Практическая работа 3

Материалы для выполнения практической работы могут быть получены с помощью команды:

```
$ wget https://cv-course.demo.3divi.com/download/task3/practical
```

Задания рекомендуется выполнять последовательно в среде Google-Colab или Jupyter

❖ В данной практической работе будут рассмотрены основные функции, необходимые для выполнения домашнего задания. Начнем с установки всех необходимых библиотек:

- opencv для работы с изображениями
- matplotlib - для визуализаций
- numpy - для работы с многомерными массивами

Код:

```
$ pip3 install numpy
$ pip3 install opencv-contrib-python
$ pip3 install matplotlib
```

Сделаем импорт всех необходимых библиотек

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
```

Импорт библиотек должен быть выполнен без ошибок

❖ Рассмотрим пример с вычислением гистограммы изображения. Для демонстрации вычисления гистограммы понадобится изображение. Убедитесь, что IMAGE_PATH содержит путь до существующего изображения.

Код:

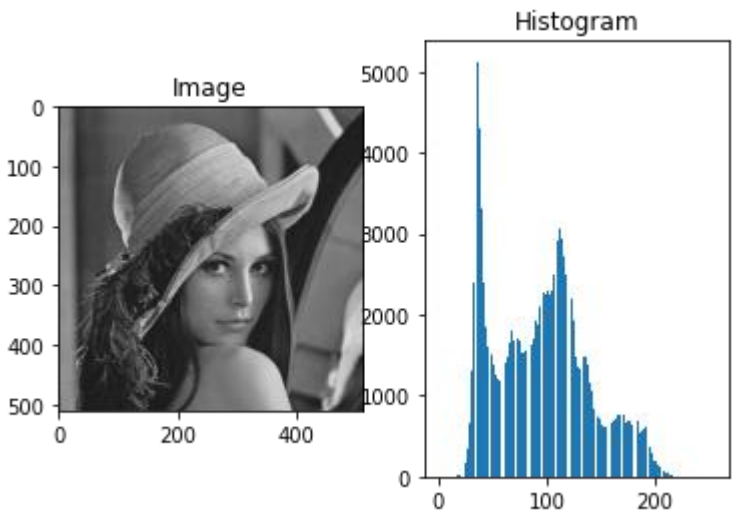
```
IMAGE_PATH = "lenna.jpg"
assert os.path.exists(IMAGE_PATH)
```

Вычислим и отрисуем гистограмму для черно-белого изображения.

Код:

```
image = cv2.imread(IMAGE_PATH)
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
histogram = cv2.calcHist([gray_image], [0], None, [256], [0, 256])
histogram = np.squeeze(histogram)
histogram_x_range = list(range(len(histogram)))

# plot
fig, ax = plt.subplots(1, 2)
ax[0].imshow(gray_image, cmap="gray")
ax[0].set_title("Image")
ax[1].bar(x=histogram_x_range, height=histogram)
ax[1].set_title("Histogram")
```

Результат:



Для самостоятельного изучения:

–изучить функцию np.squeeze;

–изменить количество бинов гистограммы.

❖ Рассмотрим решение histigram backrpojection для сегментации объекта. Для этого понадобятся:

- тестовое изображение, на котором будем искать объект;
- содержащее объект изображение, которое будет использоваться для подсчета распределения цветов объекта (будем называть его обучающее изображение).

Убедитесь, что необходимые изображения существуют.

Код:

```
TARGET_IMAGE_PATH = "full_image.jpg"
TRAIN_IMAGE_PATH = "train_crop.jpg"
assert os.path.exists(TARGET_IMAGE_PATH)
assert os.path.exists(TRAIN_IMAGE_PATH)
```

Сегментация будет происходить на основе первых двух каналов hsv пространства, но нам также понадобится rgb изображения для визуализации, введем вспомогательные функции для считывания изображений.

Код:

```
def read_rgb_from_file(file: str) -> np.ndarray:
    """Reads image and converts it to rgb"""
    image = cv2.imread(file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    return image

def read_hsv_from_file(file: str) -> np.ndarray:
    """Reads image and converts it to hsv"""
    image = cv2.imread(file)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    return image
```

Решим задачу histogram backprojection.

Код:

```
train_hsv = read_hsv_from_file(TRAIN_IMAGE_PATH)
train_image_h, train_image_w = train_hsv.shape[:2]
train_image_area = train_image_h * train_image_w
full_image_hsv = read_hsv_from_file(TARGET_IMAGE_PATH)
train_imagehist = cv2.calcHist([train_hsv], [0, 1], None, [180, 256], [0,
180, 0, 256])

backprojection_map = cv2.calcBackProject(
    [full_image_hsv], [0, 1], train_imagehist, [0, 180, 0, 256], scale=1
)
# Переведем эти значения в вероятности
backprojection_map = backprojection_map.astype(np.float32)
backprojection_map /= train_image_area
```

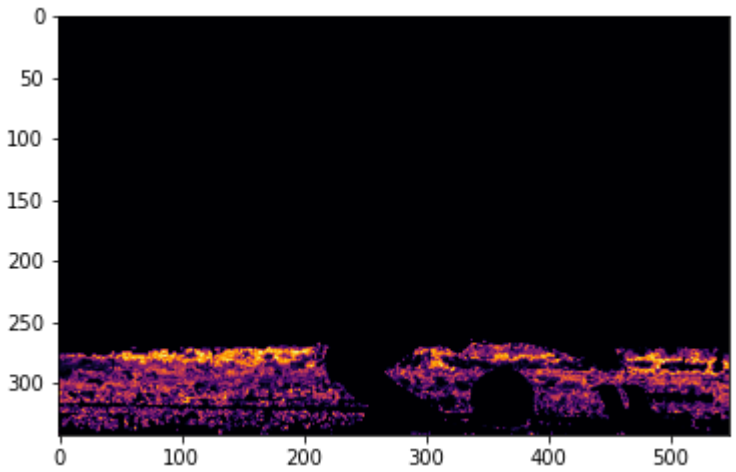
После обратного преобразования гистограммы переменная backprojection_map отражает количество пикселей целевого изображения с такими же значениями hue и saturation, как у обучающего изображения. Далее эти значения переводятся в вероятности.

Визуализируем получившиеся вероятности.

Код:

```
plt.imshow(backprojection_map, cmap="inferno")
```

Результат:



Отрисуем оригинальное изображение и получившуюся маску. Следует обратить внимание на значение переменной threshold и проверить, как изменится маска при различных ее значениях.

Код:

```
threshold = 0.001
full_image_rgb = read_rgb_from_file(TARGET_IMAGE_PATH)
mask = backprojection_map > threshold
masked_image = np.copy(full_image_rgb)
masked_image[mask == 0] = [0, 0, 0]

fig, (image1_axes, image2_axes) = plt.subplots(1, 2, figsize=(100, 50))
image1_axes.imshow(full_image_rgb)
image2_axes.imshow(masked_image)
```

Результат:

❖ Рассмотрим пример сглаживания маски с помощью билатерального фильтра, сначала зададим все необходимые параметры для его применения:

- neighborhood_diameter - диаметр окрестности вокруг пикселя, которая будет использоваться при применении фильтра
- sigmaColor - параметр контролирующий насколько далекие в цветовом пространстве пиксели могут оказывать друг на друга влияние при применении фильтра.
- sigmaSpace - параметр контролирующий насколько далекие (в терминах евклидова расстояния на изображении) пиксели могут оказывать друг на друга влияние при применении фильтра.

Код:

```
neighborhood_diameter = 5
sigmaColor = 0.35
sigmaSpace = 100
```

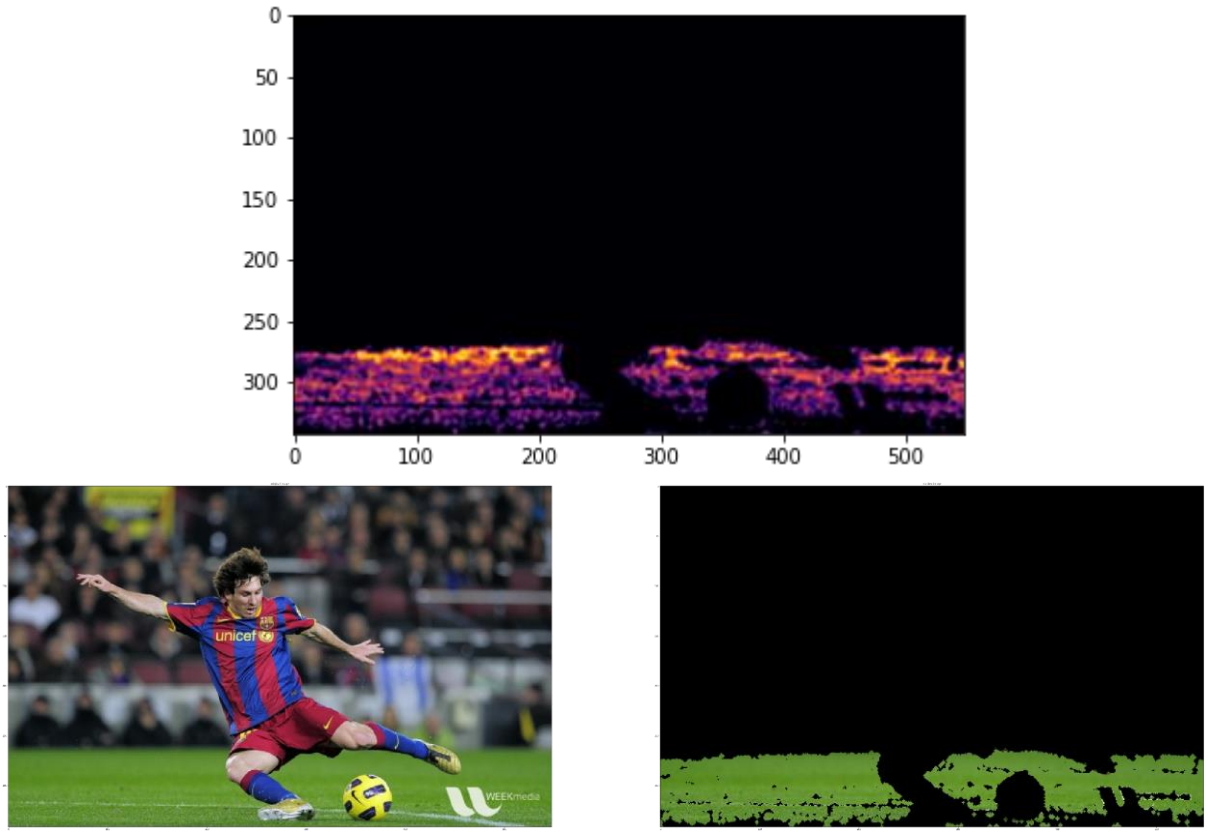
Теперь применим фильтр с выбранными параметрами и визуализируем результат.

Код:

```
threshold = 0.001
mask = cv2.bilateralFilter(backprojection_map, neighborhood_diameter,
sigmaColor, sigmaSpace)

plt.imshow(mask, cmap="inferno")
```

```
plt.show()

mask = mask > threshold
masked_image = np.copy(full_image_rgb)
masked_image[mask == 0] = [0, 0, 0]

fig, (image1_axes, image2_axes) = plt.subplots(1, 2, figsize=(100, 50))
image1_axes.imshow(full_image_rgb)
image2_axes.imshow(masked_image)
```

Результат:



❖ Далее рассмотрим сегментацию с использованием суперпикселей, будем использовать алгоритм SLIC. Ниже приведены основные константы, определяющие работу алгоритма: желаемый размер блока из суперпикселей, количество итераций, размер суперпикселей, которые будут объединены с другими на этапе постобработки.

Код:

```
block_size_pixels = 20
iter_num = 10
enforce_size_percents = 10
```

Получим сегментацию по суперпикселям.

Код:

```
image = cv2.imread(IMAGE_PATH)
# Для суперпиксельной сегментации переведем изображение в пространство LAB
image_LAB = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
slic_obj = cv2.ximgproc.createSuperpixelSLIC(
    image=image_LAB,
    algorithm=cv2.ximgproc.SLIC,
    region_size=block_size_pixels,
)
slic_obj.iterate(iter_num)
slic_obj.enforceLabelConnectivity(enforce_size_percents)
```

Теперь у нас есть объект, содержащий суперпиксельную сегментацию, доступ к ней происходит с помощью метода getLabels. В результате получим массив, в котором каждому пикселю соотвествует число, обозначающее номер суперпикселя, к которому он принадлежит. Число суперпикселей можно получить, используя метод getNumberOfSuperpixels.

Код:

```
label_map = slic_obj.getLabels()
superpixels_num = slic_obj.getNumberOfSuperpixels()
print(label_map)
print(superpixels_num)
```

Результат:

```
[[  0   0   0 ...  27  27  27]
 [  0   0   0 ...  27  27  27]
 [  0   0   0 ...  27  27  27]
 ...
 [779 779 779 ... 818 818 818]
 [779 779 779 ... 818 818 818]
 [779 779 779 ... 818 818 818]]
820
```

❖ В качестве примера работы с суперпиксельной сегментацией решим следующую задачу: заменим все пиксели средним цветом соответствующего суперпикселя.

Код:

```
superpixel_processed_image = np.zeros(image_LAB.shape, dtype=image.dtype)
image_RGB = read_rgb_from_file(IMAGE_PATH)

for superpixel_num in range(superpixels_num):
  superpixel_mask = label_map == superpixel_num
  original_image_values = image_RGB[superpixel_mask]
  mean_superpxixel_color = np.mean(original_image_values,
axis=0).astype(image.dtype)
  superpixel_processed_image[superpixel_mask] = mean_superpxixel_color

fig, (image1_axes, image2_axes) = plt.subplots(1, 2, figsize=(100, 50))
image1_axes.imshow(image_RGB)
image2_axes.imshow(superpixel_processed_image)
```

Результат:

Для самостоятельного изучения:

- *np.squeeze*
- *np.expand_dims*