Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный Исследовательский Университет ИТМО»

ЛАБОРАТОРНАЯ РАБОТА №4
ПО ПРЕДМЕТУ «ТЕХНИЧЕСКОЕ ЗРЕНИЕ»
ПО ТЕМЕ «СЕГМЕНТАЦИЯ ИЗОБРАЖЕНИЙ»

Преподаватель:
Шаветов С. В.

Выполнили:
Румянцев А. А.
Овчинников П. А.
Чебаненко Д. А.

Факультет: СУиР
Группа: R3241
Поток: ТЕХ.ЗРЕНИЕ 2.1

Санкт-Петербург
2024

Содержание

1 Начало

1.1 Цель

Цель выполнения данной работы заключается в освоении основных способов сегментации изображений на семантические области.

1.2 Оригинальная картинка

Для всех бинаризаций была выбрана следующая картинка:

Рис. 1: Оригинальная картинка

1.3 Язык программирования и необходимая библиотека

Работа выполнена на языке программирования Python с использованием библиотеки OpenCV. Следующий код позволяет считать картинку в переменную и после сохранить изображение в заданную папку:

```
import cv2

path = 'tech-vision/source/pic.png'
render_dir = 'tech-vision/renders/'

I = cv2.imread(path)
cv2.imwrite(f'{render_dir}pic.png', I)
```

Листинг 1: Пример использования библиотеки OpenCV на python

Для выполнения бинаризации нам необходимо черно-белое изображение, которое мы можем получить при считывании используя параметр cv2.IMREAD_GRAYSCALE.

```
    import cv2


    I_gray = cv2.imread("source/pic.png", cv2.IMREAD_GRAYSCALE)
    cv2.imwrite(f'{render_dir}pic1.png', I_gray)
```

Листинг 2: Пример считывания черно-белонго изображения

Рис. 2: Черно-белая картинка

2 Бинаризация изображений

2.1 Немного о бинаризации

Простейшим способом сегментации изображения на два класса (фоновые пиксели и пиксели объекта) является бинаризация. Бинаризацию можно выполнить по порогу или по двойному порогу. В первом

случае:

$$I_{new} = \begin{cases} 0, I(x,y) \leqslant t, \\ 1, I(x,y) > t, \end{cases}$$

А во втором:

$$I_{new}(x,y) = \begin{cases} 0, I(x,y) \leqslant t_1, \\ 1, t_1 < I(x,y) \leqslant t_2, \\ 0, I(x,y) > t_2, \end{cases}$$

2.2 Бинаризация по порогу

В библиотеке OpenCV бинаризация данным методом может быть выполнена с использованием функции threshold()

```
    def binarization(I, t):
        ret, Inew = cv2.threshold(I, t, 255, cv2.THRESH_BINARY)
        return Inew


    I_bin = binarization(I_gray, 127)
    cv2.imwrite(f'{render_dir}pic2.png', I_bin)
```

Листинг 3: Код для бинаризации изображения по порогу

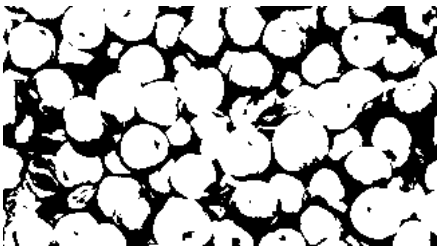

Рис. 3: Бинаризация изображения по порогу

2.3 Бинаризация по двойному порогу

```
    import numpy as np
    import cv2


    def double_thr_bin(I, t1, t2):
        ret, Inew = cv2.threshold(I, t2, 255, cv2.THRESH_TOZERO_INV)
        ret, Inew = cv2.threshold(Inew, t1, 255, cv2.THRESH_BINARY)
        return Inew


    I_db = double_thr_bin(I_gray, 127, 200)
    cv2.imwrite(f'{render_dir}pic3.png', I_db)
```

Листинг 4: Код для бинаризации изображения по двойному порогу

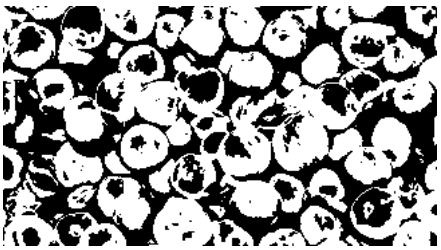

Рис. 4: Бинаризация по двойному порогу

2.4 Метод Отсу для вычисления параметра t

В библиотеке OpenCV порог t методом Отсу может быть вычислен с использованием функции cv2.threshold() с параметром cv2.THRESH_OTSU.

```
    def otsu(I):
        ret, Inew = cv2.threshold(I_gray, 0, 255, cv2.THRESH_OTSU)
        return Inew


    I_otsu = otsu(I_gray)
    cv2.imwrite(f'{render_dir}pic4.png', I_otsu)
```

Листинг 5: Код для вычисления параметра t методом Отсу

Рис. 5: Изображение при параметре t вычисленном методом Отсу

2.5 Адаптивный метод вычисления параметра t

Адаптивные методы, работающие не со всем изображением, а лишь с его фрагментами. Такие подходы зачастую используются при работе с изображениями, на которых представлены неоднородно освещенные объекты. В библиотеке OpenCV порог t адаптивным методом может быть вычислен при помощи функции cv2.adaptiveThreshold()

```
    def adaptive_bin(I):
        Inew = cv2.adaptiveThreshold(I, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY
            , 11, 2)
        return Inew

    I_ad = adaptive_bin(I_gray)
    cv2.imwrite(f'{render_dir}pic5.png', I_ad)
```

Листинг 6: Код для вычисления t адаптивным методом

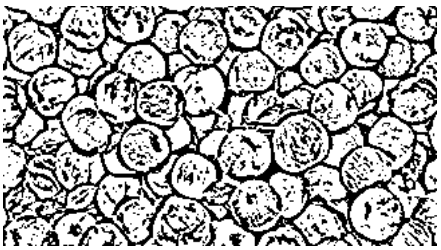

Рис. 6: Изображение при параметре t вычисленном адаптивным методом Отсу

3 Сегментация изображения по цвету кожи

Будем сегментировать величайшего!

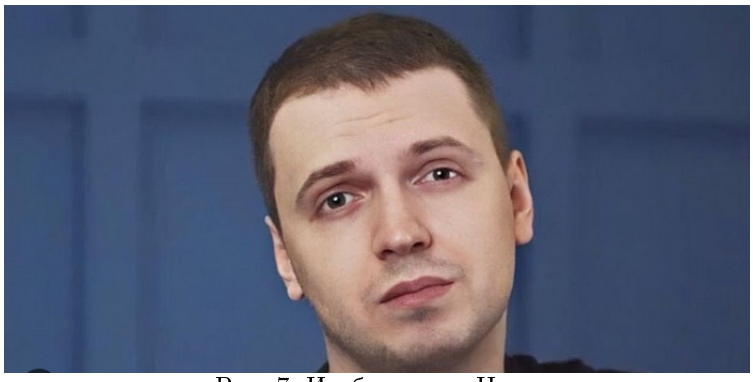

Рис. 7: Изображение Царя

Общим приницпом данного подхода является определение критерия близости интенсивности пикселей к оттенку кожи. Аналитически описать оттенок кожи довольно затруднительно, поскольку его описание базируется на человеческом восприятии цвета, меняется при изменении освещения, отличается у разных народностей, и т.д. Существует несколько аналитических описаний для изображений в цветовом пространстве RGB, позволяющих отнести пиксель к классу «кожа» при выполнении условий:

$$\begin{cases} R > 95, \\ G > 40, \\ B < 20, \\ maxR, G, B - minR, G, B > 15, \\ |R - G| > 15, \\ R > G, \\ R > B, \end{cases}$$

Данная интерпретация работает лучше всего при обычном дневном освещении, поэтому буду рассматривать ее.

```
    def skin_seg(img):
        mask = np.logical_and.reduce([img[:, :, 0] > 20, img[:, :, 1] > 40, img[:, :, 2] > 95,
                                      img[:, :, 2] > img[:, :, 1],
                                      img[:, :, 2] > img[:, :, 0],
                                      np.abs(img[:, :, 2] - img[:, :, 1]) > 15,
                                      np.max(img, axis=2) - np.min(img, axis=2) > 15])

        new_img = np.zeros_like(mask, dtype=np.uint8)
        new_img[mask] = 255
        return new_img

    skin = skin_seg(skin_img)
    cv2.imwrite(f'{render_dir}pic7.png', skin)
```

Листинг 7: Код для сегментации по цвету кожи.

Рис. 8: Изображение Царя после сегментации

4 Сегментация изображения на основе цветового пространства CIE Lab по методу k-средних

Идея метода заключается в определении центров k-кластеров и отнесении к каждому кластеру пикселей, наиболее близко относящихся к этим центрам. Все пиксели рассматриваются как векторы $x_i, i = \overline{1,p}$. Алгоритм сегментации состоит из следующих шагов:

1. Определение случайным образом k векторов $m_j, j = \overline{1,k}$, ко- торые объявляются начальными центрами кластеров.
2. Обновление значений средних векторов m_j путем вычисления расстояний от каждого вектора x_i до каждого m_j и их классификации по критерию минимальности расстояния от вектора до кластера, пересчет средних значений m_j по всем кластерам.
3. Повторение шагов 2 и 3 до тех пора, пока центры кластеров не перестанут изменяться.

```
    def k_means(img):
        Ilab = cv2.cvtColor(img, cv2.COLOR_BGR2Lab)
        Ilab = cv2.split(Ilab)
        ab = cv2.merge([Ilab[1], Ilab[2]])
        ab = ab.reshape(-1, 2).astype(np.float32)
        k = 2
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 10, 1.0)
        ret, labels, centers = cv2.kmeans(ab, k, None, criteria, 10,
                                          cv2.KMEANS_RANDOM_CENTERS)
        labels = labels.reshape(Ilab[0].shape)
        segmentedFrames = []
        for i in range(k):
            Itmp = np.zeros_like(img)
            mask = labels == i
            Itmp[mask] = img[mask, :]
            segmentedFrames.append(Itmp)
        return segmentedFrames


    ans = k_means(I)
    cv2.imwrite(f'{render_dir}pic8.png', ans[0])
    cv2.imwrite(f'{render_dir}pic9.png', ans[1])
```

Листинг 8: Код для сегментации методом k-means.

Рис. 9: Исходное изображение

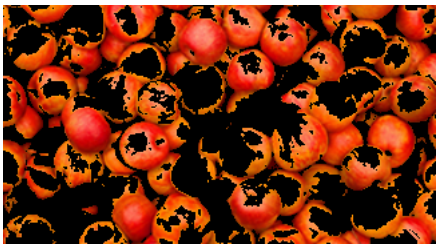

Рис. 10: Красные области

Рис. 11: Желтые области

5 Текстурная сегментация

При текстурной сегментации для описания текстуры применяются три основных метода: статистический, структурный и спек- тральный. В лабораторной работе будем рассматривать статистический подход, который описывает текстуру сегмента как гладкую, грубую или зернистую.

Рис. 12: Исходное полутоновое изображение

```
    def texture_seg(I):
        E = skimage.filters.rank.entropy(I, skimage.morphology.square(9)).astype(np.float32)
        Eim = (E - E.min()) / (E.max() - E.min())
        ret, BW1 = cv2.threshold(np.uint8(Eim * 255), 0, 255, cv2.THRESH_OTSU)
        Bwao = bwareaopen(BW1, 2000)
        nhood = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
        closeBwao = cv2.morphologyEx(Bwao, cv2.MORPH_CLOSE, nhood)
        Mask1 = imfillholes(closeBwao)
        contours, h = cv2.findContours(Mask1, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        boundary = np.zeros_like(Mask1)

        cv2.drawContours(boundary, contours, -1, 255, 1)
        segmentResults = I.copy()
        segmentResults[boundary != 0] = 255

        I2 = I.copy()
        I2[Mask1 != 0] = 0
        E2 = skimage.filters.rank.entropy(I2, skimage.morphology.square(9)).astype(np.float32)
        Eim2 = (E2 - E2.min()) / (E2.max() - E2.min())
        ret, BW2 = cv2.threshold(np.uint8(Eim2 * 255), 0, 255, cv2.THRESH_OTSU)
        BW2ao = bwareaopen(BW2, 2000)
```

```
        nhood = cv2.getStructuringElement(cv2.MORPH_RECT, (9, 9))
        closeBW2ao = cv2.morphologyEx(BW2ao, cv2.MORPH_CLOSE, nhood)
        Mask2 = imfillholes(closeBW2ao)
        contours2, h = cv2.findContours(Mask2, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
        boundary2 = np.zeros_like(Mask2)

        cv2.drawContours(boundary2, contours2, -1, 255, 1)
        segmentResults2 = I2.copy()
        segmentResults2[boundary2 != 0] = 255

        texture1 = I.copy()
        texture1[Mask2 == 0] = 0
        texture2 = I.copy()
        texture2[Mask2 != 0] = 0
        return texture1, texture2


    arr = texture_seg(city_img)
    cv2.imwrite(f'{render_dir}pic11.png', arr[0])
    cv2.imwrite(f'{render_dir}pic12.png', arr[1])
```

Листинг 9: Код для текстурной сегментации

Рис. 13: Вода

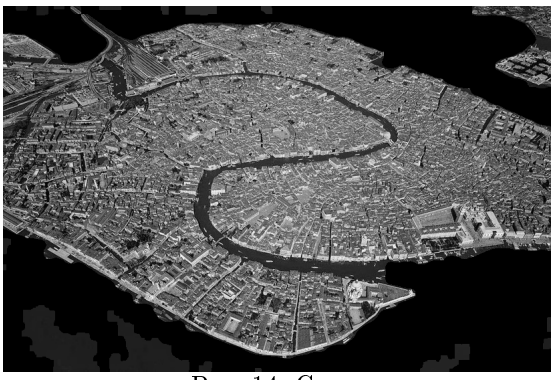

Рис. 14: Суша

	m	s	U
Вода	36.19	8.02	0.31
Суша	103.08	50.66	0.003

Вычислив параметры, мы можем сделать вывод, что изображение Воды гладкое, а суши грубое.

6 Ответы на вопросы

6.1 В каких случаях целесообразно использовать сегментацию по принципу Вебера?

Для полутоновых изображений целесообразно использовать сегментацию по принципу Вербера.

6.2 Какие значения имеют цветовые координаты a и b цветового пространства CIE Lab в полутновом изображении?

a (означает положение цветм в диапазоне от зеленого (-128) до красного (127)) и b (означает положение цвета в диапазоне от синего (-128) до желтого (127)) Мы можем расчитать это по следующей формуле:

6.3 Зачем производить сегментацию в цветовом пространстве CIE Lab, а не в исходном RGB?

Для того чтобы уменьшить влияние освещенности на результат сегментации.

6.4 Что такое цветовое пространство и цветовой охват?

Цветовое пространство – это модель, по которой цвет представляется в формате точки с конкретными координатами.
Цветовой охват - это математическое описание диапазона оттенков, которые способен отобразить дисплей.

7 Выводы

7.1 Лабораторная работа 4

В ходе проделанной работы мы познакомились с различными методами сегментации, начиная бинаризацией и заканчивая текстурной сегментацией. Мы расширили свои познания о работе с библиотекой OpenCV и языком программирования python, а также поразмыслили над интересными вопросами и дали на них ответы