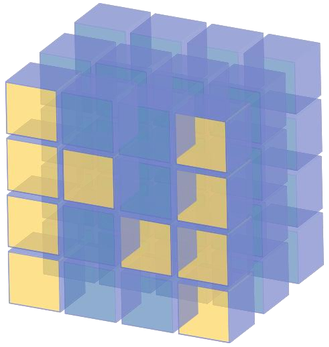
PROCESAMIENTO DIGITAL DE IMAGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Módulos principales



NumPy

matplotlib

OpenCV

pip install opencv-python

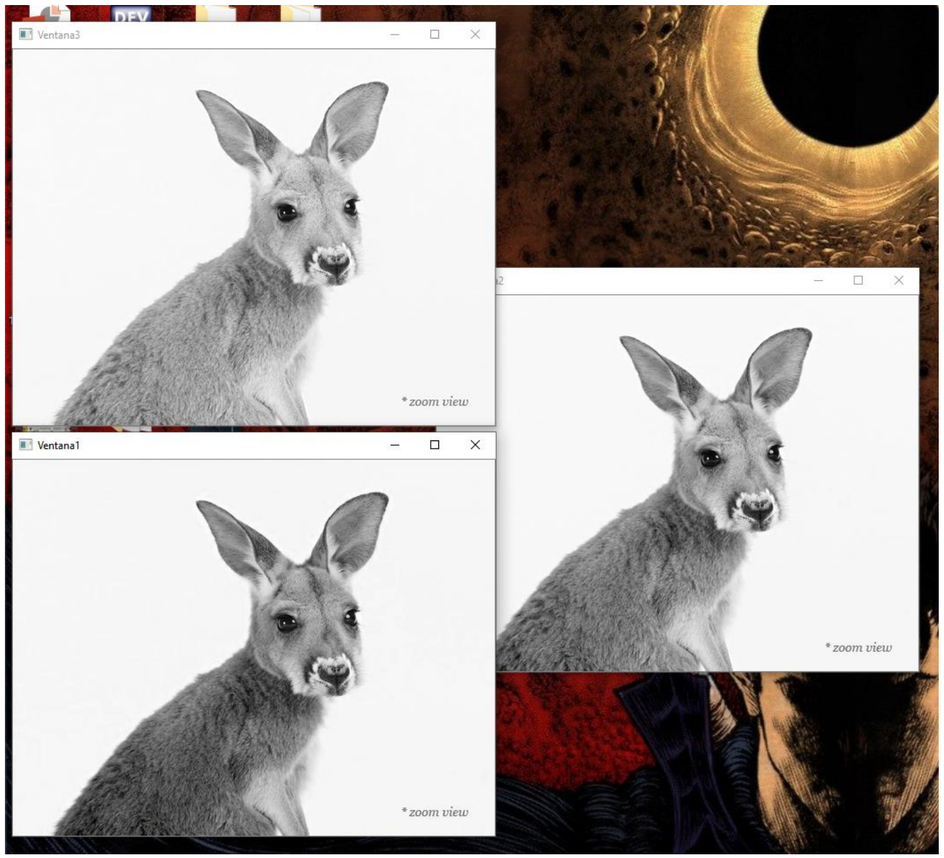
PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Dividiendo y Combinando Canales de Imagen

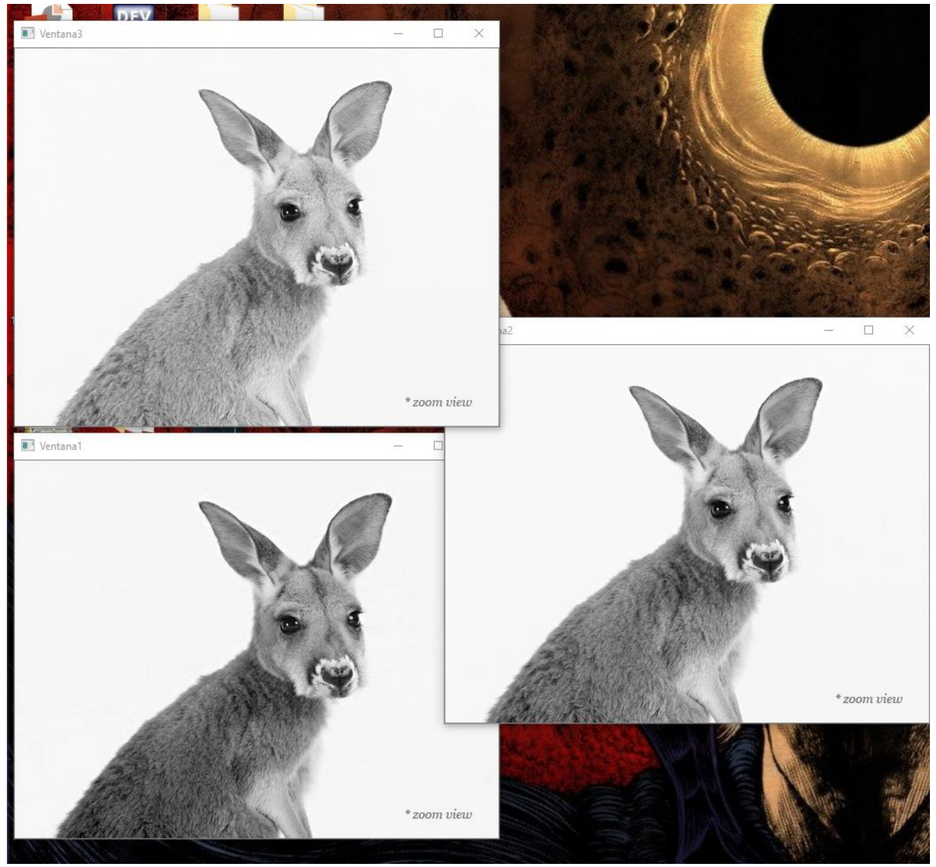
Los canales R,G,B de una imagen pueden dividirse en sus planos individuales cuando sea necesario. Luego, puede combinarse nuevamente dichos canales para nuevamente formar una imagen:

```python
import cv2

img = cv2.imread("img.jpg",1)
b,g,r = cv2.split(img)
cv2.imshow("Ventana1",b)
cv2.imshow("Ventana2",g)
cv2.imshow("Ventana3",r)
cv2.waitKey(0)
```



PROCESAMIENTO DIGITAL DE IMÁGENES

```python
import cv2

img = cv2.imread("img.jpg",1)
b = img[:,:,0]
g = img[:,:,1]
r = img[:,:,2]
cv2.imshow("Ventana1",b)
cv2.imshow("Ventana2",g)
cv2.imshow("Ventana3",r)
cv2.waitKey(0)
```



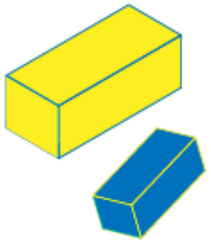
⚠ Advertencia

cv2.split() es una operación costosa (en términos de tiempo), así que sólo utilízala si es necesario. La indexación Numpy es mucho más eficiente y debería usarse siempre que sea posible.

Supongamos que quieres hacer que todos los píxeles rojos valgan cero, no necesitas dividirlos todos de esta forma y colocarlos igual a cero. Puedes simplemente usar indexación Numpy la cual es más rápida.

img[:,:,2] = 0

Operaciones Aritméticas en Imágenes



Objetivos

Aprender varias operaciones aritméticas sobre imágenes como la suma, resta, operaciones bitwise, etc.
Aprender estas funciones: ***add(), cv2.addWeighted() etc***.

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Operaciones Aritméticas en Imágenes OpcenCV con Python

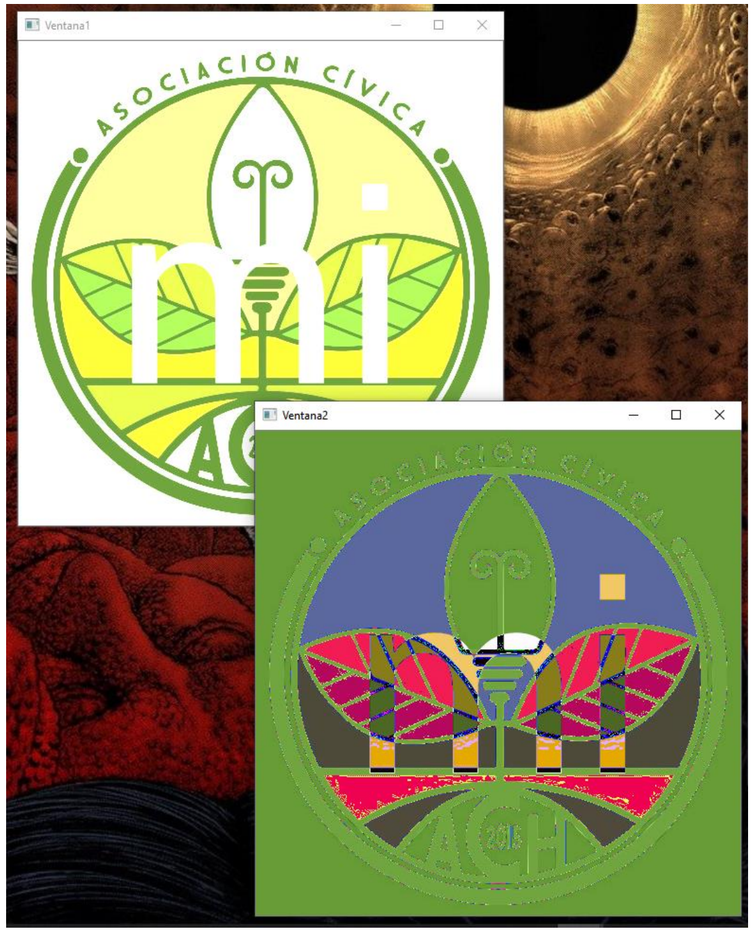
Suma de Imágenes

Puedes sumar dos imágenes usando la función de OpenCV, ***cv2.add()*** o simplemente por medio de una operación numpy, ***res = img1 + img2***. Ambas imágenes deberían tener la misma profundidad y tipo, o la segunda puede ser un valor escalar.

> ? Nota
> Existe una diferencia entre la suma OpenCV y la suma Numpy. La suma OpenCV es una operación saturada mientras que la suma Numpy es una operación modular.

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO




```python
import cv2

img1 = cv2.imread("img1.png",1)
img2 = cv2.imread("img2.png",1)
imgx = cv2.add(img1,img2)
cv2.imshow("Ventana1",imgx)
imgy = img1 + img2
cv2.imshow("Ventana2",imgy)
cv2.waitKey(0)
```

La función de OpenCV proveerá un mejor resultado. Así que siempre procura adherirte a las funciones de OpenCV.

Fusión de Imágenes

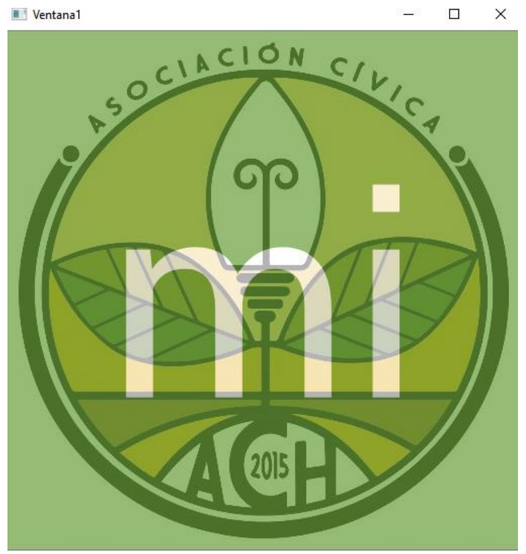
Esto también es suma de imágenes, pero con diferentes pesos lo cual da una sensación de mezcla o transparencia. Las imágenes son sumadas mediante esta ecuación:

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

Aquí tomo dos imágenes para fusionarlas. A la primera le doy un peso de 0.7 y a la segunda uno de 0.3. ***cv2.addWeighted()*** aplico la siguiente ecuación sobre la imagen.

Aquí gamma es tomado como cero.

```
import cv2

img1 = cv2.imread("img1.png",1)
img2 = cv2.imread("img2.png",1)
imgx = cv2.addWeighted(img1,0.7,img2,0.3,0)
cv2.imshow("Ventana1",imgx)
cv2.waitKey(0)
```



PROCESAMIENTO DIGITAL DE IMÁGENES

Las operaciones bit a bit se utilizan en la manipulación de imágenes y se utilizan para extraer partes esenciales de la imagen. En este artículo, las operaciones bit a bit que se utilizan son:

1. Y
2. O
3. XOR
4. NO

Además, las operaciones bit a bit ayudan en el enmascaramiento de imágenes. La creación de imágenes se puede habilitar con la ayuda de estas operaciones. Estas operaciones pueden resultar útiles para mejorar las propiedades de las imágenes de entrada.

NOTA: Las operaciones bit a bit deben aplicarse en imágenes de entrada de las mismas dimensiones

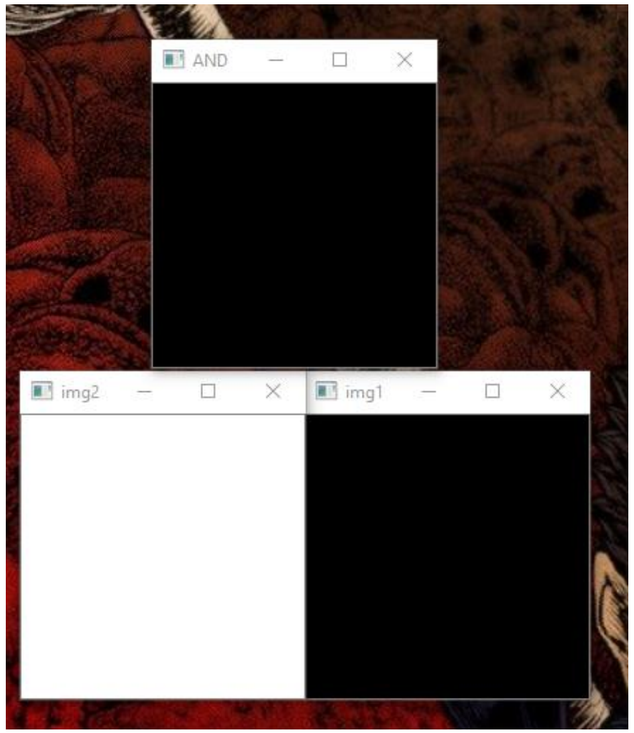
Operación AND bit a bit en la imagen:

Conjunción de bits de elementos de matriz de entrada.

Sintaxis: *cv2.bitwise_and (fuente1, fuente2, dst, máscara)*

Parámetros:

fuente1: matriz de la primera imagen de entrada (canal único, 8 bits o punto flotante)
fuente2: matriz de la segunda imagen de entrada (canal único, 8 bits o punto flotante)
dst: matriz de salida (similar a las dimensiones y tipo de matriz de imagen de entrada)
máscara: máscara de operación, máscara de canal único de entrada / salida de 8 bits




```python
import cv2
import numpy as np
img1 = np.zeros((200,200),dtype = np.uint8)
img2 = np.ones((200,200),dtype = np.uint8)
img2 = img2*255
AND = cv2.bitwise_and(img1,img2)
cv2.imshow("img1",img1)
cv2.imshow("img2",img2)
cv2.imshow("AND",AND)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Operación AND bit a bit en la imagen:

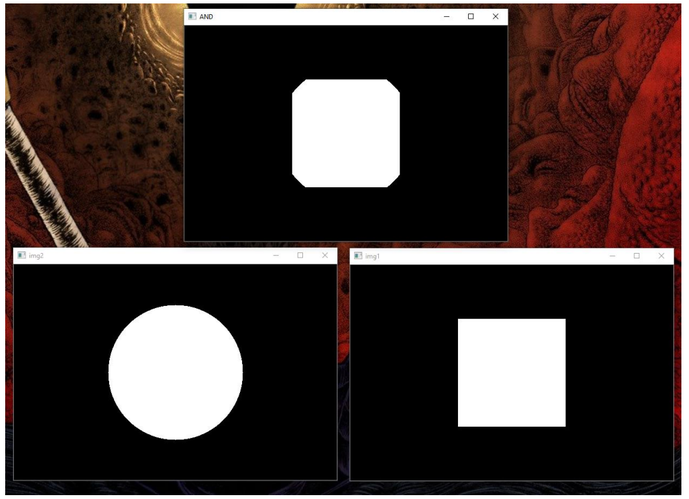
Conjunción de bits de elementos de matriz de entrada.

Sintaxis: *cv2.bitwise_and (fuente1, fuente2, dst, máscara)*

Parámetros:

fuente1: matriz de la primera imagen de entrada (canal único, 8 bits o punto flotante)
fuente2: matriz de la segunda imagen de entrada (canal único, 8 bits o punto flotante)
dst: matriz de salida (similar a las dimensiones y tipo de matriz de imagen de entrada)
máscara: máscara de operación, máscara de canal único de entrada / salida de 8 bits

```python
import cv2
import numpy as np
img1 = np.zeros((400,600),dtype = np.uint8)
img1[100:300,200:400]=255
img2 = np.zeros((400,600),dtype = np.uint8)
img2 = cv2.circle(img2,(300,200),125,(255),-1)

AND = cv2.bitwise_and(img1,img2)
cv2.imshow("AND",AND)
cv2.imshow("img1",img1)
cv2.imshow("img2",img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



Operación OR bit a bit en la imagen:

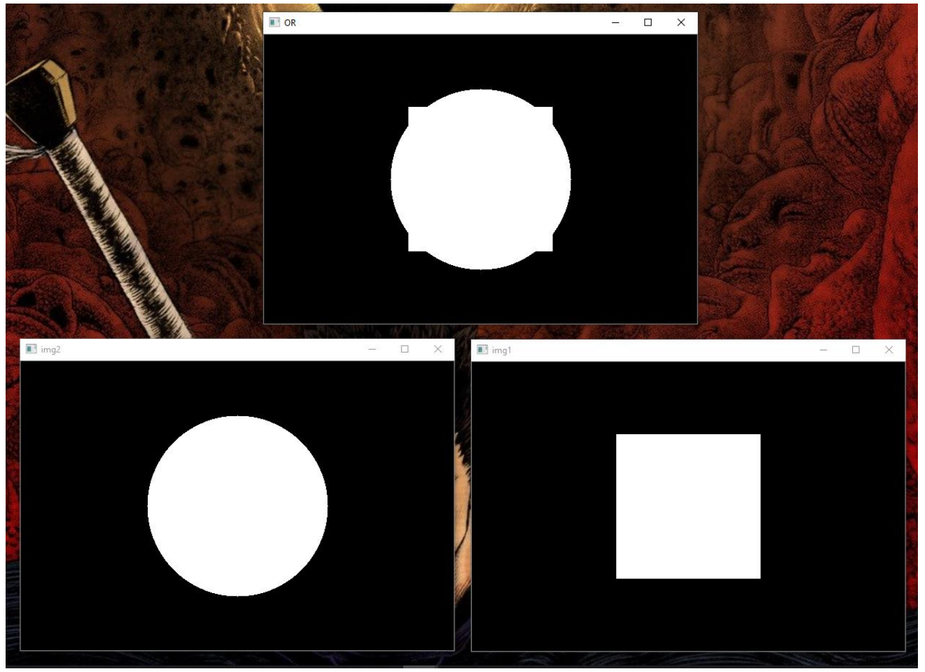
Disyunción bit a bit de los elementos de la matriz de entrada.

Sintaxis: *cv2.bitwise_or (source1, source2, dst, mask)*

Parámetros:

fuente1: matriz de la primera imagen de entrada (canal único, 8 bits o punto flotante)
fuente2: matriz de la segunda imagen de entrada (canal único, 8 bits o punto flotante)
dst: matriz de salida (similar a las dimensiones y tipo de matriz de imagen de entrada)
máscara: máscara de operación, máscara de canal único de entrada / salida de 8 bits

```python
import cv2
import numpy as np
img1 = np.zeros((400,600),dtype = np.uint8)
img1[100:300,200:400]=255
img2 = np.zeros((400,600),dtype = np.uint8)
img2 = cv2.circle(img2,(300,200),125,(255),-1)

OR = cv2.bitwise_or(img1,img2)
cv2.imshow("OR",OR)
cv2.imshow("img1",img1)
cv2.imshow("img2",img2)
cv2.waitKey(0)
cv2.destroyAllWindows()
```



PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO

Operación NO bit a bit en la imagen:

Inversión de elementos de matriz de entrada.

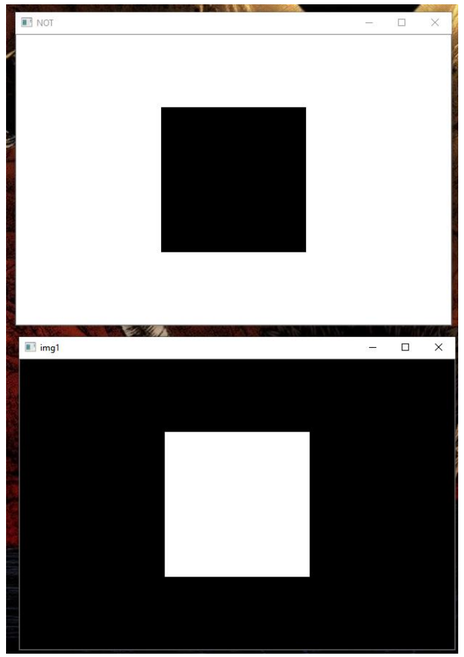
Sintaxis: *cv2.bitwise_not (origen, destino, máscara)*

Parámetros:

fuente: matriz de imagen de entrada (canal único, 8 bits o punto flotante)

dest: matriz de salida (similar a las dimensiones y el tipo de matriz de imagen de entrada)

máscara: máscara de operación, canal único de entrada / salida de 8 bits máscara




```python
import cv2
import numpy as np
img1 = np.zeros((400,600),dtype = np.uint8)
img1[100:300,200:400]=255

NOT = cv2.bitwise_not(img1)
cv2.imshow("NOT",NOT)
cv2.imshow("img1",img1)

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

PROCESAMIENTO DIGITAL DE IMÁGENES

UMAKER | CENTRO DE CAPACITACIÓN DE DESARROLLO TECNOLÓGICO