함수, 샘플 데이터((학습 데이터)와 비교하여 손실을 계산한다.

- calc_cycle_loss(real_image, cycled_image)

: 사이클 일관성 손실을 계산, 이 함수는 실제 이미지와 사이클로 복원된 이미지 사이의 차이를 계산하고, 이를 LAMBDA 값과 곱하여 반환 한다.

- identity_loss(real_image, same_image)

: 정체성 손실을 계산, 실제 이미지와 생성망의 입력으로 주어진 이미지 사이의 차이를 계산하고, 이를 LAMBDA 값과 0.5를 곱하여 반환한다.

01 코드 설명

```python
LAMBDA = 10

loss_obj = tf.keras.losses.BinaryCrossentropy(from_logits=True)

def discriminator_loss(real, generated):
  real_loss = loss_obj(tf.ones_like(real), real)

  generated_loss = loss_obj(tf.zeros_like(generated), generated)

  total_disc_loss = real_loss + generated_loss

  return total_disc_loss * 0.5

def generator_loss(generated):
  return loss_obj(tf.ones_like(generated), generated)

def calc_cycle_loss(real_image, cycled_image):
  loss1 = tf.reduce_mean(tf.abs(real_image - cycled_image))

  return LAMBDA * loss1

def identity_loss(real_image, same_image):
  loss = tf.reduce_mean(tf.abs(real_image - same_image))
  return LAMBDA * 0.5 * loss
```

→ Cycle과 Identity 네트워크가 필요한 이유는 변환 과정에서 원본 이미지의 특징을 보존하고 일관성을 유지하기 위함이다.

→ 이를 통해 Generator가 불필요한 변화를 최소화하고 입력 이미지의 특징을 보존하는 결과를 얻을 수 있다.

01 코드 설명

```python
@tf.function
def train_step(real_x, real_y):
  # persistent is set to True because the tape is used more than
  # once to calculate the gradients.
  with tf.GradientTape(persistent=True) as tape:
    # Generator G translates X -> Y
    # Generator F translates Y -> X.

    fake_y = generator_g(real_x, training=True)
    cycled_x = generator_f(fake_y, training=True)

    fake_x = generator_f(real_y, training=True)
    cycled_y = generator_g(fake_x, training=True)
```

→ 이렇게 예측된 결과는 후속 단계에서 사용한다.

Pix2Pix 모델의 학습 단계를 구현한 train_step 함수

- 주어진 코드는 TensorFlow를 사용한 훈련 루프이다
- 네 가지 기본 단계로 나눌 수 있다.

1. 가짜 이미지를 생성한다

- 생성자 모델인 generator_g와 generator_f를 사용하여 입력 이미지에 대한 가짜 이미지를 생성한다.

01 코드 설명

```
    # same_x and same_y are used for identity loss.
    same_x = generator_f(real_x, training=True)
    same_y = generator_g(real_y, training=True)

    disc_real_x = discriminator_x(real_x, training=True)
    disc_real_y = discriminator_y(real_y, training=True)

    disc_fake_x = discriminator_x(fake_x, training=True)
    disc_fake_y = discriminator_y(fake_y, training=True)

    # calculate the loss
    gen_g_loss = generator_loss(disc_fake_y)
    gen_f_loss = generator_loss(disc_fake_x)

    total_cycle_loss = calc_cycle_loss(real_x, cycled_x) +
calc_cycle_loss(real_y, cycled_y)
```

→ 생성자의 손실인 gen_g_loss와 gen_f_loss를 계산한다.

2. 손실을 계산

- 생성된 가짜 이미지와 실제 이미지 사이의 손실을 계산하는 것이다.
- disc_real_x와 disc_real_y는 실제 이미지에 대한 판별자의 예측
- disc_fake_x와 disc_fake_y는 생성된 가짜 이미지에 대한 판별자의 예측한다.

01 코드 설명

```
    # Total generator loss = adversarial loss + cycle loss
    total_gen_g_loss = gen_g_loss + total_cycle_loss + identity_loss(real_y,
same_y)
    total_gen_f_loss = gen_f_loss + total_cycle_loss + identity_loss(real_x,
same_x)

    disc_x_loss = discriminator_loss(disc_real_x, disc_fake_x)
    disc_y_loss = discriminator_loss(disc_real_y, disc_fake_y)

  # Calculate the gradients for generator and discriminator
  generator_g_gradients = tape.gradient(total_gen_g_loss,
                                        generator_g.trainable_variables)
  generator_f_gradients = tape.gradient(total_gen_f_loss,
                                        generator_f.trainable_variables)

  discriminator_x_gradients = tape.gradient(disc_x_loss,
                                            discriminator_x.trainable_variables)
  discriminator_y_gradients = tape.gradient(disc_y_loss,
                                            discriminator_y.trainable_variables)
```

3. 역전파를 사용하여 그래디언트를 계산

- tf.GradientTape를 사용하여 생성자와 판별자의 모델 파라미터들에 대한 그래디언트를 계산한다,
- discriminator_x_gradients와 discriminator_y_gradients는 판별자 모델인 discriminator_x와 discriminator_y에 대한 손실의 그래디언트이다.

01 코드 설명

```
 # Apply the gradients to the optimizer
 generator_g_optimizer.apply_gradients(zip(generator_g_gradients,
                                           generator_g.trainable_variables))

 generator_f_optimizer.apply_gradients(zip(generator_f_gradients,
                                           generator_f.trainable_variables))

 discriminator_x_optimizer.apply_gradients(zip(discriminator_x_gradients,

discriminator_x.trainable_variables))

 discriminator_y_optimizer.apply_gradients(zip(discriminator_y_gradients,

discriminator_y.trainable_variables))

 return (total_gen_g_loss, total_gen_f_loss, disc_x_loss, disc_y_loss)
```

→ 생성자와 판별자는 각각 목표에 맞는 손실을 최소화하고 최대화하여 모델을 발전시킨다.

4. 그래디언트를 옵티마이저에 적용

- generator_g_optimizer, generator_f_optimizer, discriminator_x_optimizer, discriminator_y_optimizer와 그에 대응하는 그래디언트를 사용하여 각 모델의 파라미터를 업데이트한다.

01 코드 설명

```
import matplotlib.pyplot as plt

# Reset the lists
gen_g_losses = []
gen_f_losses = []
disc_x_losses = []
disc_y_losses = []

EPOCHS = 10

for epoch in range(EPOCHS):
    start = time.time()

    n = 0
    for image_x, image_y in tf.data.Dataset.zip((train_horses, train_zebras)):
        total_gen_g_loss, total_gen_f_loss, disc_x_loss, disc_y_loss =
train_step(image_x, image_y)
        if n % 10 == 0:
            print('.', end='')
        n += 1
```

이미지 변환 모델을 학습하는 과정

- 주어진 데이터셋을 사용하여 생성자와 판별자의 손실을 계산하고, 각 에폭마다 손실 값을 저장하고 출력한다.

01 코드 설명

```
    # Calculate and store losses after each epoch
    gen_g_losses.append(total_gen_g_loss.numpy())
    gen_f_losses.append(total_gen_f_loss.numpy())
    disc_x_losses.append(disc_x_loss.numpy())
    disc_y_losses.append(disc_y_loss.numpy())

    clear_output(wait=True)
    # Using a consistent image (sample_horse) so that the progress of the model
    # is clearly visible.
    generate_images(generator_g, sample_horse)

    if (epoch + 1) % 5 == 0:
        ckpt_save_path = ckpt_manager.save()
        print ('Saving checkpoint for epoch {} at {}'.format(epoch+1,
ckpt_save_path))

    print ('Time taken for epoch {} is {} sec\n'.format(epoch + 1, time.time()-
start))

# Plot the losses
plot_losses(gen_g_losses, gen_f_losses, disc_x_losses, disc_y_losses)
```

- ckpt_save_path = ckpt_manager.save()

: 일정 주기마다 체크포인트를 저장한다.

→ 이미지 변환 모델을 학습하는 과정을 단계별로 보여주고, 학습 진행 상황과 결과를 확인할 수 있도록 도와준다.

END OF DOCUMENT

THANK YOU