МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

09.04.01 «Информатика и вычислительная техника»

Отчет по практической работе №5
«Семантическая сегментация изображений с использованием архитектуры U-Net»

МАШИННОЕ ОБУЧЕНИЕ

Исполнитель:

студент группы 8ВМ22 Ямкин Н.Н. 12.09.2023

Руководитель:

Доцент (ОИТ, ИШИТР) Друки А.А.

Томск – 2023

Содержание

1. Цель работы

Получить навыки сегментации снимков дистанционного зондирования земли на основе модели нейронной сети U-Net.

2. Задачи

1. Ознакомиться с работой сверточных нейронных сетей в библиотеке Keras;

2. Научиться решать задачу сегментации изображений на основе нейронных сетей.

3. Научиться применять модель сверточной нейронной сети U-Net.

3. Ход работы

Подключаем необходимые библиотеки:

```
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image, ImageDraw
from skimage import io
import os

from keras.models import *
from keras.layers import *
from keras.optimizers import *
```

Подключаем Google-диск:

```
from google.colab import drive
drive.mount('/content/drive')
```

Определяем функцию загрузки изображений:

```
def download_data(path):
  data = []
  for path_image in sorted(os.listdir(path=path)):
    image = Image.open(path + path_image) #Открываем изображение
    data.append(np.array(image)) #Загружаем пиксели
  return data
```

Загружаем исходный набор данных:

```
X_train = download_data(r"/content/drive/MyDrive/Машинное обучение ПР№4
/train/images/")
Y_train = download_data(r"/content/drive/MyDrive/Машинное обучение ПР№4
/train/masks/")
X_test = download_data(r"/content/drive/MyDrive/Машинное обучение ПР№4
/test/images/")
Y_test = download_data(r"/content/drive/MyDrive/Машинное обучение ПР№4
/test/masks/")
```

Отобразим изображения из обучающей и тестовой выборки:

```
# Выводим на экран 8 изображений из обучающей БД
plt.figure(figsize=(6,6))
for i in range(7):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(X_train[i], cmap=plt.cm.binary)
```

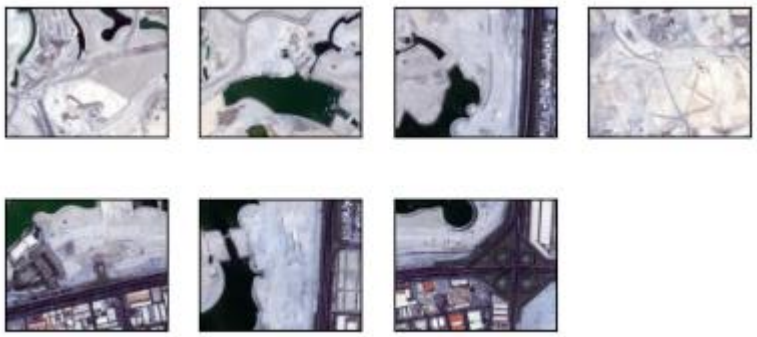

Рисунок 1 – Изображения из обучающей выборки

```
plt.figure(figsize=(6,6))
for i in range(7):
    plt.subplot(4,4,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.imshow(Y_train[i], cmap=plt.cm.binary)
```

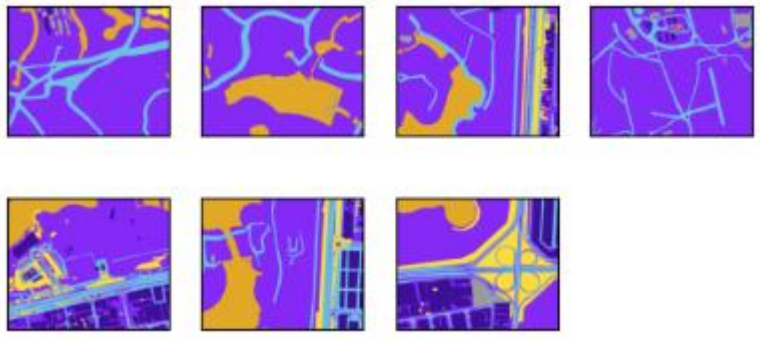

Рисунок 2 – Изображения из тестовой выборки

Определяем палитру, которая необходима для разграничивания данных. Ключом является класс объекта, значение - цвет маски в формате RGB.

```
palette = {0 : (60, 16, 152), # Building
           1 : (132, 41, 246), # Land
           2 : (110, 193, 228), # Road
           3 : (254, 221, 58), # Vegetation
           4 : (226, 169, 41), # Water
           5 : (155, 155, 155)} # Unlabeled
```

Создаём инвертированнную палитру:

```
# обратное преобразование, цвета в метку класса
invert_palette = {v: k for k, v in palette.items()}
```

Объявляем функцию, конвертирующую входной двумерный массив, содержащий класс цвета в двумерный массив, содержащий цветовую маску в формате RGB.

```
def convert_to_color(arr_2d, palette=palette):
    """ Numeric labels to RGB-color encoding """
    arr_3d = np.zeros((arr_2d.shape[0], arr_2d.shape[1], 3),
dtype=np.uint8)

    for c, i in palette.items():
        m = arr_2d == c
        arr_3d[m] = i
    return arr_3d
```

Объявляем функцию, конвертирующую входной двумерный массив, содержащий цветовую маску в формате RGB в двумерный массив, содержащий единичные вектора, соответствующие классу соответствующего цвета.

```
def convert_from_color(arr_3d, palette=invert_palette):
    """ RGB-color encoding to grayscale labels """
    arr_2d = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.uint8)

    arr_2d = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.int8) # принадлежность каждого пикселя классу
    min_distance = np.zeros((arr_3d.shape[0], arr_3d.shape[1]), dtype=np.float32)+1000 # расстояние до ближайшего класса для пикселей
    for c, i in palette.items():
      distance = np.sum((arr_3d - np.array(c).reshape(1, 1, 3))**2, axis=-1)**(1/2) # ищем расстояние для каждого пикселя до проверяемого класса по евклиду рас-ие
      condition = min_distance > distance # поиск элементов меньше min_distance
      min_distance[condition] = distance[condition] # замена дистанции найденных элементов
      arr_2d[condition] = i # замена класса найденных элементов

    for c, i in palette.items():
      m = np.all(arr_3d == np.array(c).reshape(1, 1, 3), axis=2)
      arr_2d[m] = i

    arr_2d = arr_2d.tolist()
    for i in range(len(arr_2d)):
      for j in range(len(arr_2d[0])):
        label = [0, 0, 0, 0, 0, 0]
        label[arr_2d[i][j]] = 1
        arr_2d[i][j] = label
    arr_2d = np.array(arr_2d)

    return arr_2d
```

Подготовим и нормализуем входные изображения.

```
X_train_pred = np.array(X_train).reshape([7, 644, 796, 3])/255
X_test_pred = np.array(X_test).reshape([2, 644, 796, 3])/255
```

Подготовим тестовые и проверочные выходные изображения.

```
Y_train_pred = []
for i in range(len(Y_train)):
  Y_train_pred.append(convert_from_color(Y_train[i][:, :, :3]))
Y_train_pred = np.array(Y_train_pred)

Y_test_pred = []
for i in range(len(Y_test)):
  Y_test_pred.append(convert_from_color(Y_test[i][:, :, :3]))
Y_test_pred = np.array(Y_test_pred)
```

В результате выполнения данного кода в переменные X_train_pred, Y_train_pred, X_test_pred, Y_test_pred будут записаны данные после предварительной обработки, где:

X_train_pred – исходные изображения из обучающей выборки данных.

Y_train_pred – сегментированные исходные изображения из обучающей выборки.

X_test_pred – исходные изображения из тестовой выборки данных.

Y_test_pred – сегментированные исходные изображения из тестовой выборки.

Создаём нейронную сеть.

```
inputs = Input([644, 796, 3])

conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same')(inputs)
conv1 = Conv2D(32, 3, activation = 'relu', padding = 'same')(conv1)
pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)

conv2 = Conv2D(64, 5, activation = 'relu', padding = 'same')(pool1)
conv2 = Conv2D(64, 5, activation = 'relu', padding = 'same')(conv2)
pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)

conv3 = Conv2D(128, 7, activation = 'relu', padding = 'same')(pool2)
conv3 = Conv2D(128, 7, activation = 'relu', padding = 'same')(conv3)

up1 = UpSampling2D(size = (2,2))(conv3)
merge1 = concatenate([conv2, up1], axis = 3)
conv4 = Conv2D(64, 5, activation = 'relu', padding = 'same')(merge1)
conv4 = Conv2D(64, 5, activation = 'relu', padding = 'same')(conv4)

up2 = UpSampling2D(size = (2,2))(conv4)
merge1 = concatenate([conv1, up2], axis = 3)
conv5 = Conv2D(32, 3, activation = 'relu', padding = 'same')(merge1)
conv5 = Conv2D(32, 3, activation = 'relu', padding = 'same')(conv5)

conv6 = Conv2D(6, 3, activation = 'softmax', padding = 'same')(conv5)

model = Model(inputs = inputs, outputs = conv6)
```

Обучаем нейронную сеть.

```
model.compile(optimizer = Adam(), loss = 'categorical_crossentropy')
model.fit(X_train_pred, Y_train_pred, batch_size=1, epochs=25)
```

Выполним оценку качества обучения.

```
score = model.evaluate(X_test_pred, Y_test_pred,verbose=0)
```

```
1 # evaluate() - метод оценки модели на тестовых данных.
2 score = model.evaluate(X_test_pred, Y_test_pred,verbose=0)   # тестируем НС
3 score
```

```
0.5094856023788452
```

Рисунок 3 – Качество нейронной сети

Проверим работу нейронной сети на тестовой выборке.

```
y_pred = model.predict(X_test_pred, batch_size=1)
```

Выведем результат сегментации.

```
I = 1
plt.imshow(np.array(X_test)[I])
plt.show()
plt.imshow(np.array(Y_test)[I])
plt.show()
plt.imshow(convert_to_color(np.argmax(y_pred[I], axis=-1)))
plt.show()
```

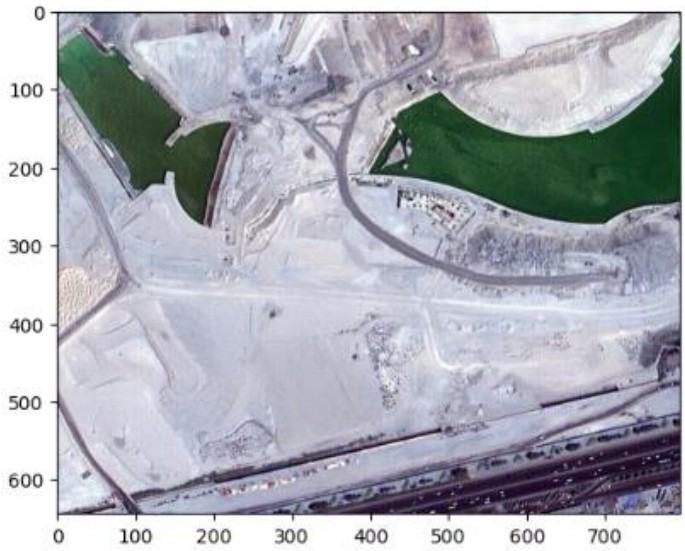

Рисунок 4 – Исходное изображение

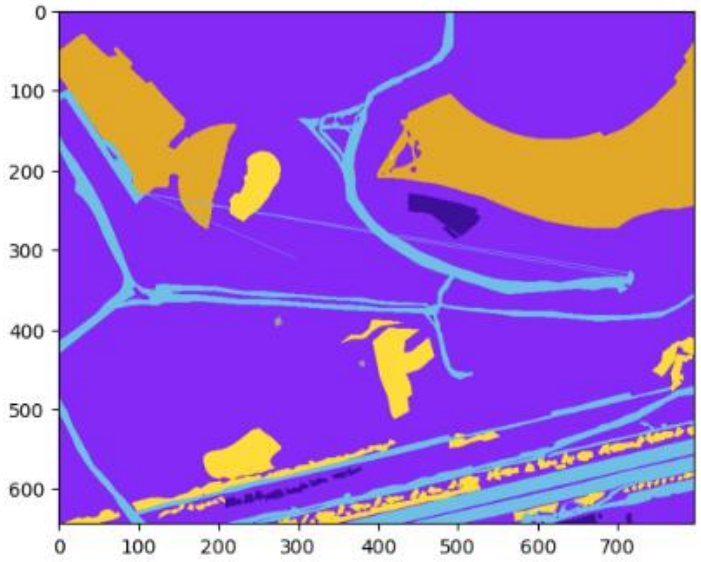

Рисунок 5 – Сегментированное изображение

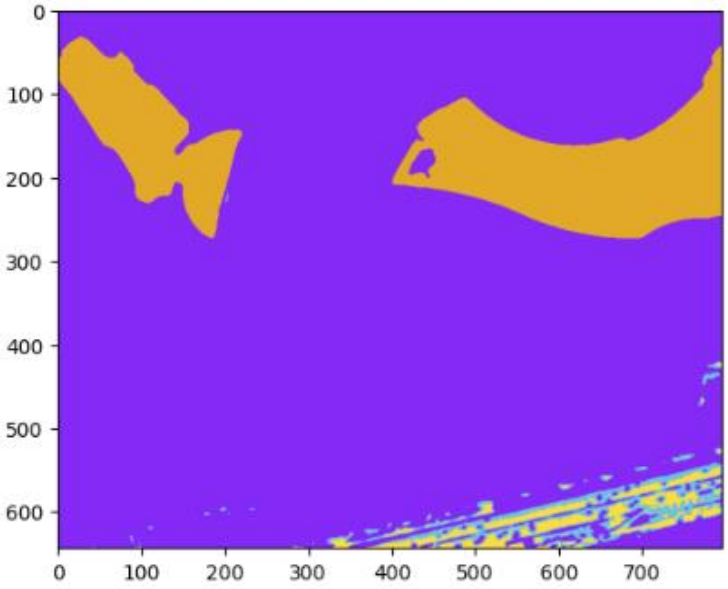

Рисунок 6 – Изображение, сегментированное нейронной сетью

4. Заключение

В результате выполнения данной лабораторной работы был получен навыки сегментации снимков дистанционного зондирования земли на основе модели нейронной сети U-Net.