# Лабораторная работа №2.

# Базовые операции с изображениями

## Основные операции с изображениями

*Цель*

*Научитесь этому:*

*- Доступ к значениям пикселей и их изменение*

*- Доступ к свойствам изображения*

*- Настройка области изображения (ROI)*

*- Разделение и слияние изображений*

Почти все операции в этом разделе в основном связаны с Numpy, а не с OpenCV. Хорошее знание Numpy необходимо для написания лучшего оптимизированного кода с помощью OpenCV.

### Pixel-значение

Вы можете получить доступ к значению пикселя по его координатам строки и столбца. Для изображения BGR он возвращает массив синих, зеленых, красных значений. Для изображения в оттенках серого возвращается только соответствующая интенсивность.

**px = image[100,100]**

**print (px)**

Таким же образом можно изменять значения пикселей.

**img[100,100] = [255,255,255]**

**print img[100,100]**

Numpy-это оптимизированная библиотека для быстрых вычислений массивов. Поэтому простой доступ к каждому значению пикселя и его изменение будут очень медленными, и это не рекомендуется.

Вышеупомянутый метод обычно используется для выбора области массива, скажем, первых 5 строк и последних 3 столбцов. Для индивидуального доступа к пикселям лучше использовать методы массива Numpy, array.item() и array.itemset (). Но он всегда возвращает скаляр. Поэтому,если вы хотите получить доступ ко всем значениям B,G, R, вам нужно вызвать array.item() отдельно для всех.

**# accessing RED value**

**>>> img.item(10,10,2)**

**59**

**# modifying RED value**

**>>> img.itemset((10,10,2),100)**

**>>> img.item(10,10,2)**

**100**

### Свойства изображения

Свойства изображения включают в себя количество строк, столбцов и каналов, тип данных изображения, количество пикселей и т.д.

Размеры изображения доступна с помощью img.shape. Он возвращает кортеж из числа строк, столбцов и каналов (если изображение цветное):

**>>> print img.shape**

**(342, 548, 3)**

Если изображение имеет оттенки серого, возвращаемый кортеж содержит только количество строк и столбцов. Таким образом, это хороший метод, чтобы проверить, является ли загруженное изображение оттенками серого или цветным изображением.

Общее количество пикселей доступно по img.size:

**>>> print img.size**

**562248**

Тип данных изображения получается с помощью img.type:

**>>> print img.dtype**

**uint8**

img.dtype очень важен при отладке, поскольку большое количество ошибок в коде OpenCV-Python вызвано недопустимым типом данных.

### ROI - Опорные области изображения

Иногда вам придется играть с определенной областью изображений. Для обнаружения глаз на изображениях сначала выполните распознавание лиц по изображению до тех пор, пока лицо не будет найдено, а затем выполните поиск глаз в области лица. Этот подход повышает точность (потому что глаза всегда находятся на лицах :D ) и производительность (потому что мы ищем небольшую область).

Выбор и копирование региона изображения:

**img2 = image[280:340, 330:390]**

**image[273:333, 100:160] = img2**

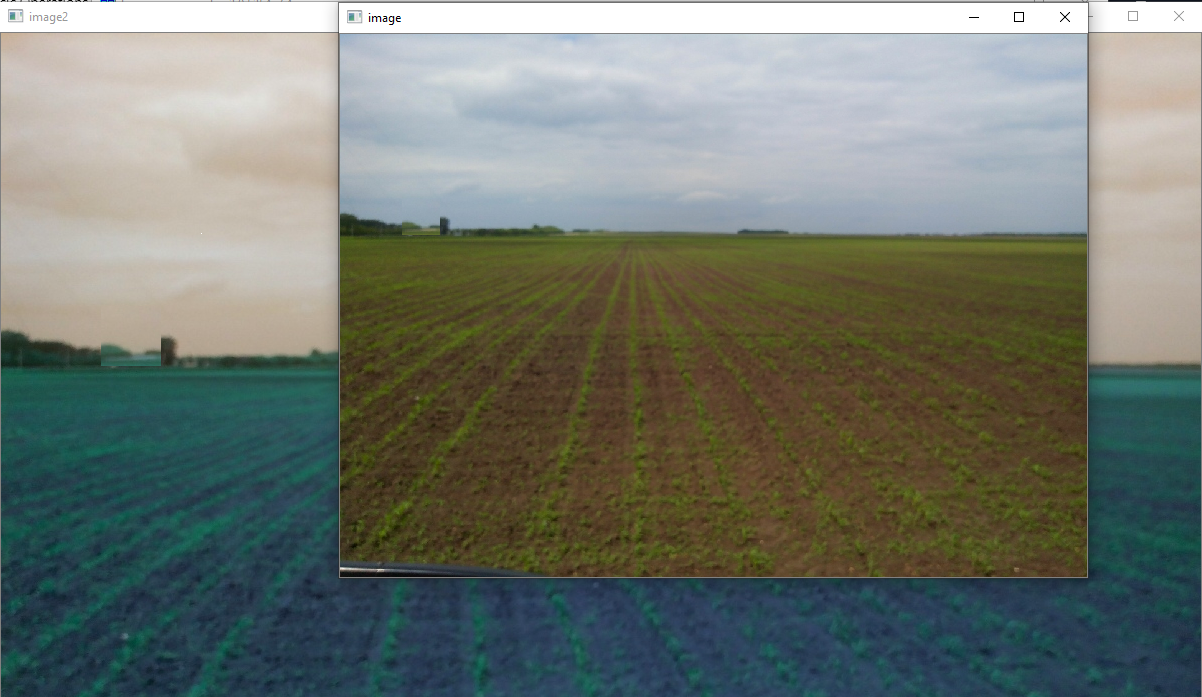
**cv2.imshow("image", image)**

## Разделение на цветовые плоскости

Каналы B,G,R изображения могут быть разделены на их отдельные плоскости, когда это необходимо. Затем отдельные каналы могут быть объединены обратно вместе, чтобы снова сформировать изображение BGR (и каналы можно «тусовать»)):

**b,g,r = cv2.split(image)**

**image2 = cv2.merge((r,g,b))**



Второй вариант:

**bb = image[:,:,0]**

**gg = image[:,:,1]**

**rr = image[:,:,2]**

Предположим, вы хотите сделать все красные пиксели равными нулю, вам не нужно разбивать их вот так и ставить равными нулю. Вы можете просто использовать индексацию Numpy, которая работает быстрее.

**img[:,:,2] = 0**

Вывод изображения в «зеленых» тонах:

**image[:,:,2] = 0**

**image[:,:,0] = 0**

**bb = image[:,:,0]**

**gg = image[:,:,1]**

**rr = image[:,:,2]**

**image3 = cv2.merge((bb,gg,rr))**

**cv2.imshow("image3", image3)**



## Арифметические операции над изображениями

*Цель*

*Изучите несколько арифметических операций над изображениями, таких как сложение, вычитание, побитовые операции и т. д.*

*Вы узнаете эти функции : cv2.add(), cv2.addWeighted() и т. д.*

### Сложение изображений

Вы можете складывать два изображения с помощью функции OpenCV, **cv2.add()** или просто с помощью операции **numpy, res = img1 + img2**. Оба изображения должны быть одинаковой глубины и типа, или второе изображение может быть просто скалярным значением.

!!!Существует разница между сложением OpenCV и сложением Numpy. Сложение OpenCV выполняет сложение с насыщением, Numpy - сложение по модулю.

**>>> x = np.uint8([250])**

**>>> y = np.uint8([10])**

**>>> print cv2.add(x,y) # 250+10 = 260 => 255**

**[[255]]**

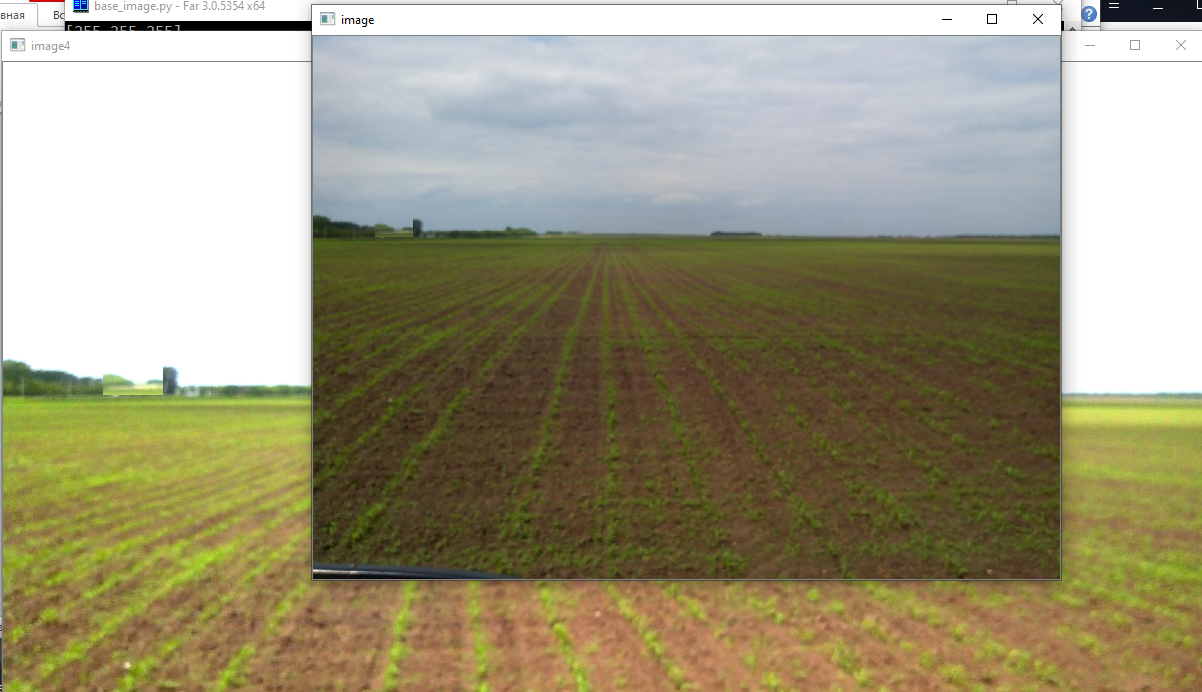
**>>> print x+y # 250+10 = 260 % 256 = 4**

**[4]**

Пример:

**image4 = cv2.add(image, image, image)**

**cv2.imshow("image4", image4)**



### Битовые операции

Это включает в себя побитовые операции **AND, OR, NOT** и **XOR**. Они будут очень полезны при извлечении любой части изображения (как мы увидим в следующих главах), определении и работе с непрямоугольным ROI и т. д. Ниже мы рассмотрим пример того, как изменить определенную область изображения.

Поместим логотип OpenCV над изображением. Если сложить два изображения, они изменят цвет (да и их размер разный). Если применить смешивание, то получим прозрачный эффект. Если бы это была прямоугольная область, можно было бы использовать ROI, как делалось в ранее. Но логотип OpenCV - это не прямоугольная форма. Таким образом, вы можете сделать это с помощью побитовых операций, как показано ниже:

**import cv2**

**import numpy as np**

**image = cv2.imread("dHWei9jpqrM.jpg",1)**

**logo = cv2.imread("logo.jpg",1)**

**cv2.namedWindow("image", cv2.WINDOW\_NORMAL)**

**# Я хочу поместить логотип в верхний левый угол, поэтому я создаю ROI**

**rows,cols,channels = logo.shape**

**roi = image[0:rows, 0:cols ]**

**# Теперь создайте маску логотипа и создайте также его инверсную маску**

**img2gray = cv2.cvtColor(logo,cv2.COLOR\_BGR2GRAY)**

**ret, mask = cv2.threshold(img2gray, 10, 255, cv2.THRESH\_BINARY)**

**mask\_inv = cv2.bitwise\_not(mask)**

**# Теперь затемните область логотипа в ROI**

**img1\_bg = cv2.bitwise\_and(roi,roi,mask = mask\_inv)**

**# Возьмите только область логотипа из изображения логотипа.**

**img2\_fg = cv2.bitwise\_and(logo,logo,mask = mask)**

**# Поместите логотип в ROI и измените основное изображение**

**dst = cv2.add(img1\_bg,img2\_fg)**

**image[0:rows, 0:cols ] = dst**

**cv2.imshow('res',image)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



****

**mask\_inv, img1\_bg, img2\_fg, dst**

### Смешивание изображений

Это также добавление изображения, но различные веса даются изображениям, так что это дает ощущение смешивания или прозрачности. Изображения добавляются в соответствии с приведенным ниже уравнением:

g(x) = (1 - \alpha)f_{0}(x) + \alpha f_{1}(x)

Изменяя \alpha от 0 \rightarrow 1, Вы можете выполнить крутой переход между одним изображением и другим.

Берутся два изображения, чтобы смешать их вместе. Первое изображение имеет вес 0,7, а второе-0,3. cv2.addWeighted() применяет к изображению следующее уравнение.

dst = \alpha \cdot img1 + \beta \cdot img2 + \gamma

Здесь \gamma принимается за ноль:

**img1 = cv2.imread('ml.png')**

**img2 = cv2.imread('opencv\_logo.jpg')**

**dst = cv2.addWeighted(img1,0.7,img2,0.3,0)**

**cv2.imshow('dst',dst)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

Как и в случае со сложением, смешение требует одинакового размера изображений.

****

Вопросы к лабораторной работе

1. привести подробное описание основных функций, рассмотренных в работе - функция, что делает, аргументы, параметры, возвращаемые результаты, типы данных и тп.
2. выполнить примеры, описанные в лабораторной - предоставить комментированный код, поэтапные результаты выполнения действий (исходные фото-, видео- данные для примеров берутся собственные, или из открытых источников или из базового набора библиотеки OpenCV)