OpenCV

OpenCV

Open-source Computer Vision Library

Библиотека компьютерного зрения с открытым исходным кодом — это open source библиотека компьютерного зрения, которая предназначена **для анализа, классификации и обработки изображений**.

Широко используется в таких языках как C, C++, Python и Java.

- Панорамы улиц в картах Google (+ другие продукты)
- Система зрения робота PR2 компании Willow Garage
- Роботы для исследования поверхности Марса (проект NASA)
- Контроль качества монет (Центробанк Китая)

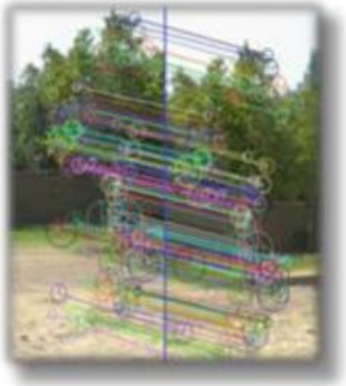
Функциональность

Обработка изображений

Фильтрация

Трансформации

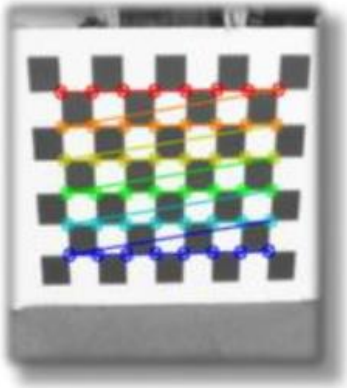
Ребра, контурный анализ



Сопоставление с помощью особых точек

Сегментация

Видео, Стерео, 3D



Калибрация камер

Вычисление положения в пространстве

Оптический поток

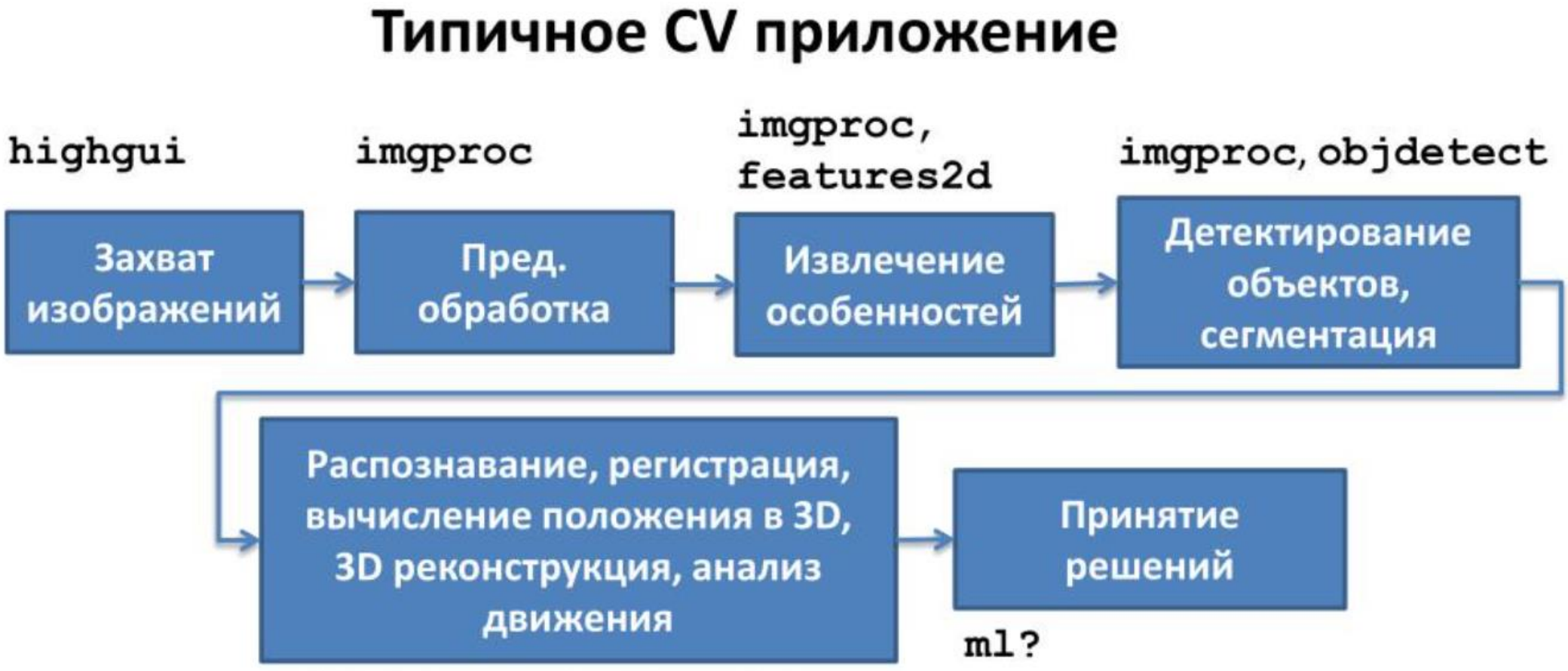
Построение карты глубины

Нахождение объектов

OpenCV в приложениях

OpenCV – базовая, в основном низкоуровневая библиотека.

Предоставляет строительные блоки, кирпичики для приложений.



Загрузка изображения

RGB — cv2.IMREAD_COLOR,
Оттенки серого — cv2.IMREAD_GRAYSCALE

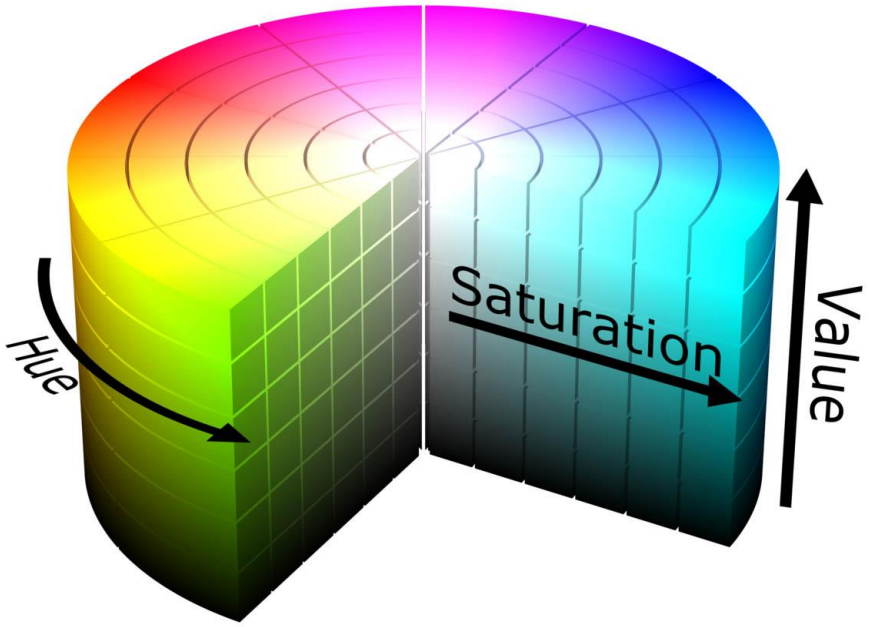
```
import cv2


def loading_displaying_saving():

  img = cv2.imread('girl.jpg', cv2.IMREAD_GRAYSCALE)

  cv2.imshow('girl', img)

  cv2.waitKey(0)

  cv2.imwrite('graygirl.jpg', img)
```

Цветовое пространство - BGR

```
(b, g, r) = img[0, 0]
print("Красный: {}, Зелёный: {}, Синий: {}".format(r, g, b))
nemo = cv2.cvtColor(nemo, cv2.COLOR_BGR2RGB)
```

```
print("Высота:"+str(img.shape[0]))
print("Ширина:" + str(img.shape[1]))
print("Количество каналов:" + str(img.shape[2]))
```

Цветовая схема HSV



HSV — это **Hue, Saturation и Value (оттенок, насыщенность и яркость).**

Это цилиндрическое цветовое пространство. Цвета, или оттенки, меняются при движении пс кругу цилиндра.

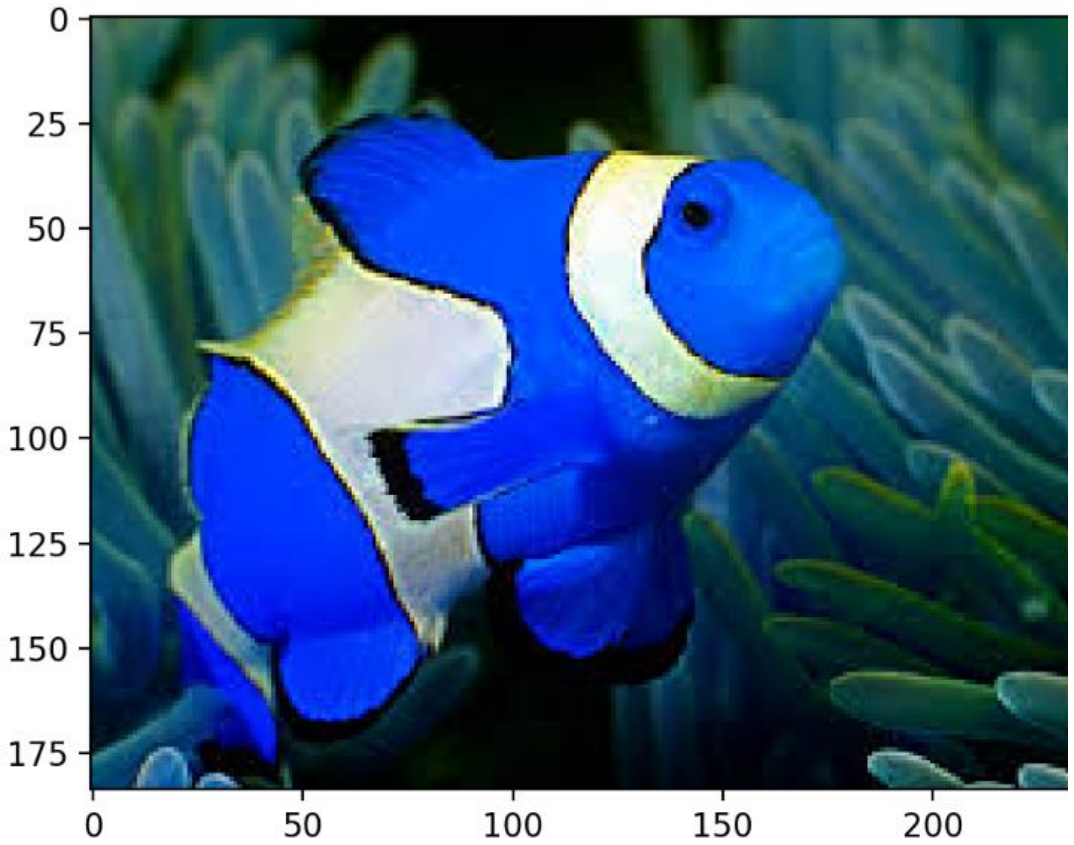
Вертикальная ось отвечает за яркость: от темного (0 в нижней части) до светлого сверху.

Третья ось, насыщенность, определяет тени оттенков при движении от центра к краю вдоль радиуса цилиндра (от менее к более насыщенному)

```
hsv_nemo = cv2.cvtColor(nemo, cv2.COLOR_RGB2HSV)
```

Пример

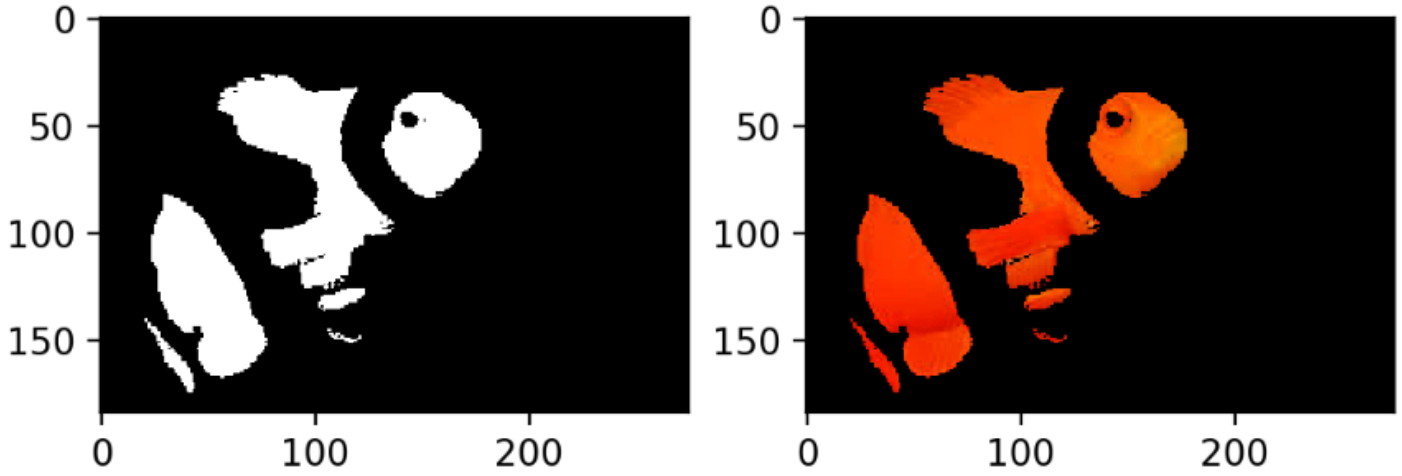
```
nemo = cv2.imread('./images/nemo0.jpg')
plt.imshow(nemo)
plt.show()
```



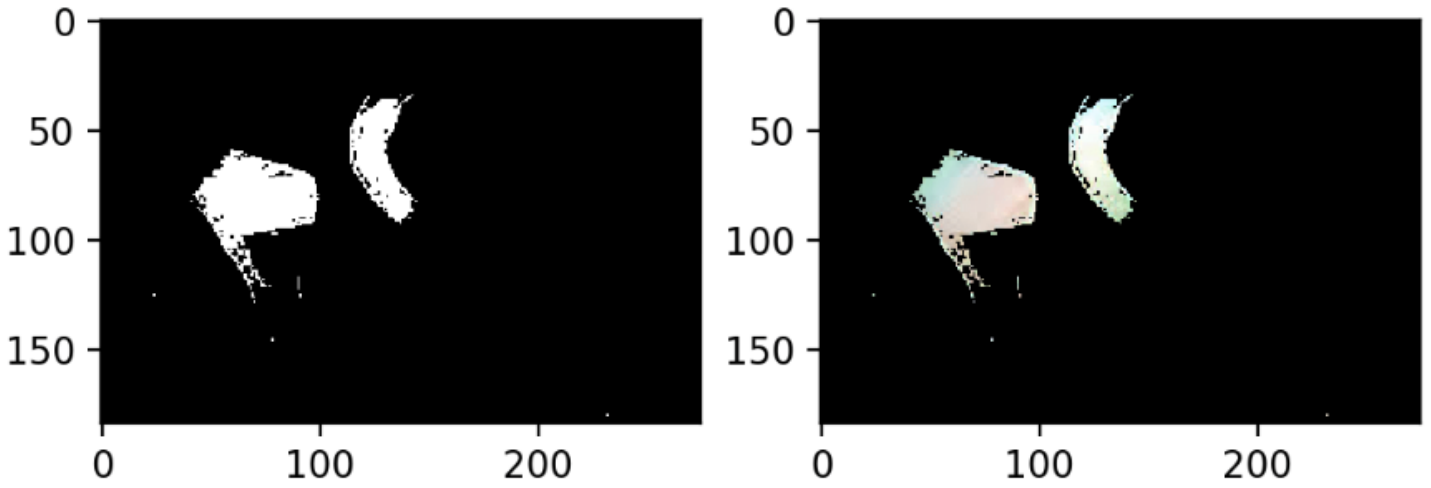

```
nemo = cv2.cvtColor(nemo, cv2.COLOR_BGR2RGB)
plt.imshow(nemo)
plt.show()
```

Сегментация по цвету

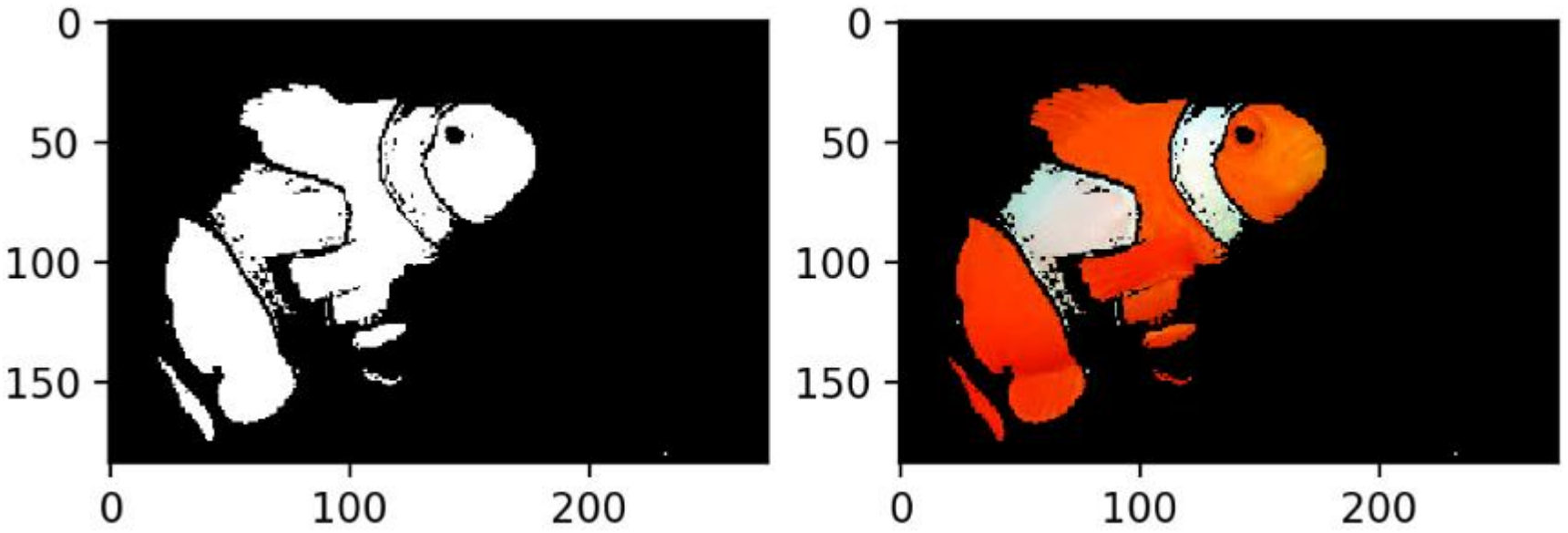
```
light_orange = (1, 190, 200)
dark_orange = (18, 255, 255)
#бинарная маска
mask = cv2.inRange(hsv_nemo, light_orange, dark_orange)
result = cv2.bitwise_and(nemo, nemo, mask=mask)
plt.subplot(1, 2, 1)
plt.imshow(mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
```



```
light_white = (0, 0, 200)
dark_white = (145, 60, 255)
mask_white = cv2.inRange(hsv_nemo, light_white, dark_white)
result_white = cv2.bitwise_and(nemo, nemo, mask=mask_white)
plt.subplot(1, 2, 1)
plt.imshow(mask_white, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result_white)
plt.show()
```

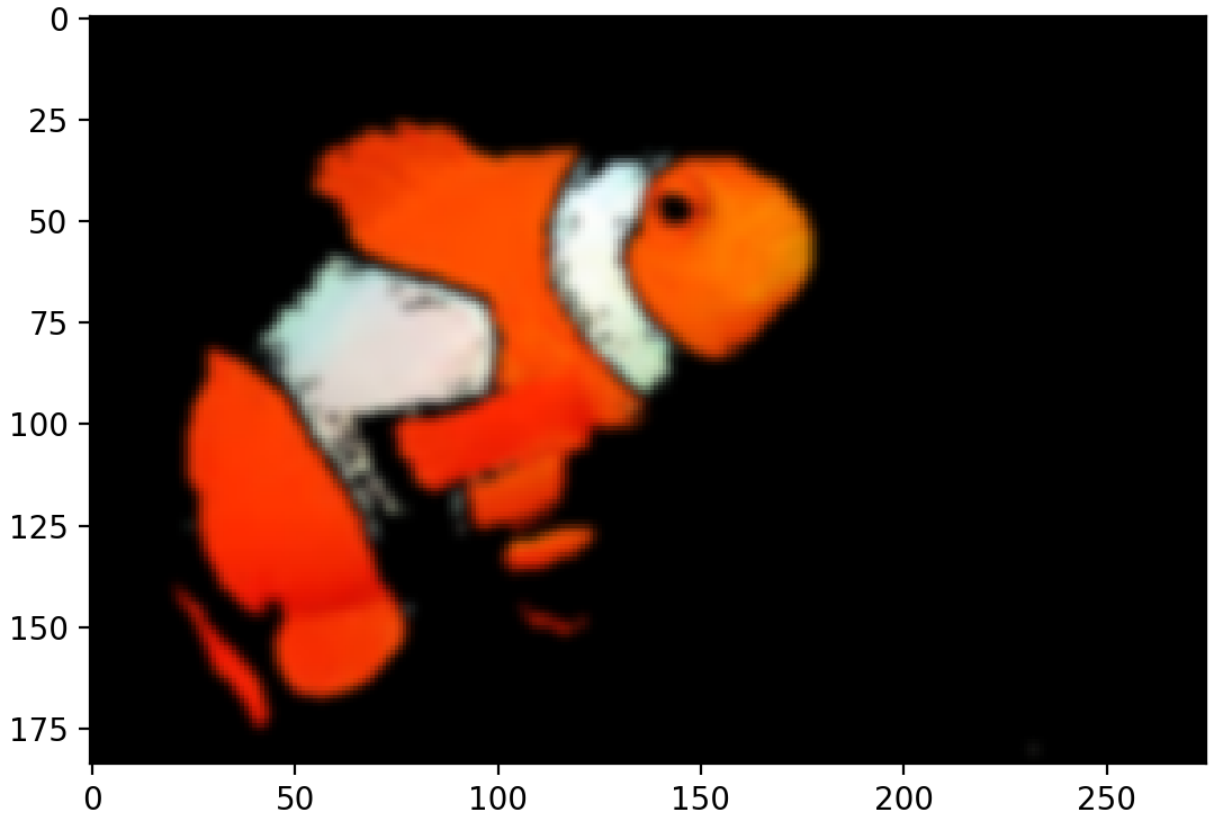


```
final_mask = mask + mask_white
final_result = cv2.bitwise_and(nemo, nemo, mask=final_mask)
plt.subplot(1, 2, 1)
plt.imshow(final_mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(final_result)
plt.show()
```



Гауссовское размытие

```
blur = cv2.GaussianBlur(final_result, (7, 7), 0)
plt.imshow(blur)
plt.show()
```



Фильтры

```
1.  import cv2
2.
3.  # Загрузка первоначального изображения
4.  original_image = cv2.imread('my_bike.png')
5.
6.  # Фильтрация изображения усредняющим фильтром 3X3
7.  average_image = cv2.blur(original_image,(3,3))
8.
9.  # Применение фильтра Гаусса к первоначальному изображению
10. gaussian_image = cv2.GaussianBlur((original_image,(3,3),0))
11.
12. # Применение медианного фильтра к первоначальному изображению
13. median_image = cv2.medianBlur(original_image,3)
```

Кадрирование

```python
import cv2
cropped = image[10:500, 500:2000]
viewImage(cropped, "Пёсик после кадрирования")
```

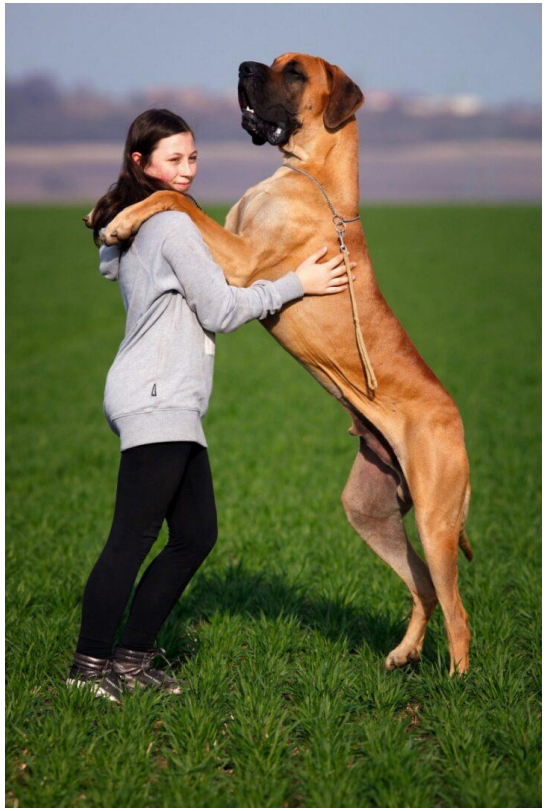
```python
def viewImage(image, name_of_window):
    cv2.namedWindow(name_of_window, cv2.WINDOW_NORMAL)
    cv2.imshow(name_of_window, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

Где `image[10:500, 500:2000]` — это `image[y:y + высота, x:x + ширина]`.

Изменение размера

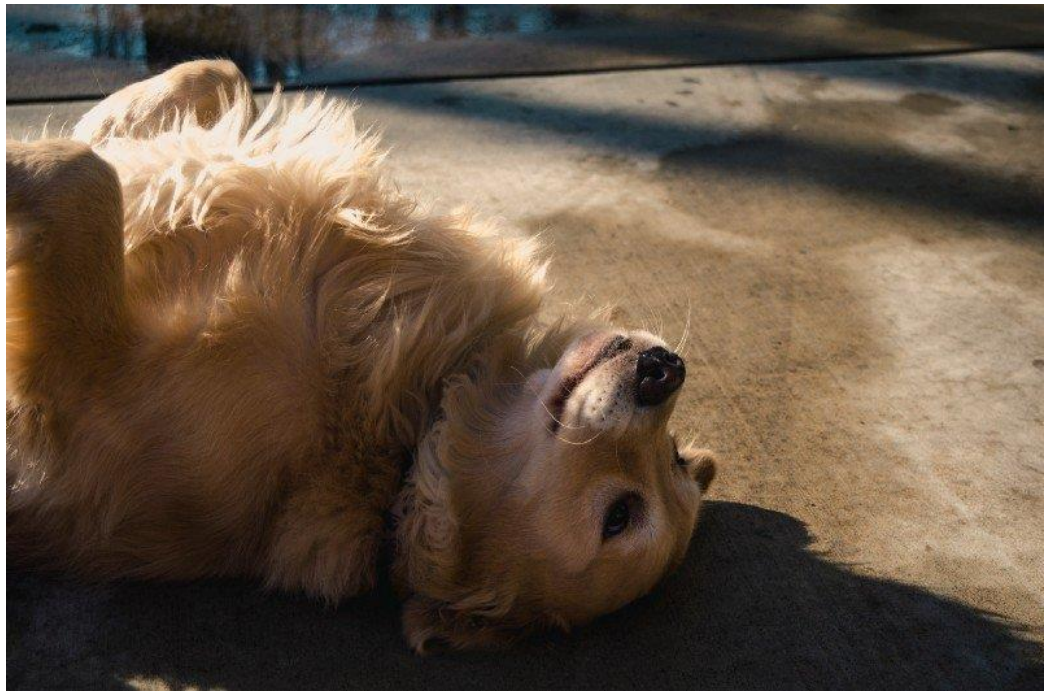
```python
def viewImage(image, name_of_window):
    cv2.namedWindow(name_of_window, cv2.WINDOW_NORMAL)
    cv2.imshow(name_of_window, image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```python
scale_percent = 20 # Процент от изначального размера
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)
viewImage(resized, "После изменения размера на 20 %")
```



Поворот

```
(h, w, d) = image.shape
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, 180, 1.0)
rotated = cv2.warpAffine(image, M, (w, h))
viewImage(rotated, "Пёсик после поворота на 180 градусов")
```



Полутоновое и черно-белое изображение

```
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
ret, threshold_image = cv2.threshold(im, 127, 255, 0)
viewImage(gray_image, "Пёсик в градациях серого")
viewImage(threshold_image, "Чёрно-белый пёсик")
```

```
ret, threshold = cv2.threshold(im, 150, 200, 10)
```

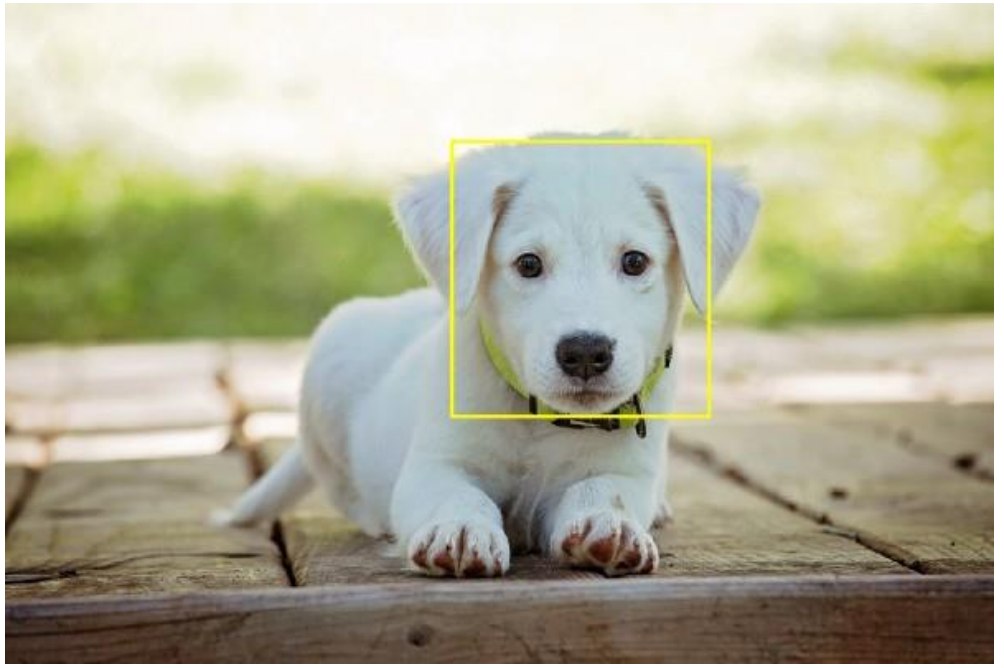
Здесь всё, что темнее, чем 150, заменяется на 10, а всё, что ярче, — на 200.

Размытие

```
blurred = cv2.GaussianBlur(image, (51, 51), 0)
viewImage(blurred, "Размытый пёсик")
```

Прямоугольник

```
output = image.copy()
cv2.rectangle(output, (2600, 800), (4100, 2400), (0, 255, 255), 10)
viewImage(output, "Обводим прямоугольником лицо пёсика")
```



```
cv2.line(output, (60, 20), (400, 200), (0, 0, 255), 5)
```

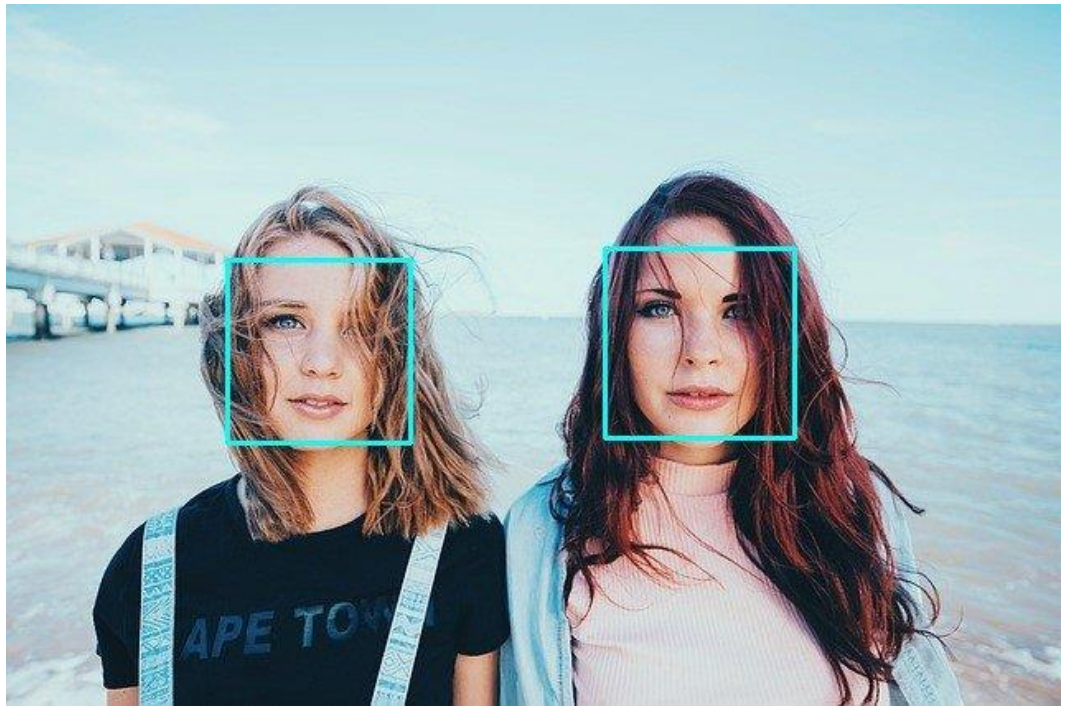
Текст

```
output = image.copy()
cv2.putText(output, "We <3 Dogs", (1500, 3600),cv2.FONT_HERSHEY_SIMPLEX,
                                                                    15, (30, 105, 210), 40)
```

Распознавание лиц

Классический каскадный классификатор на основе признаков-фильтров (метод Виолы-Джонса)

Классификатор - предварительно обученные модели по умолчанию для определения лица, глаз и рта.



```
cascPath = "/usr/local/lib/python3.7/site-
packages/cv2/data/haarcascade_frontalface_default.xml"
eyePath = "/usr/local/lib/python3.7/site-packages/cv2/data/haarcascade_eye.xml"
smilePath = "/usr/local/lib/python3.7/site-packages/cv2/data/haarcascade_smile.xml"
```

```
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
```

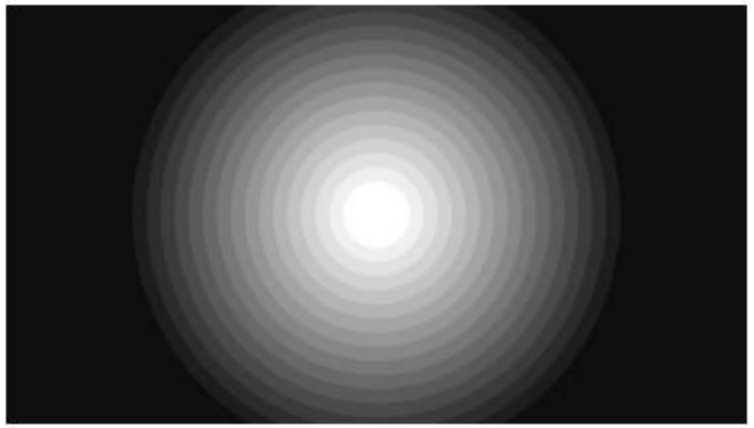
```python
import cv2
image_path = "./путь/к/фото.расширение"
face_cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml'
image = cv2.imread(image_path)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(
    gray,
    scaleFactor= 1.1,
    minNeighbors= 5,
    minSize=(10, 10)
)
faces_detected = "Лиц обнаружено: " + format(len(faces))
print(faces_detected)
# Рисуем квадраты вокруг лиц
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 255, 0), 2)
viewImage(image,faces_detected)
```

Параметр scaleFactor. Некоторые лица могут быть больше других, поскольку находятся ближе, чем остальные. Этот параметр компенсирует перспективу.
Параметр minNeighbors - Слишком маленькое значение увеличит количество ложных срабатываний, а слишком большое сделает алгоритм более требовательным (скользящее окно).
Параметр minSize – размер области

Распознавание на основе цветовой сегментации

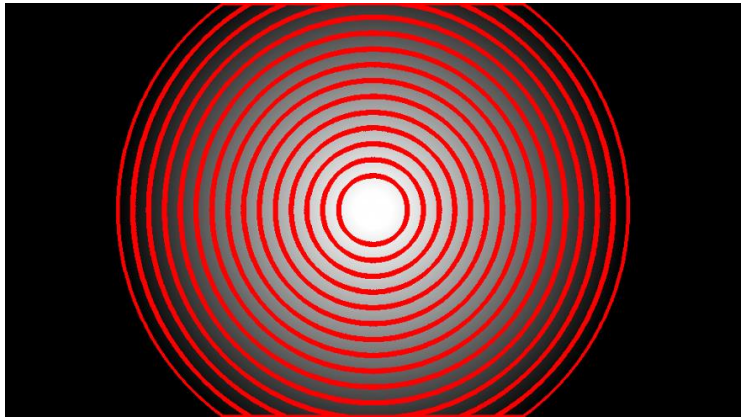
Контуром называется кривая, которая объединяет все непрерывные точки (по границе) одного цвета или интенсивности. Контуры являются весьма полезными инструментами для анализа форм, обнаружения и распознавания объектов.

Пороговая обработка полутонового изображения (в оттенках серого) превращает его в бинарное. Задается некое пороговое значение, и все значения ниже порога становятся черными, а выше — белыми.



17 уровней

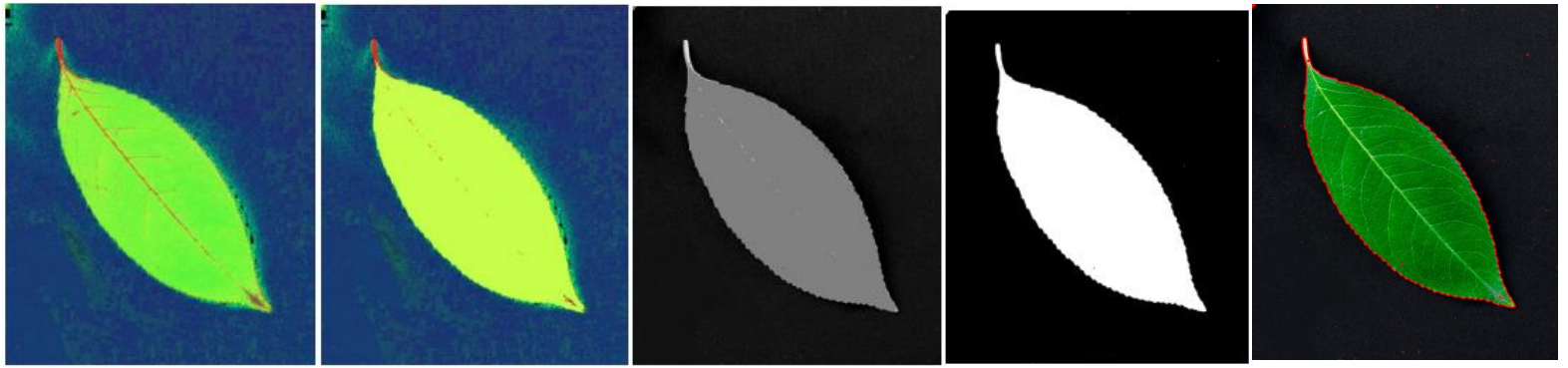
```
import cv2
import numpy as np


def viewImage(image):
    cv2.namedWindow('Display', cv2.WINDOW_NORMAL)
    cv2.imshow('Display', image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


def grayscale_17_levels (image):
    high = 255
    while(1):
        low = high - 15
        col_to_be_changed_low = np.array([low])
        col_to_be_changed_high = np.array([high])
        curr_mask = cv2.inRange(gray,
col_to_be_changed_low,col_to_be_changed_high)
        gray[curr_mask > 0] = (high)
        high -= 15
        if(low == 0 ):
            break


image = cv2.imread('./path/to/image')
viewImage(image)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
grayscale_17_levels(gray)
viewImage(gray)
```



```
def get_area_of_each_gray_level(im):


## convert image to gray scale (must br done before contouring)
    image = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
    output = []
    high = 255
    first = True
    while(1):


low = high - 15
        if(first == False):
```

```
# making values that are of a greater gray level black
            ## so it won't get detected
            to_be_black_again_low = np.array([high])
            to_be_black_again_high = np.array([255])
            curr_mask = cv2.inRange(image, to_be_black_again_low,
            to_be_black_again_high)
            image[curr_mask > 0] = (0)

        # making values of this gray level white so we can calculate
        # it's area
        ret, threshold = cv2.threshold(image, low, 255, 0)
        contours, hirerchy = cv2.findContours(threshold,
        cv2.RETR_LIST, cv2.CHAIN_APPROX_NONE)


if(len(contours) > 0):


output.append([cv2.contourArea(contours[0])])
            cv2.drawContours(im, contours, -1, (0,0,255), 3)


high -= 15
        first = False
        if(low == 0 ):


break


return output
```

Сегментация листа



```
image = cv2.imread('./path/to/image.jpg')
 hsv_img = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
 viewImage(hsv_img) ## 1
 green_low = np.array([45 , 100, 50] )
 green_high = np.array([75, 255, 255])
 curr_mask = cv2.inRange(hsv_img, green_low, green_high)
 hsv_img[curr_mask > 0] = ([75,255,200])
 viewImage(hsv_img) ## 2## Преобразование HSV-изображения к оттенкам серого
для дальнейшего## оконтуривания
 RGB_again = cv2.cvtColor(hsv_img, cv2.COLOR_HSV2RGB)
 gray = cv2.cvtColor(RGB_again, cv2.COLOR_RGB2GRAY)
 viewImage(gray) ## 3
ret, threshold = cv2.threshold(gray, 90, 255, 0)
 viewImage(threshold) ## 4
contours, hierarchy =
cv2.findContours(threshold,cv2.RETR_TREE,cv2.CHAIN_APPROX_SIMPLE)
 cv2.drawContours(image, contours, -1, (0, 0, 255), 3)
 viewImage(image) ## 5
```

Работа с камерой

```
#выполняем видеозахват с помощью метода VideoCapture(1) библиотеки cv2:
0 – встроенная камера ноутбука;  1 – например подключаемая к USB WEB камера;
C:\video\video1.mp3 – видеозахват из файла.

cap = cv2.VideoCapture(0)
while(True):
        ret, frame = cap.read()
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        cv2.imshow('Video', frame)
        cv2.imshow('Frame', gray)
        if cv2.waitKey(1) & 0xFF == ord('q'):
                break
cap.release()  #Закрываем канал видеозахвата;
cv2.destroyAllWindows() #Закрываем все окна открытые программой
```

Сложение изображений

```
1.  import cv2
2.
3.  # Считываем два изображения
4.  image_1 = cv2.imread('bike.jpg')
5.  image_2 = cv2.imread('car.jpg')
6.
7.  # Суммируем массивы двух изображений по всем каналам
8.  result = cv2.add(image_1, image_2)
9.
10. cv2.imshow('result', result)
11. cv2.waitKey(0)
12. cv2.destroyAllWindows()
```

Смешение изображений

```
1.  import cv2
2.
3.  # Считываем два изображения
4.  image_1 = cv2.imread('bike.jpg')
5.  image_2 = cv2.imread('car.jpg')
6.
7.  result = cv2.addWeighted(image_1, 0.9, image_2, 0.1)
8.
9.  cv2.imshow('result', result)
10. cv2.waitKey(0) # Программа останавливается до нажатия любой клавиши
11. cv2.destroyAllWindows()
```