

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Московский государственный технический университет имени Н.Э.
Баумана»
(МГТУ им. Н.Э. Баумана)

Факультет ИУ – «Информатика и управление»

Кафедра ИУ-3 – «Информационные системы и телекоммуникации»

Отчёт по лабораторной работе 2 по дисциплине

«Цифровая обработка изображений»

Вариант 4

Выполнил: Цветков А. А.
Группа: ИУЗ-48М
Проверил: Алфимцев А.Н.

Москва, 2017

Разработка программы распознавания блюд бауманской столовой

По заданию лабораторной работы 2 необходимо произвести классификацию блюд на цифровых изображениях с использованием преобразований в цветовом пространстве (в соответствии с вариантом – цветовое пространство YUV).

YUV – цветовая модель, в которой цвет представляется как 3 компоненты – яркость (Y) и две цветоразностных (U и V).

Y - яркостная компонента, если оставить только её получим изображение в оттенках серого, компонента получается из исходного RGB сигнала, каждая составляющая множится на свой вес (сумма весов - 1)

U - разностная компонента для голубого цвета ($B' - Y'$)

V - разностная компонента для красного цвета ($R' - Y'$)

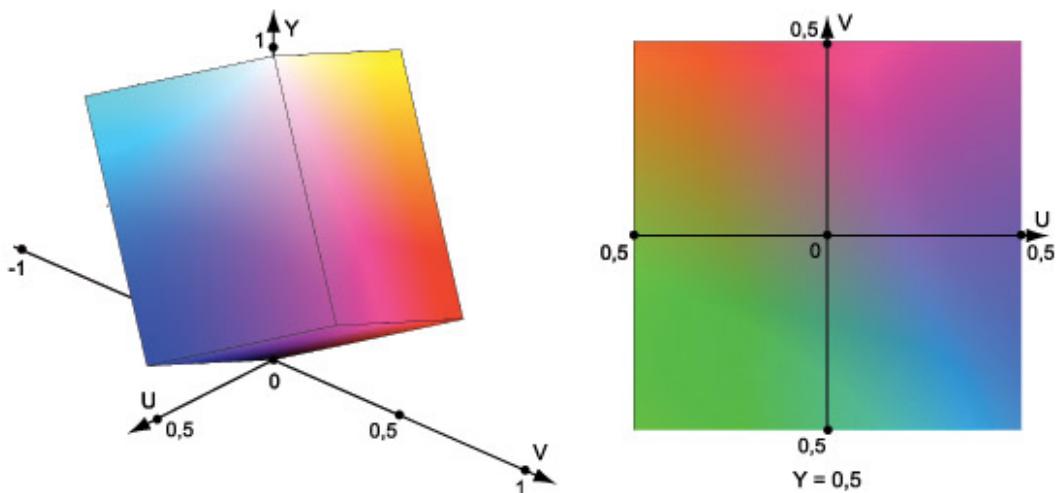


Рисунок 1 – RGB-куб в пространстве YUV, диаграмма UV при $Y = 0,5$

Данная цветовая модель получила особое распространение в областях передачи цифрового сигнала (телевидение) с небольшими изменениями в формулах отображении на соответственные цветовые пространства.

Цветовые модели YCbCr и YPbPr являются вариациями YUV с другими весами для U и V (им соответствуют Cb/Pb и Cr/Pr). YPbPr применяется для описания аналоговых сигналов (преимущественно в телевидении), а YCbCr - для цифровых. Для их определения используются два коэффициента: Kb и Kr.

Однако в задачах обработки изображений данное пространство практически не используется. Исключением являются задачи корректировки

яркости в фотографиях, т. к. для манипуляции параметрами яркости достаточно модифицировать только составляющую Y.

На данный момент нет широко распространённых практик использованиях YUV для выделения областей одинакового цвета, таких как RGB-маски. Более того необходимо добавить, что все современные графические редакторы используют именно RGB-маски.

В рамках данной работы принято решение использовать RGB модель для выделения каналов. Однако стоит заметить, что отображения RGB-to-YUV и YUV-to-RGB приводят к потере и искажению информации о цвете.

Для перевода используются формулы, изображенные на рисунке 2.

$$R = Y + 1,13983 \times (V - 128);$$

$$G = Y - 0,39465 \times (U - 128) - 0,58060 \times (V - 128);$$

$$B = Y + 2,03211 \times (U - 128);$$

$$Y = 0,299 \times R + 0,587 \times G + 0,114 \times B;$$

$$U = -0,14713 \times R - 0,28886 \times G + 0,436 \times B + 128;$$

$$V = 0,615 \times R - 0,51499 \times G - 0,10001 \times B + 128$$

Рисунок 2 – Формулы конверсии в RGB и обратно.

Для классификации блюд необходимо определить верхние и нижние границы трех компонент Y, U, V в цветовом пространстве YUV для каждого составляющего в блюде. Например, морковь имеет следующие характеристики: $81 < Y < 140$, $88 < U < 100$, $174 < V < 178$. Затем, необходимо определить верхнюю и нижнюю границы по площадям в пикселях каждого составляющего в 4люде. Например, для картофельного пюре это $290000 < S < 300000$. Значения соответствующих компонент YUV определялись с помощью встроенного в MAC OS X ПО Digital Color Meter.

Исходное изображение изображено на рисунке 3.



Рисунок 3 – Исходное изображение

Алгоритм классификации блюд состоит из следующих этапов:

1. Перевод из цветового пространства YUV в RGB.
2. Преобразования в цветовом пространстве + заполнение замкнутых областей;
3. Если площадь обнаруженного объекта не входит в указанные границы, применение фильтра расширения (dilation) + заполнение замкнутых областей;
4. Если площадь обнаруженного объекта не входит в указанные границы, применение фильтра замыкания (closing) + заполнение замкнутых областей;
5. Распознавание объекта, отображение его названия и площасти в пикселях.

В качестве примера рассмотрим применение фильтров расширения и замыкания на примере поиска моркови (рисунок 4).



Рисунок 4 – Исходное изображение для поиска моркови

Результат преобразований в цветовом пространстве с последующим заполнением замкнутых областей отображен на рисунке 5.



Рисунок 5 – Результат преобразований в цветовом пространстве с последующим заполнением замкнутых областей (поиск моркови)

В данном примере для обнаружения объекта с необходимой площадью необходимо использовать фильтры расширения и замыкания (рисунки 6, 7).

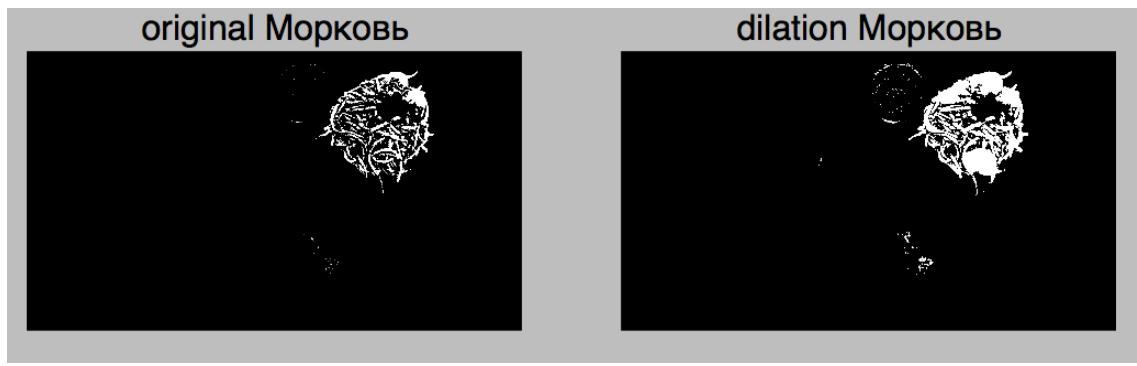


Рисунок 6 – Применение фильтра расширения

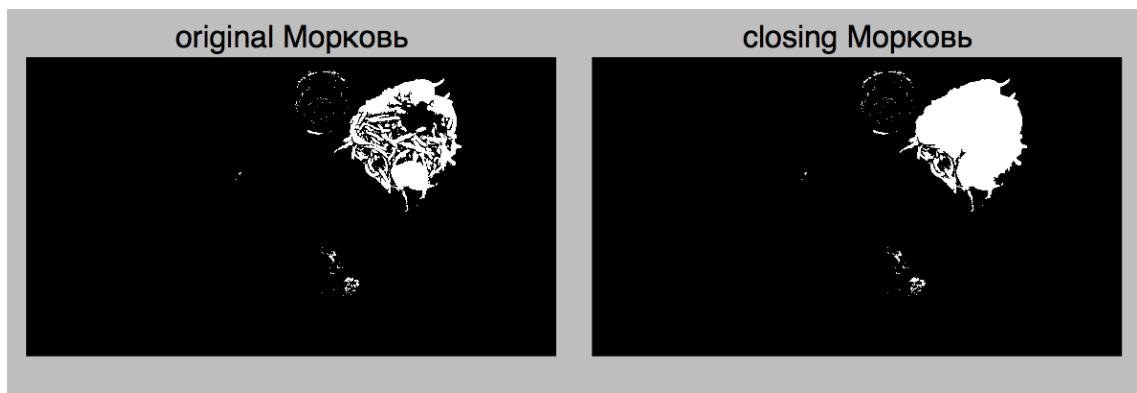


Рисунок 7 – Применение фильтра замыкания

Пример работы программы

На рисунках 8 – 13 отображены результаты работы алгоритма в задаче поиска черного хлеба, а так в задаче поиска всех возможных блюд.

Food detection

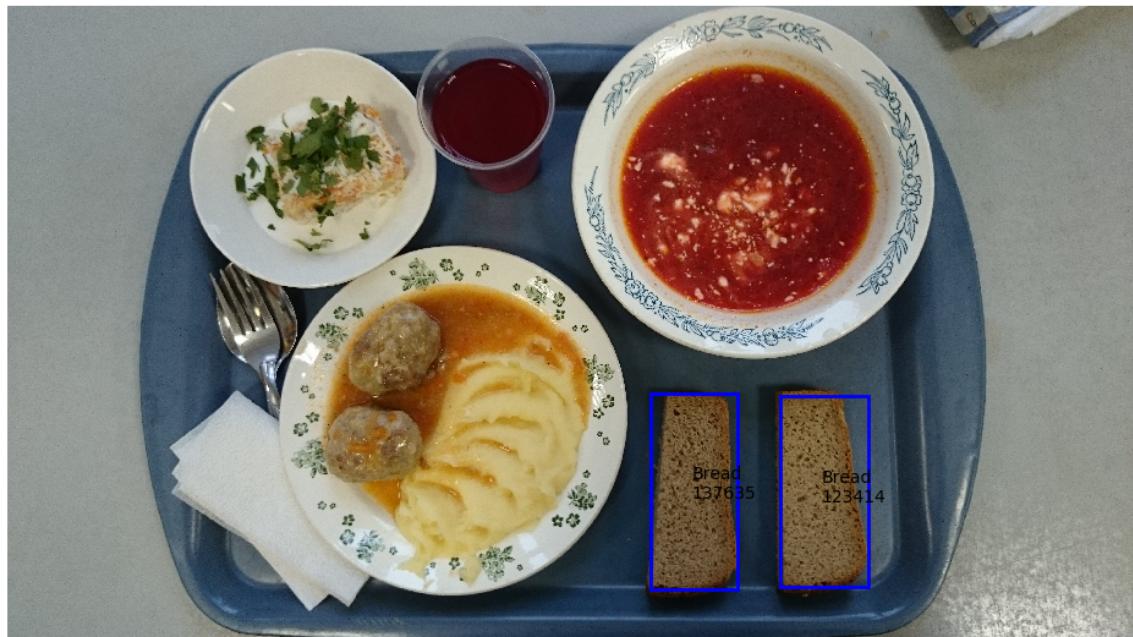


Рисунок 8 – Пример работы

Food detection



Рисунок 9 – Пример работы

Food detection



Рисунок 10 – Пример работы

Food detection



Рисунок 11 – Пример работы

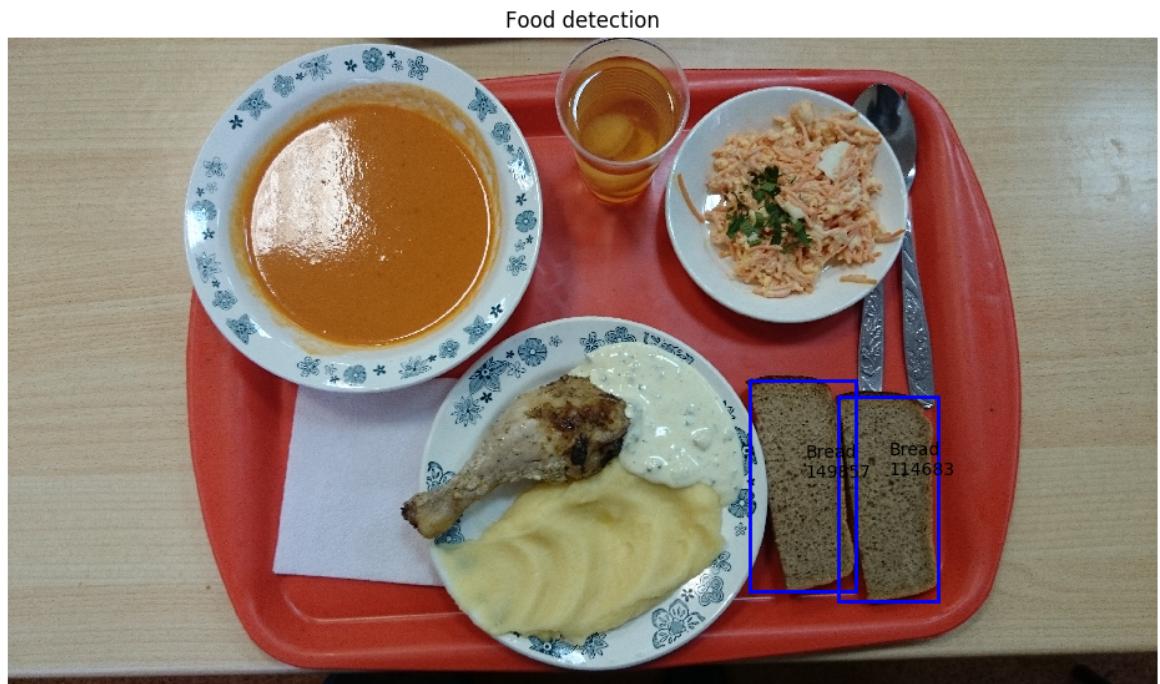


Рисунок 12 – Пример работы

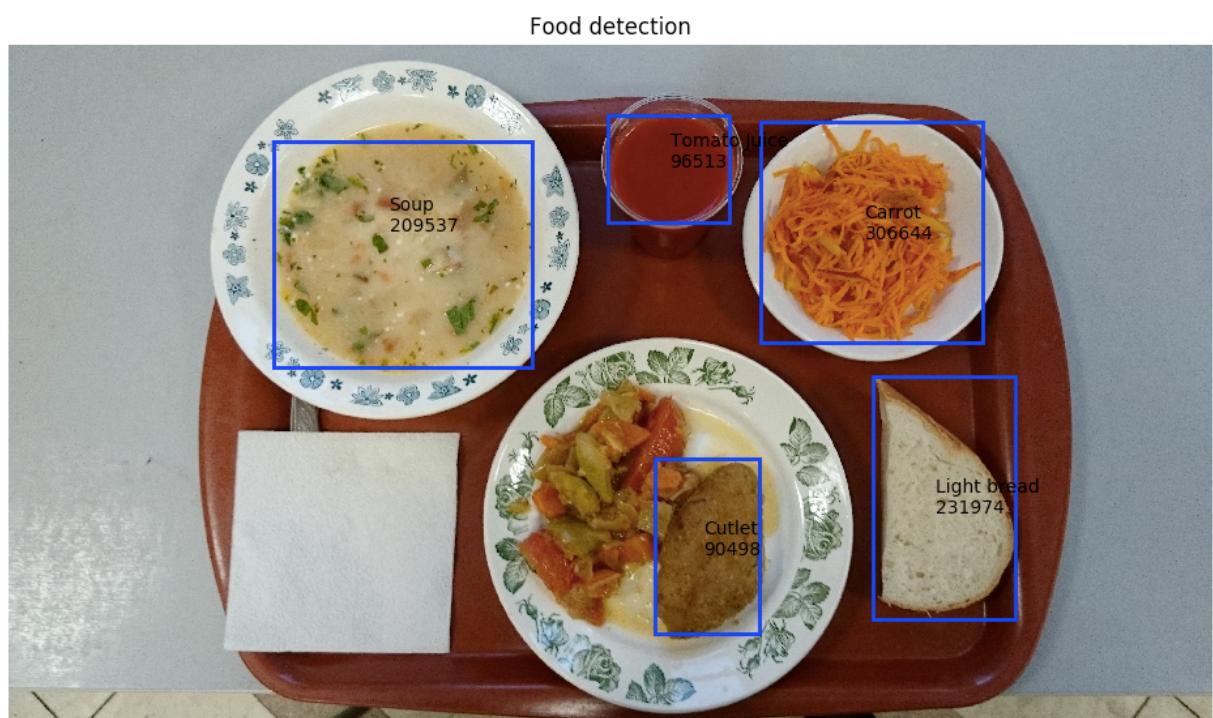


Рисунок 13 – Пример работы

Полученные результаты отображены в таблице 1.

Таблица 1 – Результаты распознавания блюд

Номер изображения	Объекты для возможного распознавания	Распознано правильно	Распознано неправильно (ошибка 2 рода)	Не распознано (ошибка 1 рода)
1	7	5	4	1
2	5	8	1	3
3	5	2	1	2
4	6	4	1	1
5	6	4	4	1
6	6	1	5	2
7	7	1	8	1
8	6	1	0	0
9	6	2	5	0
10	6	2	5	3

Из-за потери информации при конвертации из YUV в RGB существенно повышается количество ошибок 1 и 2 рода. Также большое влияние оказывает похожий цвет разных блюд, например, белый хлеб и пюре.

В результате выполнения лабораторной работы можно сделать вывод, что для распознавания блюд на основе цвета использование пространства YUV не рекомендуется в связи с большой вероятностью ошибок 1-го и 2-го родов.

Листинг программы

```
import os

import matplotlib.patches as mpatches
from matplotlib import pyplot as plt
from scipy import ndimage as ndi
from skimage import io, color
from skimage.measure import label, regionprops
from skimage.morphology import dilation, closing, square

basedir = os.path.abspath(os.path.dirname(__file__))

INPUT_DIR = os.path.join(basedir, 'input')
OUTPUT_DIR = os.path.join(basedir, 'output')

"""
RGB-YCbCR
"""

def yuv_to_rgb(y, u, v):
    def clamp(n, smallest, largest): return max(smallest, min(n, largest))

    return [
        clamp(int(1.164 * (y - 16) + 1.596 * (v - 128)), 0, 255),
        clamp(int(1.164 * (y - 16) - 0.813 * (v - 128) - 0.391 * (u - 128)), 0, 255),
        clamp(int(1.164 * (y - 16) + 2.018 * (u - 128)), 0, 255)
    ]

def rgb_to_yuv(r, r_max, g, g_max, b, b_max):
    return [
        (0.257 * r) + (0.504 * g) + (0.098 * b) + 16,
        (0.257 * r_max) + (0.504 * g_max) + (0.098 * b_max) + 16,
        -(0.148 * r) - (0.291 * g) + (0.439 * b) + 128,
        -(0.148 * r_max) - (0.291 * g_max) + (0.439 * b_max) + 128,
        (0.439 * r) - (0.368 * g) - (0.071 * b) + 128,
        (0.439 * r_max) - (0.368 * g_max) - (0.071 * b_max) + 128
    ]

# check the area of interest zone
def check_area(image, min_area, max_area):
    label_image = label(image)
    flag = False
    for region in regionprops(label_image):

        # take regions with large enough areas
        if min_area <= region.area <= max_area:
            flag = True
            break

    return flag

# search food
def find_food(image, y_min, y_max, u_min, u_max, v_min, v_max, name, min_area, max_area):
    print("Search: " + name)

    r_min, g_min, b_min = yuv_to_rgb(y_min, u_min, v_min)
    r_max, g_max, b_max = yuv_to_rgb(y_max, u_max, v_max)

    # color channels
    r = image[:, :, 0]
    g = image[:, :, 1]
    b = image[:, :, 2]

    # mask for color channels
```

```

mask_r = (r > r_min) & (r < r_max)
mask_g = (g > g_min) & (g < g_max)
mask_b = (b > b_min) & (b < b_max)

# summary mask
mask = mask_r & mask_g & mask_b

# convert from rgb to gray
mask1 = color.rgb2gray(mask)

# fill holes
mask1 = ndi.binary_fill_holes(mask1)

# kernel for morphology
s = 10
selem = square(s)

# dilation filter
if not check_area(mask1, min_area, max_area):
    print("dilation " + str(s))
    mask2 = dilation(mask1, selem)
    mask2 = ndi.binary_fill_holes(mask2)
else:
    mask2 = mask1

# closing filter
if not check_area(mask2, min_area, max_area):
    print("closing")
    mask3 = closing(mask2, selem)
    mask3 = ndi.binary_fill_holes(mask3)
else:
    mask3 = mask2

# segment an image with image labelling
label_image = label(mask3)

for region in regionprops(label_image):

    # take regions with large enough areas
    if min_area <= region.area <= max_area:
        # draw rectangle around segmented objects
        minr, minc, maxr, maxc = region.bbox
        rect = mpatches.Rectangle((minc, minr), maxc - minc, maxr - minr, fill=False, edgecolor='blue', linewidth=2)
        ax.add_patch(rect)

        # find center points in segmented objects
        y0, x0 = region.centroid

        # draw the name of food and its area
        ax.text(x0, y0, name + '\n' + str(region.area), fontsize=10, color='black')
        ax.set_title(u'Food detection')

if __name__ == '__main__':
    # open the input image in RGB Color space
    input_image_rgb = io.imread(os.path.join(INPUT_DIR, 'Мено (51).JPG'))

    fig, ax = plt.subplots(figsize=(10, 6))
    ax.imshow(input_image_rgb)

    # advanced
    find_food(input_image_rgb, 81.71, 140.675, 100.03, 88.315, 174.03, 178.0, 'Carrot', 290000, 400000)
    find_food(input_image_rgb, 72.8, 116.19, 112.5850, 114.805, 142.945, 136.36, 'Light bread', 180000, 250000)
    find_food(input_image_rgb, 102.935, 138.53, 100.155, 99.44, 142.88, 138.845, 'Soup', 200000, 800000)
    find_food(input_image_rgb, 44.122, 73.678, 116.752, 108.125, 161.364, 161.099, 'Tomato Juice', 96000, 120000)
    find_food(input_image_rgb, 56, 85, 110, 107, 152, 145, 'Cutlet', 50000, 100000)
    find_food(input_image_rgb, 140.938, 173.227, 94.586, 104.698, 145.856, 140.485, 'Puree', 180000, 270000)
    find_food(input_image_rgb, 37.98, 44.05, 122.77, 124.225, 155.825, 157.665, 'Juice 2', 40000, 200000)
    find_food(input_image_rgb, 21.742, 29.73, 125.188, 125.04, 136.341, 136.78, 'Juice 2', 40000, 200000)
    find_food(input_image_rgb, 84.435, 103.645, 93.495, 89.845, 162.635, 161.15, 'Juice 3', 40000, 200000)

```

```

find_food(input_image_rgb, 34.164, 59.867, 119.017, 117.567, 143.565, 140.479, 'Apple juice', 50000, 100000)
find_food(input_image_rgb, 102.081, 117.861, 109.073, 107.899, 143.513, 144.533, 'Bread', 40000, 150000)

# set axis off
ax.set_axis_off()
plt.tight_layout()
plt.show()

"""
Uncomment if u want to process multiple image
"""

# i = 1
# for img in io.imread_collection(os.path.join(INPUT_DIR, '*.JPG')):
#     fig, ax = plt.subplots(figsize=(10, 6))
#     ax.imshow(img)
#     ax.set_axis_off()
#     plt.tight_layout()
#
#     # find_food(img, 81.71, 140.675, 100.03, 88.315, 174.03, 178.0, 'Carrot', 300000, 315000)
#     # find_food(img, 72.8, 116.19, 112.5850, 114.805, 142.945, 136.36, 'Light bread', 180000, 250000)
#     # find_food(img, 72.8, 116.19, 112.5850, 114.805, 142.945, 136.36, 'Bread', 90000, 155000)
#     # find_food(img, 102.935, 138.53, 100.155, 99.44, 142.88, 138.845, 'Soup', 200000, 800000)
#     # find_food(img, 44.122, 73.678, 116.752, 108.125, 161.364, 161.099, 'Tomato Juice', 96000, 120000)
#     # find_food(img, 56, 85, 110, 107, 152, 145, 'Cutlet', 50000, 100000)
#     # find_food(img, 140.938, 173.227, 94.586, 104.698, 145.856, 140.485, 'Puree', 180000, 270000)
#     # find_food(img, 37.98, 44.05, 122.77, 124.225, 155.825, 157.665, 'Juice 2', 40000, 200000)
#     # find_food(img, 21.742, 29.73, 125.188, 125.04, 136.341, 136.78, 'Juice 2', 40000, 200000)
#     # find_food(img, 84.435, 103.645, 93.495, 89.845, 162.635, 161.15, 'Juice 3', 40000, 200000)
#     # find_food(img, 34.164, 59.867, 119.017, 117.567, 143.565, 140.479, 'Apple juice', 50000, 100000)
#     # find_food(img, 102.081, 117.861, 109.073, 107.899, 143.513, 144.533, 'Bread', 40000, 150000)
#
#     # plt.savefig(os.path.join(OUTPUT_DIR, str(i) + '-result.png'))
#     # i += 1

```