Лабораторная работа №3

# Обработка изображений в OpenCV

## Изменение цветовых пространств

*Цель*

*В этом уроке вы узнаете, как конвертировать изображения из одного цветового пространства в другое, например BGR <> Gray, BGR <> HSV и т. д.*

*Кроме того, мы создадим приложение, которое извлекает цветной объект в видео*

*Вы узнаете следующие функции : cv2.cvtColor(), cv2.inRange() и т.д.*

**Изменение цветового пространства**

В OpenCV доступно более 150 методов преобразования цветового пространства. Но мы рассмотрим только два наиболее широко используемых из них: BGR <> Gray и BGR <> HSV.

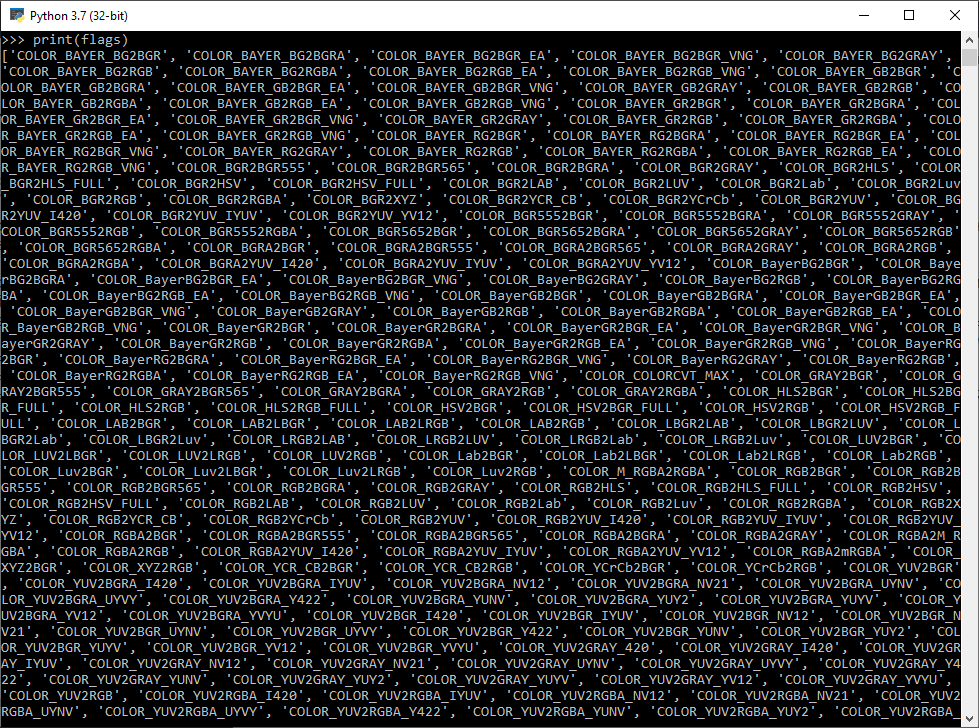
Для преобразования цвета мы используем функцию **cv2.cvtColor(input\_image, flag)**, где flag определяет тип преобразования.

Для преобразования BGR<>Gray мы используем флаги cv2.COLOR\_BGR2GRAY. Аналогично для BGR<>HSV мы используем флаг cv2.COLOR\_BGR2HSV. Чтобы получить другие флаги, просто выполните следующие команды в вашем терминале Python:

**>>> import cv2**

**>>> flags = [i for i in dir(cv2) if i.startswith('COLOR\_')]**

**>>> print flags**



Для HSV диапазон оттенков составляет [0,179], диапазон насыщенности - [0,255], а диапазон значений - [0,255]. Разные программы используют разные масштабы. Поэтому, если вы сравниваете значения OpenCV с ними, вам нужно нормализовать эти диапазоны.

### HSV

Пример разложения исходного изображения на каналы HSV:

**import cv2**

**import numpy as np**

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

**image = cv2.imread("IMG\_20190819\_155021.jpg",1)**

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

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

**# Convert BGR to HSV**

**hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)**

**h = hsv[:,:,0]**

**s = hsv[:,:,1]**

**v = hsv[:,:,2]**

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

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

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

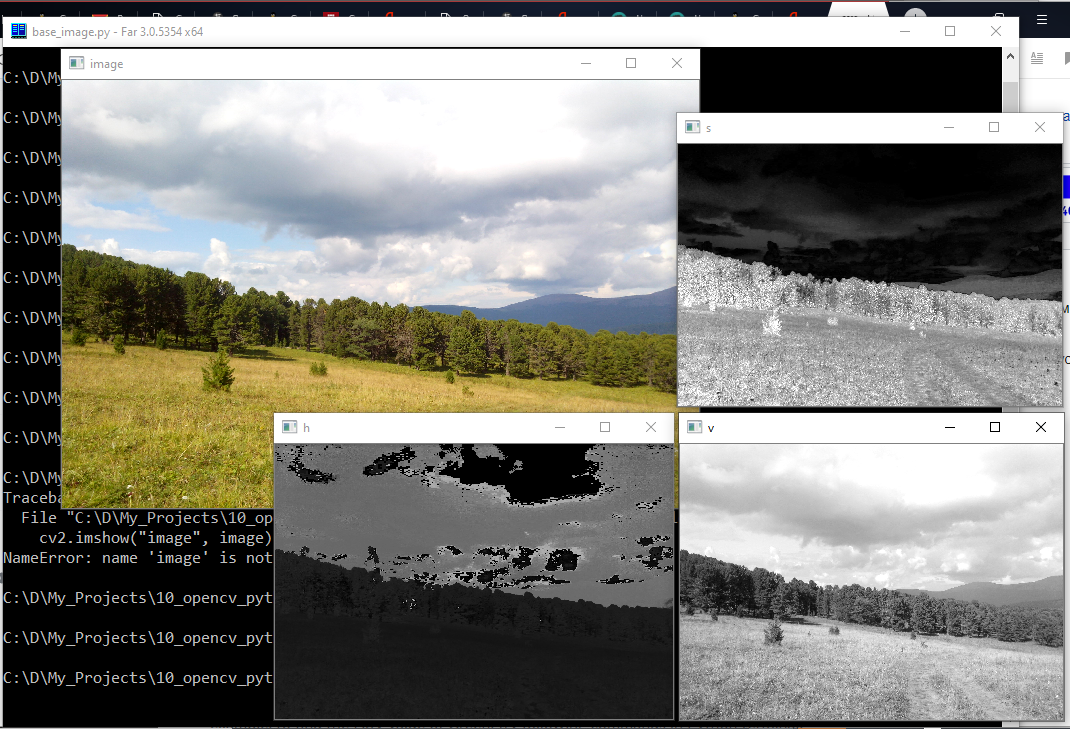
**cv2.imshow("s", s)**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



### Как найти значения HSV для отслеживания?

Это распространенный вопрос, который можно найти в stackoverflow.com. Это очень просто, и вы можете использовать ту же функцию, cv2.cvtColor(). Вместо того чтобы передавать изображение, вы просто передаете нужные значения BGR. Например, чтобы найти значение HSV зеленого цвета, попробуйте выполнить следующие команды в терминале Python:

**>>> green = np.uint8([[[0,255,0 ]]])**

**>>> hsv\_green = cv2.cvtColor(green,cv2.COLOR\_BGR2HSV)**

**>>> print hsv\_green**

**[[[ 60 255 255]]]**

Теперь вы берете [H-10, 100,100] и [H+10, 255, 255] как нижнюю границу и верхнюю границу соответственно. Помимо этого метода, вы можете использовать любые инструменты редактирования изображений, такие как GIMP или любые онлайн-конвертеры, чтобы найти эти значения, но не забудьте настроить диапазоны HSV.

Например, попробуем выделить на фотографии деревья:

**import cv2**

**import numpy as np**

**image = cv2.imread("IMG\_20190819\_155021.jpg",1)**

**# Convert BGR to HSV**

**hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)**

**lower\_blue = np.array([28,50,10])**

**upper\_blue = np.array([45,255,255])**

**# Threshold the HSV image to get only blue colors**

**mask = cv2.inRange(hsv, lower\_blue, upper\_blue)**

**# Bitwise-AND mask and original image**

**res = cv2.bitwise\_and(image,image, mask= mask)**

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

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

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

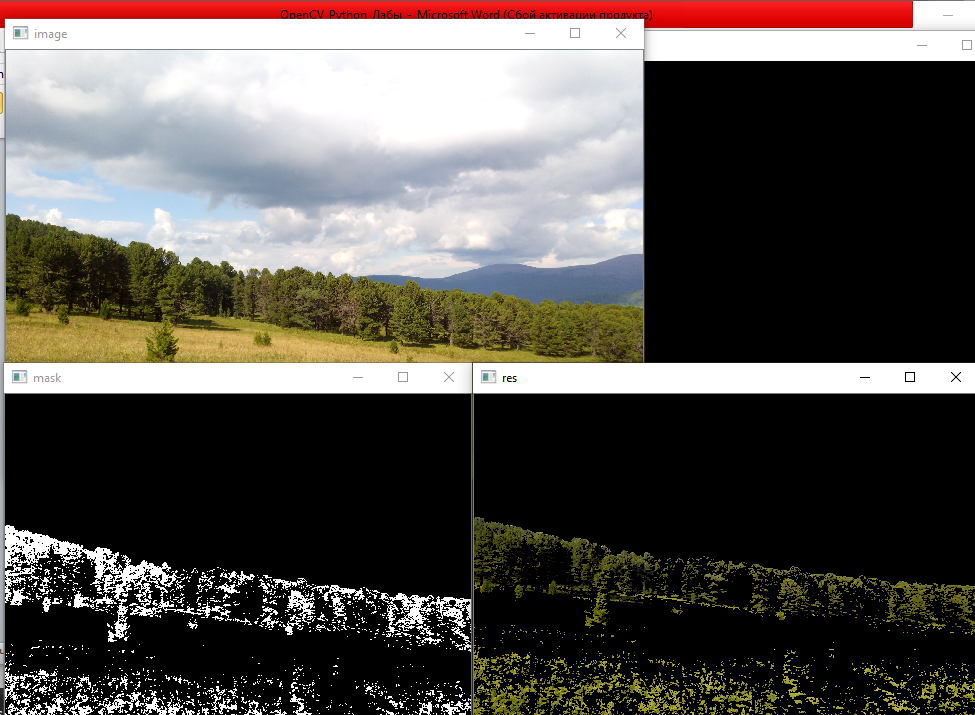
**cv2.imshow('mask',mask)**

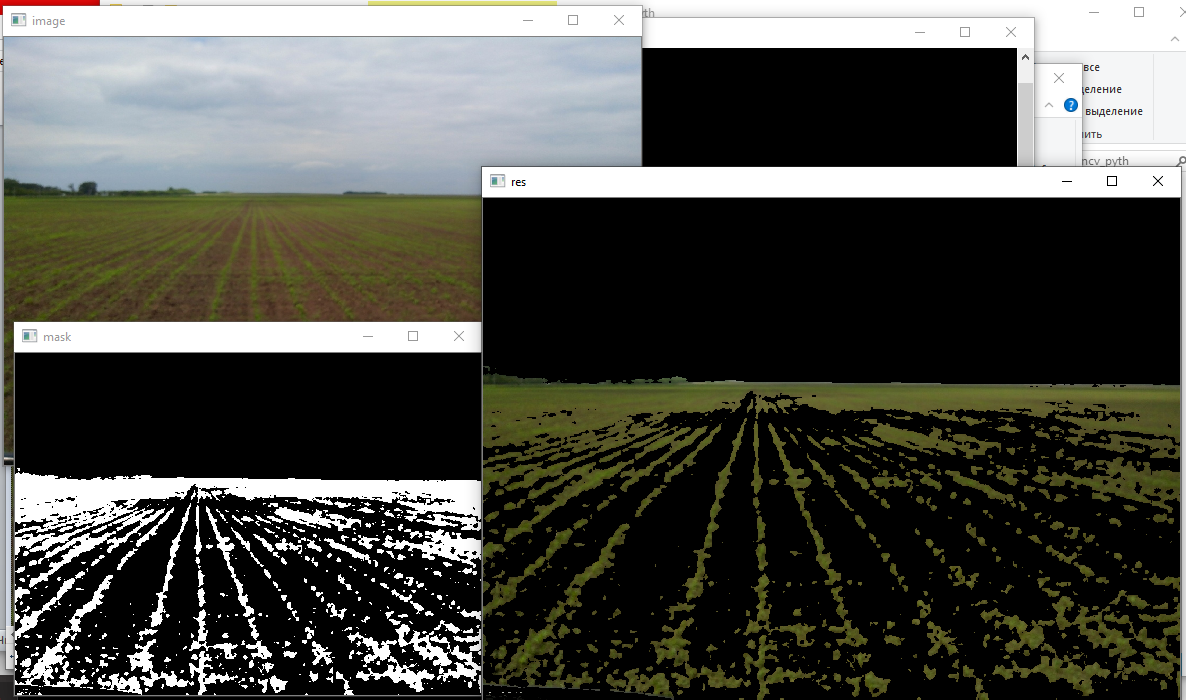
**cv2.namedWindow("res", cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**





Выделение травы на поле

**lower\_blue = np.array([27,0,0])**

**upper\_blue = np.array([50,255,255])**

## Бинаризация изображений

*В этом уроке вы узнаете о простом пороговом удержании, адаптивном пороговом удержании, пороговом удержании и т. д.*

*Вы узнаете эти функции : cv2.threshold, cv2.adaptiveThreshold и т. д.*

### Простая пороговая бинаризация

Если значение пикселя больше порогового значения, ему присваивается одно значение (может быть белым), в противном случае ему присваивается другое значение (может быть черным). Используемая функция- **cv2.threshold**. Первый аргумент-это исходное изображение, которое должно быть изображением в оттенках серого. Второй аргумент-это пороговое значение, которое используется для классификации значений пикселей. Третий аргумент-это maxVal, который представляет значение, которое должно быть задано, если значение пикселя больше (иногда меньше) порогового значения. OpenCV предост OpenCV предоставляет различные типы порогового значения, и оно определяется четвертым параметром функции. Существуют различные типы:

cv2.THRESH\_BINARY

cv2.THRESH\_BINARY\_INV

cv2.THRESH\_TRUNC

cv2.THRESH\_TOZERO

cv2.THRESH\_TOZERO\_INV

Получается два выхода. Первый-это retval , который будет объяснен позже. Второй вывод-это наше пороговое изображение.

**import cv2**

**import numpy as np**

**img = cv2.imread('image\_gray.jpg',0)**

**ret,thresh1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)**

**ret,thresh2 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY\_INV)**

**ret,thresh3 = cv2.threshold(img,127,255,cv2.THRESH\_TRUNC)**

**ret,thresh4 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO)**

**ret,thresh5 = cv2.threshold(img,127,255,cv2.THRESH\_TOZERO\_INV)**

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

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

**cv2.namedWindow("thresh1", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh1',thresh1)**

**cv2.namedWindow("thresh2", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh2',thresh2)**

**cv2.namedWindow("thresh3", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh3',thresh3)**

**cv2.namedWindow("thresh4", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh4',thresh4)**

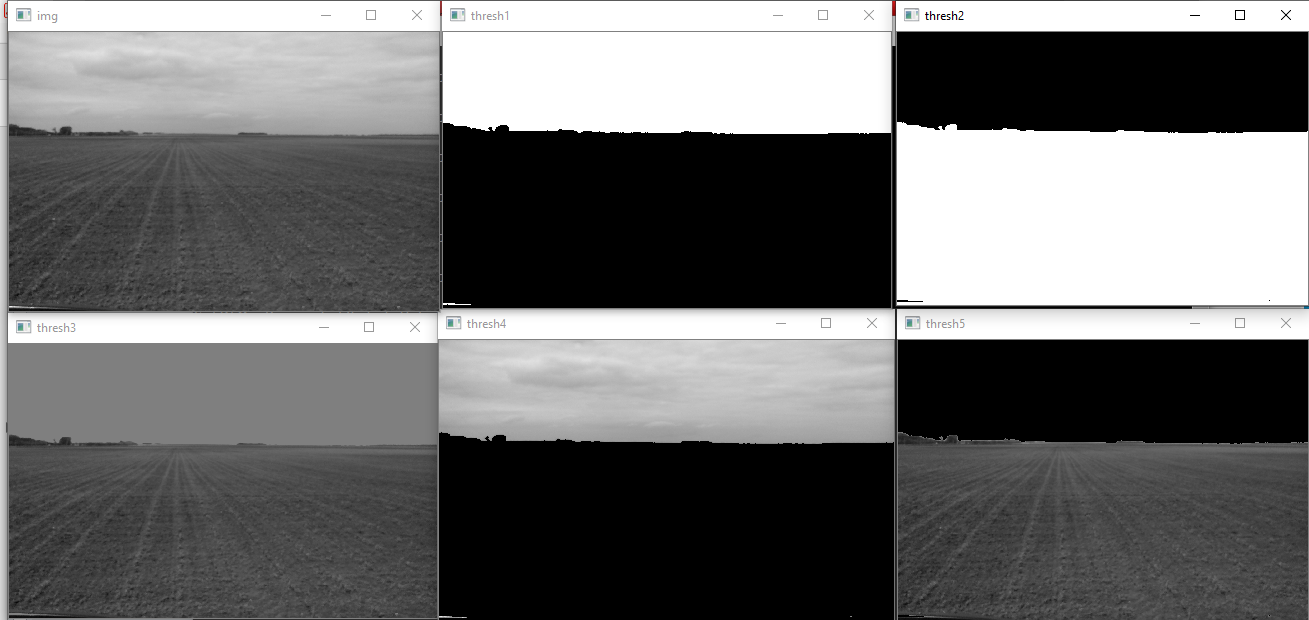
**cv2.namedWindow("thresh5", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh5',thresh5)**

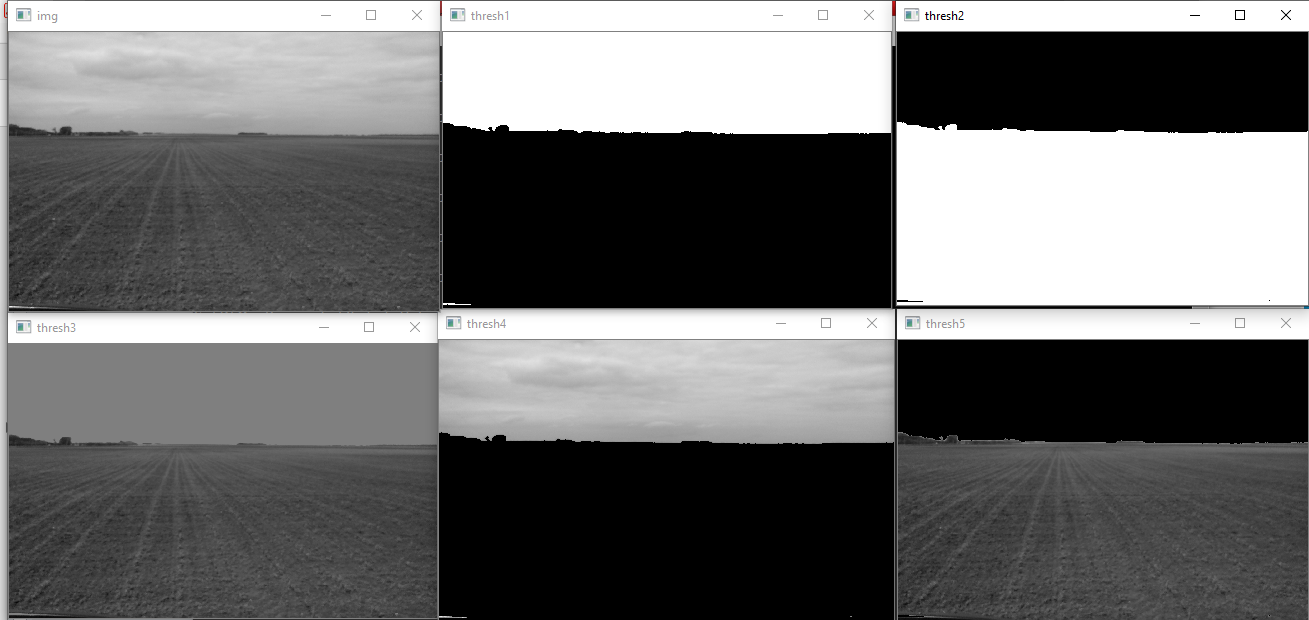
**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

Примеры различных вариантов бинаризации



Исходное cv2.THRESH\_BINARY cv2.THRESH\_BINARY\_INV



cv2.THRESH\_TRUNC cv2.THRESH\_TOZERO cv2.THRESH\_TOZERO\_INV

### Адаптивная пороговая бинаризация

В предыдущем разделе мы использовали глобальное значение в качестве порогового значения. Но это может быть не очень хорошо во всех условиях, когда изображение имеет разные условия освещения в разных областях. В этом случае мы переходим к адаптивному пороговому значению. При этом алгоритм вычисляет порог для небольших областей изображения. Таким образом, мы получаем разные пороговые значения для разных областей одного и того же изображения, и это дает нам лучшие результаты для изображений с различной освещенностью.

Он имеет три " специальных’ входных параметра и только один выходной аргумент.

Адаптивный метод - он решает, как вычисляется пороговое значение.

cv2.ADAPTIVE\_THRESH\_MEAN\_C : пороговое значение-это среднее значение площади окрестности.

cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C : пороговое значение-это взвешенная сумма значений окрестностей, где веса являются гауссовым окном.

Block Sze - Размер блока - он определяет размер района соседства.

С - это просто константа, которая вычитается из среднего или взвешенного среднего, рассчитанного.

Ниже фрагмент кода сравнивает глобальное пороговое значение и адаптивное пороговое значение для изображения с различной освещенностью:

**import cv2**

**import numpy as np**

**img = cv2.imread('WhatsApp Image 2020-04-01 at 21.39.37.jpg' ,0)**

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

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

**ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)**

**th2 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_MEAN\_C,\**

**cv2.THRESH\_BINARY,11,2)**

**th3 = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,\**

**cv2.THRESH\_BINARY,11,2)**

**cv2.namedWindow("thresh1", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh1',th1)**

**cv2.namedWindow("thresh2", cv2.WINDOW\_NORMAL)**

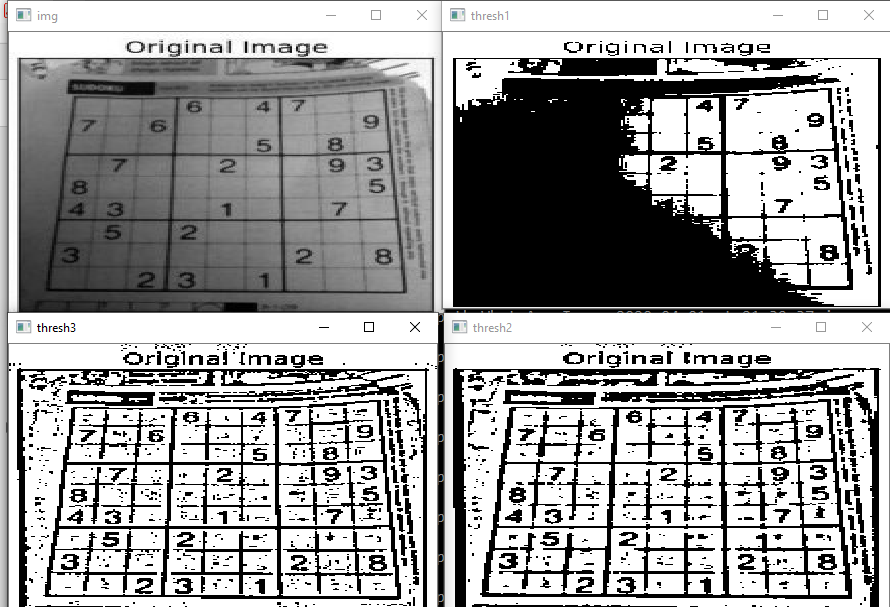
**cv2.imshow('thresh2',th2)**

**cv2.namedWindow("thresh3", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh3',th3)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



### Бинаризации OTSU

В первом разделе я говорил вам, что есть второй параметр retVal. Его использование приходит, когда мы идем на бинаризацию Оцу. Так в чем же дело?

В глобальном пороговом значении мы использовали произвольное значение для порогового значения, верно? Итак, как мы можем знать, является ли выбранная нами ценность хорошей или нет? Ответ-методом проб и ошибок. Но рассмотрим бимодальное изображение (проще говоря, бимодальное изображение-это изображение, гистограмма которого имеет два пика). Для этого изображения мы можем приблизительно принять значение в середине этих пиков в качестве порогового значения, не так ли ? Именно это и делает бинаризация Оцу. Таким образом, простыми словами, он автоматически вычисляет пороговое значение из гистограммы изображения для бимодального изображения. (Для изображений, которые не являются бимодальными, бинаризация не будет точной.)

Для этого используется наша функция cv2.threshold(), но передайте дополнительный флаг cv2.THRESH\_OTSU. Для порогового значения просто передайте ноль. Затем алгоритм находит оптимальное пороговое значение и возвращает вам в качестве второго выхода retval. Если пороговое значение Otsu не используется, retVal совпадает с пороговым значением, которое вы использовали.

Проверьте приведенный ниже пример. Входное изображение-это зашумленное изображение. В первом случае применяется глобальное пороговое значение для значения 127. Во втором случае - пороговое значение Оцу напрямую. В третьем случае - фильтрация изображения с помощью гауссовского ядра 5x5, чтобы удалить шум, а затем применяется пороговое значение Otsu. Посмотрите, как фильтрация шума улучшает результат:



**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20190819\_155021.jpg',0)**

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

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

**ret,th1 = cv2.threshold(img,127,255,cv2.THRESH\_BINARY)**

**ret,th2 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)**

**# Otsu's thresholding after Gaussian filtering**

**blur = cv2.GaussianBlur(img,(5,5),0)**

**ret,th3 = cv2.threshold(img,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)**

**cv2.namedWindow("thresh1", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh1',th1)**

**cv2.namedWindow("thresh2", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh2',th2)**

**cv2.namedWindow("thresh3", cv2.WINDOW\_NORMAL)**

**cv2.imshow('thresh3',th3)**

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

Поскольку мы работаем с бимодальными изображениями, алгоритм Otsu пытается найти пороговое значение (t), которое минимизирует взвешенную внутриклассовую дисперсию, заданную соотношением :

\sigma_w^2(t) = q_1(t)\sigma_1^2(t)+q_2(t)\sigma_2^2(t)

q_1(t) = \sum_{i=1}^{t} P(i) \quad \& \quad q_1(t) = \sum_{i=t+1}^{I} P(i)

\mu_1(t) = \sum_{i=1}^{t} \frac{iP(i)}{q_1(t)} \quad \& \quad \mu_2(t) = \sum_{i=t+1}^{I} \frac{iP(i)}{q_2(t)}

\sigma_1^2(t) = \sum_{i=1}^{t} [i-\mu_1(t)]^2 \frac{P(i)}{q_1(t)} \quad \& \quad \sigma_2^2(t) = \sum_{i=t+1}^{I} [i-\mu_1(t)]^2 \frac{P(i)}{q_2(t)}

Он фактически находит значение t, которое лежит между двумя пиками, так что отклонения для обоих классов минимальны.

# Геометрические преобразования изображений

*Цели*

*Научитесь применять различные геометрические преобразования к изображениям, такие как перевод, вращение, аффинное преобразование и т. д.*

*Вы увидите следующие функции: cv2.getPerspectiveTransform*

## Преобразования изображений

OpenCV предоставляет две функции преобразования, **cv2.warpAffine** и **cv2.warpPerspective**, с помощью которых вы можете иметь все виды преобразований. cv2.warpAffine принимает матрицу преобразования 2x3, в то время как cv2.warperspective принимает матрицу преобразования 3x3 в качестве входных данных.

## Масштабирование

Масштабирование-это просто изменение размера изображения. OpenCV поставляется с функцией **cv2.resize()** для этой цели. Размер изображения можно задать вручную или задать коэффициент масштабирования. Используются различные методы интерполяции. Предпочтительными методами интерполяции являются cv2.INTER\_AREA для сжатия и cv2.INTER\_CUBIC (slow) & cv2.INTER\_LINEAR для масштабирования. По умолчанию используется метод интерполяции cv2.INTER\_LINEAR для всех целей изменения размера. Вы можете изменить размер входного изображения одним из следующих способов:

**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20191109\_180417.jpg')**

**#res = cv2.resize(img,None,fx=2, fy=2, interpolation = cv2.INTER\_CUBIC)**

**#OR**

**height, width = img.shape[:2]**

**res = cv2.resize(img,(2\*width, 2\*height), interpolation = cv2.INTER\_CUBIC)**

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

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

## Сдвиг

Сдвиг - это смещение местоположения объекта. Если вы знаете сдвиг в направлении (x,y), пусть это будет (t\_x,t\_y), вы можете создать матрицу преобразования \textbf{M} следующим образом:

M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y  \end{bmatrix}

Вы можете взять make it в массив Numpy типа np.float32 и передать его в функцию cv2.warpAffine (). См. ниже пример для сдвига (100,50). Третьим аргументом функции cv2.warpAffine() является размер выходного изображения, который должен быть в виде (ширина, высота). Помните, что ширина = количество столбцов, а высота = количество строк:

**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20191109\_180417.jpg')**

**rows, cols = img.shape[:2]**

**# задаем матрицу преобразования**

**M = np.float32([[1,0,100],[0,1,50]])**

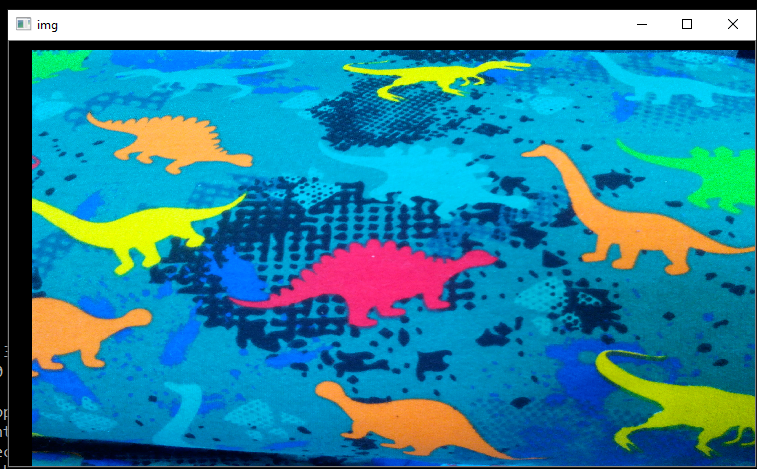
**dst = cv2.warpAffine(img,M,(cols,rows))**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Вращение

Поворот изображения на угол ϑ достигается матрицей преобразования вида:

M = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta   \end{bmatrix}

Но OpenCV обеспечивает масштабированное вращение с регулируемым центром вращения, так что вы можете вращаться в любом месте, которое вы предпочитаете. Модифицированная матрица преобразования задается формулой

\begin{bmatrix} \alpha &  \beta & (1- \alpha )  \cdot center.x -  \beta \cdot center.y \\ - \beta &  \alpha &  \beta \cdot center.x + (1- \alpha )  \cdot center.y \end{bmatrix},

где

\begin{array}{l} \alpha =  scale \cdot \cos \theta , \\ \beta =  scale \cdot \sin \theta \end{array}

Чтобы найти эту матрицу преобразования, OpenCV предоставляет функцию **cv2.getRotationMatrix2D**. Проверьте приведенный ниже пример, который поворачивает изображение на 37 градусов относительно центра без какого-либо масштабирования.

**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20191109\_180417.jpg')**

**rows, cols = img.shape[:2]**

**M = cv2.getRotationMatrix2D((cols/2,rows/2),37,1)**

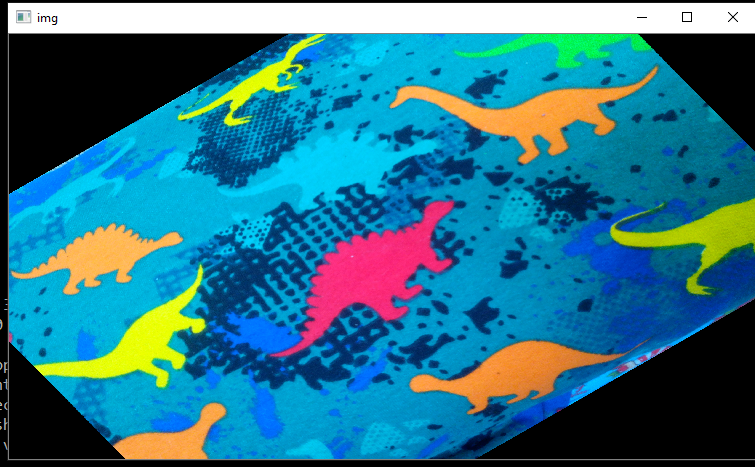
**dst = cv2.warpAffine(img,M,(cols,rows))**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



## Аффинное преобразование

При аффинном преобразовании все параллельные линии в исходном изображении будут по-прежнему параллельны в выходном изображении. Чтобы найти матрицу преобразования, нам нужны три точки из входного изображения и их соответствующие местоположения в выходном изображении. Затем cv2.getAffineTransform создаст матрицу 2x3, которая будет передана в cv2.warpAffine.

**Import cv2**

**import numpy as np**

**img = cv2.imread('dHWei9jpqrM.jpg')**

**rows, cols = img.shape[:2]**

**pts1 = np.float32([[50,50],[200,50],[50,200]])**

**pts2 = np.float32([[10,100],[200,50],[100,250]])**

**M = cv2.getAffineTransform(pts1,pts2)**

**dst = cv2.warpAffine(img,M,(cols,rows))**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

## Перспективное преобразование

Для перспективного преобразования вам понадобится матрица преобразования 3x3. Прямые линии останутся прямыми даже после трансформации. Чтобы найти эту матрицу преобразования, вам нужно 4 точки на входном изображении и соответствующие точки на выходном изображении. Среди этих 4 точек 3 из них не должны быть коллинеарными. Тогда матрицу преобразования можно найти с помощью функции cv2.getPerspectiveTransform. Затем примените cv2.warpPerspective с этой матрицей преобразования 3x3.

**import cv2**

**import numpy as np**

**img = cv2.imread('perspective1.jpg')**

**rows, cols = img.shape[:2]**

**pts1 = np.float32([[27,34],[182,27],[13,190],[191,191]])**

**pts2 = np.float32([[0,0],[300,0],[0,300],[300,300]])**

**M = cv2.getPerspectiveTransform(pts1,pts2)**

**dst = cv2.warpPerspective(img,M,(300,300))**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

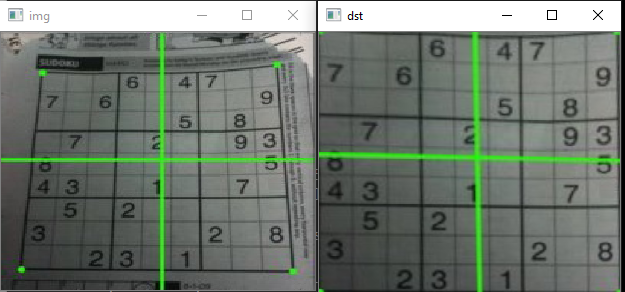
**cv2.imshow('img',img)**

**cv2.namedWindow('dst', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



# Сглаживание Изображений

*Цели*

*Научитесь этому:*

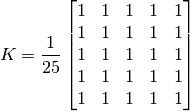
*Размытие изображений с помощью различных фильтров нижних частот*

*Применение пользовательских фильтров к изображениям (2D свертка)*

## 2D свертка ( фильтрация изображений )

Как и в случае одномерных сигналов, изображения также могут быть отфильтрованы с помощью различных фильтров нижних частот (ФНЧ), фильтров верхних частот (ФВЧ) и т. д. ФНЧ помогает устранить шум или размыть изображение. Фильтры ФВЧ помогают находить края в изображении.

OpenCV предоставляет функцию, **cv2.filter2D()**, сворачиваться ядра фильтра с изображением. В качестве примера мы попробуем применить усредняющий фильтр к изображению. Ядро усредняющего фильтра 5х5 можно определить следующим образом:



Фильтрация с вышеприведенным ядром приводит к следующему: для каждого пикселя окно 5x5 центрируется на этом пикселе, все пиксели, попадающие в это окно, суммируются, а затем результат делится на 25. Это равносильно вычислению среднего значения пикселей внутри этого окна. Эта операция выполняется для всех пикселей изображения, чтобы получить выходное отфильтрованное изображение:

**import cv2**

**import numpy as np**

**img = cv2.imread('IMG\_20190819\_155021.jpg')**

**#img = cv2.imread('opencv\_logo.png')**

**kernel = np.ones((5,5),np.float32)/25**

**dst = cv2.filter2D(img,-1,kernel)**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

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

**cv2.namedWindow('dst', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**

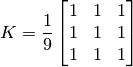


## Размытие Изображения (Сглаживание Изображения)

Размытие изображения достигается путем свертки изображения с ядром фильтра нижних частот. Это полезно для удаления шума. Он фактически удаляет высокочастотное содержимое (например, шум, края) из изображения, что приводит к размытию краев при применении этого фильтра. (+есть методы размытия, которые не размывают края). OpenCV предоставляет в основном четыре типа методов размытия.

**Усреднение**

Это делается путем свертки изображения с нормализованным прямоугольным фильтром. Он просто берет среднее значение всех пикселей под областью ядра и заменяет центральный элемент этим средним значением. Это делается с помощью функции **cv2.blur()** или **cv2.box Filter()**. RTFM для получения более подробной информации о ядре. Мы должны указать ширину и высоту ядра. Нормализованный фильтр 3x3 будет выглядеть следующим образом:



Если вы не хотите использовать нормализованный коробочный фильтр, используйте cv2.boxFilter() и передайте функции аргумент normalize=False.

**Гауссовская Фильтрация**

В этом подходе вместо коробочного фильтра, состоящего из равных коэффициентов фильтра, используется гауссово ядро. Это делается с помощью функции **cv2.GaussianBlur()**. Мы должны указать ширину и высоту ядра, которые должны быть положительными и нечетными. Мы также должны указать стандартное отклонение в направлениях X и Y, sigmaX и sigmaY соответственно. Если указан только sigmaX, то Сигма принимается равной Сигме X. Если оба значения заданы как нули, то они вычисляются исходя из размера ядра. Гауссовская фильтрация очень эффективна для удаления гауссовского шума из изображения.

Если вы хотите, вы можете создать гауссово ядро с помощью функции cv2.getGaussianKernel().

**Медианная Фильтрация**

Здесь функция **cv2.medianBlur()** вычисляет медиану всех пикселей под окном ядра, и центральный пиксель заменяется этим медианным значением. Это очень эффективно в устранении шума соли и перца. Интересно отметить, что в Гауссовском и боксовом фильтрах фильтруемое значение для центрального элемента может быть значением, которое может не существовать в исходном изображении. Однако это не относится к медианной фильтрации, так как центральный элемент всегда заменяется некоторым значением пикселя в изображении. Это эффективно снижает уровень шума. Размер ядра должен быть положительным нечетным целым числом.

**Двусторонняя Фильтрация**

Как мы уже отмечали, фильтры, представленные ранее, имеют тенденцию размывать края. Это не относится к двустороннему фильтру **cv2.bilateralFilter()**, который был определен для и очень эффективен при удалении шума при сохранении краев. Но работа происходит медленнее по сравнению с другими фильтрами. Мы уже видели, что гауссовский фильтр берет окрестность а вокруг пикселя и находит его Гауссовское средневзвешенное значение. Этот гауссовский фильтр является функцией только пространства, то есть при фильтрации учитываются соседние пиксели. Он не учитывает, имеют ли пиксели почти одинаковое значение интенсивности, и не учитывает, лежит ли пиксель на краю или нет. В результате получается, что Гауссовские фильтры имеют тенденцию размывать края, что нежелательно.

Двусторонний фильтр также использует Гауссов фильтр в пространственной области, но он также использует еще один (мультипликативный) Компонент гауссовского фильтра, являющийся функцией разности интенсивностей пикселей. Гауссова функция пространства гарантирует, что для фильтрации учитываются только пиксели, являющиеся "пространственными соседями", в то время как Гауссова компонента, применяемая в области интенсивности (Гауссова функция разности интенсивностей), гарантирует, что для вычисления размытого значения интенсивности включаются только те пиксели, интенсивность которых аналогична интенсивности центрального пикселя ("соседи интенсивности"). В результате этот метод сохраняет ребра, так как для пикселей, лежащих вблизи ребер, соседние пиксели, расположенные с другой стороны ребра и поэтому проявляющие большие вариации интенсивности по сравнению с центральным пикселем, не будут включены для размытия.

**import cv2**

**import numpy as np**

**img = cv2.imread('pl.jpg')**

**#img = cv2.imread('opencv-logo-white.png')**

**blur = cv2.blur(img,(5,5))**

**gaus = cv2.GaussianBlur(img,(5,5),0)**

**medi = cv2.medianBlur(img,5)**

**bila = cv2.bilateralFilter(img,9,75,75)**

**cv2.namedWindow('img', cv2.WINDOW\_NORMAL)**

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

**cv2.namedWindow('blur', cv2.WINDOW\_NORMAL)**

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

**cv2.namedWindow('gaus', cv2.WINDOW\_NORMAL)**

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

**cv2.namedWindow('medi', cv2.WINDOW\_NORMAL)**

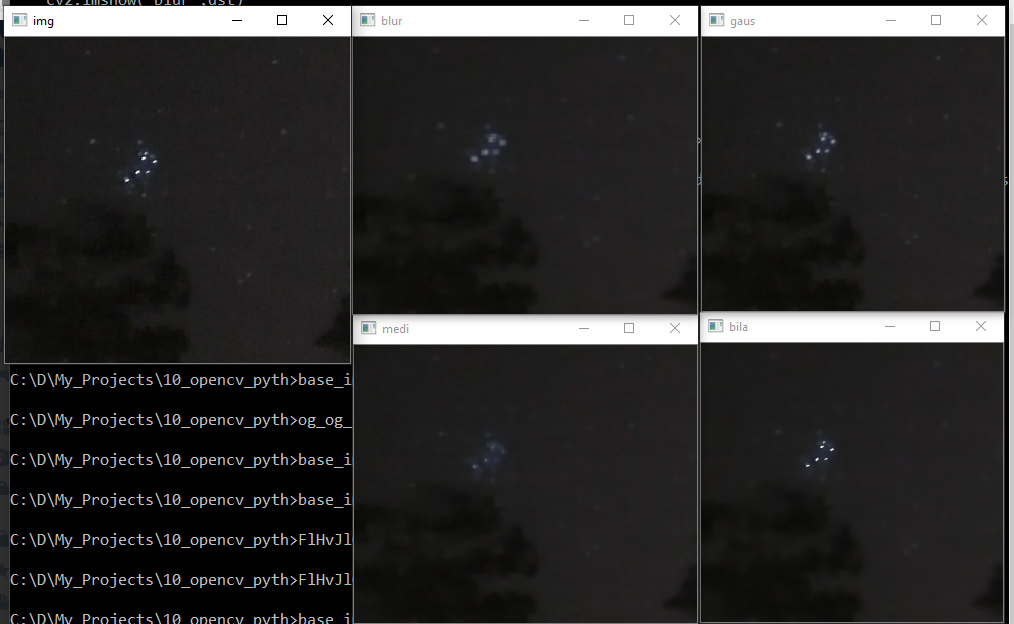
**cv2.imshow('medi',medi)**

**cv2.namedWindow('bila', cv2.WINDOW\_NORMAL)**

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

**cv2.waitKey(0)**

**cv2.destroyAllWindows()**



# Морфологические Преобразования

*Цель*

*В этой главе, мы будем изучать различные морфологические операции, такие как эрозия, расширение, открытие, закрытие и т. д.*

*Мы увидим различные функции, такие как : cv2.erode (), cv2.dilate(), cv2.morphologyEx() и т. д.*

Теория

Морфологические преобразования-это несколько простых операций, основанных на форме изображения. Обычно он выполняется на бинарных изображениях. Он нуждается в двух входах, один из которых является нашим исходным изображением, второй называется структурирующим элементом или ядром, которое определяет характер работы. Двумя основными морфологическими операторами являются эрозия и дилатация. Затем в игру также вступают его вариативные формы, такие как открытие, закрытие, градиент и т. д.

**Эрозия**

Основная идея эрозии так же, как эрозия почвы только, она размывает границы объекта переднего плана (всегда старайтесь держать передний план в белом цвете). Так что же он делает? Ядро скользит по изображению (как в 2D свертке). Пиксель в исходном изображении (либо 1, либо 0) будет считаться 1 только в том случае, если все пиксели под ядром равны 1, в противном случае он будет размыт (сведен к нулю).

Таким образом, происходит то, что все пиксели вблизи границы будут отброшены в зависимости от размера ядра. Таким образом, толщина или размер объекта переднего плана уменьшается или просто белая область уменьшается на изображении. Он полезен для удаления небольших белых шумов (как мы видели в главе цветовое пространство), отсоединения двух связанных объектов и т. д.

**erosion = cv2.erode(img,kernel,iterations = 1)**

**Дилатация**

Это прямо противоположно эрозии. Здесь пиксельный элемент равен "1", если хотя бы один пиксель под ядром равен "1". Таким образом, он увеличивает белую область на изображении или увеличивает размер объекта переднего плана. Обычно в таких случаях, как удаление шума, эрозия сопровождается расширением. Потому что эрозия удаляет белые шумы, но она также сжимает наш объект. Поэтому мы ее расширяем. Поскольку шум исчез, они не вернутся, но наша площадь объекта увеличивается. Это также полезно при соединении сломанных частей объекта.

**dilation = cv2.dilate(img,kernel,iterations = 1)**

**Открытие**

Открытие-это просто другое название эрозии, за которой следует расширение. Это полезно для устранения шума, как мы объяснили выше. Здесь мы используем функцию cv2.morphologyEx()

**opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)**

**Закрытие**

Закрытие-это обратная сторона открытия, расширение, за которым следует эрозия. Он полезен для закрытия небольших отверстий внутри объектов переднего плана или небольших черных точек на объекте.

**closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)**

**Морфологический Градиент**

Это разница между расширением и размыванием изображения.

Результат будет выглядеть как контур объекта.морфологический градиент

**gradient = cv2.morphologyEx(img, cv2.MORPH\_GRADIENT, kernel)**

**TopHat**

Это разница между входным изображением и открытием изображения. Ниже приведен пример для ядра 9x9.

**tophat = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel)**

**BlackHat**

Это разница между закрытием входного изображения и входного изображения.

**blackhat = cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)**

**Структурирующий Элемент**

Мы вручную создали структурирующие элементы в предыдущих примерах с помощью Numpy. Он имеет прямоугольную форму. Но в некоторых случаях вам могут понадобиться ядра эллиптической/круглой формы. Поэтому для этой цели в OpenCV есть функция cv2.getStructuringElement(). Вы просто передаете форму и размер ядра, вы получаете желаемое ядро.

**# Rectangular Kernel**

**>>> cv2.getStructuringElement(cv2.MORPH\_RECT,(5,5))**

**array([[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1]], dtype=uint8)**

**# Elliptical Kernel**

**>>> cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(5,5))**

**array([[0, 0, 1, 0, 0],**

**[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1],**

**[1, 1, 1, 1, 1],**

**[0, 0, 1, 0, 0]], dtype=uint8)**

**# Cross-shaped Kernel**

**>>> cv2.getStructuringElement(cv2.MORPH\_CROSS,(5,5))**

**array([[0, 0, 1, 0, 0],**

**[0, 0, 1, 0, 0],**

**[1, 1, 1, 1, 1],**

**[0, 0, 1, 0, 0],**

**[0, 0, 1, 0, 0]], dtype=uint8)**

Код:

**import cv2**

**import numpy as np**

**img = cv2.imread('j.png',0)**

**kernel = np.ones((5,5),np.uint8)**

**erosion = cv2.erode(img,kernel,iterations = 1)**

**dilation = cv2.dilate(img,kernel,iterations = 1)**

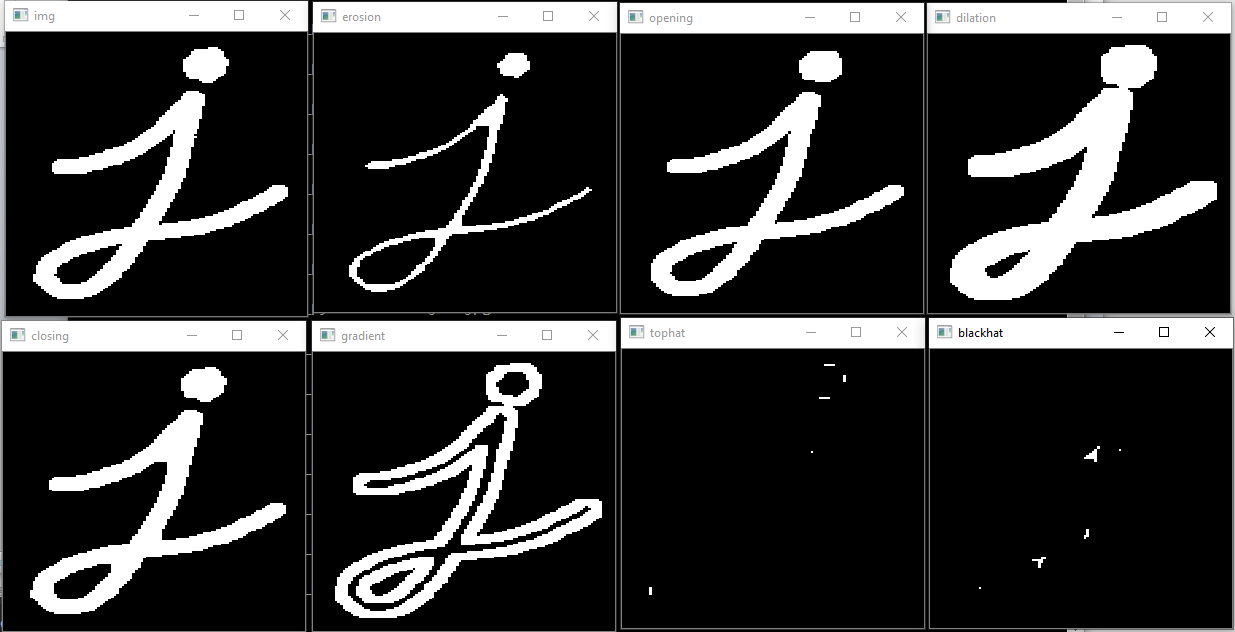
**opening = cv2.morphologyEx(img, cv2.MORPH\_OPEN, kernel)**

**closing = cv2.morphologyEx(img, cv2.MORPH\_CLOSE, kernel)**

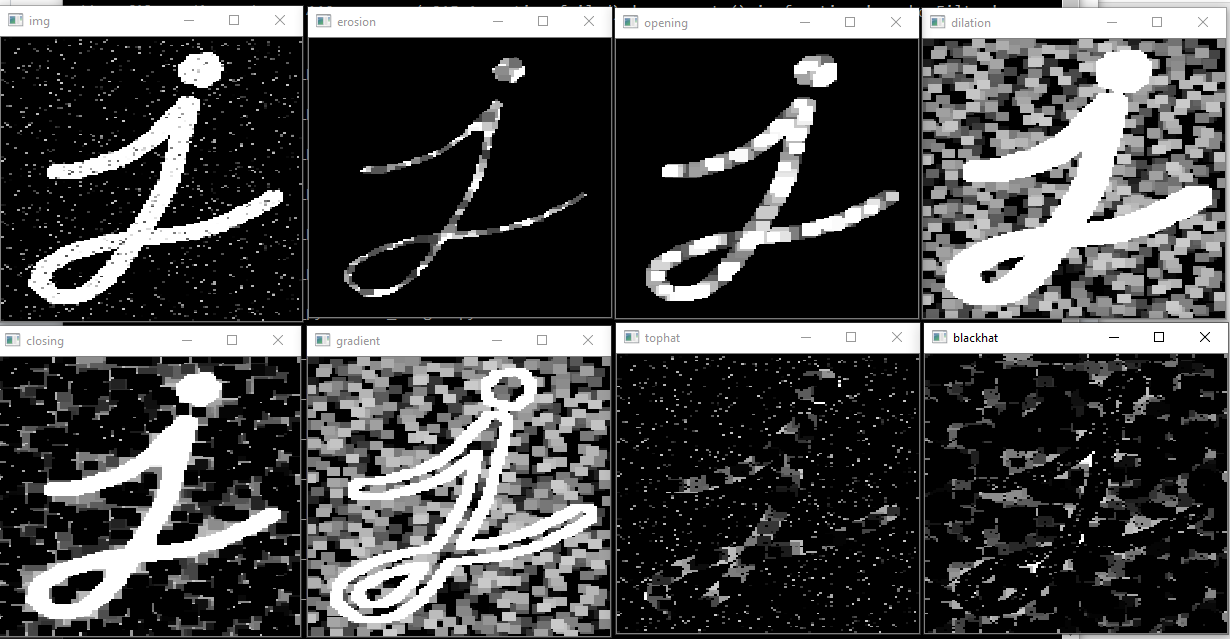
**gradient = cv2.morphologyEx(img, cv2.MORPH\_GRADIENT, kernel)**

**tophat = cv2.morphologyEx(img, cv2.MORPH\_TOPHAT, kernel)**

**blackhat = cv2.morphologyEx(img, cv2.MORPH\_BLACKHAT, kernel)**



Второй вариант с зашумленным начальным изображением



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)