GeekBrains

Введение в искусственные нейронные сети

Сегментация

На этом уроке

1. Познакомимся с сегментацией изображений
2. Изучим виды архитектур
3. Рассмотрим сегментацию на практике

Оглавление

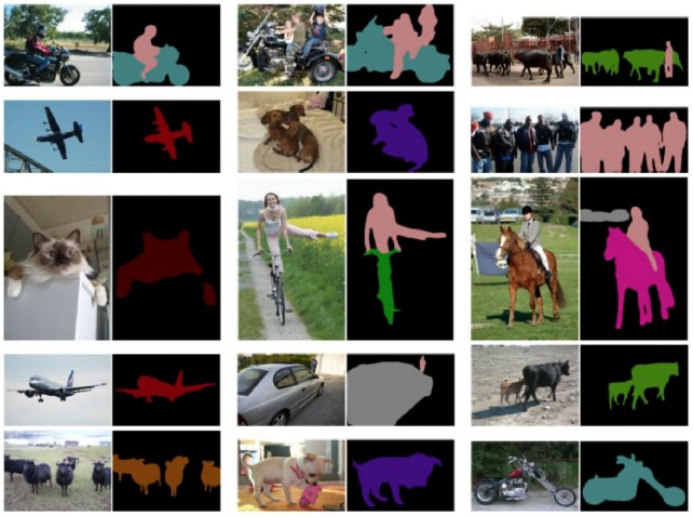
Что такое сегментация изображения

В компьютерном зрении есть несколько основных видов задач, например классификация, сегментация и детектирование объектов.

До сих пор задачи в области компьютерного зрения, с которыми мы сталкивались, относились к задаче классификации. Она подразумевает получение от нейронной сети предсказания названия класса объекта, представленного ей на изображении.

Решение такой задачи интересно для академических кругов, для поиска новых архитектур и определённого практического применения, однако для продвинутого компьютерного зрения, как правило, требуется не только знать, что за объект находится на изображении, но и где он находится, какой формы этот объект и т.д. Нейронные сети для сегментации и нейронные сети для детектирования объектов призваны решить эти задачи.

В этом уроке мы познакомимся с задачей сегментации. Сегментация изображения подразумевает отнесение каждого пикселя изображения к определённому классу. По сути мы получаем маску изображения.



Нейронные сети для сегментирования изображений находят широкое применение в беспилотном транспорте, «умном» видеонаблюдении, медицине и других областях.

Подобного рода нейронные сети обычно состоят как правило из части, ответственной за восприятие объектов на изображении, и части, ответственной за определение контуров изображения. Такие нейросети относительно тяжеловесны (с точки зрения вычислительных затрат), в научных кругах ведётся поиск более легковесных решений.

Для тренировки нейронных сетей, решающих задачу сегментирования, применяются свои специальные датасеты, в которых размечена маска изображения, например, PASCAL Visual Object Classes или COCO (Common Objects in Context). Датасетов для сегментации меньше и они сложнее в составлении. Поэтому существует распространённая практика использования предобученнной

нейронной сети (например, на ImageNet) в качестве части, ответственной за восприятие объектов. Затем нейронная сеть дообучается на размеченных масках с помощью специальных датасетов.

На сегодняшний день существует множество архитектур для сегментации изображений, далее разберём основные из них.

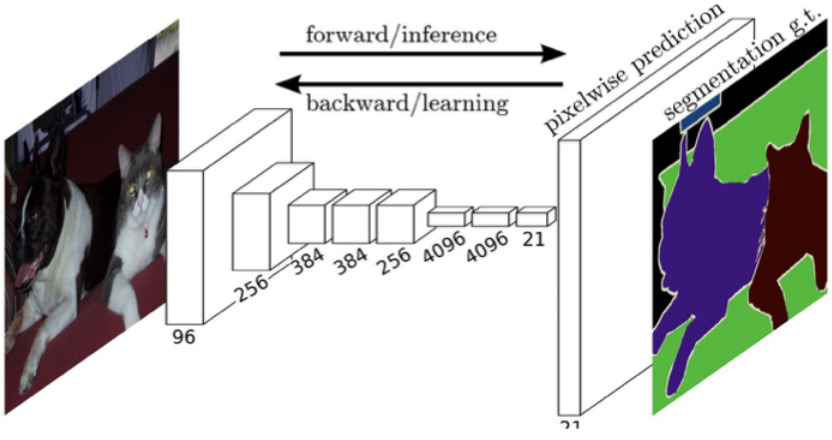
Виды архитектур для сегментации изображений

Как уже было упомянуто ранее, существует много архитектур для решения задачи сегментации, и новые всё продолжают появляться. Мы рассмотрим такие архитектуры, как FCN (fully convolutional network), SegNet, U-net, Mask-RCNN. Они отражают основные вехи в эволюции архитектур нейронных сетей для решения задачи сегментации. Также мы рассмотрим архитектуру Autoencoder — её полезно изучить как вводную часть к большинству архитектур в области сегментации изображений.

FCN (fully convolutional network)

Эта архитектура появилась в 2014 г., буквально через несколько лет после того, как глубокое обучение получило широкое распространение.

FCN (в отличие от обычной CNN) не имеет на конце полносвязных слоёв, вместо этого на конце нейронной сети располагается модуль, позволяющий увеличить то представление изображения, которое имеют обычные CNN перед передачей данных в полносвязные слои.

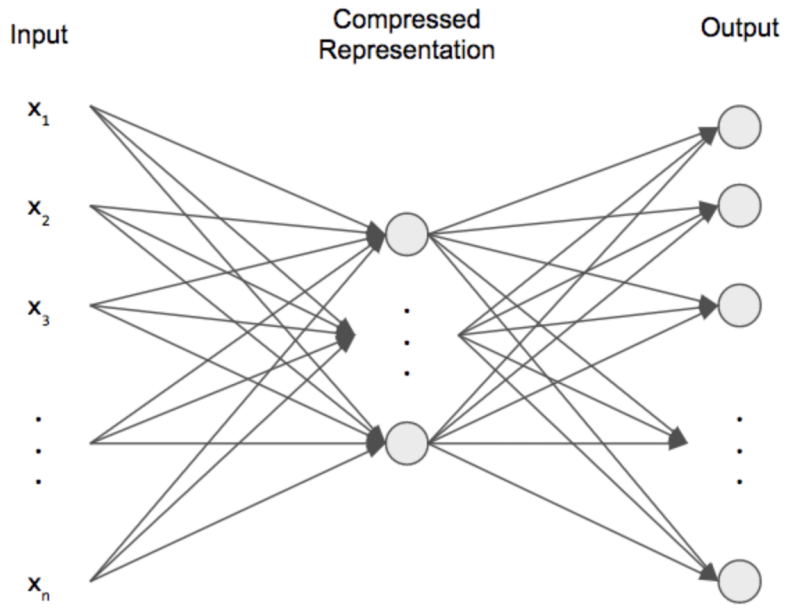


Такой подход позволил превзойти предыдущие подходы в этой области, базировавшиеся не на глубоком обучении, но и он имел свои изъяны.

Autoencoder

Прежде чем разобрать архитектуру SegNet, изучим Autoencoder. Эта архитектура представляет собой набор слоёв, сжимающих входящие данные во всё более маленькое представление, а также слоёв, впоследствии разжимающих эти данные.

Если после такой процедуры сжатия и разжатия на выходе удаётся получить изначальные данные, значит в центральных слоях архитектуры располагается сжатое представление данных. Отбросив вторую часть (разжимающую данные), можно использовать сжатые данные для различных целей.

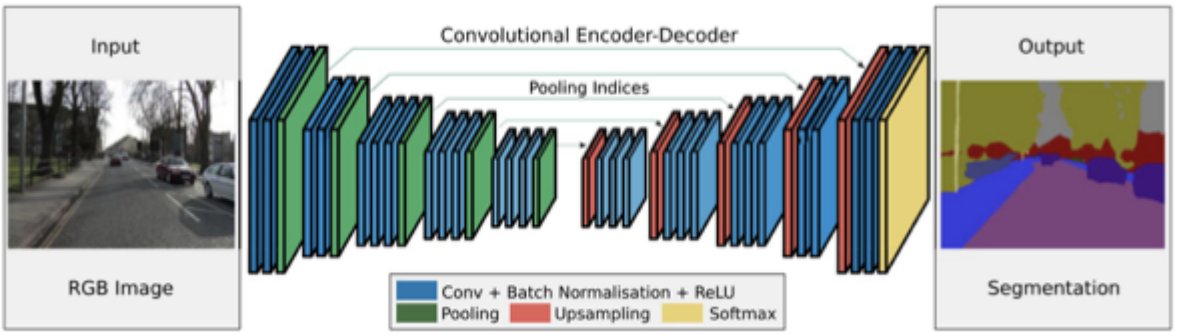


SegNet

SegNet появилась в 2015 г. Она состоит из конволюционной и деконволюционной части и отчасти повторяет собой архитектуру Autoencoder, где есть кодирующая и декодирующая части.

Декодирующая часть в SegNet позволяет сделать более плавное разворачивание изображения после того как отработала конволюционная (свёрточная) часть архитектуры. Благодаря этому границы объектов на изображение определяются более корректно.

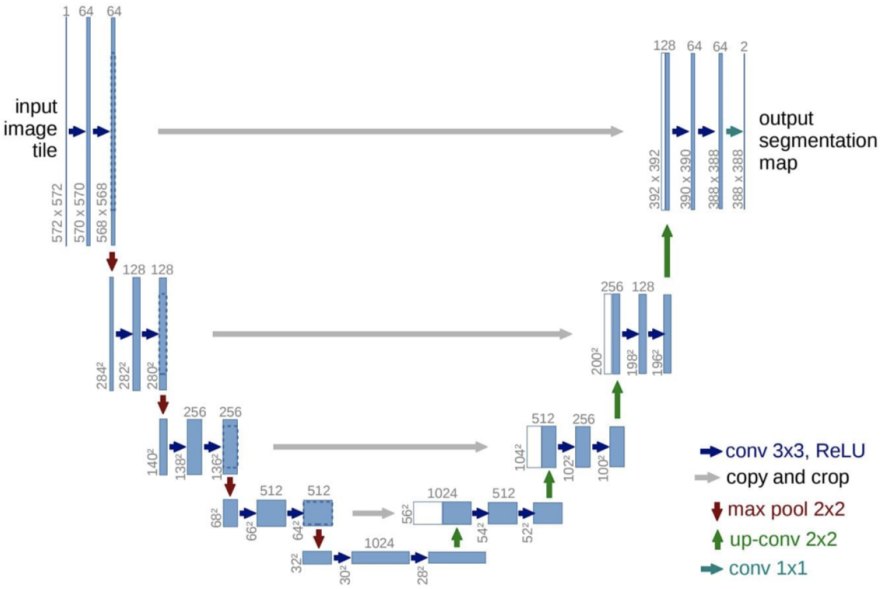
Стоит отметить, что в качестве конволюционной (свёрточной) части могут использоваться различные предтренированные нейронные сети для решения задач классификации, например различные модификации VGG и ResNet.



U-Net

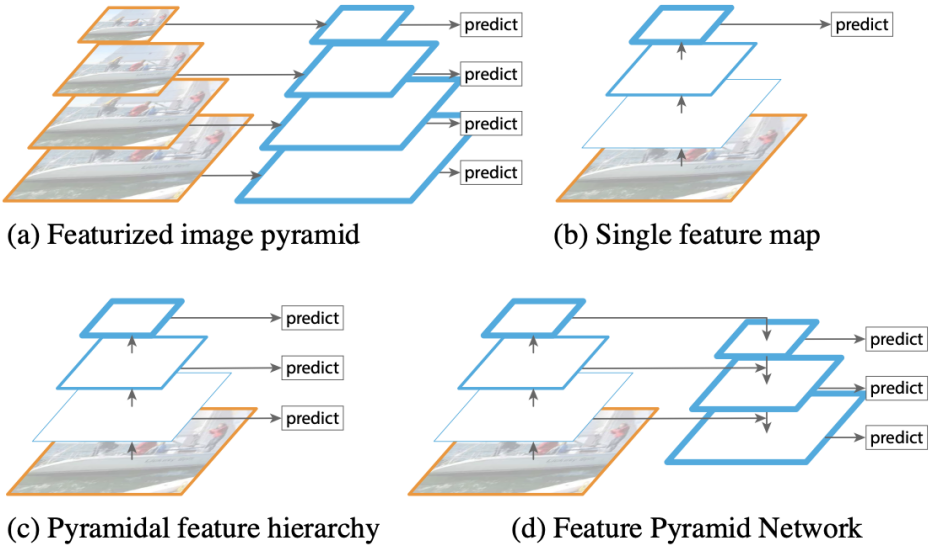
U-net появился в 2015 г. для решения медицинских задач. Эта архитектура и её модификации являются одним из основных практических инструментов для решения современных задач сегментации изображений. Архитектура отличается более высокой степенью точности сегментирования, достаточной для работы с медицинскими снимками и картами, где располагаются множество мелких объектов, и других задач.

U-net, так же как и SegNet, берёт input и реконструирует output как сегментированную карту изображения. Однако в данной архитектуре используются skip connections, похожие на те, что помогли решить проблему исчезающего градиента в задаче классификации (ResNet архитектура). Эти skip connections позволяют сигналу не только проходить строго по траектории (сначала по нисходящей лестницы слоёв, а потом по восходящей лестнице апсемплинга (увеличение маленькой репрезентации изображения в полноценную карту)), но и перескакивать между слоями одного уровня, что решает проблему дублирования функций слоёв и улучшает эффективность обучения.



FPN (The Feature Pyramid Network)

Архитектура появилась в 2017 г. и визуально похожа на пирамиду. Она может применяться как для задач детектирования объектов, так и для задач сегментации изображений. FPN во многом похожа на U-Net, но есть важное нововведение: предсказания снимаются не только с самой высокой части апсемплинга, но и на каждой ступени, начинающейся в фазе декодирования. Это позволяет совершать детекцию на разных масштабах восстанавливаемого изображения, что положительно сказывается на точности сегментирования.



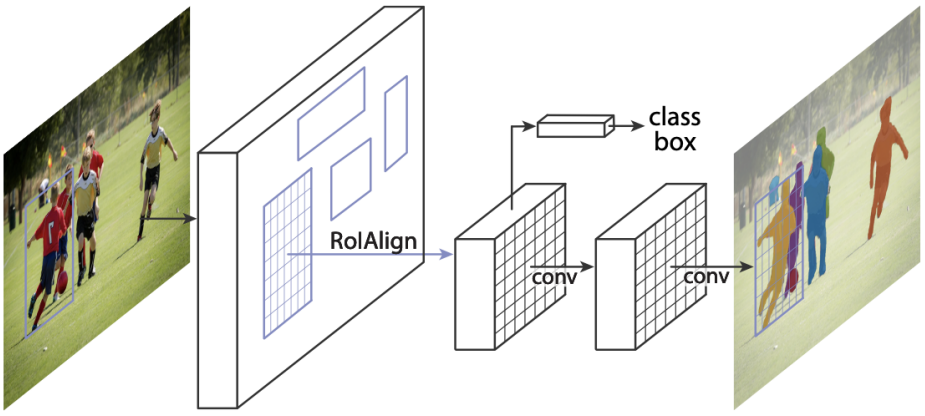
(a) Featurized image pyramid (b) Single feature map

(c) Pyramidal feature hierarchy (d) Feature Pyramid Network

Mask R-CNN

Архитектура появилась в 2018 г. и является надстройкой над Faster R-CNN, одной из лучших архитектур в сфере object detection. Она будет рассмотрена в следующем уроке.

В Mask R-CNN, так же как и в ранее рассмотренных архитектурах, используется часть с энкодером (например ResNet101), но в неё добавлен модуль Region Proposal Network (RPN) из сферы object detection, позволяющий эффективно находить места на изображении, где могут находиться объекты. После чего Mask R-CNN генерирует сегментационную маску.



Практический пример сегментации

В качестве практики попробуем сегментировать изображение домашних животных из датасета Oxford-IIIT Pet Dataset, с использованием модифицированной версии U-Net. Данный пример тестировался и корректно работал при составлении методического пособия на python 3.7.7 и tensorflow 2.1.0. Также может понадобиться установка модуля для python под названием tensorflow-datasets.

```
##### Copyright 2019 The TensorFlow Authors.


Licensed under the Apache License, Version 2.0 (the "License");
#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
!pip install git+https://github.com/tensorflow/examples.git
try:
  # %tensorflow_version only exists in Colab.
  %tensorflow_version 2.x
except Exception:
  pass
import tensorflow as tf
from __future__ import absolute_import, division, print_function, unicode_literals


from tensorflow_examples.models.pix2pix import pix2pix


import tensorflow_datasets as tfds
tfds.disable_progress_bar()
```

```
from IPython.display import clear_output

import matplotlib.pyplot as plt



import tensorflow as tf

tf.config.experimental.set_visible_devices([], 'GPU')
```

Загрузка датасета Oxford-IIIT Pets

Датасет является стандартным для TensorFlow, однако, как упоминалось выше, возможно будет необходимо установить модуль tensorflow-datasets для python.

```
dataset, info = tfds.load('oxford_iiit_pet:3.*.*', with_info=True)
```

Следующий код выполнит простую аугментацию данных посредством переворота изображений. В дополнение изображение будет нормализовано к 0 и 1. Пиксели сегментационной маски будут помечены {1, 2, 3}, но для удобства из данного цифрового ряда будет вычтено по 1 и в итоге получится {0, 1, 2}.

```
def normalize(input_image, input_mask):

  input_image = tf.cast(input_image, tf.float32) / 255.0

  input_mask -= 1

  return input_image, input_mask

@tf.function

def load_image_train(datapoint):

  input_image = tf.image.resize(datapoint['image'], (128, 128))

  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))



  if tf.random.uniform(()) > 0.5:

    input_image = tf.image.flip_left_right(input_image)

    input_mask = tf.image.flip_left_right(input_mask)



  input_image, input_mask = normalize(input_image, input_mask)



  return input_image, input_mask

def load_image_test(datapoint):

  input_image = tf.image.resize(datapoint['image'], (128, 128))

  input_mask = tf.image.resize(datapoint['segmentation_mask'], (128, 128))
```

```
  input_image, input_mask = normalize(input_image, input_mask)


  return input_image, input_mask
```

Датасет уже содержит необходимые тестовый и тренеровочный сплиты, поэтому будем использовать их.

```
TRAIN_LENGTH = info.splits['train'].num_examples

BATCH_SIZE = 64

BUFFER_SIZE = 1000

STEPS_PER_EPOCH = TRAIN_LENGTH // BATCH_SIZE

train = dataset['train'].map(load_image_train, num_parallel_calls=tf.data.experimental.AUTOTUNE)

test = dataset['test'].map(load_image_test)

train_dataset = train.cache().shuffle(BUFFER_SIZE).batch(BATCH_SIZE).repeat()

train_dataset = train_dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)

test_dataset = test.batch(BATCH_SIZE)
```

Рассмотрим изображение из датасета и соответствующую ему маску из датасета.

```
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.preprocessing.image.array_to_img(display_list[i]))

    plt.axis('off')

  plt.show()

for image, mask in train.take(1):

  sample_image, sample_mask = image, mask

display([sample_image, sample_mask])
```

Определение модели

Будем использовать модифицированный U-Net. В качестве энкодера будет использоваться предтренированный MobileNetV2. Декодером будет апсемпл блок, заранее имплементированный в TensorFlow examples Pix2pix tutorial.

Причина, по который используются три канала, заключается в наличии 3-х возможных лейблов на каждый пиксель. Это можно воспринимать как классификацию, где каждый пиксель будет принадлежать одному из трёх классов.

```
OUTPUT_CHANNELS = 3
```

Энкодером будет предтренированный MobileNetV2, который подготовлен и готов к использованию — tf.keras.applications. Энкодер состоит из определённых аутпутов из средних слоёв модели. Обратите внимание: энкодер не будет участвовать в процессе тренировки модели.

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


# Use the activations of these layers
layer_names = [
    'block_1_expand_relu',   # 64x64
    'block_3_expand_relu',   # 32x32
    'block_6_expand_relu',   # 16x16
    'block_13_expand_relu',  # 8x8
    'block_16_project',      # 4x4
]
layers = [base_model.get_layer(name).output for name in layer_names]


# Create the feature extraction model
down_stack = tf.keras.Model(inputs=base_model.input, outputs=layers)


down_stack.trainable = False
```

Декодер/апсемплер — это просто серия апсемпл блоков, имплементированных в TensorFlow examples.

```
up_stack = [
```

```
    pix2pix.upsample(512, 3),  # 4x4 -> 8x8
    pix2pix.upsample(256, 3),  # 8x8 -> 16x16
    pix2pix.upsample(128, 3),  # 16x16 -> 32x32
    pix2pix.upsample(64, 3),   # 32x32 -> 64x64
]
def unet_model(output_channels):
  inputs = tf.keras.layers.Input(shape=[128, 128, 3])
  x = inputs

  # Downsampling through the model
  skips = down_stack(x)
  x = skips[-1]
  skips = reversed(skips[:-1])

  # Upsampling and establishing the skip connections
  for up, skip in zip(up_stack, skips):
    x = up(x)
    concat = tf.keras.layers.Concatenate()
    x = concat([x, skip])

  # This is the last layer of the model
  last = tf.keras.layers.Conv2DTranspose(
      output_channels, 3, strides=2,
      padding='same')  #64x64 -> 128x128

  x = last(x)

  return tf.keras.Model(inputs=inputs, outputs=x)
```

Тренировка модели

Теперь осталось скомпилировать модель, и начать процесс её тренировки. Loss-функция, которую будем использовать: `losses.SparseCategoricalCrossentropy(from_logits=True)`. Причина использования заключается в том, что нейросеть пытается назначить каждому пикселю лейбл, так же как в задачах предсказания класса. Для модели с 3 каналами, каждый из которых пытается предсказать класс, `losses.SparseCategoricalCrossentropy(from_logits=True)` обычно

также рекомендуется. На выходе нейросети каждому пикселю назначается лейбл с наибольшим значением. Это то, что делает функция `create_mask`.

```
model = unet_model(OUTPUT_CHANNELS)

model.compile(optimizer='adam',

              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),

              metrics=['accuracy'])
```

Посмотрим на получившуюся архитектуру модели.

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

С помощью модели попробуем сделать предсказание до того, как началось обучение.

```
def create_mask(pred_mask):

  pred_mask = tf.argmax(pred_mask, axis=-1)

  pred_mask = pred_mask[..., tf.newaxis]

  return pred_mask[0]

def show_predictions(dataset=None, num=1):

  if dataset:

    for image, mask in dataset.take(num):

      pred_mask = model.predict(image)

      display([image[0], mask[0], create_mask(pred_mask)])

  else:

    display([sample_image, sample_mask,

             create_mask(model.predict(sample_image[tf.newaxis, ...]))])

show_predictions()
```

Осуществим мониторинг того, как улучшается работа модели в процессе обучения. Callback функция для завершения этой задачи определена ниже.

```
class DisplayCallback(tf.keras.callbacks.Callback):

  def on_epoch_end(self, epoch, logs=None):

    clear_output(wait=True)

    show_predictions()

    print ('\nSample Prediction after epoch {}\n'.format(epoch+1))
```

```
EPOCHS = 5 # увеличьте при необходимости

VAL_SUBSPLITS = 5

VALIDATION_STEPS = info.splits['test'].num_examples//BATCH_SIZE//VAL_SUBSPLITS


model_history = model.fit(train_dataset, epochs=EPOCHS,

                          steps_per_epoch=STEPS_PER_EPOCH,

                          validation_steps=VALIDATION_STEPS,

                          validation_data=test_dataset,

                          callbacks=[DisplayCallback()])

loss = model_history.history['loss']

val_loss = model_history.history['val_loss']


epochs = range(EPOCHS)


plt.figure()

plt.plot(epochs, loss, 'r', label='Training loss')

plt.plot(epochs, 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()
```

Make predictions

Сделаем несколько предсказаний. Для экономии времени использовалось небольшое количество эпох, но вы можете его увеличить (для того, чтобы модель давала более точные результаты).

```
show_predictions(test_dataset, 3)
```

Используемые источники

- https://www.tensorflow.org/tutorials/images/segmentation
- Image Segmentation Using Deep Learning: A Survey. Shervin Minaee и др. 15 Jan 2020
- Mask R-CNN, Kaiming He и др. 24 янв. 2018 г.

- https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4