УДК 519.68:007.5

Лабораторная работа 4. Сети со сквозными соединениями и задача сегментации изображений

Цели работы: подготовить модель типа U-net для решения задачи сегметнации изображений на основе предобученной сверточной сети, путем внедрения в ее структуру сквозных соединений между дальними слоями.

Ключевые слова: сверточные сети, сегментация изображений, перенос обучения, сквозные соединения, модели U-Net.

§ 1. Построение моделей со сквозными соединениями

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

```
import tensorflow as tf
import tensorflow_datasets as tfds
from IPython.display import clear_output
import matplotlib.pyplot as plt
```

Как вариант, можно вместо локально установленного Python'а использовать облачный сервис Google Colab с его реализацией тетради jupyter.

```
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
# загружаем набор изображений вместе с готовыми масками для обучения
TRAIN_LENGTH = info.splits['train'].num_examples
# набор данных уже размечен для валидации и обучения
BATCH_SIZE = 64 # можете установить 32 или 128, как вариант
BUFFER_SIZE = 1000 # размер буфера для перемешивания обучающих данных
STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE
# количество шагов градиентного спуска за одну эпоху обучения
```

Предусмотрим предобработку изображений: приведение к одному размеру (128,128) и нормализацию пикселей изображений и их эталонных масок.

```
def normalize(input_image, input_mask):
  input_image = tf.cast(input_image, tf.float32) / 255.0
  # значения пикселей для изображений отобразим на отрезок [0,1]
  input_mask -= 1 # значение пикселей для маски полагаем: -1, 0, 1
  return input_image, input_mask
def load_image(datapoint):
  input_image = tf.image.resize(datapoint['image'], (128, 128))
  input_mask = tf.image.resize(
    datapoint['segmentation_mask'],
    (128, 128),
    method = tf.image.ResizeMethod.NEAREST_NEIGHBOR,
  )
  input_image, input_mask = normalize(input_image, input_mask)
  return input_image, input_mask
```

Используя готовое разбиение, загрузим данные для обучения и валидации:

```
train_images = dataset['train'].map(load_image,
                                    num_parallel_calls=tf.data.AUTOTUNE)
test_images = dataset['test'].map(load_image,
                                  num_parallel_calls=tf.data.AUTOTUNE)
```

Для обучающей выборки подготовим класс для аугментации:

```
class Augment(tf.keras.layers.Layer):
  def __init__(self, seed=42):
    super().__init__()
    self.augment_inputs = tf.keras.layers.RandomFlip(mode="horizontal",
                                                     seed=seed)
    self.augment_labels = tf.keras.layers.RandomFlip(mode="horizontal",
                                                     seed=seed)
  def call(self, inputs, labels):
    inputs = self.augment_inputs(inputs)
    labels = self.augment_labels(labels)
    return inputs, labels
```

1.2. Искуственное расширение обучающей выборки. Применим к обучающей выборке аугментацию, искусственное расширение, разбиение на пакеты, перестановку примеров и кеширование для чтения с диска. Для валидационной выборки применим только разбиение на пакеты.

```
train_batches = (
    train_images
    .cache()
    .shuffle(BUFFER_SIZE)
    .batch(BATCH_SIZE)
    .repeat()
    .map(Augment())
    .prefetch(buffer_size=tf.data.AUTOTUNE))
test_batches = test_images.batch(BATCH_SIZE)
```

Подготовим функцию для визуализации обучающих данных, эталонной маски, а также маски полученной в результате обучения:

```
def display(display_list):
  plt.figure(figsize=(15, 15))
  title = ['Input Image', 'True Mask', 'Predicted Mask']
  for i in range(len(display_list)):
    plt.subplot(1, len(display_list), i+1)
    plt.title(title[i])
    plt.imshow(tf.keras.utils.array_to_img(display_list[i]))
    plt.axis('off')
  plt.show()
```

Проверим работу функции визуализиации, а заодно и корректность загрузки обучающих данных:

```
for images, masks in train_batches.take(2):
  sample_image, sample_mask = images[0], masks[0]
  display([sample_image, sample_mask])
```

1.3. Перенос обучения с добавлением сквозных соединений. Возмьмем за основу модель MobileNet2, отключив у неё выходные слои.

```
base_model = tf.keras.applications.MobileNetV2( input_shape=[128, 128, 3],
                                                 include_top=False)
```

Выделим часть из промежуточных слоев исходной модели MobileNet2 для создания сквозных соединений:

```
layer_names = [
    'block_1_expand_relu',   # 64x64 размерность
    'block_3_expand_relu',   # 32x32 размерность
    'block_6_expand_relu',   # 16x16 размерность
    'block_13_expand_relu',  # 8x8   размерность
    'block_16_project',      # 4x4   размерность
]
```

В соответствии со спецификацией модели U-Net предусмотрим энкодерную часть (downsampler), которая будет использовать эти слои как выходные.

```
base_model_outputs =
    [base_model.get_layer(name).output for name in layer_names]
# Создаем энкодерную часть модели (downsampler):
down_stack = tf.keras.Model(inputs=base_model.input,
                            outputs=base_model_outputs)
down_stack.trainable = False
# замораживаем коэффициенты предобученной модели MobileNet2
```

Для реализации декодерной части подготовим следующую функцию:

```
def upsample(filters, size, apply_dropout=False):
  initializer = tf.random_normal_initializer(0., 0.02)
  # инициализатор с нормальным распределением (m=0, sigma=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
```

Декодерная часть U-net будет организована в стек слоев:

```
up_stack = [
    upsample(512, 3),  # 4x4 -> 8x8 после Conv2DTranspose
    upsample(256, 3),  # 8x8 -> 16x16 ...
    upsample(128, 3),  # 16x16 -> 32x32 ...
    upsample(64, 3),   # 32x32 -> 64x64 ...
]
```

Итоговая U-net модель со сквозными соединениями (skip connection) будет создаваться в отдельной функции. При этом, output_channels - число сегментов в итоговом сегментированном изображении:

```
def unet_model(output_channels:int):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  skips = down_stack(inputs)
  x = skips[-1]
  skips = reversed(skips[:-1])

  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # Итоговый слой должен выдать маску 128x128
  last = tf.keras.layers.Conv2DTranspose(
      filters=output_channels, kernel_size=3, strides=2,
      padding='same')  #64x64 -> 128x128
  x = last(x)
  return tf.keras.Model(inputs=inputs, outputs=x)
```

1.4. Обучение модели и визуализация результатов. Скомпилируем итоговую модель, выбрав в качестве метода градиентного спуска метод Adam.

```
OUTPUT_CLASSES = 3

model = unet_model(output_channels=OUTPUT_CLASSES)
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(
                    from_logits=True),
              # Выходы не нормализованы, поэтому мы используем logits
              metrics=['accuracy'])
```

Визуализируем в виде графа итоговую U-net модель:

```
tf.keras.utils.plot_model(model, show_shapes=True)
```

Для итоговой маски предусмотрим функцию для её презентации:

```
def create_mask(pred_mask):
  pred_mask = tf.math.argmax(pred_mask, axis=-1)
  pred_mask = pred_mask[..., tf.newaxis]
  return pred_mask[0]

def show_predictions(dataset, num=1):
  for image, mask in dataset.take(num):
    pred_mask = model.predict(image)
    display([image[0], mask[0], create_mask(pred_mask)])
```

Промежуточные результаты будем визуализировать с помощью класса:

```
class DisplayCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs=None):
```

```
    clear_output(wait=True)
    show_predictions(train_batches)
    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

Запустим обучение модели на двадцать эпох, подключив нашу функцию для визуализации промежуточных результатов по итогам каждой эпохи обучения.

```
EPOCHS = 20
VAL_SUBSPLITS = 5 # разобъем процесс валидации на 5 этапов
VALIDATION_STEPS =
    info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS
model_history = model.fit(train_batches, epochs=EPOCHS,
                          steps_per_epoch=STEPS_PER_EPOCH,
                          validation_steps=VALIDATION_STEPS,
                          validation_data=test_batches,
                          callbacks=[DisplayCallback()])
```

После обучения построим графики для функций потерь на обучающих и валидатационных данных:

```
loss = model_history.history['loss']
val_loss = model_history.history['val_loss']
plt.plot(model_history.epoch, loss, 'r', label='Training loss')
plt.plot(model_history.epoch, val_loss, 'bo', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.ylim([0, 1])
plt.legend()
plt.show()
```

§ 2. Практические задания

Задание № 1: Доработка модели.

1. Доработать нашу модель, включив опциональный dropout, а также тонкую настройку (разморозив коэффициенты).
2. Визуализировать результаты, сравнить с базовой моделью.
3. Сравнить качество и скорость обучения при использовании заморозки коэффициентов предобученных слоев и без заморозки.

Задание № 2: Подготовка новых наборов данных.

1. Выбрать один новый тип объекта для сегментации: пожар (1), водоем (2), свалка (3) в соответсвии с формулой N mod 3 + 1, где N - это номер в списке группы.
2. Собрать датасет, который будет содержать изображения с нужным объектом (минимум 100 изображений).
3. Подготовить для изображений маски, разметив копии изображений вручную, либо с помощью специальной программы. К примеру, можно воспользовать VIA (VGG Image Annotator).
4. Обучить модель U-net и проверить качество её работы на новом изображении (не входившем в собранный датасет).