Практическое занятие №1 Начало работы с библиотекой OpenCV

1. Установка Python
2. Установка библиотек OpenCV-Python, Numpy и Matplotlib
3. Тестирование

Установка Python

Скачиваем нужную версию Python с официального сайта Скачиваем и запускаем .exe файл. Не забываем установить галочку add path. Открываем командную строку.

Пакеты устанавливаются с помощью команды pip. Обычно он идет с Python, но требуется его обновить командой:
>python -m pip install --upgrade pip
Обновили pip, установим OpenCV и дополнительные библиотеки:

Установка библиотек OpenCV-Python, Numpy и Matplotlib

Они понадобятся для тестирования функций opencv.

Самый легкий и быстрый вариант установки с неофициальной версии.
Устанавливаем его из командной строки:
>pip install opencv-python

Вместе с opencv-python с этим пакетом идет пакет numpy. Дополнительно установим matplotlib:

>pip install matplotlib.

Тестирование

Теперь осталось проверить все ли работает для этого запускаем python и вводим команду:

```
>>>import cv2 as cv
>>>print(cv.__version__)
```

На экране должна отобразится версия установленной **библиотеки**

Работа с файлами изображений

Для чтения изображений используется метод cv2.imread(). Этот метод загружает изображение из указанного файла. Если изображение не может быть прочитано (из-за отсутствия файла, неправильных разрешений, неподдерживаемого или недопустимого формата), этот метод возвращает пустую матрицу.

Используемое изображение:

Пример: Python OpenCV для чтения изображения

- Python

```
# Python code to read image
import cv2

# To read image from disk, we use
# cv2.imread function, in below method,
img = cv2.imread("geeks.png", cv2.IMREAD_COLOR)
 print(img)
```

Отображение изображений

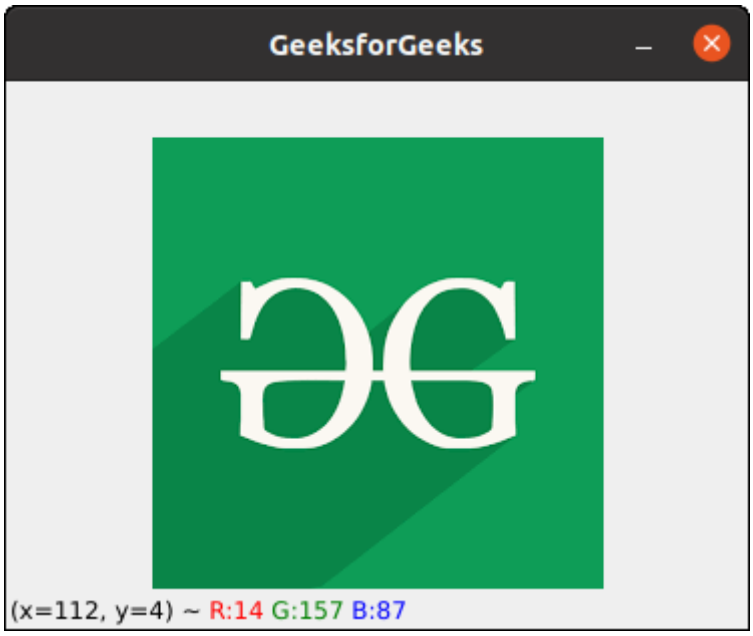
метод cv2.imshow() используется для отображения изображения в окне. Окно автоматически соответствует размеру изображения.

Пример: Python OpenCV отображает изображения

- Python

```
# Python code to read image
import cv2

# To read image from disk, we use
# cv2.imread function, in below method,
img = cv2.imread("geeks.png", cv2.IMREAD_COLOR)

# Creating GUI window to display an image on screen
# first Parameter is windows title (should be in string format)
# Second Parameter is image array
cv2.imshow("GeeksforGeeks", img)

# To hold the window on screen, we use cv2.waitKey method
# Once it detected the close input, it will release the control
# To the next line
# First Parameter is for holding screen for specified milliseconds
# It should be positive integer. If 0 pass an parameter, then it will
# hold the screen until user close it.
cv2.waitKey(0)

# It is for removing/deleting created GUI window from screen
# and memory
cv2.destroyAllWindows()
```

Вывод:



Сохранение изображений

метод cv2.imwrite() используется для сохранения изображения на любое запоминающее устройство. Это сохранит изображение в соответствии с указанным форматом в текущем рабочем каталоге.

Пример: Python OpenCV Сохраняет изображения

- Python

```
# Python program to explain cv2.imwrite() method

# importing cv2
import cv2

image_path = 'geeks.png'

# Using cv2.imread() method
# to read the image
img = cv2.imread(image_path)

# Filename
filename = 'savedImage.jpg'

# Using cv2.imwrite() method
# Saving the image
cv2.imwrite(filename, img)

# Reading and showing the saved image
img = cv2.imread(filename)
cv2.imshow("GeeksforGeeks", img)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:

Вращение изображений

[метод cv2.rotate()](#) используется для поворота 2D-массива на 90 градусов. Функция cv::rotate поворачивает массив тремя различными способами.

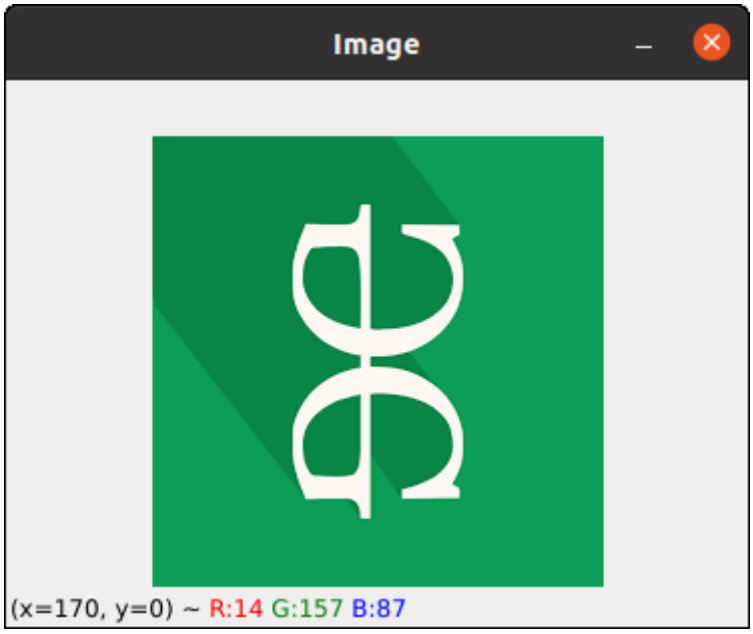
Пример: Python OpenCV поворачивает изображение

- Python3

```
# Python program to explain cv2.rotate() method

# importing cv2
import cv2

# path
path = 'geeks.png'

# Reading an image in default mode
src = cv2.imread(path)

# Window name in which image is displayed
window_name = 'Image'

# Using cv2.rotate() method
# Using cv2.ROTATE_90_CLOCKWISE rotate
# by 90 degrees clockwise
image = cv2.rotate(src, cv2.cv2.ROTATE_90_CLOCKWISE)
```

```
# Displaying the image
cv2.imshow(window_name, image)
cv2.waitKey(0)
```

Вывод:



Вышеуказанные функции ограничивают нас возможностью поворота изображения только на 90 градусов. Мы также можем поворачивать изображение под любым углом, определяя точку поворота матрицы поворота, степень поворота и коэффициент масштабирования.

Пример: Python OpenCV поворачивает изображение под любым углом

- Python

```
import cv2
import numpy as np

FILE_NAME = 'geeks.png'

# Read image from the disk.
img = cv2.imread(FILE_NAME)

# Shape of image in terms of pixels.
(rows, cols) = img.shape[:2]

# getRotationMatrix2D creates a matrix needed
```

```
# for transformation. We want matrix for rotation
# w.r.t center to 45 degree without scaling.
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 45, 1)
res = cv2.warpAffine(img, M, (cols, rows))

cv2.imshow("GeeksforGeeks", res)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:

Изменение размера изображения

Изменение размера изображения относится к масштабированию изображений. Это помогает уменьшить количество пикселей в изображении и имеет ряд преимуществ, например. Это может сократить время обучения нейронной сети, поскольку чем больше количество пикселей в изображении, тем больше количество входных узлов, что, в свою очередь, увеличивает сложность модели. Это также помогает в увеличении изображений. Много раз нам нужно изменить размер изображения, т.Е. Либо уменьшить его, либо увеличить в соответствии с требованиями к размеру.

OpenCV предоставляет нам несколько методов интерполяции для изменения размера изображения. Выбор метода интерполяции для изменения размера –

- **cv2.INTER_AREA:** используется, когда нам нужно уменьшить изображение.
- **cv2.INTER_CUBIC:** это медленно, но более эффективно.

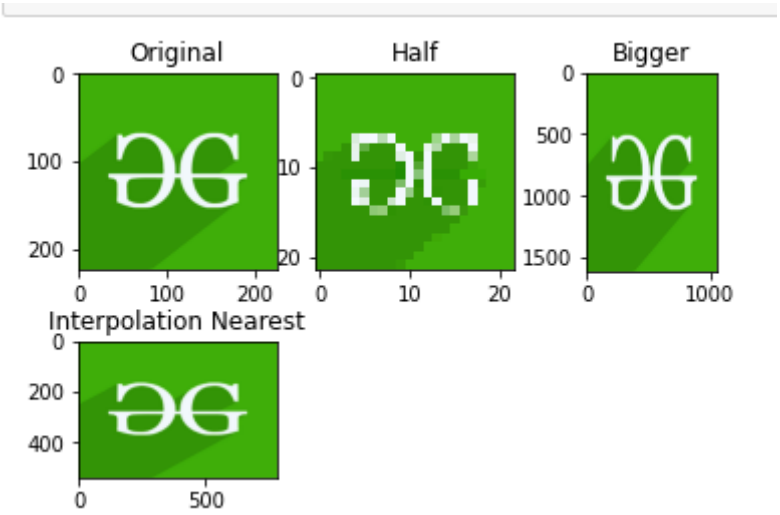
- **cv2.INTER_LINEAR:** это в основном используется, когда требуется масштабирование. Это метод интерполяции по умолчанию в OpenCV.

Пример: Изменение размера изображения Python OpenCV

- Python

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

image = cv2.imread("geeks.png", 1)
# Loading the image

half = cv2.resize(image, (0, 0), fx = 0.1, fy = 0.1)
bigger = cv2.resize(image, (1050, 1610))

stretch_near = cv2.resize(image, (780, 540),
           interpolation = cv2.INTER_NEAREST)


Titles =["Original", "Half", "Bigger", "Interpolation Nearest"]
images =[image, half, bigger, stretch_near]
count = 4

for i in range(count):
   plt.subplot(2, 3, i + 1)
   plt.title(Titles[i])
   plt.imshow(images[i])

plt.show()
```

Вывод:



Цветовые пространства

Цветовые пространства - это способ представления цветовых каналов, присутствующих в изображении, который придает изображению определенный оттенок. Существует несколько разных цветовых пространств, и каждое из них имеет свое значение. Некоторые из популярных цветовых пространств - RGB (красный, зеленый, синий), CMYK (голубой, пурпурный, желтый, черный), HSV (оттенок, насыщенность, значение) и т.д. метод cv2.cvtColor() используется для преобразования изображения из одного цветового пространства в другое. В OpenCV доступно более 150 методов преобразования цветового пространства.

Пример: Цветовые пространства Python OpenCV

- Python

```
# Python program to explain cv2.cvtColor() method

# importing cv2
import cv2

# path
path = 'geeks.png'

# Reading an image in default mode
src = cv2.imread(path)

# Window name in which image is displayed
window_name = 'GeeksforGeeks'
```

```
# Using cv2.cvtColor() method
# Using cv2.COLOR_BGR2GRAY color space
# conversion code
image = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY )

# Displaying the image
cv2.imshow(window_name, image)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:

Арифметические операции

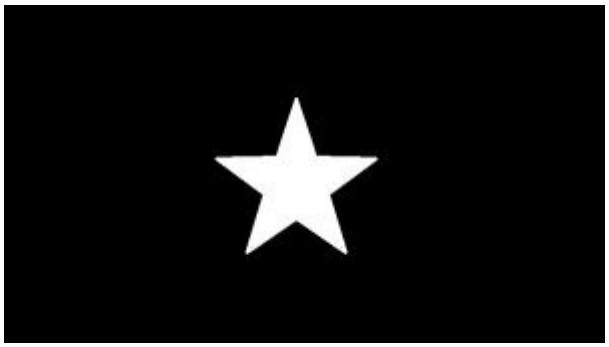
К входным изображениям могут быть применены арифметические операции, такие как сложение, вычитание и побитовые операции (И, ИЛИ, НЕ, XOR). Эти операции могут быть полезны для улучшения свойств входных изображений. Арифметика изображений важна для анализа свойств входного изображения. Управляемые изображения могут быть дополнительно использованы в качестве улучшенного входного изображения, и многие другие операции могут быть применены для уточнения, порогового значения, расширения и т.д. изображения.

Добавление изображения:

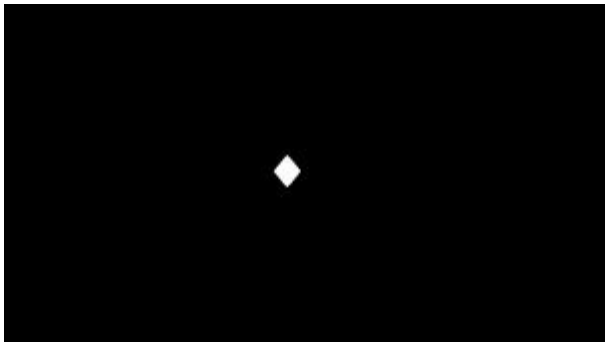
Мы можем добавить два изображения с помощью функции cv2.add(). Это напрямую суммирует пиксели изображения в двух изображениях. Но добавление пикселей - не идеальная ситуация. Итак, мы используем

cv2.addweighted(). Помните, что оба изображения должны быть одинакового размера и глубины.

Ввод изображения 1:



Ввод изображения 2:



- Python3

```
# Python program to illustrate
# arithmetic operation of
# addition of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
image1 = cv2.imread('star.jpg')
image2 = cv2.imread('dot.jpg')

# cv2.addWeighted is applied over the
```

```
# image inputs with applied parameters
weightedSum = cv2.addWeighted(image1, 0.5, image2, 0.4, 0)

# the window showing output image
# with the weighted sum
cv2.imshow('Weighted Image', weightedSum)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:

Вычитание изображения:

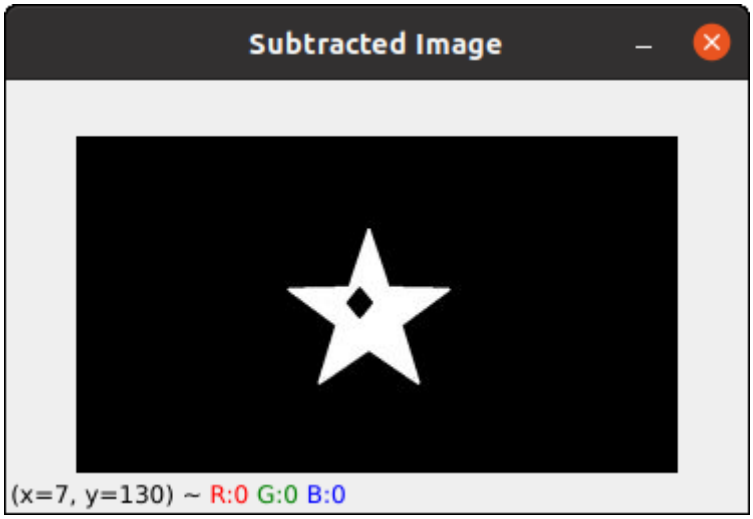
Точно так же, как и в дополнение, мы можем вычесть значения пикселей в двух изображениях и объединить их с помощью cv2.subtract() . Изображения должны быть одинакового размера и глубины.

- Python3

```
# Python program to illustrate
# arithmetic operation of
# subtraction of pixels of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
```

```
image1 = cv2.imread('star.jpg')
image2 = cv2.imread('dot.jpg')

# cv2.subtract is applied over the
# image inputs with applied parameters
sub = cv2.subtract(image1, image2)

# the window showing output image
# with the subtracted image
cv2.imshow('Subtracted Image', sub)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:



Побитовые операции над двоичным изображением

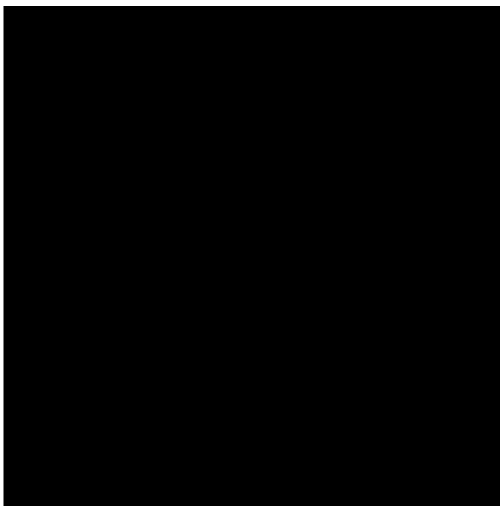
Побитовые операции используются при обработке изображений и используются для извлечения важных частей изображения. Используемые побитовые операции :

- И
- или
- XOR
- НЕ

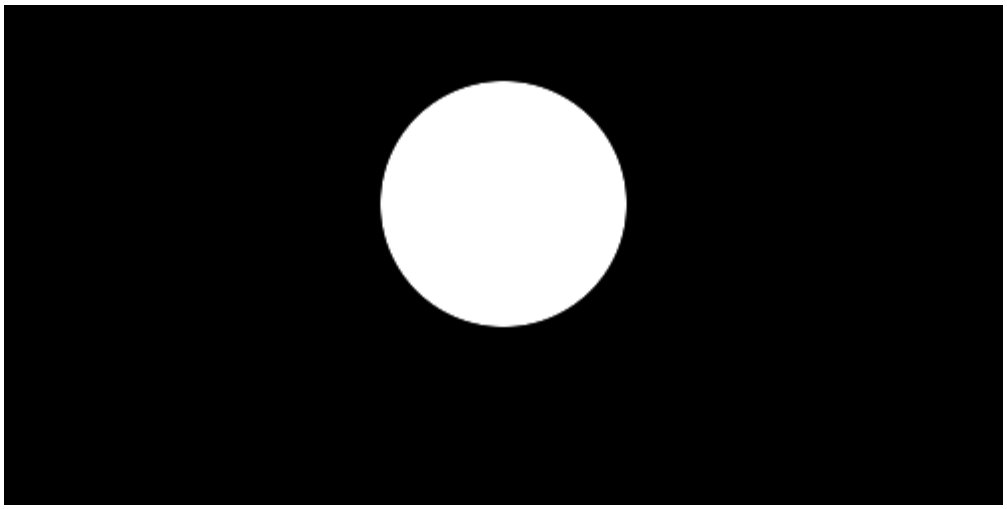
Побитовая обработка и операция

Побитовое объединение элементов входного массива.

Входное изображение 1:



Входное изображение 2:

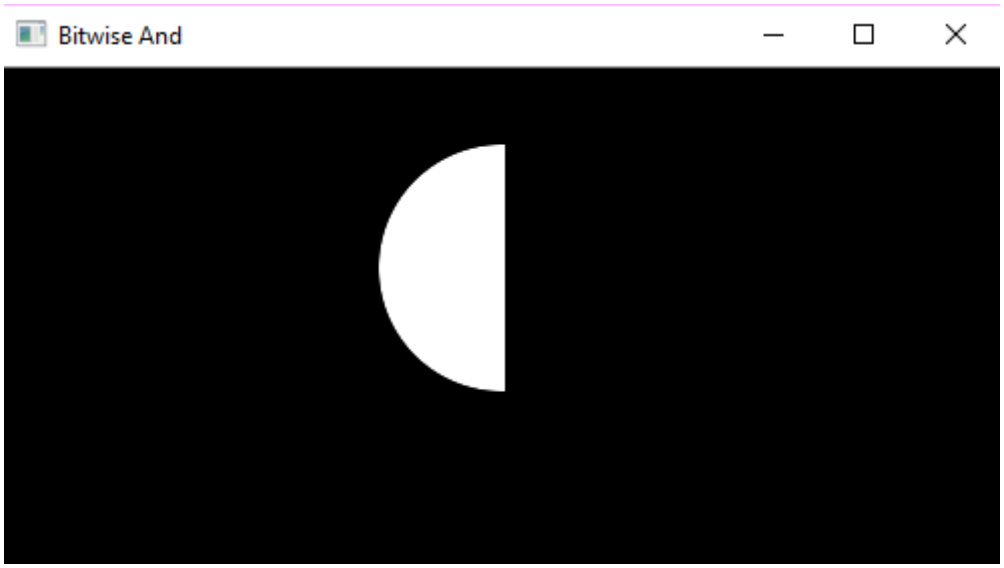


- Python3

```
# Python program to illustrate
# arithmetic operation of
# bitwise AND of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')

# cv2.bitwise_and is applied over the
# image inputs with applied parameters
```

```
dest_and = cv2.bitwise_and(img2, img1, mask = None)

# the window showing output image
# with the Bitwise AND operation
# on the input images
cv2.imshow('Bitwise And', dest_and)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:



Побитовая операция ИЛИ

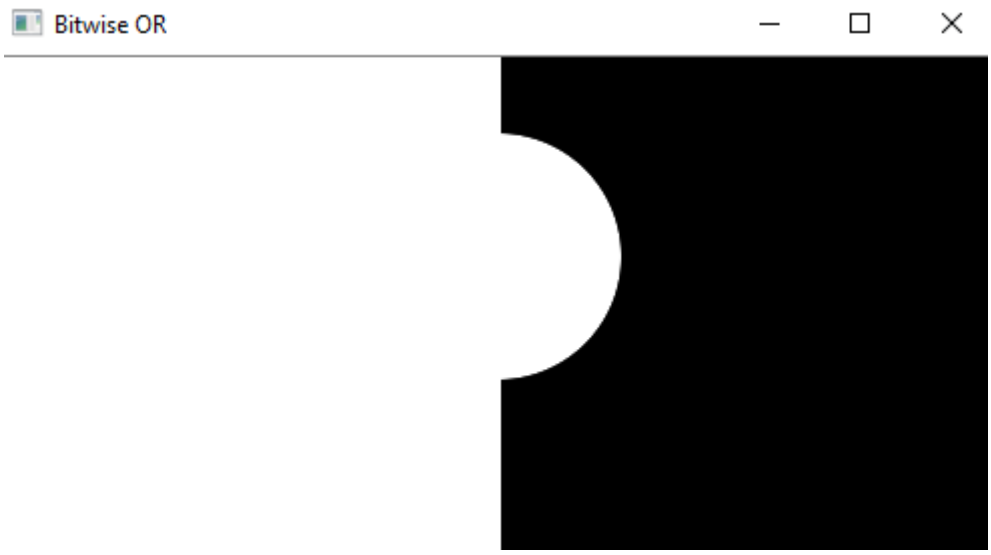
Побитовая дизъюнкция элементов входного массива.

- Python3

```
# Python program to illustrate
# arithmetic operation of
# bitwise OR of two images

# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
img1 = cv2.imread('input1.png')
```

```
img2 = cv2.imread('input2.png')

# cv2.bitwise_or is applied over the
# image inputs with applied parameters
dest_or = cv2.bitwise_or(img2, img1, mask = None)

# the window showing output image
# with the Bitwise OR operation
# on the input images
cv2.imshow('Bitwise OR', dest_or)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:



Побитовая операция XOR

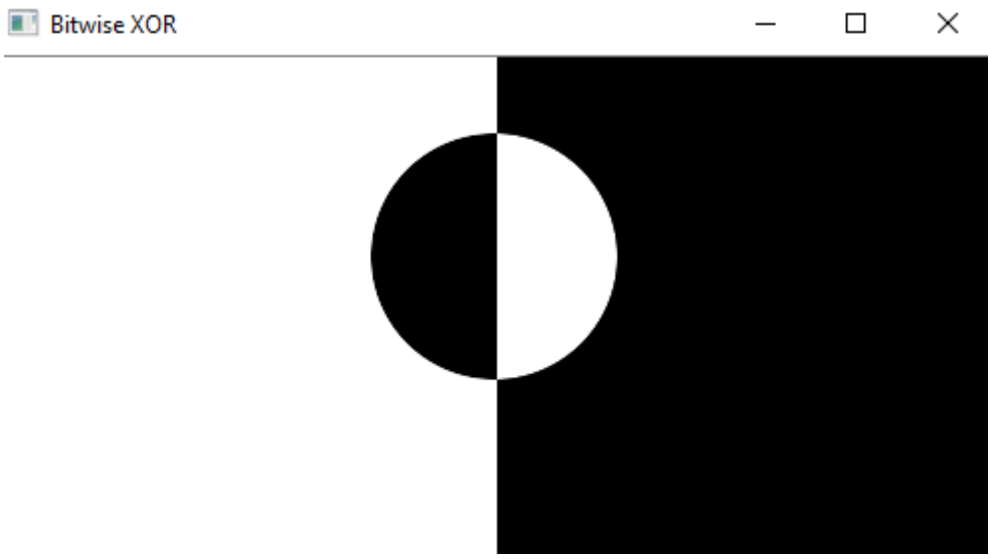
Побитовая операция exclusive-OR над элементами входного массива.

- Python3

```
# Python program to illustrate
# arithmetic operation of
# bitwise XOR of two images

# organizing imports
import cv2
import numpy as np
```

```
# path to input images are specified and
# images are loaded with imread command
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')

# cv2.bitwise_xor is applied over the
# image inputs with applied parameters
dest_xor = cv2.bitwise_xor(img1, img2, mask = None)

# the window showing output image
# with the Bitwise XOR operation
# on the input images
cv2.imshow('Bitwise XOR', dest_xor)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:



Побитовая операция NOT

Инверсия элементов входного массива.

- Python

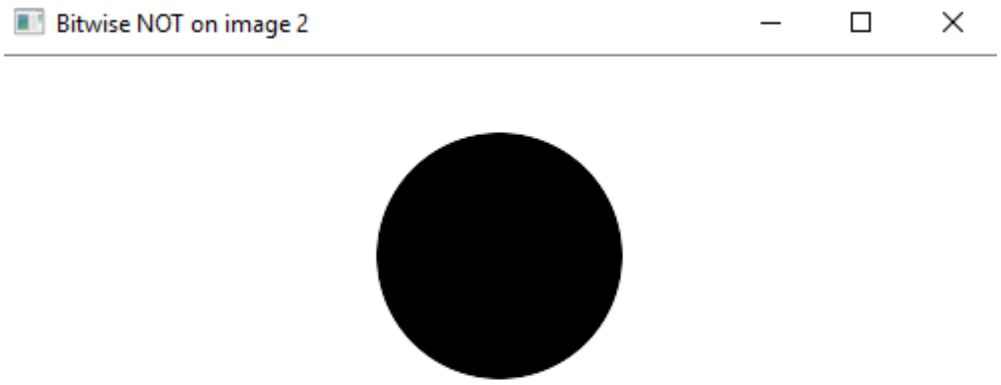
```
# Python program to illustrate
# arithmetic operation of
# bitwise NOT on input image
```

```
# organizing imports
import cv2
import numpy as np

# path to input images are specified and
# images are loaded with imread command
img1 = cv2.imread('input1.png')
img2 = cv2.imread('input2.png')

# cv2.bitwise_not is applied over the
# image input with applied parameters
dest_not1 = cv2.bitwise_not(img1, mask = None)
dest_not2 = cv2.bitwise_not(img2, mask = None)

# the windows showing output image
# with the Bitwise NOT operation
# on the 1st and 2nd input image
cv2.imshow('Bitwise NOT on image 1', dest_not1)
cv2.imshow('Bitwise NOT on image 2', dest_not2)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:

Побитовый НЕ на изображении 1

Побитовый НЕ на изображении 2



Перевод изображений

Перевод относится к прямолинейному перемещению объекта, то есть изображения, из одного местоположения в другое. Если мы знаем величину сдвига в горизонтальном и вертикальном направлениях, скажем (tx, ty), тогда мы можем составить матрицу преобразования. Теперь мы можем использовать функцию cv2.wrapAffine() для реализации переводов. Для этой функции требуется массив размером 2 × 3. Массив numpy должен иметь тип float.

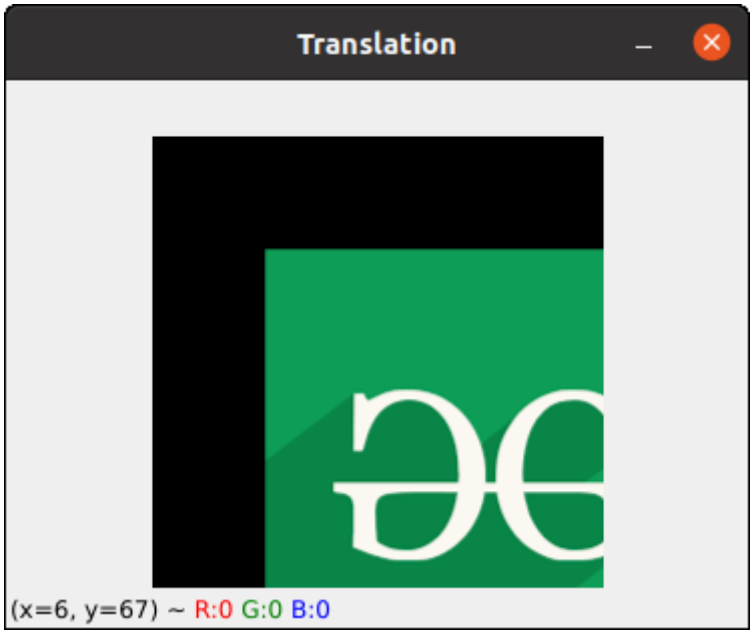
Пример: Смещение изображений Python OpenCV

- Python

```
import cv2
import numpy as np

image = cv2.imread('geeks.png')

# Store height and width of the image
height, width = image.shape[:2]

quarter_height, quarter_width = height / 4, width / 4

T = np.float32([[1, 0, quarter_width], [0, 1, quarter_height]])

# We use warpAffine to transform
# the image using the matrix, T
img_translation = cv2.warpAffine(image, T, (width, height))
```

```
cv2.imshow('Translation', img_translation)
cv2.waitKey(0)

cv2.destroyAllWindows()
```

Вывод:



Обнаружение границ

Процесс обнаружения изображения включает в себя обнаружение острых краев на изображении. Это обнаружение границ важно в контексте распознавания изображений или локализации / обнаружения объектов. Существует несколько алгоритмов для обнаружения ребер из-за его широкой применимости. Мы будем использовать один из таких алгоритмов, известный как Canny Edge Detection.

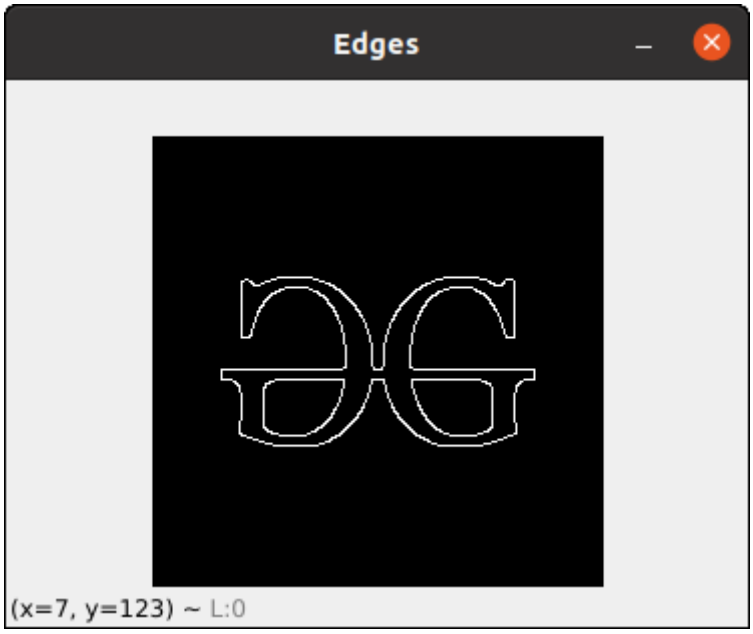
Пример: обнаружение границ Python OpenCV

- Python

```
import cv2

FILE_NAME = 'geeks.png'

# Read image from disk.
img = cv2.imread(FILE_NAME)

# Canny edge detection.
```

```
edges = cv2.Canny(img, 100, 200)

# Write image back to disk.
cv2.imshow('Edges', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



Простое определение порога

Пороговое значение - это метод в OpenCV, который заключается в присвоении значений пикселей по отношению к предоставленному пороговому значению. При пороговом значении каждое значение пикселя сравнивается с пороговым значением. Если значение пикселя меньше порогового значения, ему присваивается значение 0, в противном случае ему присваивается максимальное значение (обычно 255). Пороговое значение - очень популярный метод сегментации, используемый для отделения объекта, рассматриваемого как передний план, от его фона. Пороговое значение - это значение, которое имеет две области с обеих сторон, т.Е. Ниже порогового значения или выше порогового значения.

В компьютерном зрении этот метод определения порога выполняется для изображений в оттенках серого. Итак, изначально изображение должно быть преобразовано в цветовое пространство в оттенках серого.

If f (x, y) < T
  then f (x, y) = 0
else
  f (x, y) = 255

where
f (x, y) = Coordinate Pixel Value
T = Threshold Value.
В OpenCV с Python функция cv2.threshold используется для определения порогового значения.
Основной метод определения порога - двоичное определение порога. Для каждого пикселя применяется одно и то же пороговое значение. Если значение пикселя меньше порогового значения, ему присваивается значение 0, в противном случае ему присваивается максимальное значение. Различные простые методы определения порога:

- **cv2.THRESH_BINARY:** если интенсивность пикселей превышает установленный порог, значение устанавливается равным 255, в противном случае устанавливается значение 0 (черный).
- **cv2.THRESH_BINARY_INV:** инвертированный или противоположный случай cv2.THRESH_BINARY.
- **cv.THRESH_TRUNC:** если значение интенсивности пикселей превышает пороговое значение, оно усекается до порогового значения. Значения пикселей устанавливаются такими же, как и пороговое значение. Все остальные значения остаются прежними.
- **cv.THRESH_TOZERO:** для интенсивности всех пикселей установлено значение 0, меньшее порогового значения.
- **cv.THRESH_TOZERO_INV:** инвертированный или противоположный случай cv2.THRESH_TOZERO.

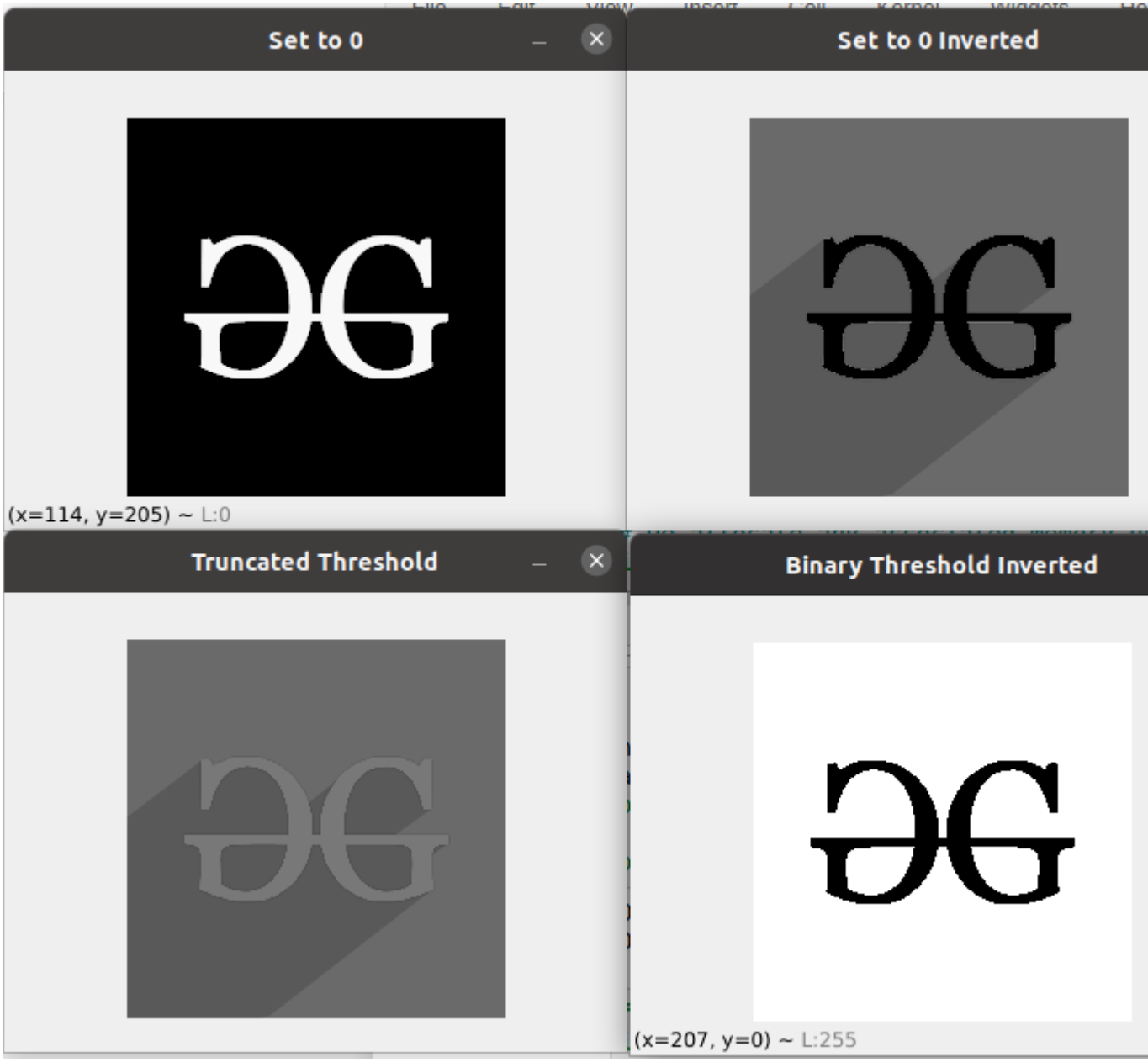
Пример: простое пороговое значение Python OpenCV

- Python

```
# Python program to illustrate
# simple thresholding type on an image

# organizing imports
import cv2
import numpy as np

# path to input image is specified and
# image is loaded with imread command
image1 = cv2.imread('geeks.png')
```

```
# cv2.cvtColor is applied over the
# image input with applied parameters
# to convert the image in grayscale
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

# applying different thresholding
# techniques on the input image
# all pixels value above 120 will
# be set to 255
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)

# the window showing output images
# with the corresponding thresholding
# techniques applied to the input images
cv2.imshow('Binary Threshold', thresh1)
cv2.imshow('Binary Threshold Inverted', thresh2)
cv2.imshow('Truncated Threshold', thresh3)
cv2.imshow('Set to 0', thresh4)
cv2.imshow('Set to 0 Inverted', thresh5)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:



Адаптивное пороговое значение

Адаптивное пороговое значение - это метод, при котором пороговое значение вычисляется для небольших областей. Это приводит к различным пороговым значениям для разных регионов в отношении изменения освещения. Для этого мы используем cv2.adaptiveThreshold.

Пример: адаптивное пороговое значение Python OpenCV

- Python

```
# Python program to illustrate
# adaptive thresholding type on an image

# organizing imports
import cv2
```

```
import numpy as np

# path to input image is specified and
# image is loaded with imread command
image1 = cv2.imread('geeks.png')

# cv2.cvtColor is applied over the
# image input with applied parameters
# to convert the image in grayscale
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

# applying different thresholding
# techniques on the input image
thresh1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                cv2.THRESH_BINARY, 199, 5)

thresh2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                cv2.THRESH_BINARY, 199, 5)

# the window showing output images
# with the corresponding thresholding
# techniques applied to the input image
cv2.imshow('Adaptive Mean', thresh1)
cv2.imshow('Adaptive Gaussian', thresh2)



# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:

Пороговое значение Otsu

При пороговом значении Otsu значение порога не выбирается, а определяется автоматически. Рассматривается бимодальное изображение (два разных значения изображения). Сгенерированная гистограмма содержит два пика. Итак, общим условием было бы выбрать пороговое значение, которое лежит в середине обоих пиковых значений гистограммы. Мы используем традиционную функцию cv2.threshold и используем cv2.THRESH_OTSU в качестве дополнительного флага.

Пример: пороговое значение Otsu для Python OpenCV

- Python

```
# Python program to illustrate
# Otsu thresholding type on an image

# organizing imports
import cv2
import numpy as np

# path to input image is specified and
# image is loaded with imread command
image1 = cv2.imread('geeks.png')

# cv2.cvtColor is applied over the
# image input with applied parameters
# to convert the image in grayscale
```

```
img = cv2.cvtColor(image1, cv2.COLOR_BGR2GRAY)

# applying Otsu thresholding
# as an extra flag in binary
# thresholding
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY +
                             cv2.THRESH_OTSU)

# the window showing output image
# with the corresponding thresholding
# techniques applied to the input image
cv2.imshow('Otsu Threshold', thresh1)

# De-allocate any associated memory usage
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

Вывод:

Размытие изображения

Размытие изображения означает, что изображение становится менее четким или отчетливым. Это делается с помощью различных ядер фильтров нижних частот. Важные типы размытия:

- **Размытие по Гауссу: размытие** по Гауссу является результатом размытия изображения с помощью функции Гаусса. Это широко используемый эффект в графическом программном обеспечении, обычно для уменьшения шума изображения и уменьшения детализации. Он также

используется в качестве этапа предварительной обработки перед применением наших моделей машинного обучения или глубокого обучения. Например. гауссовского ядра (3 × 3)

- **Медианное размытие:** Медианный фильтр - это метод нелинейной цифровой фильтрации, часто используемый для удаления шума из изображения или сигнала. Медианная фильтрация очень широко используется в цифровой обработке изображений, поскольку при определенных условиях она сохраняет границы при удалении шума. Это один из лучших алгоритмов для удаления шума соли и перца.
- **Двустороннее размытие:** двусторонний фильтр - это нелинейный сглаживающий фильтр с сохранением границ и шумоподавлением для изображений. Он заменяет интенсивность каждого пикселя средневзвешенным значением интенсивности из соседних пикселей. Этот вес может быть основан на распределении по Гауссу. Таким образом, острые края сохраняются при отбрасывании слабых.

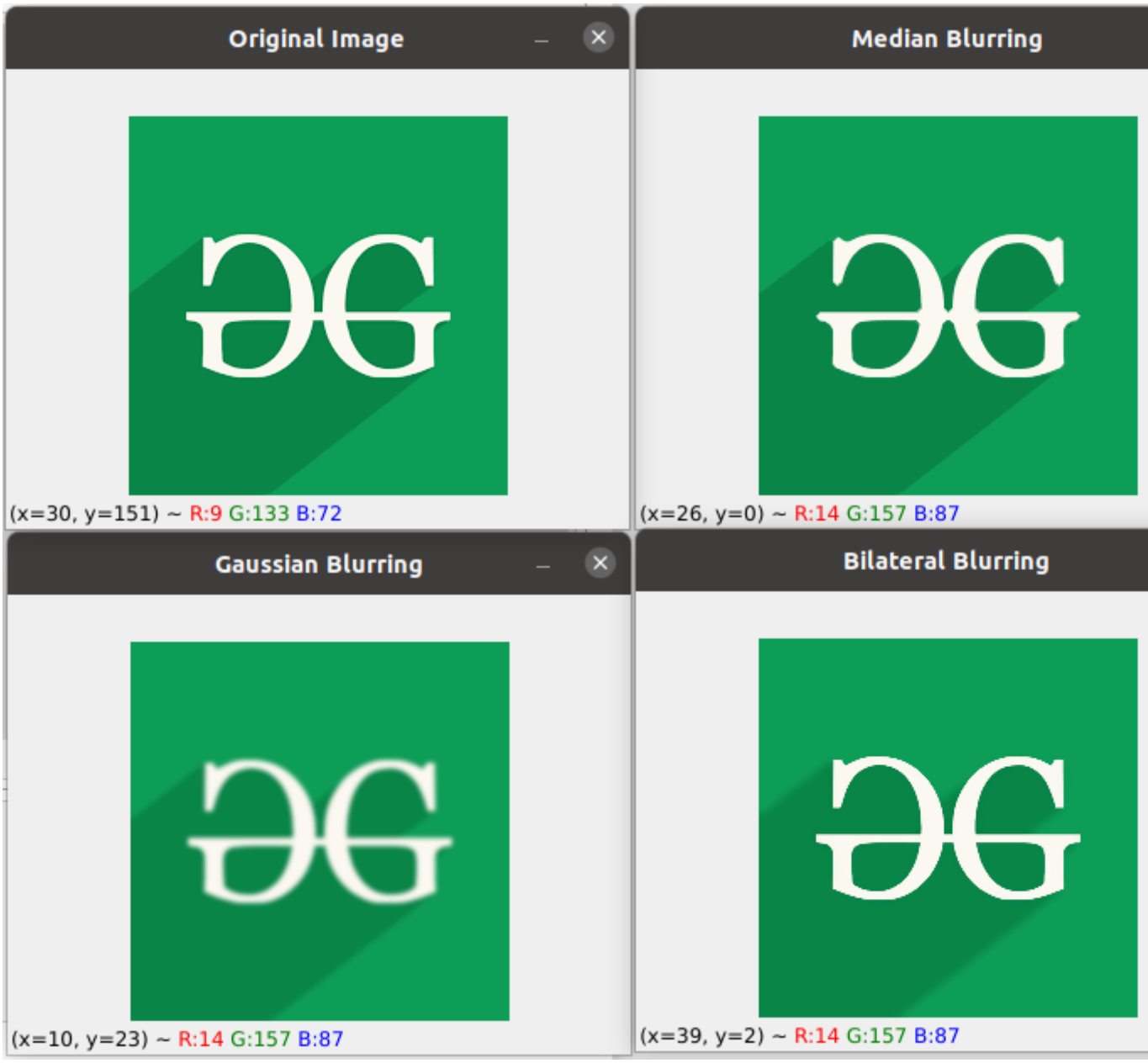
Пример: размытое изображение Python OpenCV

- Python

```
# importing libraries
import cv2
import numpy as np

image = cv2.imread('geeks.png')

cv2.imshow('Original Image', image)
cv2.waitKey(0)

# Gaussian Blur
Gaussian = cv2.GaussianBlur(image, (7, 7), 0)
cv2.imshow('Gaussian Blurring', Gaussian)
cv2.waitKey(0)

# Median Blur
median = cv2.medianBlur(image, 5)
cv2.imshow('Median Blurring', median)
cv2.waitKey(0)
```

```
# Bilateral Blur
bilateral = cv2.bilateralFilter(image, 9, 75, 75)
cv2.imshow('Bilateral Blurring', bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



Двусторонняя фильтрация

Двусторонний фильтр используется для сглаживания изображений и уменьшения шума при сохранении краев. Однако эти свертки часто приводят к потере важной информации о границе, поскольку они размывают все, независимо от того, является ли это шумом или границей. Для решения этой

проблемы был введен нелинейный двусторонний фильтр. В OpenCV есть функция bilateralFilter() со следующими аргументами:

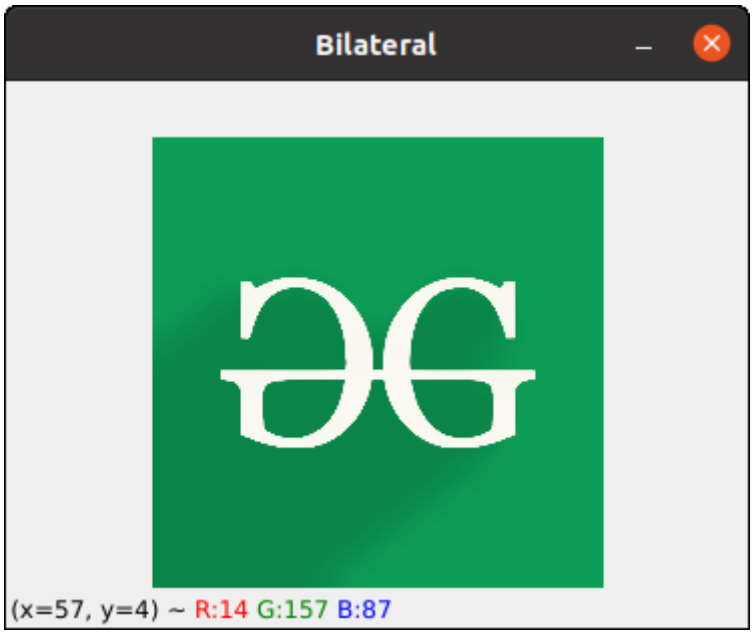
- d: диаметр каждой окрестности пикселя.
- sigmaColor: значение в цветовом пространстве. Чем больше значение, тем дальше друг от друга цвета начнут смешиваться.
- sigmaColor: значение в координатном пространстве. Чем больше его значение, тем больше дополнительных пикселей будет смешиваться вместе, учитывая, что их цвета находятся в диапазоне sigmaColor.

Пример: двустороннее изображение Python OpenCV

- Python3

```
import cv2

# Read the image
img = cv2.imread('geeks.png')

# Apply bilateral filter with d = 30,
# sigmaColor = sigmaSpace = 100
bilateral = cv2.bilateralFilter(img, 15, 100, 100)

# Save the output
cv2.imshow('Bilateral', bilateral)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



Контуры изображения

Контуры определяются как линия, соединяющая все точки вдоль границы изображения, которые имеют одинаковую интенсивность. Контуры удобны при анализе формы, определении размера интересующего объекта и обнаружении объекта. В OpenCV есть функция findContour(), которая помогает извлекать контуры из изображения. Он лучше всего работает с двоичными изображениями, поэтому сначала мы должны применить методы определения порога, ребер Собеля и т. Д.

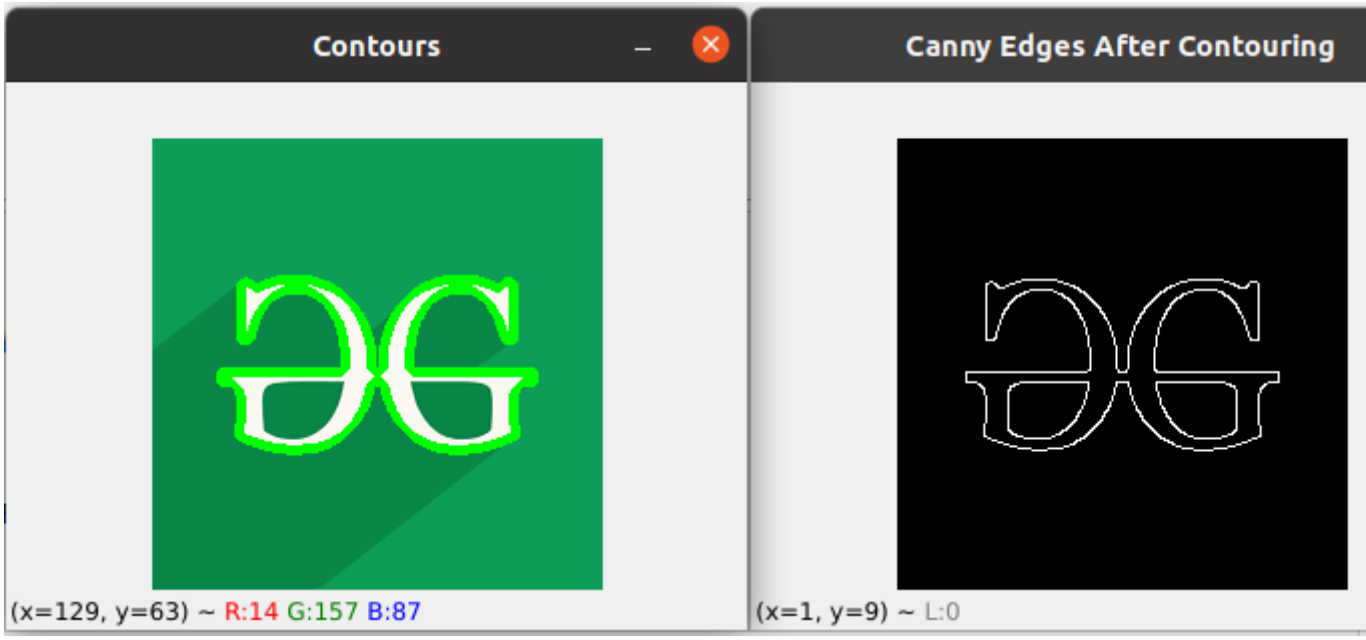
Пример: Контур изображения Python OpenCV

- Python3

```
import cv2
import numpy as np

# Let's load a simple image with 3 black squares
image = cv2.imread('geeks.png')
cv2.waitKey(0)

# Grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Find Canny edges
edged = cv2.Canny(gray, 30, 200)
cv2.waitKey(0)

# Finding Contours
# Use a copy of the image e.g. edged.copy()
# since findContours alters the image
contours, hierarchy = cv2.findContours(edged,
    cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

cv2.imshow('Canny Edges After Contouring', edged)
cv2.waitKey(0)

print("Number of Contours found = " + str(len(contours)))

# Draw all contours
# -1 signifies drawing all contours
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)
```

```
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



Размывание и расширение

Самыми основными морфологическими операциями являются две: эрозия и расширение

Основы эрозии:

- Размывает границы объекта переднего плана
- Используется для уменьшения возможностей изображения.

Работа с erosion:

- Ядро (матрица нечетного размера (3,5,7) свернуто с изображением.
- Пиксель в исходном изображении (либо 1, либо 0) будет считаться 1, только если все пиксели под ядром равны 1, в противном случае он стирается (обнуляется).
- Таким образом, все пиксели вблизи границы будут отброшены в зависимости от размера ядра.
- Таким образом, толщина или размер объекта переднего плана уменьшается или просто уменьшается белая область на изображении.

Основы расширения:

- Увеличивает площадь объекта
- Используется для выделения функций

Работа с расширением:

- Ядро (матрица нечетного размера (3,5,7) свернуто с изображением

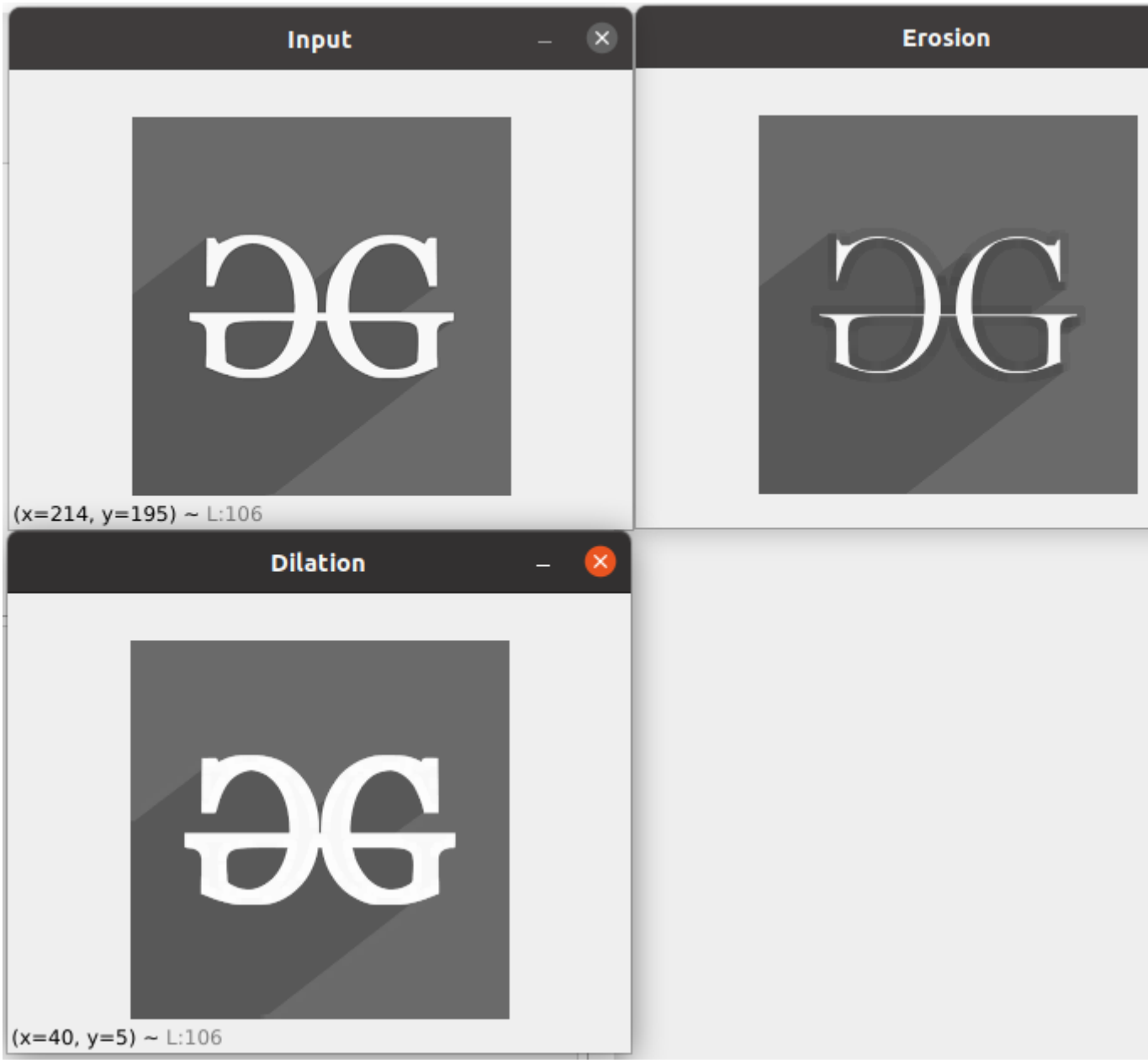
- Пиксельный элемент в исходном изображении равен ‘1’, если хотя бы один пиксель под ядром равен ‘1’.
- Это увеличивает белую область на изображении или увеличивает размер объекта переднего плана

Пример: размывание и расширение Python OpenCV

- Python

```
# Python program to demonstrate erosion and
# dilation of images.
import cv2
import numpy as np

# Reading the input image
img = cv2.imread('geeks.png', 0)

# Taking a matrix of size 5 as the kernel
kernel = np.ones((5,5), np.uint8)

# The first parameter is the original image,
# kernel is the matrix with which image is
# convolved and third parameter is the number
# of iterations, which will determine how much
# you want to erode/dilate a given image.
img_erosion = cv2.erode(img, kernel, iterations=1)
img_dilation = cv2.dilate(img, kernel, iterations=1)

cv2.imshow('Input', img)
cv2.imshow('Erosion', img_erosion)
cv2.imshow('Dilation', img_dilation)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод



Сопоставление функций

ORB - это сочетание БЫСТРОГО детектора ключевых точек и КРАТКОГО дескриптора с некоторыми дополнительными функциями для повышения производительности. FAST - это функции из ускоренного сегментного теста, используемые для обнаружения функций в предоставленном изображении. Он также использует пирамиду для создания многомасштабных функций. Теперь он не вычисляет ориентацию и дескрипторы для функций, так что именно здесь BRIEF играет роль.

ORB использует КРАТКИЕ дескрипторы, но КРАТКОЕ описание плохо работает с вращением. Итак, что делает ORB, так это поворачивает БРИФ в соответствии с ориентацией ключевых точек. Используя ориентацию патча, его матрица вращения найдена и поворачивает БРИФ, чтобы получить повернутую версию. ORB является эффективной альтернативой алгоритмам

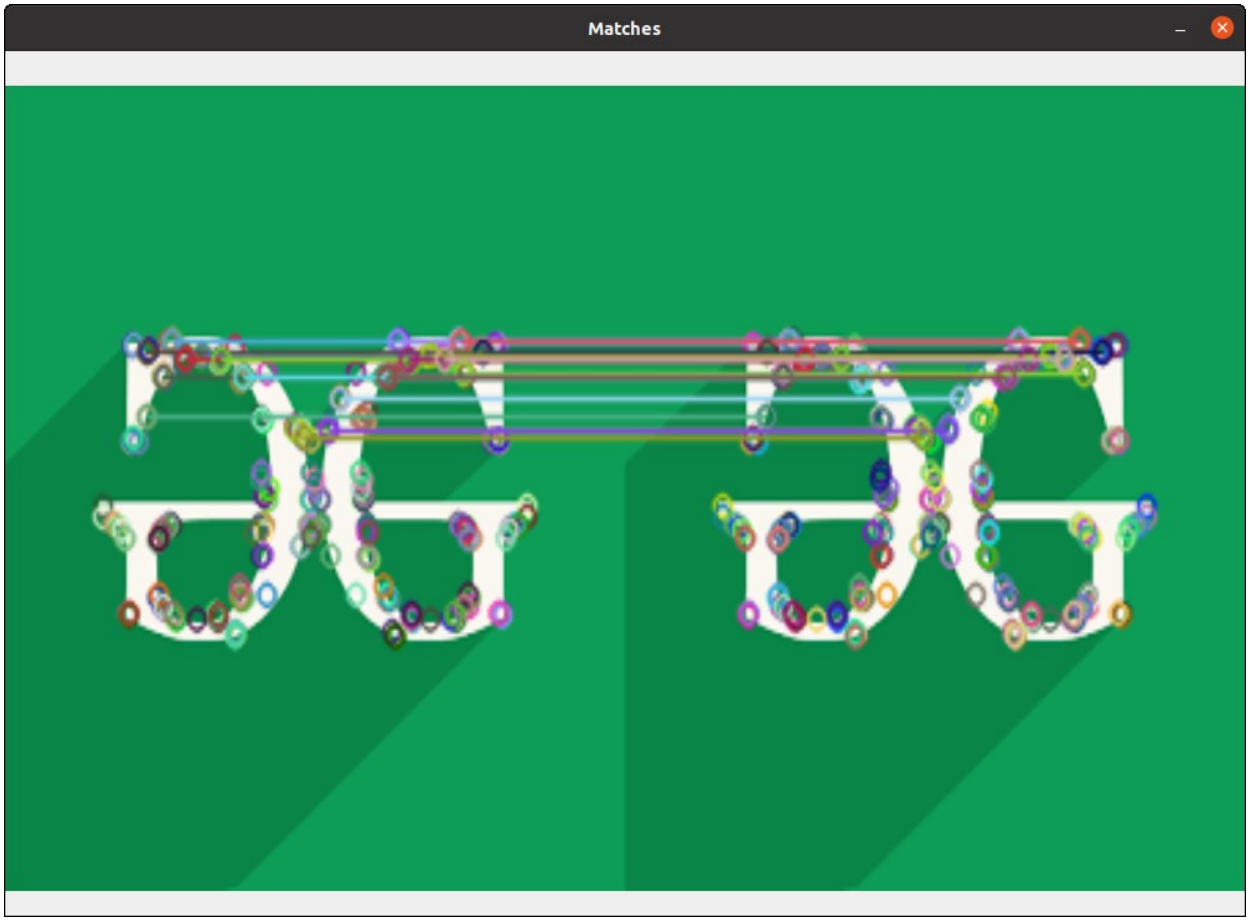
SIFT или SURF, используемым для извлечения функций, с точки зрения стоимости вычислений, соответствия производительности и, главным образом, патентов. SIFT и SURF запатентованы, и вы должны платить им за их использование. Но ORB не запатентован.

- Python

```
import numpy as np
import cv2


# Read the query image as query_img
# and train image This query image
# is what you need to find in train image
# Save it in the same directory
# with the name image.jpg
query_img = cv2.imread('geeks.png')
train_img = cv2.imread('geeks.png')

# Convert it to grayscale
query_img_bw = cv2.cvtColor(query_img,cv2.COLOR_BGR2GRAY)
train_img_bw = cv2.cvtColor(train_img, cv2.COLOR_BGR2GRAY)

# Initialize the ORB detector algorithm
orb = cv2.ORB_create()

# Now detect the keypoints and compute
# the descriptors for the query image
# and train image
queryKeypoints, queryDescriptors = orb.detectAndCompute(query_img_bw,None)
trainKeypoints, trainDescriptors = orb.detectAndCompute(train_img_bw,None)

# Initialize the Matcher for matching
# the keypoints and then match the
# keypoints
matcher = cv2.BFMatcher()
matches = matcher.match(queryDescriptors,trainDescriptors)

# draw the matches to the final image
```

```
# containing both the images the drawMatches()
# function takes both images and keypoints
# and outputs the matched query image with
# its train image
final_img = cv2.drawMatches(query_img, queryKeypoints,
train_img, trainKeypoints, matches[:20],None)

final_img = cv2.resize(final_img, (1000,650))

# Show the final image
cv2.imshow("Matches", final_img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



Рисование на изображениях

Давайте посмотрим некоторые функции рисования и нарисуем геометрические фигуры на изображениях с помощью OpenCV. Некоторые из функций рисования :

- cv2.line() : используется для рисования линии на изображении.

- cv2.rectangle() : используется для рисования прямоугольника на изображении.
- cv2.circle() : используется для рисования круга на изображении.
- cv2.putText() : используется для записи текста на изображении.

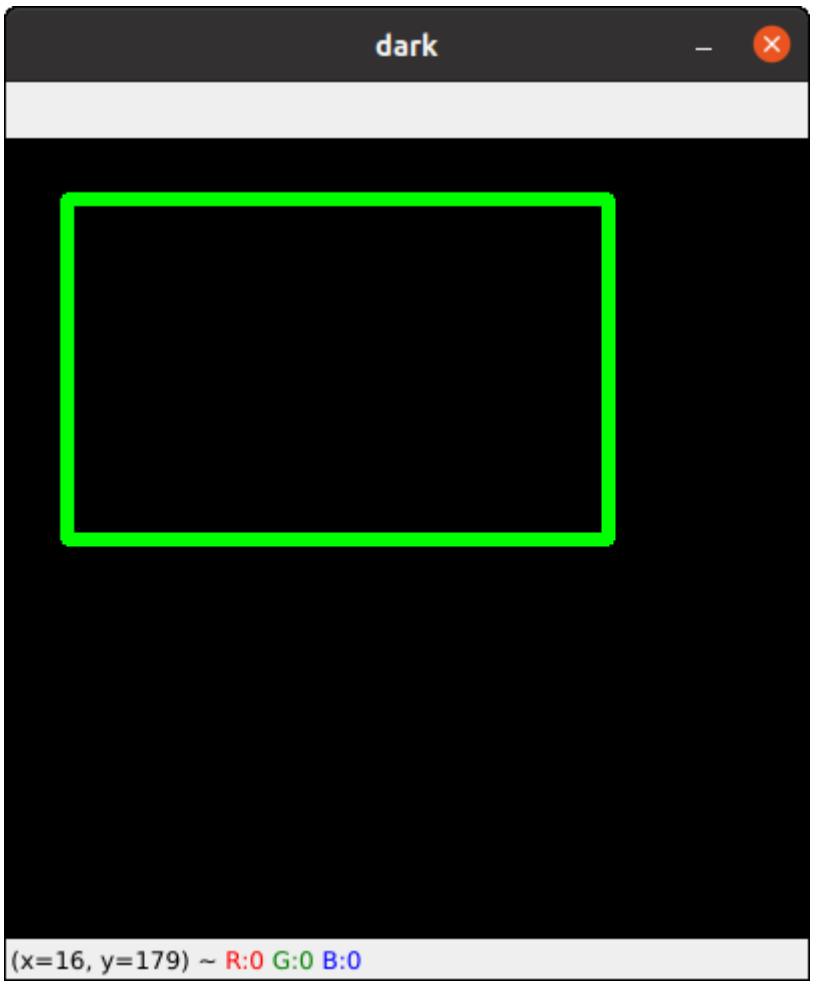
Для демонстрации использования вышеупомянутых функций понадобится изображение размером 400 X 400, заполненное сплошным цветом (в данном случае черным). Для этого можно использовать функцию numpy.zeroes для создания требуемого изображения.

Пример: Python OpenCV рисует на изображении

- Python

```
# Python3 program to draw rectangle
# shape on solid image
import numpy as np
import cv2

# Creating a black image with 3
# channels RGB and unsigned int datatype
img = np.zeros((400, 400, 3), dtype = "uint8")

# Creating rectangle
cv2.rectangle(img, (30, 30), (300, 200), (0, 255, 0), 5)

cv2.imshow('dark', img)

# Allows us to see image
# until closed forcefully
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Вывод:



При выполнении заданий студенты формируют собственные файлы в соответствии с упражнением.

Порядок и результаты их обработки представляются в отчете по практической работе.

Для получения дополнительной информации о Python OpenCV обратитесь к руководству по Python OpenCV.