УДК 519.68:007.5

Лабораторная работа 5. Генеративно состязательные сети

Цели работы: реализовать генеративно состязательную модель для создания реалистичных изображений на основе специально размеченных эталонов.

Ключевые слова: модели U-Net, генерация изображений, обучение на основе состязания генератора и дискриминатора.

§ 1. Построение моделей для генератора и дискриминатора

1.1. Подготовка к работе. Введем следующие команды в рабочей тетради jupyter и проверим, что они исполнились без ошибок:

```
import tensorflow as tf
import os
import pathlib
import time
import datetime
from matplotlib import pyplot as plt
from IPython import display
```

Как вариант, можно вместо локально установленного Python'а использовать облачный сервис Google Colab с его реализацией тетради jupyter. Загрузим набор эталонных изображений из датасета фасадов зданий вместе с эталонными масками для их разметки.

```
dataset_name = "facades"
_URL = f'http://efrosgans.eecs.berkeley.edu/pix2pix/datasets/facades.tar.gz'
path_to_zip = tf.keras.utils.get_file(
    fname=f"{dataset_name}.tar.gz",
    origin=_URL,
    extract=True)
path_to_zip  = pathlib.Path(path_to_zip)
PATH = path_to_zip.parent/dataset_name
```

Визуализируем как выглядит отдельное изображение из загруженного датасета

```
sample_image = tf.io.read_file(str(PATH / 'train/1.jpg'))
sample_image = tf.io.decode_jpeg(sample_image)
plt.figure()
plt.imshow(sample_image)
```

Подготовим функцию для загрузки изображений и разделения их на две части: эталонное изображение и размеченную область.

```
def load(image_file):
  image = tf.io.read_file(image_file)
  image = tf.io.decode_jpeg(image)
  w = tf.shape(image)[1]
```

```
  w = w // 2
  input_image = image[:, w:, :]
  real_image = image[:, :w, :]
  input_image = tf.cast(input_image, tf.float32)
  real_image = tf.cast(real_image, tf.float32)
  return input_image, real_image
```

Проверим работу функции, распечатав несколько разделенных изображений.

```
inp, re = load(str(PATH / 'train/100.jpg'))
plt.figure()
plt.imshow(inp / 255.0)
plt.figure()
plt.imshow(re / 255.0)
```

Зададим размеры изображений, а также число обучающих примеров в одном батче и глубину буфера для их псевдослучайной перестановки.

```
BUFFER_SIZE = 400
BATCH_SIZE = 1
IMG_WIDTH = 256
IMG_HEIGHT = 256
```

Предусмотрим три функции для предобработки изображений: изменения масштаба, случайного обрезания и нормализации.

```
def resize(input_image, real_image, height, width):
  input_image = tf.image.resize(input_image, [height, width],
                        method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
  real_image = tf.image.resize(real_image, [height, width],
                        method=tf.image.ResizeMethod.NEAREST_NEIGHBOR)
  return input_image, real_image
def random_crop(input_image, real_image):
  stacked_image = tf.stack([input_image, real_image], axis=0)
  cropped_image = tf.image.random_crop(
      stacked_image, size=[2, IMG_HEIGHT, IMG_WIDTH, 3])
  return cropped_image[0], cropped_image[1]
def normalize(input_image, real_image):
  input_image = (input_image / 127.5) - 1
  real_image = (real_image / 127.5) - 1
  return input_image, real_image
```

Объединим эти функции в единую для итоговой предобработки изображений в соответсвии с изложенной в публикации https://arxiv.org/abs/1611.07004.

```
@tf.function()
def random_jitter(input_image, real_image):
  input_image, real_image = resize(input_image, real_image, 286, 286)
  input_image, real_image = random_crop(input_image, real_image)
  if tf.random.uniform(()) > 0.5:
    input_image = tf.image.flip_left_right(input_image)
    real_image = tf.image.flip_left_right(real_image)
  return input_image, real_image
```

Для обучающей выборки применим наши функции предобработки и нормализации, а для тестовой только изменения масштаба и нормализации.

```
def load_image_train(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = random_jitter(input_image, real_image)
  input_image, real_image = normalize(input_image, real_image)
  return input_image, real_image
def load_image_test(image_file):
  input_image, real_image = load(image_file)
  input_image, real_image = resize(input_image, real_image,
                                   IMG_HEIGHT, IMG_WIDTH)
  input_image, real_image = normalize(input_image, real_image)
  return input_image, real_image
```

Загрузим датасеты для обучения и валидации с помощью наших функций.

```
train_dataset = tf.data.Dataset.list_files(str(PATH / 'train/*.jpg'))
train_dataset = train_dataset.map(load_image_train,
                                  num_parallel_calls=tf.data.AUTOTUNE)
train_dataset = train_dataset.shuffle(BUFFER_SIZE)
train_dataset = train_dataset.batch(BATCH_SIZE)

test_dataset = tf.data.Dataset.list_files(str(PATH / 'val/*.jpg'))
test_dataset = test_dataset.map(load_image_test)
test_dataset = test_dataset.batch(BATCH_SIZE)
```

В соответствии со спецификацией pix2pix модели её энкодерная часть строится на основе сверточных слоев и опциональных слоев пакетной нормализации (batch norm), а декодерная - на основе слоев инверсной свертки и дропаута.

```
def downsample(filters, size, apply_batchnorm=True):
  initializer = tf.random_normal_initializer(0., 0.02)
  result = tf.keras.Sequential()
  result.add(
      tf.keras.layers.Conv2D(filters, size, strides=2, padding='same',
                             kernel_initializer=initializer,
                             use_bias=False))
  if apply_batchnorm:
    result.add(tf.keras.layers.BatchNormalization())
  result.add(tf.keras.layers.LeakyReLU())
  return result
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)
  result = tf.keras.Sequential()
  result.add(
    tf.keras.layers.Conv2DTranspose(filters, size, strides=2,
                                    padding='same',
                                    kernel_initializer=initializer,
                                    use_bias=False))
  result.add(tf.keras.layers.BatchNormalization())
```

```
  if apply_dropout:
      result.add(tf.keras.layers.Dropout(0.5))
  result.add(tf.keras.layers.ReLU())
  return result
```

Проверим работу наших функций на эталонном загруженном изображении.

```
down_model = downsample(3, 4)
down_result = down_model(tf.expand_dims(inp, 0))
print (down_result.shape)
up_model = upsample(3, 4)
up_result = up_model(down_result)
print (up_result.shape)
```

Модель генератора организуем в соответсвии со спецификацией pix2pix, выделив энкодерную часть (down_stack) и декодерную (up_stack) и предусмотрев сквозные соединения между ними.

```
OUTPUT_CHANNELS = 3
def Generator():
  inputs = tf.keras.layers.Input(shape=[256, 256, 3])
  down_stack = [
    downsample(64, 4, apply_batchnorm=False),  # (batch_size, 128, 128, 64)
    downsample(128, 4),  # (batch_size, 64, 64, 128)
    downsample(256, 4),  # (batch_size, 32, 32, 256)
    downsample(512, 4),  # (batch_size, 16, 16, 512)
    downsample(512, 4),  # (batch_size, 8, 8, 512)
    downsample(512, 4),  # (batch_size, 4, 4, 512)
    downsample(512, 4),  # (batch_size, 2, 2, 512)
    downsample(512, 4),  # (batch_size, 1, 1, 512)
  ]
  up_stack = [
    upsample(512, 4, apply_dropout=True),  # (batch_size, 2, 2, 1024)
    upsample(512, 4, apply_dropout=True),  # (batch_size, 4, 4, 1024)
    upsample(512, 4, apply_dropout=True),  # (batch_size, 8, 8, 1024)
    upsample(512, 4),  # (batch_size, 16, 16, 1024)
    upsample(256, 4),  # (batch_size, 32, 32, 512)
    upsample(128, 4),  # (batch_size, 64, 64, 256)
    upsample(64, 4),  # (batch_size, 128, 128, 128)
  ]
  initializer = tf.random_normal_initializer(0., 0.02)
  last = tf.keras.layers.Conv2DTranspose(OUTPUT_CHANNELS, 4,
                                         strides=2,
                                         padding='same',
                                         kernel_initializer=initializer,
                                         activation='tanh')
  x = inputs
  skips = []
  for down in down_stack:
    x = down(x)
```

```
    skips.append(x)
  skips = reversed(skips[:-1])
  for up, skip in zip(up_stack, skips):
    x = up(x)
    x = tf.keras.layers.Concatenate()([x, skip])
  x = last(x)
  return tf.keras.Model(inputs=inputs, outputs=x)
```

Визуализируем граф подключений для генератора.

```
generator = Generator()
tf.keras.utils.plot_model(generator, show_shapes=True, dpi=64)
```

Проверим, что генератор выдает изображения в нужном масштабе.

```
gen_output = generator(inp[tf.newaxis, ...], training=False)
plt.imshow(gen_output[0, ...])
```

Модель дискриминатора будет более компактной, так как в его задачу входит только оценка искусственности сгенерированных изображений.

```
def Discriminator():
  initializer = tf.random_normal_initializer(0., 0.02)
  inp = tf.keras.layers.Input(shape=[256, 256, 3], name='input_image')
  tar = tf.keras.layers.Input(shape=[256, 256, 3], name='target_image')
  x = tf.keras.layers.concatenate([inp, tar])
  down1 = downsample(64, 4, False)(x)  # (batch_size, 128, 128, 64)
  down2 = downsample(128, 4)(down1)  # (batch_size, 64, 64, 128)
  down3 = downsample(256, 4)(down2)  # (batch_size, 32, 32, 256)
  zero_pad1 = tf.keras.layers.ZeroPadding2D()(down3)
  conv = tf.keras.layers.Conv2D(512, 4, strides=1,
                                kernel_initializer=initializer,
                                use_bias=False)(zero_pad1)
  batchnorm1 = tf.keras.layers.BatchNormalization()(conv)
  leaky_relu = tf.keras.layers.LeakyReLU()(batchnorm1)
  zero_pad2 = tf.keras.layers.ZeroPadding2D()(leaky_relu)
  last = tf.keras.layers.Conv2D(1, 4, strides=1,
                                kernel_initializer=initializer)(zero_pad2)
  return tf.keras.Model(inputs=[inp, tar], outputs=last)
```

Визуализируем граф подключений для дискриминатора.

```
discriminator = Discriminator()
tf.keras.utils.plot_model(discriminator, show_shapes=True, dpi=64)
```

Проверим работу дискриминатора.

```
disc_out = discriminator([inp[tf.newaxis, ...], gen_output], training=False)
plt.imshow(disc_out[0, ..., -1], vmin=-20, vmax=20, cmap='RdBu_r')
plt.colorbar()
```

1.2. Задание функций потерь для организации состязательного обучения генератора и дискриминатора. Функция потерь для генератора будет складываться из функции сравнения с эталонным изображением, а также из функции попиксельной оценки качества, которую будет выдавать дискриминатор (disc_generated_output).

```
LAMBDA = 100 # выбрано в pix2pix
```

```
loss_object = tf.keras.losses.BinaryCrossentropy(from_logits=True)
def generator_loss(disc_generated_output, gen_output, target):
  gan_loss = loss_object(tf.ones_like(disc_generated_output),
                         disc_generated_output)
  l1_loss = tf.reduce_mean(tf.abs(target - gen_output))
  total_gen_loss = gan_loss + (LAMBDA * l1_loss)
  return total_gen_loss, gan_loss, l1_loss
```

Дискриминатор в свою очередь должен отличать естественные изображения от созданных с помощью генератора. Поэтому его функция потерь также собирается из двух частей.

```
def discriminator_loss(disc_real_output, disc_generated_output):
  real_loss = loss_object(tf.ones_like(disc_real_output),
                          disc_real_output)
  generated_loss = loss_object(tf.zeros_like(disc_generated_output),
                               disc_generated_output)
  total_disc_loss = real_loss + generated_loss
  return total_disc_loss
```

Подготовим функцию для записи промежуточных результатов обучения (checkpoint saver) и выберем методы градиентного спуска для обучения генератора и дискрминатора.

```
generator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
discriminator_optimizer = tf.keras.optimizers.Adam(2e-4, beta_1=0.5)
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt")
checkpoint = tf.train.Checkpoint(generator_optimizer=generator_optimizer,
                                 discriminator_optimizer=discriminator_optimizer,
                                 generator=generator,
                                 discriminator=discriminator)
```

Для визуализации промежуточных результатов добавим следующую функцию.

```
def generate_images(model, test_input, tar):
  prediction = model(test_input, training=True)
  plt.figure(figsize=(15, 15))
  display_list = [test_input[0], tar[0], prediction[0]]
  title = ['Input Image', 'Ground Truth', 'Predicted Image']
  for i in range(3):
    plt.subplot(1, 3, i+1)
    plt.title(title[i])
    plt.imshow(display_list[i] * 0.5 + 0.5)
    plt.axis('off')
  plt.show()
```

Проверим ее работу на одном примере из валидатационного датасета.

```
for example_input, example_target in test_dataset.take(1):
  generate_images(generator, example_input, example_target)
```

1.3. Обучение модели и визуализация результатов. Подготовим директорию для записи логов по эпохам обучения.

```
log_dir="logs/"
```

```
summary_writer = tf.summary.create_file_writer(
  log_dir + "fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S"))
```

Для отдельного шага обучения реализуем свою функцию для последовательного обучения дискриминатора и генератора.

```
@tf.function
def train_step(input_image, target, step):
  with tf.GradientTape() as gen_tape, tf.GradientTape() as disc_tape:
    gen_output = generator(input_image, training=True)
    disc_real_output = discriminator([input_image, target], training=True)
    disc_generated_output = discriminator([input_image, gen_output],
                                          training=True)
    gen_total_loss, gen_gan_loss, gen_l1_loss =
        generator_loss(disc_generated_output, gen_output, target)
    disc_loss = discriminator_loss(disc_real_output, disc_generated_output)
  generator_gradients = gen_tape.gradient(gen_total_loss,
                                          generator.trainable_variables)
  discriminator_gradients = disc_tape.gradient(disc_loss,
                                       discriminator.trainable_variables)
  generator_optimizer.apply_gradients(zip(generator_gradients,
                                        generator.trainable_variables))
  discriminator_optimizer.apply_gradients(zip(discriminator_gradients,
                                       discriminator.trainable_variables))
  with summary_writer.as_default():
    # запишем промежуточные результаты в лог по одному разу в тычу эпох.
    tf.summary.scalar('gen_total_loss', gen_total_loss, step=step//1000)
    tf.summary.scalar('gen_gan_loss', gen_gan_loss, step=step//1000)
    tf.summary.scalar('gen_l1_loss', gen_l1_loss, step=step//1000)
    tf.summary.scalar('disc_loss', disc_loss, step=step//1000)
```

Итоговая функция для реализации конкуретного обучения генератора и дискриминатора.

```
def fit(train_ds, test_ds, steps):
  example_input, example_target = next(iter(test_ds.take(1)))
  start = time.time()
  for step,(input_image, target) in train_ds.repeat().take(steps).enumerate():
    if (step) % 1000 == 0:
      display.clear_output(wait=True)
      if step != 0:
        print(f'Time taken for 1000 steps: {time.time()-start:.2f} sec\n')
      start = time.time()
      generate_images(generator, example_input, example_target)
      print(f"Step: {step//1000}k")
    train_step(input_image, target, step)
    # Каждые 10 шагов печатаем точку
    if (step+1) % 10 == 0:
      print('.', end='', flush=True)
    # Каждые 5000 эпох запоминаем чекпоинт
```

```
    if (step + 1) % 5000 == 0:
      checkpoint.save(file_prefix=checkpoint_prefix)
```

Подключаем tensorboard для визуализации данных из логов и запускаем обучение.

```
%load_ext tensorboard
%tensorboard --logdir {log_dir}
fit(train_dataset, test_dataset, steps=30000)
```

§ 2. Практические задания

Задание № 1: Визуализация работы и доработка модели.

1. Извлечь данные из логов и найти точку на которой у модели были оптимальные параметры с точки зрения валидатационной выборки.
2. Сохранить коэффициенты модели, которые соответсвуют этому идеальному состоянию.
3. Сравнить на графиках и визуально на генерируемых изображениях качество в оптимальной точке и на последней итерации.
4. Предложить как можно было бы переработать наш алгоритм обучения. Какие наблюдаются недостатки у нашей версии?

Задание № 2: Адаптировать модель под набор данных из лабораторной работы по сегментации изображений.

1. В качестве эталонных размеченных данных взять сегментационные маски для животных.
2. В качестве эталонных изображений - картинки самих животных.
3. Провести обучение и сравнить качество работы на оптимальной итерации и на последней.