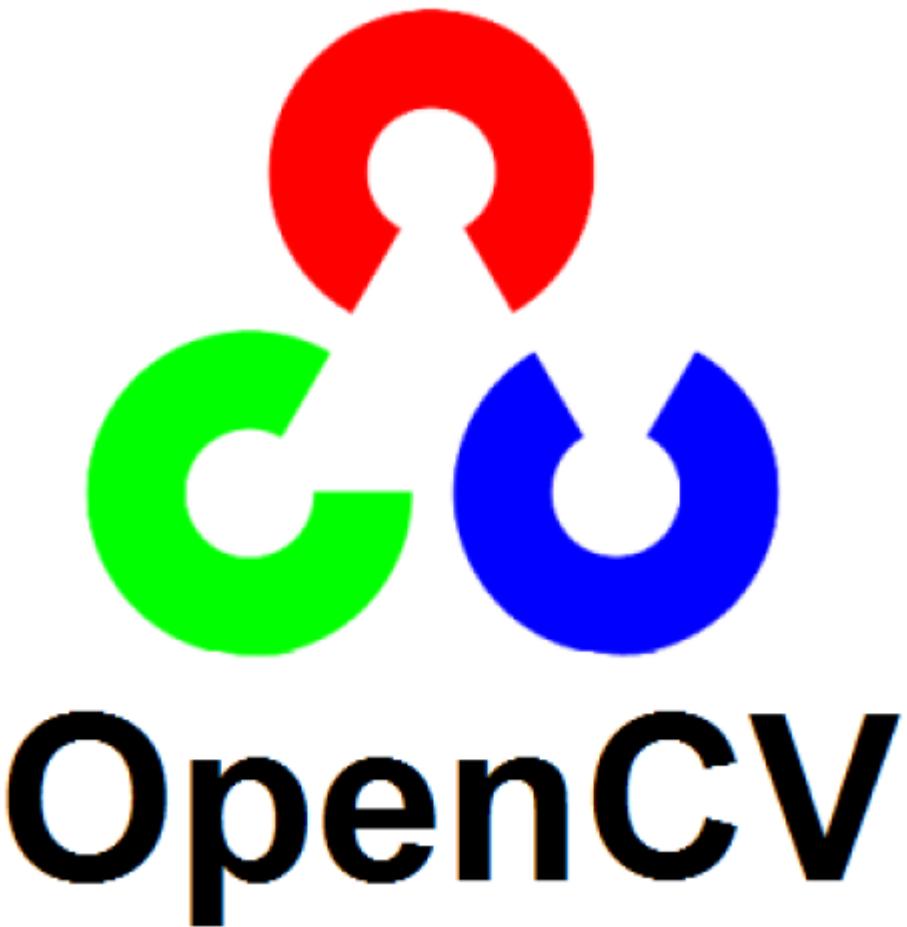


Introducción a la Visión por Computador

Noviembre 2020



1. Imágenes y Matrices

La estructura más importante en una visión artificial es, sin duda, las imágenes. La imagen en visión artificial es una representación del mundo físico capturado con un dispositivo digital. Esta imagen es solo una secuencia de números almacenados en un formato de matriz, como se muestra en la siguiente imagen. Cada número es una medida de la intensidad de la luz para la longitud de onda considerada (por ejemplo, en rojo, verde o azul en imágenes en color) o para un rango de longitud de onda (para dispositivos pancromáticos). Cada punto de una imagen se llama píxel (para un elemento de imagen), y cada píxel puede almacenar uno o más valores dependiendo de si es una imagen gris, negra o blanca (también llamada imagen binaria) que almacena solo uno valor, como 0 o 1, una imagen de nivel de escala de grises que puede almacenar solo un valor, o una imagen en color que puede almacenar tres valores. Estos valores suelen ser números enteros entre 0 y 255, pero puede usar el otro rango. Por ejemplo, 0 a 1 en números de coma flotante como HDRI (imágenes de alto rango dinámico) o imágenes térmicas. La imagen es almacenado en un formato matricial, donde cada pixel tiene una posición y puede ser 1 referenciado por su fila y columna. En el caso de una imagen en escala de grises una matriz simple representa la imagen

159	165	185	187	185	190	189	198	193	197	184	152	123
174	167	186	194	185	196	204	191	200	178	149	129	125
168	184	185	188	195	192	191	195	169	141	116	115	129
178	188	190	195	196	199	195	164	128	120	118	126	135
188	194	189	195	201	196	166	114	113	120	128	131	129
187	200	197	198	190	144	107	106	113	120	125	125	125
198	195	202	183	134	98	97	112	114	115	116	116	118
194	206	178	111	87	99	97	101	107	105	101	97	95
206	168	107	82	80	100	102	91	98	102	104	99	72
160	97	80	86	80	92	80	79	71	74	81	81	64
98	66	76	86	76	83	72	71	55	53	61	61	56
60	76	74	70	67	64	63	60	55	49	54	52	54

En el caso de una imagen a color, como mostramos en la figura, sera representada por una matriz de ancho * alto * numero de colores (RGB)

2. Introducción a OpenCV

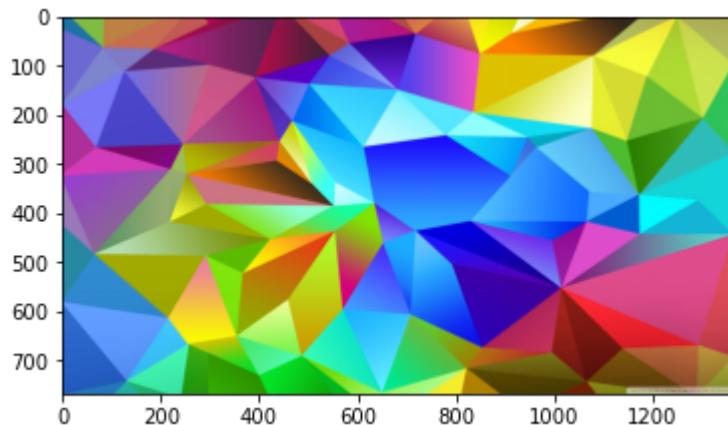
2.1 Leer, escribir y mostrar una imagen

OpenCV provee las funciones imread() e imwrite() que soportan varios formatos de archivos para trabajar con imágenes, la imagen debe estar en el directorio de trabajo actual o se debe pasar la dirección absoluta o relativa de la imagen.

imread()

```
In [1]: #importar Librería vc
import cv2 as cv
import matplotlib.pyplot as plt
#Lectura de una imagen
img = cv.imread("imagen1.jpg")
#mostrar imagen
plt.imshow(img)
```

Out[1]: <matplotlib.image.AxesImage at 0x1126844fe20>



Por defecto imread() retorna una imagen en formato de color BGR, incluso si usamos una imagen en formato de escala de grises. BGR representa el mismo espacio de color que RGB pero el orden de byte esta invertido.

Modos de imread() (Banderas)

IMREAD_UNCHANGED Python: cv.IMREAD_UNCHANGED Si se establece, devuelve la imagen cargada como está (con canal alfa; de lo contrario, se recorta). Ignore la orientación EXIF.

IMREAD_GRAYSCALE Python: cv.IMREAD_GRAYSCALE Si está configurado, siempre convierta la imagen a la imagen en escala de grises de un solo canal (conversión interna del códec).

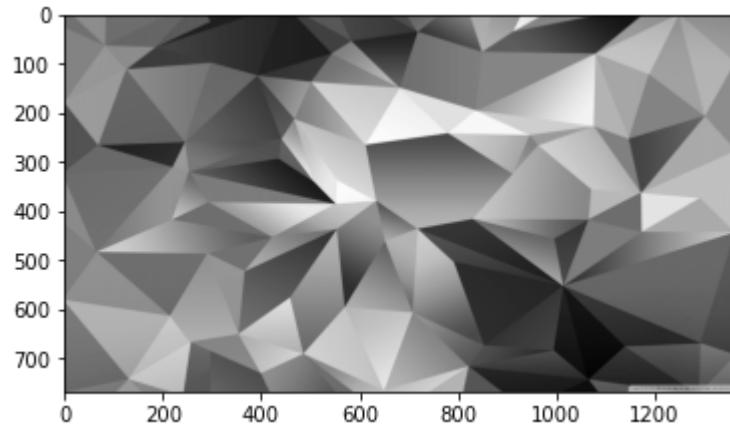
IMREAD_COLOR Python: cv.IMREAD_COLOR Si está configurado, siempre convierta la imagen a la imagen en color BGR de 3 canales.

IMREAD_ANYCOLOR Python: cv.IMREAD_ANYCOLOR Si se establece, la imagen se lee en cualquier formato de color posible.

cv.IMREAD_GRAYSCALE

```
In [4]: import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("imagen1.jpg", cv.IMREAD_GRAYSCALE)
plt.imshow(img, cmap="gray")
```

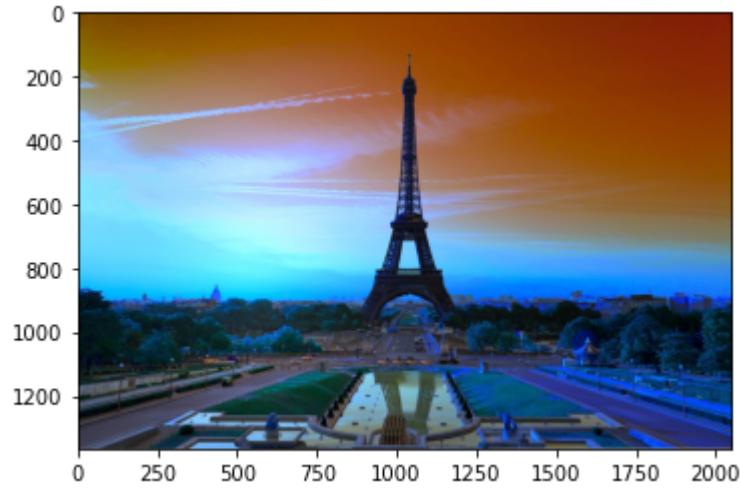
Out[4]: <matplotlib.image.AxesImage at 0x112686f58b0>



cv. IMREAD_COLOR

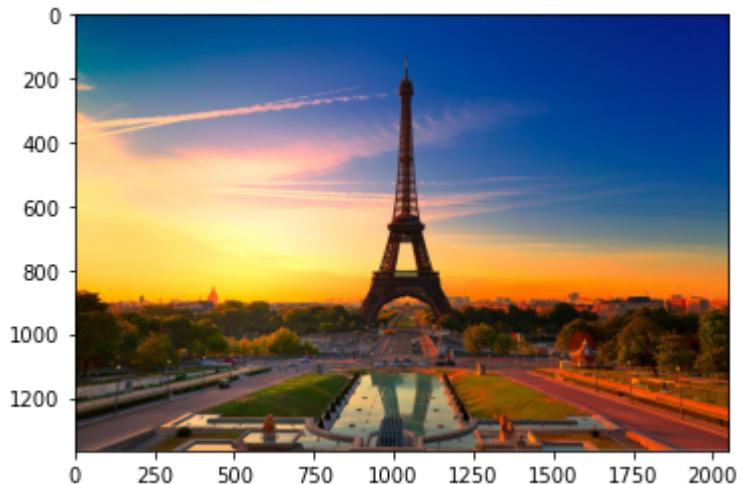
```
In [9]: import cv2 as cv
import matplotlib.pyplot as plt
img = cv.imread("imagen2.jpg", cv.IMREAD_COLOR)
plt.imshow(img, cmap="gray")
```

Out[9]: <matplotlib.image.AxesImage at 0x1126b22eeb0>



```
In [10]: #Transformar de BGR a RGB
img = cv.imread("Imagen2.jpg", cv.IMREAD_COLOR)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img)
```

Out[10]: <matplotlib.image.AxesImage at 0x1126b578910>



imshow()

```
In [13]: import cv2 as cv
img = cv.imread("Imagen1.jpg")
#muestra la imagen en una ventana
cv.imshow("Imagen",img)
#cierra la imagen
cv.waitKey(0)
cv.destroyAllWindows()
```

imwrite()

La función imwrite nos permite escribir en disco o guardar una imagen, le pasamos como argumento la ruta donde guardar con el nombre a guardar y la extensión, y como segundo argumento la imagen a guardar si todo sale sin errores devuelve True

```
In [2]: import cv2 as cv
img = cv.imread("Imagen1.jpg")
img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
cv.imwrite("result/Imagen1gray.jpg",img_gray)
```

Out[2]: True

2.2 Lectura y Escritura de Archivos de Video

OpenCV dispone de las clases **VideoCapture()** y **VideoWriter()** que soporta varios formatos de archivos de video. Los formatos admitidos varían según el sistema, pero siempre deben incluir AVI. A través de su método `read()`, una clase de `VideoCapture` puede sondarse para nuevos cuadros hasta llegar al final de su archivo de video. Cada cuadro es una imagen en formato BGR.

Reproducir video desde un archivo

```
In [15]: import numpy as np
import cv2 as cv

# crear un objeto VideoCapture su parametro sera La direccion del video
cap = cv.VideoCapture("videoTBT.mp4")
# creamos un bucle

while(cap.isOpened()):

    ret,frame = cap.read()
    if ret:

        cv.imshow("Video Motor a escala", frame)

        if(cv.waitKey(10) & 0xFF ==ord("q")):
            break

    else:
        break
cap.release()
cv.destroyAllWindows()
```

Acceder a la cámara

```
In [16]: import numpy as np
import cv2 as cv

# crear un objeto VideoCapture su parametro sera la direccion del video (0 para una webcam)
cap = cv.VideoCapture(0)
# creamos un bucle

while(cap.isOpened()):

    ret,frame = cap.read()
    if ret:

        cv.imshow("Video Motor a escala", frame)

        if(cv.waitKey(10) & 0xFF ==ord("q")):
            break

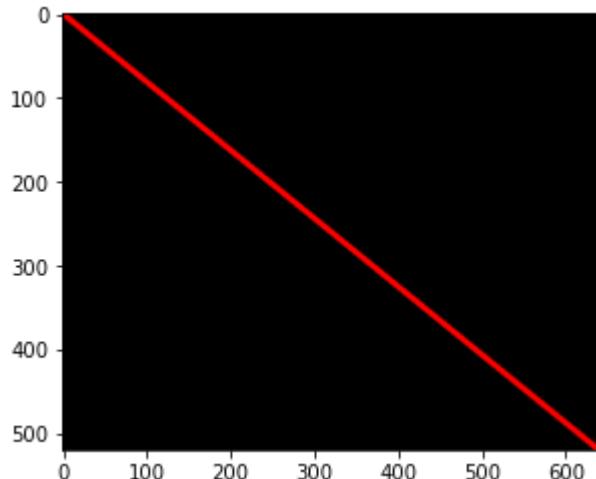
    else:
        break
cap.release()
cv.destroyAllWindows()
```

Funciones para dibujar figuras geométricas

Línea

```
In [5]: #función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar la linea
img = cv.line(img,(0,0),(640,520),(255,0,0),6)
#mostrar imagen
plt.imshow(img)
```

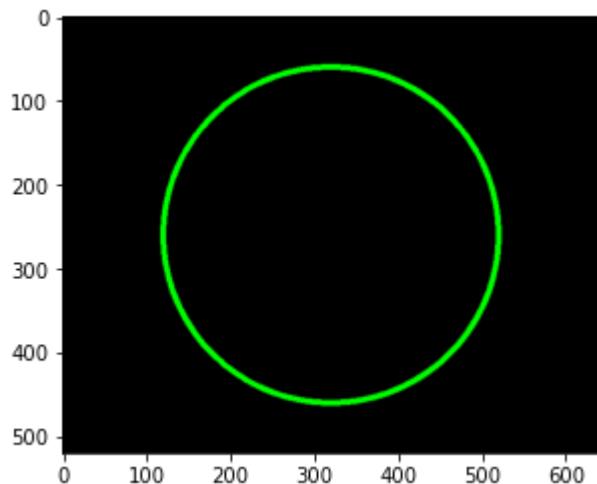
Out[5]: <matplotlib.image.AxesImage at 0x21461bc1940>



Dibujar un círculo

```
In [9]: #función cv.circle()
#parámetros: img, coordenadas centro, radio, color, grosor
#función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar círculo
img = cv.circle(img,(320,260),200,(0,255,0),5)
#mostrar imagen
plt.imshow(img)
```

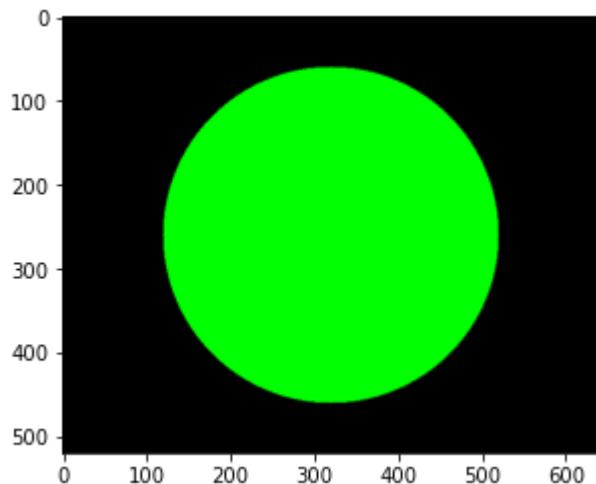
Out[9]: <matplotlib.image.AxesImage at 0x21461d7c400>



Círculo relleno

```
In [17]: #función cv.circle()
#parámetros: img, coordenadas centro, radio, color, grosor
#función cv.line(imagen, coordenadas iniciales,
#coordenadas finales, color, grosor)
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar círculo (-1 para llenar el círculo)
img = cv.circle(img,(320,260),200,(0,255,0),-1)
#mostrar imagen
plt.imshow(img)
```

Out[17]: <matplotlib.image.AxesImage at 0x1126bbcc910>

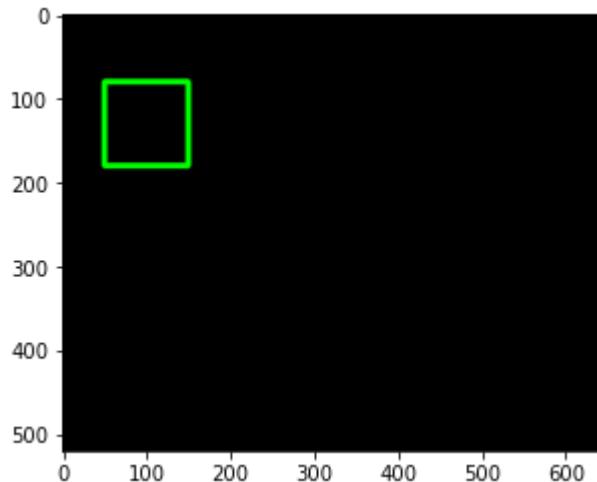


Dibujar rectángulo

```
In [10]: #función cv.rectangle()
#parámetros: img, coordenadas esquina superior izq, coordenada esquina inf de
r, color, grosor

import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
#crear una imagen
img = np.zeros((520,640,3), np.uint8)
#dibujar rectangulo
img = cv.rectangle(img,(50,80),(150,180),(0,255,0),5)
#mostrar imagen
plt.imshow(img)
```

Out[10]: <matplotlib.image.AxesImage at 0x21461dcafa0>



Captura de eventos del Mouse

OpenCV permite que las ventanas con nombre se creen, redibujen y destruyan usando las funciones **namedWindow()**, **imshow()** y **destroyWindow()**. Además, cualquier ventana puede capturar la entrada del teclado a través de la función **waitKey ()** y la entrada del mouse a través de la función **setMouseCallback()**. Veamos un ejemplo donde mostramos cuadros de entrada de cámara en vivo

Función setMouseCallback()

```
In [13]: import cv2 as cv
clicked = False
#funcion encargada de capturar el mouse
def onMouse(evento,x,y,flags,param):
    #variable global
    global clicked

    if(evento == cv.EVENT_LBUTTONUP):
        clicked = True
#crear objeto video cv.VideoCapture()
camCap = cv.VideoCapture(0)
cv.namedWindow("MyWindow")
#configurar envío de eventos a función onMouse
cv.setMouseCallback("MyWindow",onMouse)

print("Mostrar estado cámara, click en la ventana para detener")
ret,frame = camCap.read()

while(ret and cv.waitKey(1)==-1 and not clicked):
    cv.imshow("MyWindow", frame)
    ret, frame = camCap.read()
camCap.release()
cv.destroyAllWindows()
```

Mostrar estado cámara, click en la ventana para detener

El argumento para waitKey () es un número de milisegundos para esperar la entrada del teclado. El valor de retorno es -1 (lo que significa que no se ha presionado ninguna tecla) o un código de tecla ASCII, como 27 para Esc. Podemos enumerar todos los eventos del mouse disponibles con el siguiente código:

```
In [18]: import cv2 as cv
events = [i for i in dir(cv) if 'EVENT' in i ]
events
```

```
Out[18]: ['EVENT_FLAG_ALTKEY',
 'EVENT_FLAG_CTRLKEY',
 'EVENT_FLAG_LBUTTON',
 'EVENT_FLAG_MBUTTON',
 'EVENT_FLAG_RBUTTON',
 'EVENT_FLAG_SHIFTKEY',
 'EVENT_LBUTTONDOWNDBLCLK',
 'EVENT_LBUTTONDOWN',
 'EVENT_LBUTTONUP',
 'EVENT_MBUTTONDOWNDBLCLK',
 'EVENT_MBUTTONDOWN',
 'EVENT_MBUTTONUP',
 'EVENT_MOUSEHWHEEL',
 'EVENT_MOUSEMOVE',
 'EVENT_MOUSEWHEEL',
 'EVENT_RBUTTONDOWNDBLCLK',
 'EVENT_RBUTTONDOWN',
 'EVENT_RBUTTONUP']
```

Dibujar circulos en donde se haga click

```
In [ ]: import numpy as np
import cv2 as cv
def draw_circle(event, x, y, flags, param):

    if(event == cv.EVENT_LBUTTONDOWN):
        cv.circle(img, (x, y), 25, (255,0,0), -1 )

img = np.zeros((512,512,3), np.uint8)

cv.namedWindow('Dibujando Circulos')
cv.setMouseCallback('Dibujando Circulos', draw_circle)

while(True):
    cv.imshow('Dibujando Circulos',img)
    if (cv.waitKey(1) & 0xFF ==ord('q')):
        break

cv.destroyAllWindows()
```

TAREA 1

Realizar un ejercicio capturando eventos con el mouse donde se pulse el Botón izquierdo dibuje un circulo, pulsamos el botón derecho dibuja un rectángulo

```
In [30]: import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
#Función que dibuja círculos o rectángulos
def draw_form(event, x, y, flags, param):

    #cuando se presione el botón izquierdo (círculo)
    if(event == cv.EVENT_LBUTTONDOWN):
        cv.circle(img, (x, y), 25, (0,200,255), -1 )

    #cuando se presione el botón derecho (rectángulo)
    if(event == cv.EVENT_RBUTTONDOWN):
        cv.rectangle(img, (x-25, y-25), (x+25,y+25),(100,200,0),-1)

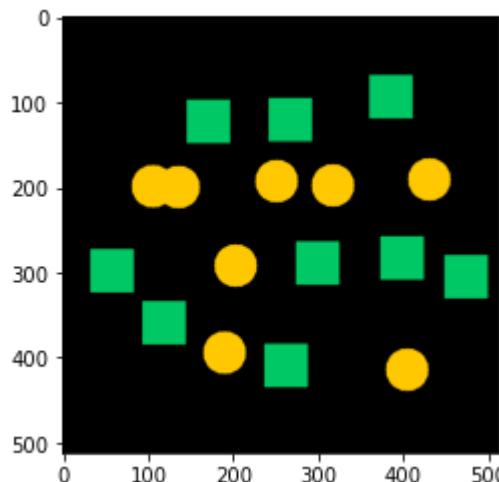
    #crea una imagen
img = np.zeros((512,512,3), np.uint8)

cv.namedWindow('TAREA 1')
cv.setMouseCallback('TAREA 1', draw_form)

while(True):
    cv.imshow('TAREA 1',img)
    if (cv.waitKey(1) & 0xFF ==ord('q')):
        break

cv.destroyAllWindows()
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
plt.imshow(img)
```

Out[30]: <matplotlib.image.AxesImage at 0x21464257df0>



3. Operaciones con Imágenes

Las operaciones con imágenes es la implementación de operaciones aritméticas estándar, como suma, resta, multiplicación y división, en imágenes. Estas operaciones tienen muchos usos en el procesamiento de imágenes, por ejemplo: la resta de imagen se puede utilizar para detectar diferencias entre dos o más imágenes de la misma escena u objeto.

3.1 Operaciones básicas

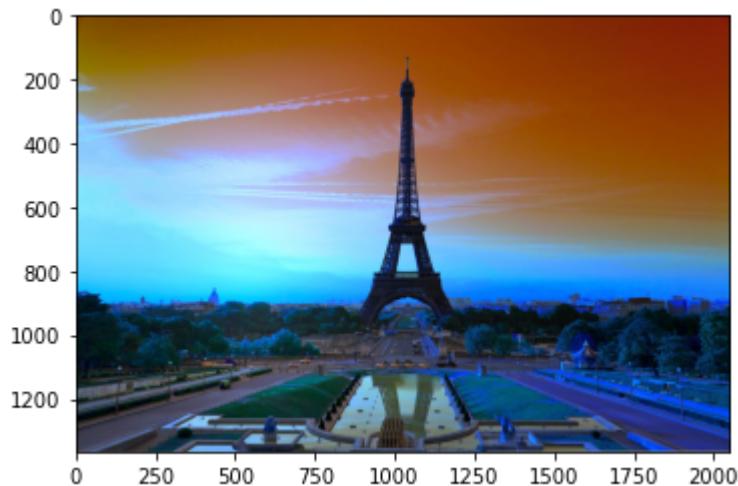
La mayoría de operaciones con imágenes están relacionadas principalmente con la librería numpy en lugar de OpenCV. Se requiere de conocimientos básicos en su uso para poder entender y aplicar de mejor manera las operaciones.

Acceso y modificación de pixeles

Puede acceder a un valor de píxel por sus coordenadas de fila y columna. Para la imagen BGR, devuelve una matriz de valores azules, verdes y rojos. Para la imagen en escala de grises, solo se devuelve la intensidad correspondiente entre 0 y 255 siendo 0 igual a negro y 255 equivalente a blanco.

```
In [19]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("imagen2.jpg")
plt.imshow(img)
```

```
Out[19]: <matplotlib.image.AxesImage at 0x1126b9099a0>
```



```
In [28]: img = cv.imread("imagen2.jpg")
#acceder al valor gbr del pixel
px = img[100,100]
px
```

```
Out[28]: array([137, 94, 1], dtype=uint8)
```

```
In [29]: img = cv.imread("imagen2.jpg")
#Acceder al canal B de bgr
px = img[100,100,0]
px
```

```
Out[29]: 137
```

```
In [30]: img = cv.imread("imagen2.jpg")
#acceder al tercer canal R de bgr
px = img[100,100,2]
px
```

```
Out[30]: 1
```

Modificar valores de pixeles

```
In [31]: img = cv.imread("imagen2.jpg")
#modificación
img[100,100]=[255,255,255]
img[100,100]
```

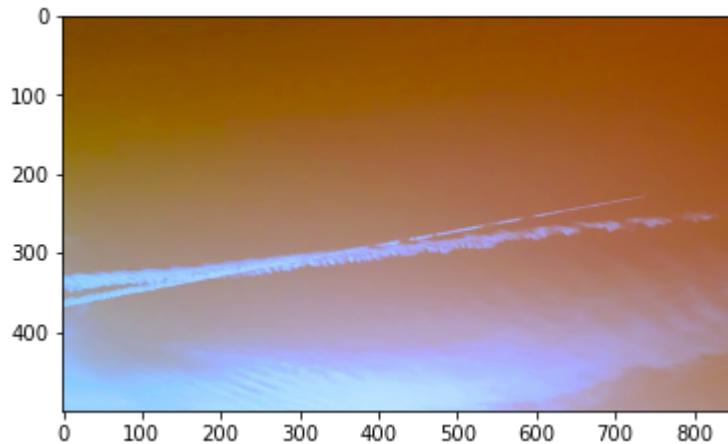
```
Out[31]: array([255, 255, 255], dtype=uint8)
```

Numpy es una biblioteca o librería optimizada para cálculos rápidos de matrices. No se recomienda simplemente acceder a cada valor de pixel de las formas ya vistas y modificarlos, porque sería un proceso muy lento. Estos métodos se usan para seleccionar una región de una matriz, supongamos las 5 primeras filas y las últimas 3 columnas de la forma que la veremos a continuación

Acceso a una parte (recorte) de una imagen

```
In [109]: img = cv.imread("imagen2.jpg")
img2 = img[:500,150:1000]
plt.imshow(img2)
```

```
Out[109]: <matplotlib.image.AxesImage at 0x11200906340>
```



Métodos item() y itemset() Para el acceso individual de píxeles, los métodos de Numpy, array.item () y array.itemset () se consideran mejores. Sin embargo, siempre devuelven un escalar,

por lo que si desea acceder a todos los valores B, G, R, deberá llamar a array.item () por separado para cada valor.

```
In [37]: #otra forma de acceder a pixel
img.item(100,2000,2)
```

```
Out[37]: 5
```

```
In [107]: img.itemset((100,2000,2),100)
```

```
In [108]: img.item(100,2000,2)
```

```
Out[108]: 100
```

Accediendo a las propiedades de una imagen

Las Imagenes poseen propiedades como numero de filas, numero de columnas, numero de pixeles, tipo de dato

Método shape

```
In [39]: img.shape
```

```
Out[39]: (1365, 2048, 3)
```

Método dtype

```
In [40]: img.dtype
```

```
Out[40]: dtype('uint8')
```

Método size

```
In [110]: img.size
```

```
Out[110]: 8386560
```

```
In [42]: type(img)
```

```
Out[42]: numpy.ndarray
```

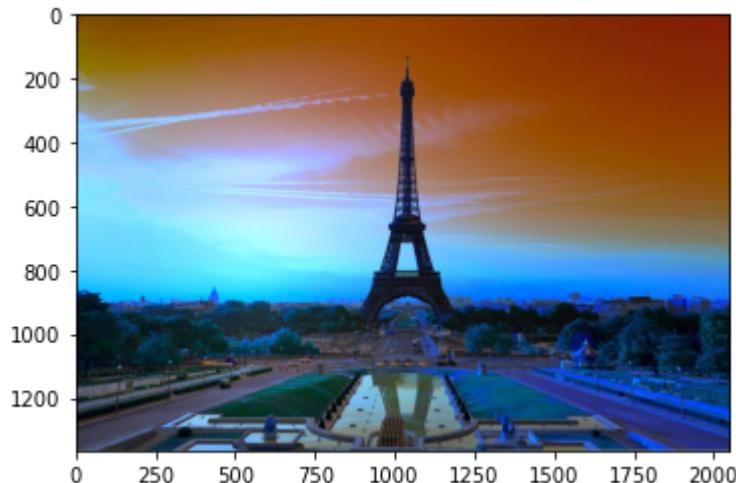
3.2 ROI (region of interest) de una imagen

A veces, tendrás que jugar con ciertas regiones de imágenes. Para la detección de ojos en imágenes, la detección de la primera cara se realiza sobre toda la imagen. Cuando se obtiene una cara, seleccionamos solo la región de la cara y buscamos ojos dentro de ella en lugar de buscar en toda la imagen. Mejora la precisión (porque los ojos siempre están en las caras) y el rendimiento (porque buscamos en un área pequeña) El ROI lo podemos obtener utilizando la indexación antes vista, en este ejemplo seleccionaremos la torre y la copiaremos en otro lugar de la imagen:

```
In [111]: import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

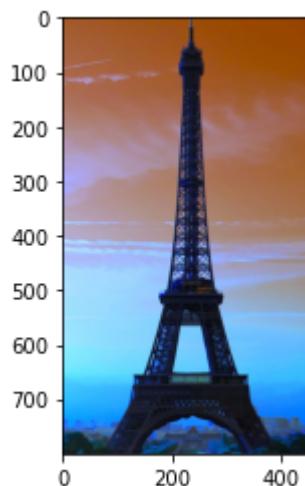
img = cv.imread("Imagen2.jpg")
plt.imshow(img)
```

Out[111]: <matplotlib.image.AxesImage at 0x11200a9b340>



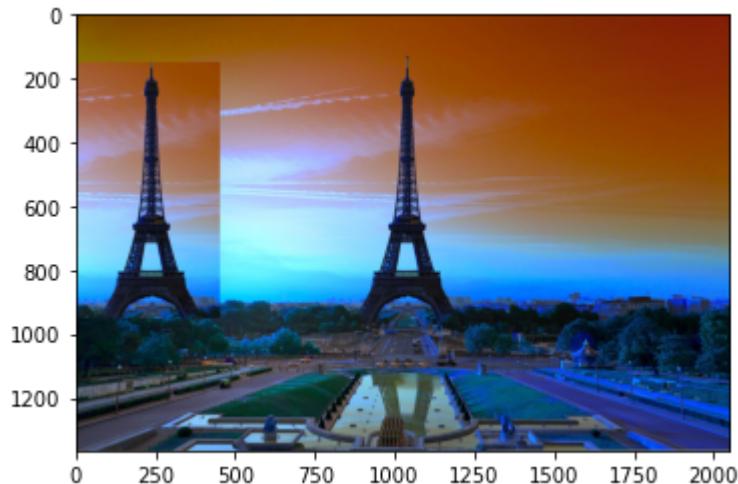
```
In [112]: torre = img[150:950, 800:1250]
plt.imshow(torre)
```

Out[112]: <matplotlib.image.AxesImage at 0x11200aeecd0>



```
In [113]: img[150:950,0:450] = torre  
plt.imshow(img)
```

```
Out[113]: <matplotlib.image.AxesImage at 0x11200c6e1f0>
```



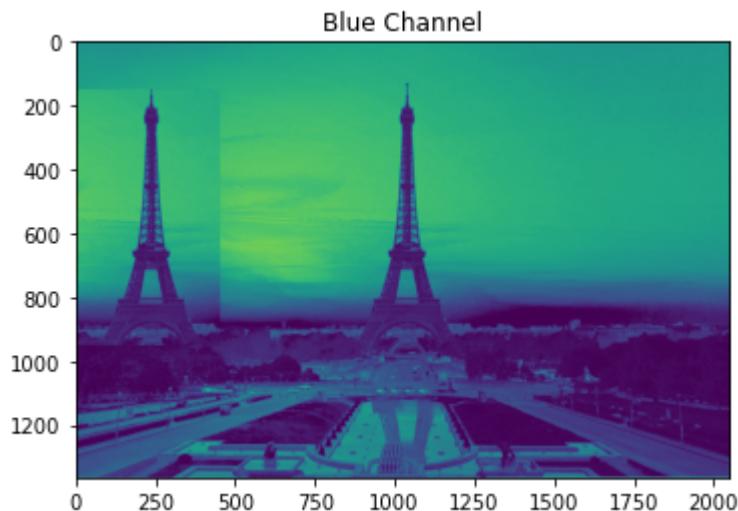
3.3 División y fusión de canales RGB

split()

```
In [47]: ## split() permite dividir los canales de una imagen  
b,g,r = cv.split(img)
```

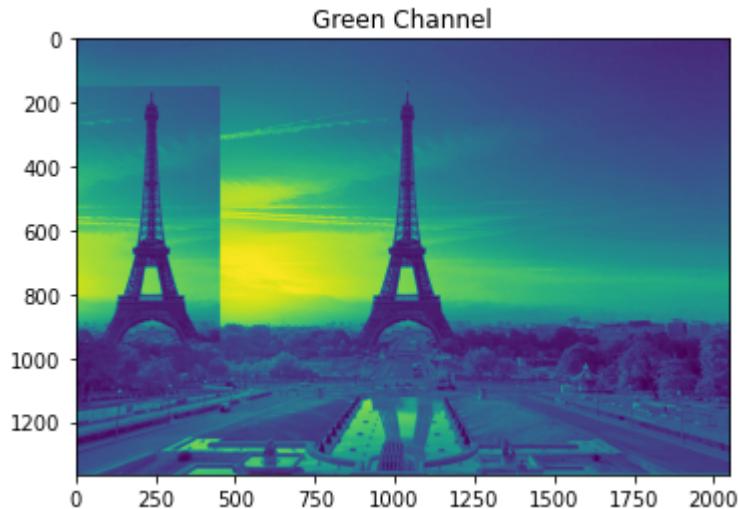
```
In [48]: plt.imshow(b)  
plt.title("Blue Channel")
```

```
Out[48]: Text(0.5, 1.0, 'Blue Channel')
```



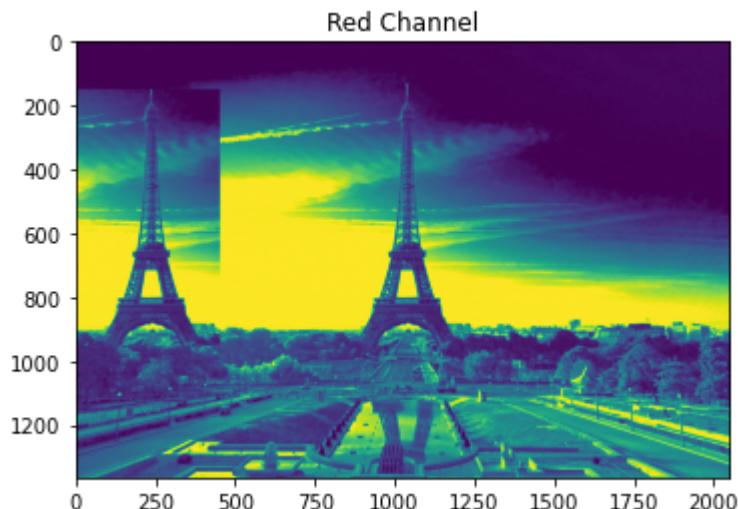
```
In [49]: plt.imshow(g)
plt.title("Green Channel")
```

```
Out[49]: Text(0.5, 1.0, 'Green Channel')
```



```
In [50]: plt.imshow(r)
plt.title("Red Channel")
```

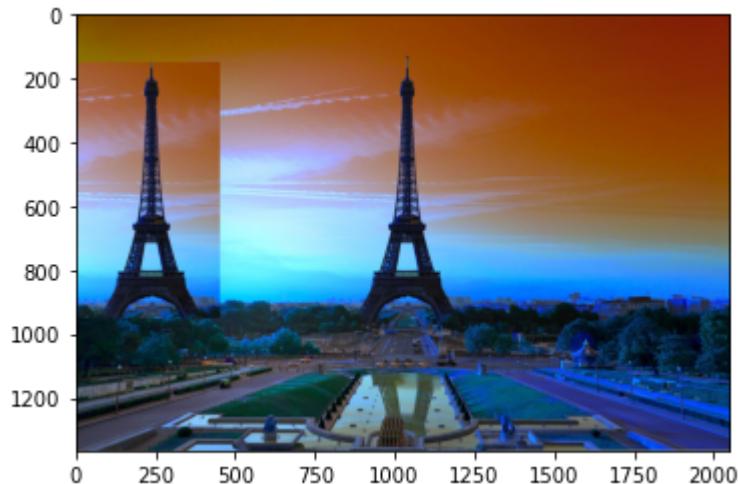
```
Out[50]: Text(0.5, 1.0, 'Red Channel')
```



merge()

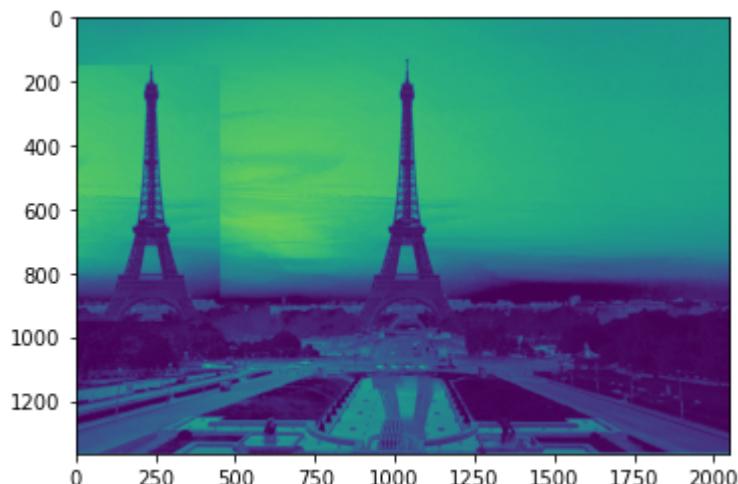
```
In [51]: # merge() permite unir canales de una imagen  
img_unida = cv.merge((b,g,r))  
plt.imshow(img_unida)
```

Out[51]: <matplotlib.image.AxesImage at 0x1126b7dbe80>



```
In [52]: #acceder al canal blue  
channel_blue = img[:, :, 0]  
plt.imshow(channel_blue)
```

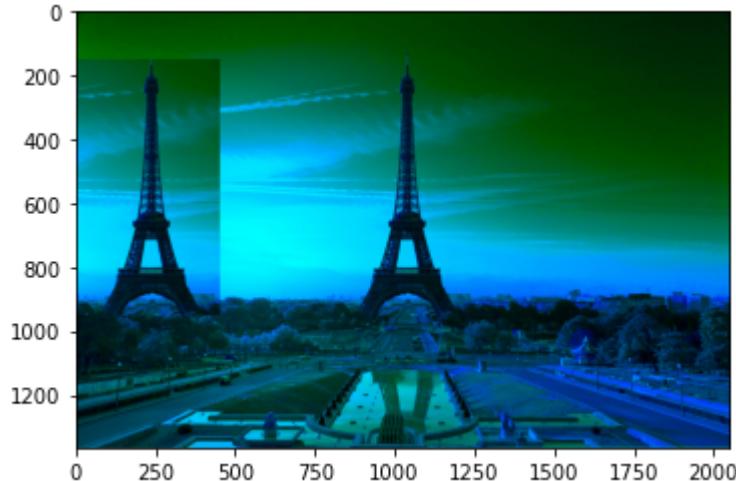
Out[52]: <matplotlib.image.AxesImage at 0x1126b6dd220>



```
In [53]: #modificar valores del canal azul
```

```
In [55]: img[:, :, 0] = 0  
plt.imshow(img)
```

```
Out[55]: <matplotlib.image.AxesImage at 0x1126b63c7f0>
```



3.4 Operaciones Aritméticas con Imágenes

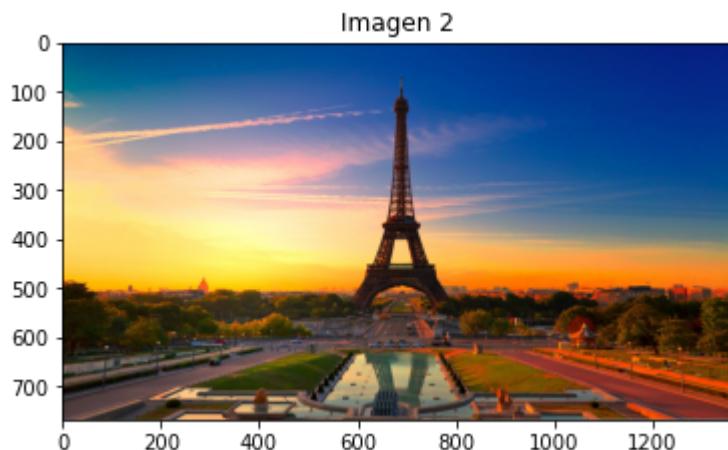
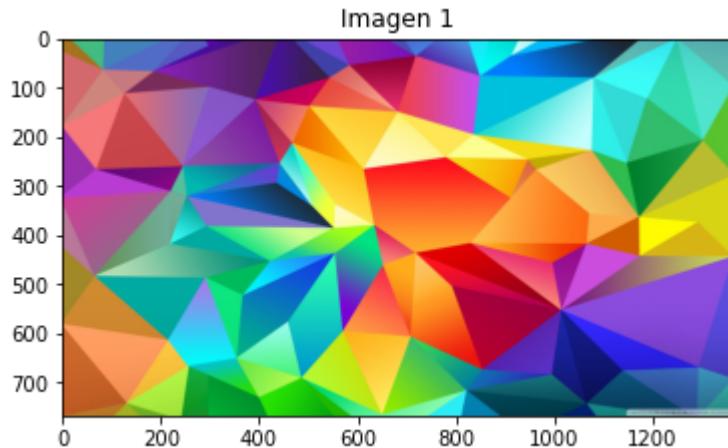
Mezcla de imágenes

Se toma en cuenta los pesos hasta completar 1 que es el 100%, con eso se controla la transparencia de una imagen con respecto a la otra

```
In [56]: import cv2 as cv  
import numpy as np  
import matplotlib.pyplot as plt  
  
img1 = cv.imread("imagen1.jpg")  
img2 = cv.imread("imagen2.jpg")  
  
#cambio de espacio de color  
img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)  
img2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)  
  
#igualar el tamaño  
fil,col,chan = img1.shape  
img2 = cv.resize(img2,(col,fil))
```

```
In [57]: #mostrar imágenes  
plt.figure(1)  
plt.imshow(img1)  
plt.title("Imagen 1")  
  
plt.figure(2)  
plt.imshow(img2)  
plt.title("Imagen 2")
```

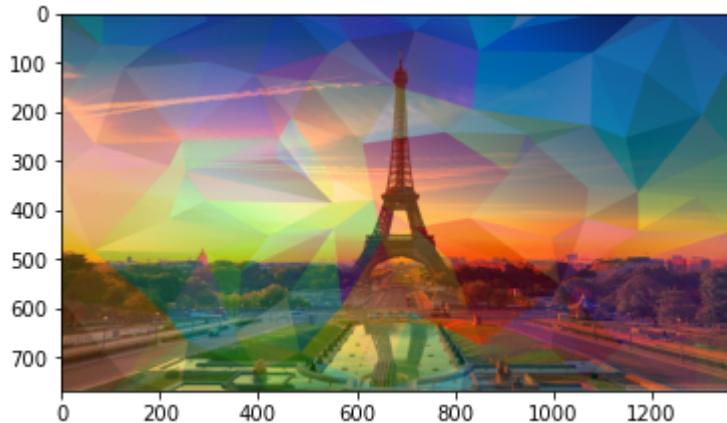
Out[57]: Text(0.5, 1.0, 'Imagen 2')



In [62]: *#combinar las imágenes*

```
img_out = cv.addWeighted(img1,0.3,img2,0.7,0)
plt.imshow(img_out)
```

Out[62]: <matplotlib.image.AxesImage at 0x1126d626700>

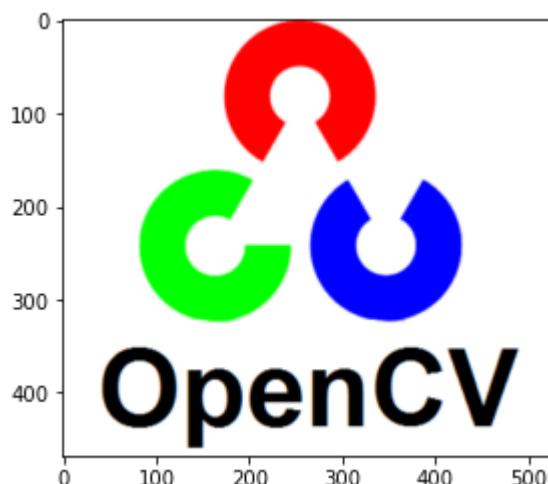


3.5 Operaciones bit a bit

Esto incluye las operaciones bit a bit AND, OR, NOT y XOR. Serán muy útiles al extraer cualquier parte de la imagen (como iremos viendo al transcurso de este aprendizaje), definir y trabajar con ROI no rectangulares, etc. A continuación veremos un ejemplo de cómo cambiar una región particular de una imagen. Quiero poner el logotipo de OpenCV sobre una imagen. Si agrego dos imágenes, cambiará el color. Si los mezclo, obtengo un efecto transparente. Pero quiero que sea opaco. Si fuera una región rectangular, podría usar el ROI como lo hicimos en el último capítulo. Pero el logotipo de OpenCV no es una forma rectangular. Entonces puede hacerlo con operaciones bit a bit como se muestra a continuación:

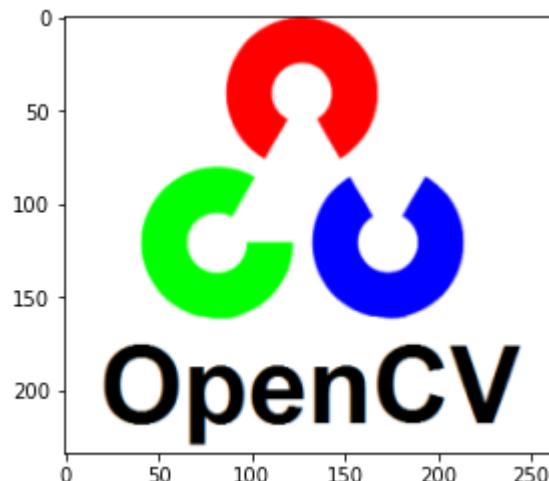
```
In [114]: #importamos las librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# cargar nuestras imagenes
img1 = cv.imread('Imagen2.jpg')
img2 = cv.imread('imagenes/opencv.png')
# cambiando espacio de color
torre = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
cv2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)
# mostrando las imagenes leidas
plt.figure(1)
plt.imshow(torre)
plt.figure(2)
plt.imshow(cv2)
```

Out[114]: <matplotlib.image.AxesImage at 0x1120150e340>



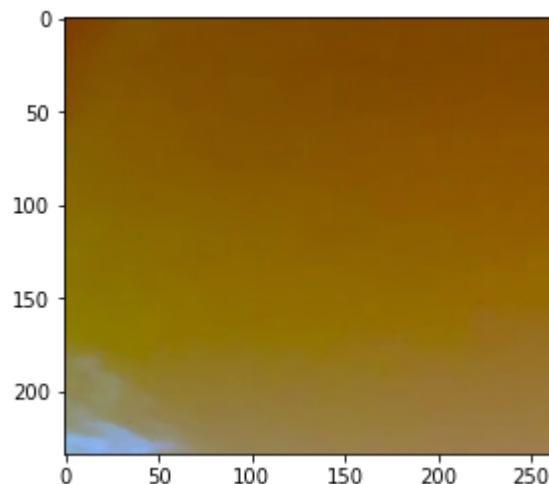
```
In [117]: # cambiamos el tamano de imagen  
fil,col,_ = torre.shape  
fil2,col2,_ = cv2.shape  
cv2 = cv.resize(cv2,(col2//2,fil2//2))  
plt.imshow(cv2)
```

Out[117]: <matplotlib.image.AxesImage at 0x1120157e5b0>



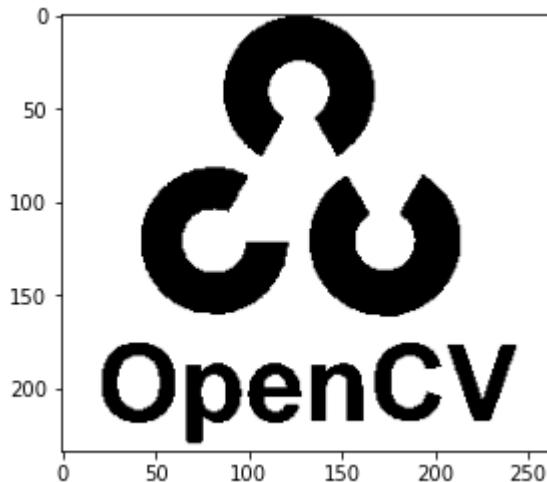
```
In [118]: fil, col,_ =cv2.shape  
roi = img1[0:fil,0:col]  
plt.imshow(roi)
```

Out[118]: <matplotlib.image.AxesImage at 0x112015cc4f0>



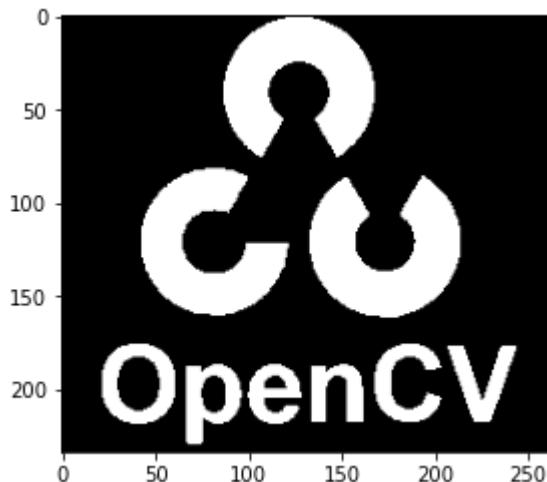
```
In [120]: #Crear una máscara del Logo opencv  
cv2_gray = cv.cvtColor(cv2, cv.COLOR_RGB2GRAY)  
ret,mask = cv.threshold(cv2_gray, 150, 255, cv.THRESH_BINARY)  
plt.imshow(mask, cmap="gray")
```

Out[120]: <matplotlib.image.AxesImage at 0x11201868fa0>



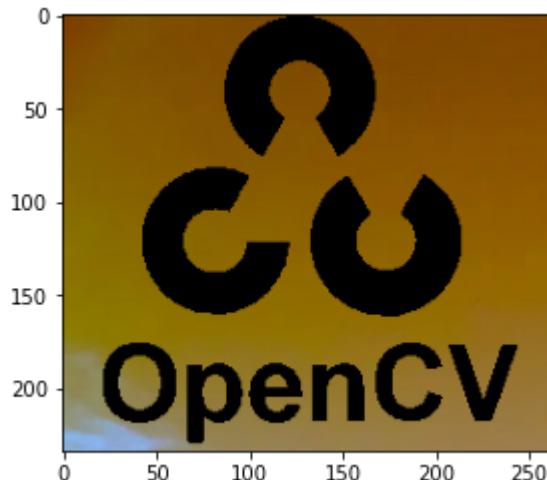
```
In [121]: # Creamos una mascara invertida  
mask_inv = cv.bitwise_not(mask)  
plt.imshow(mask_inv, cmap="gray")
```

Out[121]: <matplotlib.image.AxesImage at 0x112018b9ee0>



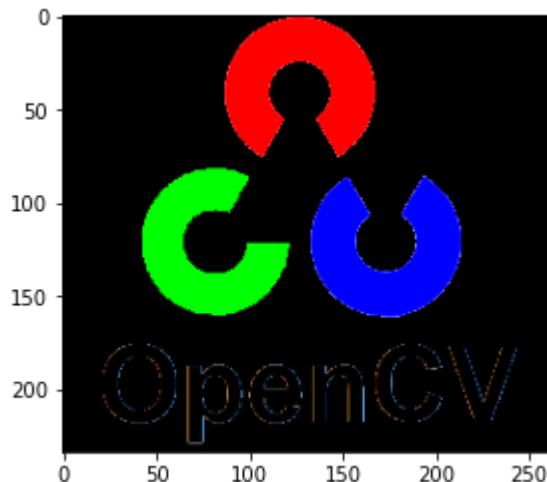
```
In [122]: # tomamos el roi menos la mascara  
img1_bg = cv.bitwise_and(roi,roi,mask = mask)  
plt.imshow(img1_bg)
```

Out[122]: <matplotlib.image.AxesImage at 0x1120316af10>



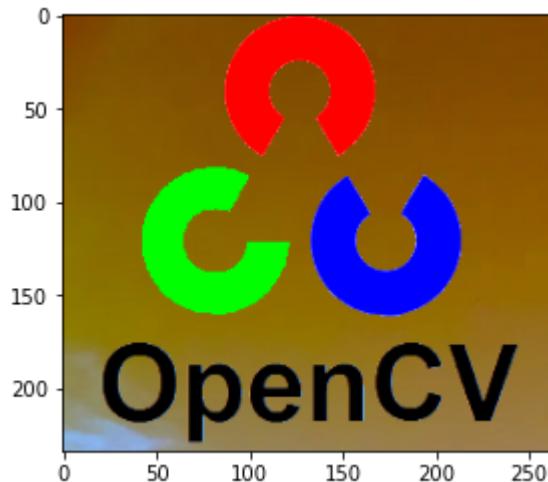
```
In [123]: #tomamos region de intere del logotipo opencv  
img2_bg = cv.bitwise_and(cv2, cv2, mask=mask_inv)  
plt.imshow(img2_bg)
```

Out[123]: <matplotlib.image.AxesImage at 0x112031baee0>



```
In [124]: img = img1_bg + img2_bg  
plt.imshow(img)
```

```
Out[124]: <matplotlib.image.AxesImage at 0x11203191d30>
```



```
In [126]: img1[0:fil,0:col]=img  
plt.imshow(img1)
```

```
Out[126]: <matplotlib.image.AxesImage at 0x112002b1490>
```



4 Procesamiento de imágenes

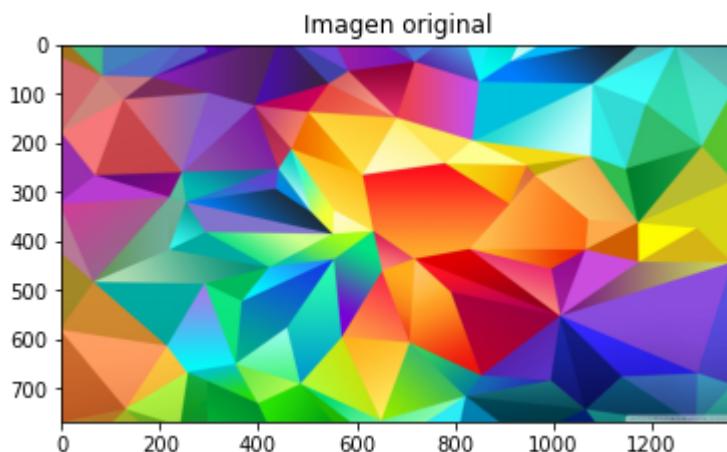
4.1 Cambio de Espacio de Color de las Imágenes

En la visión por computadora y el procesamiento de imágenes, el espacio de color se refiere a una forma específica de organizar los colores. Un espacio de color es en realidad una combinación de dos cosas, un modelo de color y una función de mapeo. La razón por la que queremos modelos de color es porque nos ayuda a representar valores de píxeles usando tuplas. La función de mapeo asigna el modelo de color al conjunto de todos los colores posibles que se pueden representar. Hay muchos espacios de color diferentes que son útiles. Algunos de los espacios de color más populares son RGB, YUV, HSV, Lab, etc. Diferentes espacios de color ofrecen diferentes ventajas. Solo tenemos que elegir el espacio de color adecuado para el problema dado. Tomemos un par de espacios de color y veamos qué información proporcionan:

RGB: Probablemente el espacio de color más popular. Es sinónimo de rojo, verde y azul. En este espacio de color, cada color se representa como una combinación ponderada de rojo, verde y azul. Por lo tanto, cada valor de píxel se representa como una tupla de tres números correspondientes a rojo, verde y azul. Cada valor oscila entre 0 y 255.

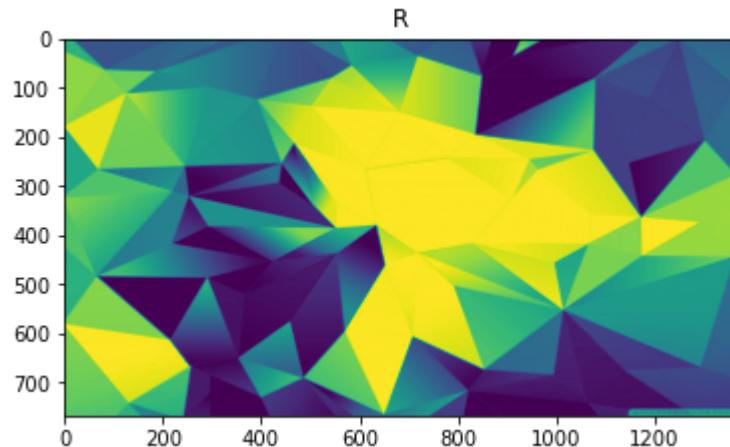
```
In [128]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("Imagen1.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
plt.imshow(img), plt.title("Imagen original")
```

```
Out[128]: (<matplotlib.image.AxesImage at 0x1126f3e4e80>,
Text(0.5, 1.0, 'Imagen original'))
```



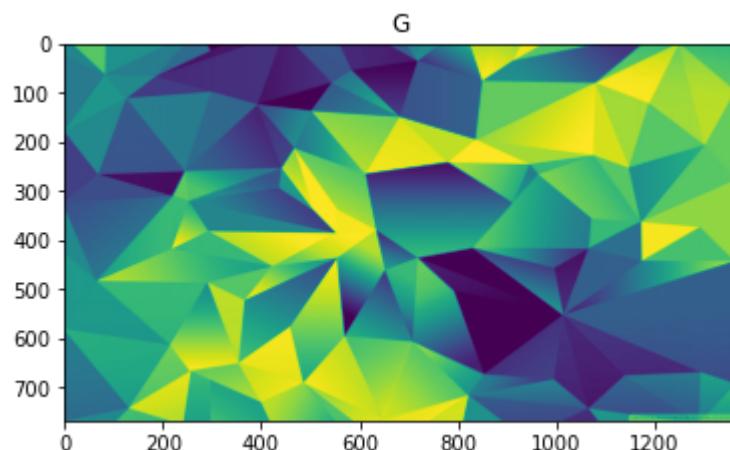
```
In [129]: plt.imshow(img[:, :, 0]), plt.title("R")
```

```
Out[129]: (<matplotlib.image.AxesImage at 0x11278652910>, Text(0.5, 1.0, 'R'))
```



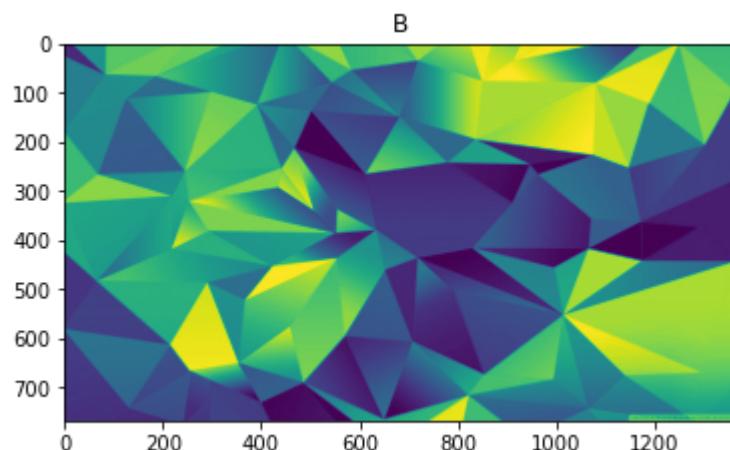
```
In [130]: plt.imshow(img[:, :, 1]), plt.title("G")
```

```
Out[130]: (<matplotlib.image.AxesImage at 0x112016a4c10>, Text(0.5, 1.0, 'G'))
```



```
In [131]: plt.imshow(img[:, :, 2]), plt.title("B")
```

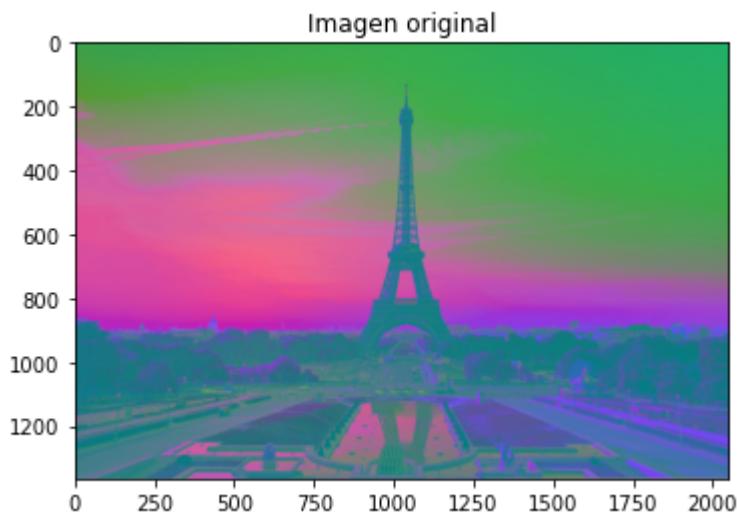
```
Out[131]: (<matplotlib.image.AxesImage at 0x112017361c0>, Text(0.5, 1.0, 'B'))
```



YUV: Aunque RGB es bueno para muchos propósitos, tiende a ser muy limitado para muchas aplicaciones de la vida real. La gente comenzó a pensar en diferentes métodos para separar la información de intensidad, de la información de color. Por lo tanto, se les ocurrió el espacio de color YUV. Y se refiere a la luminancia o intensidad, y los canales U / V representan información de color. Esto funciona bien en muchas aplicaciones porque el sistema visual humano percibe la información de intensidad de manera muy diferente a la información de color.

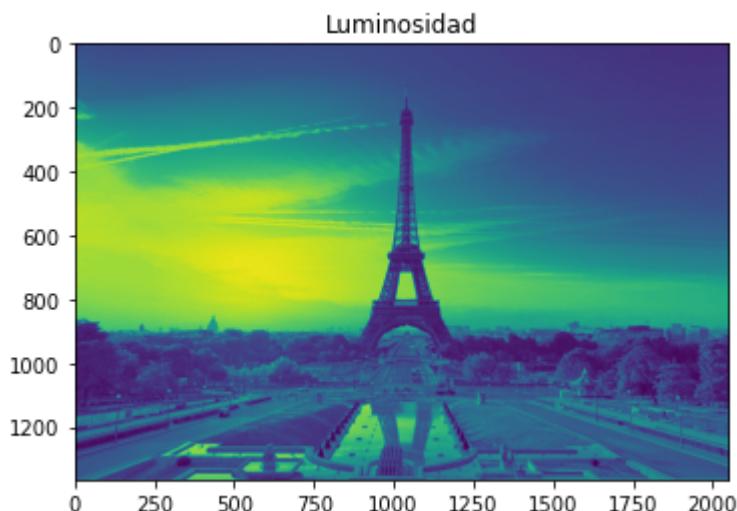
```
In [99]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("Imagen2.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2YUV)
plt.imshow(img),plt.title("Imagen original")
```

```
Out[99]: (<matplotlib.image.AxesImage at 0x1127b270a60>,
Text(0.5, 1.0, 'Imagen original'))
```



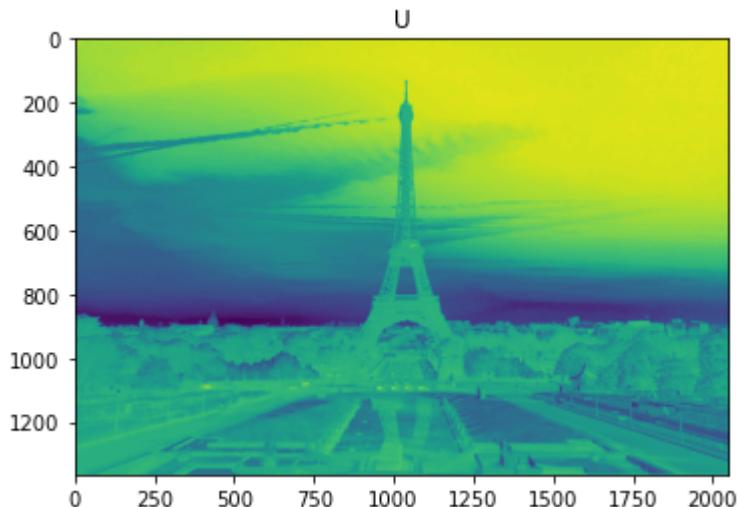
```
In [100]: plt.imshow(img[:, :, 0]), plt.title("Luminosidad")
```

```
Out[100]: (<matplotlib.image.AxesImage at 0x1127b2d11f0>, Text(0.5, 1.0, 'Luminosida
d'))
```



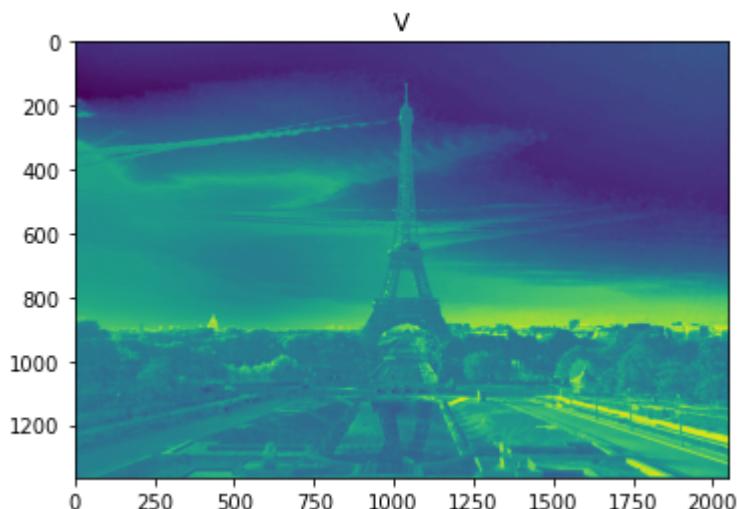
```
In [101]: plt.imshow(img[:, :, 1]), plt.title("U")
```

```
Out[101]: (<matplotlib.image.AxesImage at 0x1127b30bf10>, Text(0.5, 1.0, 'U'))
```



```
In [102]: plt.imshow(img[:, :, 2]), plt.title("V")
```

```
Out[102]: (<matplotlib.image.AxesImage at 0x112791a5e20>, Text(0.5, 1.0, 'V'))
```

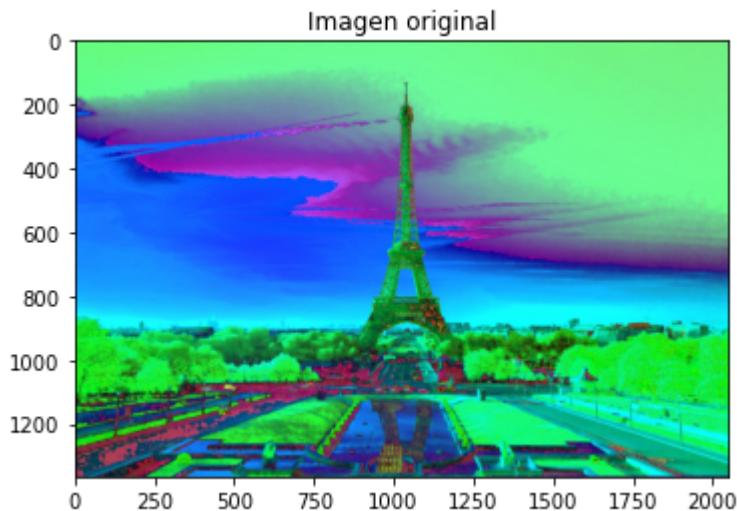


HSV

Al final resultó que, incluso YUV todavía no era lo suficientemente bueno para algunas aplicaciones. Entonces la gente comenzó a pensar en cómo los humanos perciben el color, y se les ocurrió el espacio de color HSV. HSV significa Hue, Saturación y Valor(Matiz, Saturacion, Valor). Este es un sistema cilíndrico donde separamos tres de las propiedades más primarias de los colores y las representamos usando diferentes canales. Esto está estrechamente relacionado con la forma en que el sistema visual humano comprende el color. Esto nos da mucha flexibilidad en cuanto a cómo podemos manejar las imágenes.

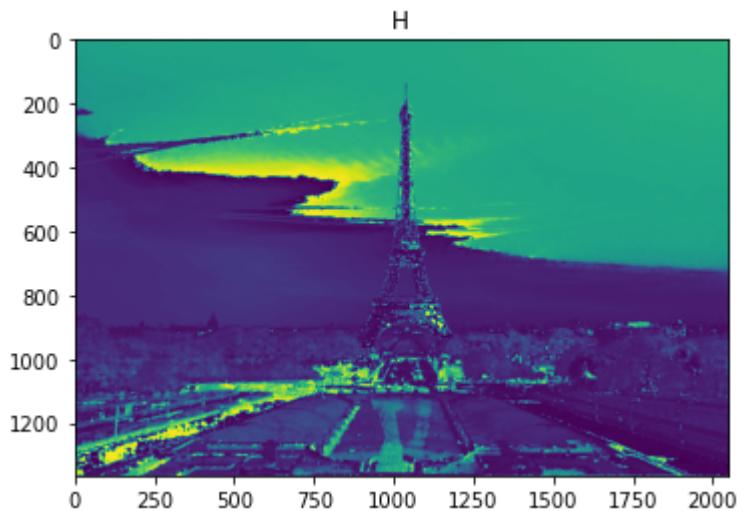
```
In [103]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
img = cv.imread("Imagen2.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2HSV)
plt.imshow(img),plt.title("Imagen original")
```

```
Out[103]: (<matplotlib.image.AxesImage at 0x11278827940>,
Text(0.5, 1.0, 'Imagen original'))
```



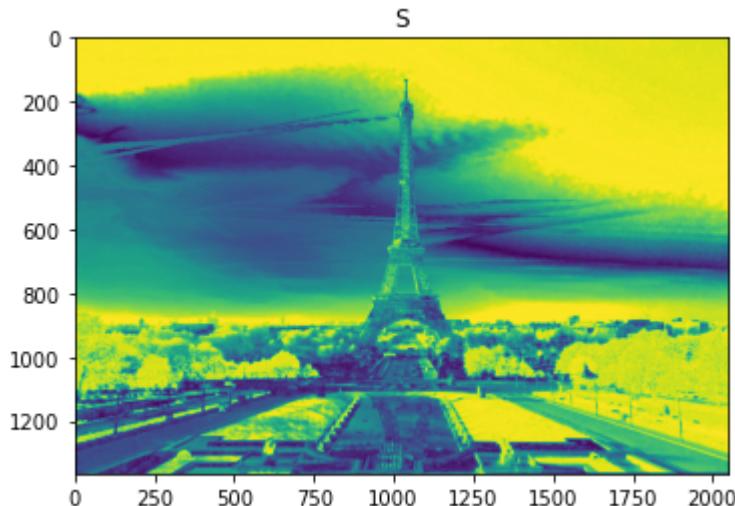
```
In [104]: plt.imshow(img[:, :, 0]), plt.title("H")
```

```
Out[104]: (<matplotlib.image.AxesImage at 0x11279100250>, Text(0.5, 1.0, 'H'))
```



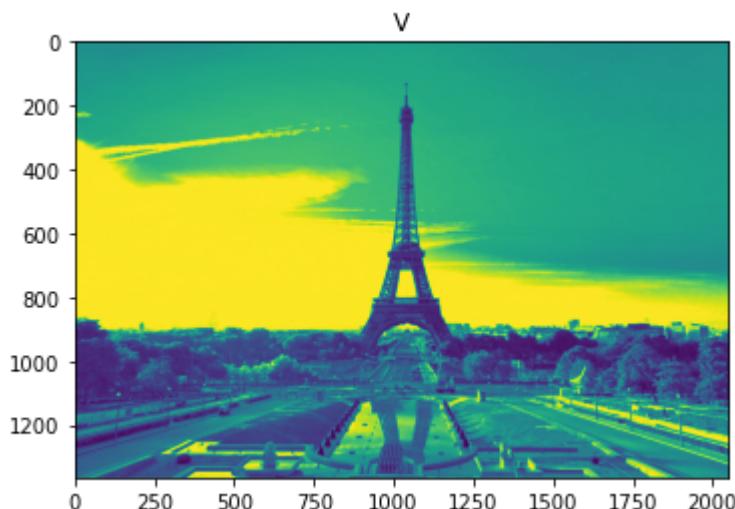
```
In [105]: plt.imshow(img[:, :, 1]), plt.title("S")
```

```
Out[105]: (<matplotlib.image.AxesImage at 0x112002c9be0>, Text(0.5, 1.0, 'S'))
```



```
In [106]: plt.imshow(img[:, :, 2]), plt.title("V")
```

```
Out[106]: (<matplotlib.image.AxesImage at 0x112005e5700>, Text(0.5, 1.0, 'V'))
```



Tarea 2

Realizar 3 ejercicios: 1> fusión de imágenes con cv.addWeighted() 2>extracción del roi (porción de una imagen) y ubicarla en otro lugar de la misma 3>Ejercicio con operaciones Bit a Bit como el visto en clase

Fusión de imágenes

```
In [1]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

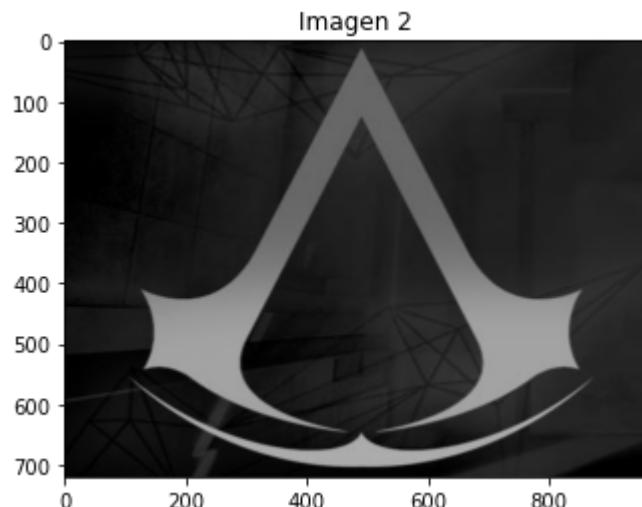
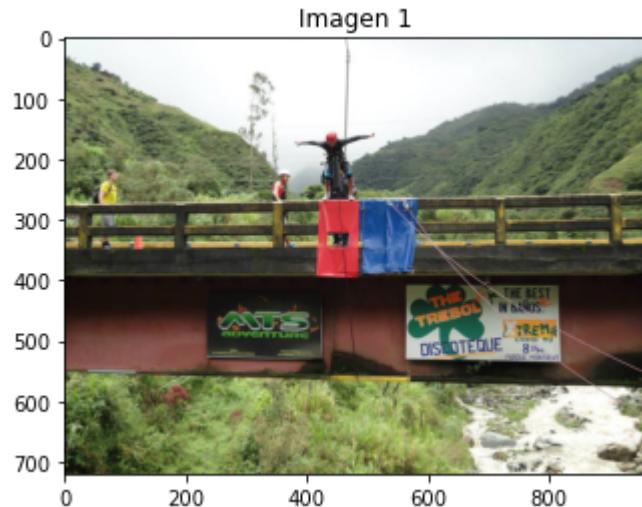
img1 = cv.imread("imagen3.jpg")
img2 = cv.imread("imagen4.jpg")

#cambio de espacio de color
img1 = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
img2 = cv.cvtColor(img2, cv.COLOR_BGR2RGB)

#igualar el tamaño
fil,col,chan = img1.shape
img2 = cv.resize(img2,(col,fil))
```

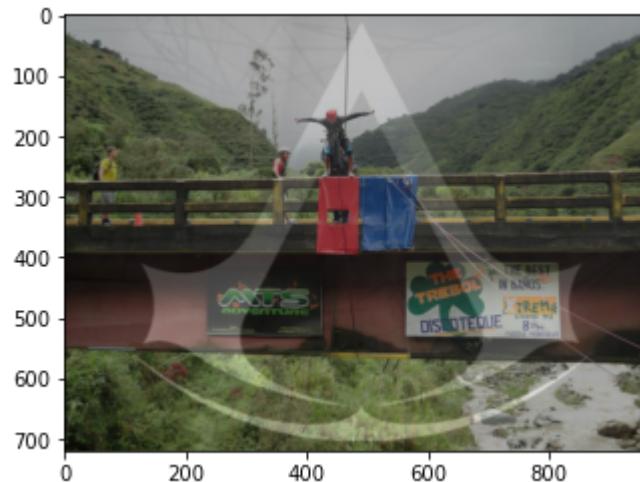
```
In [2]: #mostrar imágenes  
plt.figure(1)  
plt.imshow(img1)  
plt.title("Imagen 1")  
  
plt.figure(2)  
plt.imshow(img2)  
plt.title("Imagen 2")
```

Out[2]: Text(0.5, 1.0, 'Imagen 2')



```
In [8]: #combinar las imágenes  
img_out = cv.addWeighted(img1,0.6,img2,0.4,0)  
plt.imshow(img_out)
```

Out[8]: <matplotlib.image.AxesImage at 0x1c85e448df0>



Extracción de ROI

```
In [17]: import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
  
img = cv.cvtColor(cv.imread("imagen3.jpg"), cv.COLOR_BGR2RGB)  
plt.imshow(img)
```

Out[17]: <matplotlib.image.AxesImage at 0x1c8615ea1f0>



```
In [18]: roi = img[50:270,380:520]
plt.imshow(roi)
```

```
Out[18]: <matplotlib.image.AxesImage at 0x1c861694ac0>
```



```
In [20]: img[50:270,600:740] = roi
plt.imshow(img)
```

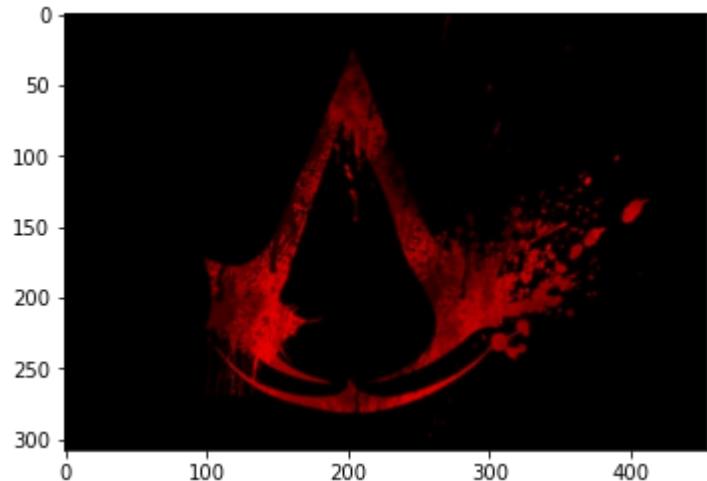
```
Out[20]: <matplotlib.image.AxesImage at 0x1c8614bbd60>
```



Operaciones bit a bit

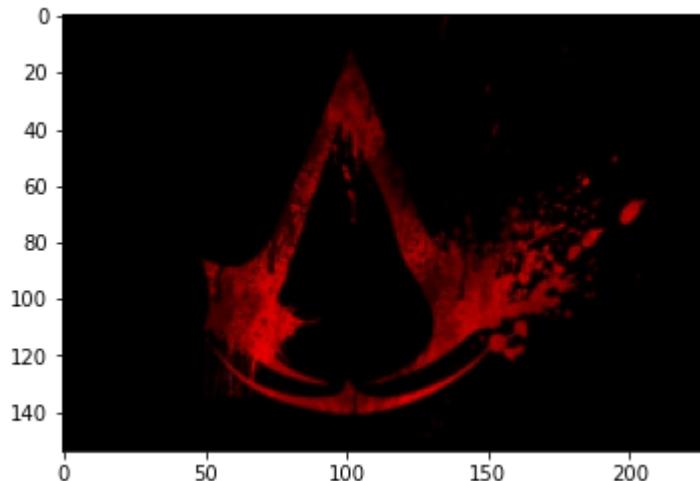
```
In [57]: #importamos las librerias
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt
# cargar nuestras imagenes
img1 = cv.imread('imagenes/assassinsCreed.jpg')
img2 = cv.imread('imagenes/assassinsLogo.png')
# cambiando espacio de color
AC = cv.cvtColor(img1, cv.COLOR_BGR2RGB)
logo = cv.cvtColor(img2, cv.COLOR_BGR2RGB)
# mostrando las imagenes leidas
plt.figure(1)
plt.imshow(AC)
plt.figure(2)
plt.imshow(logo)
```

Out[57]: <matplotlib.image.AxesImage at 0x1c862ad9d30>



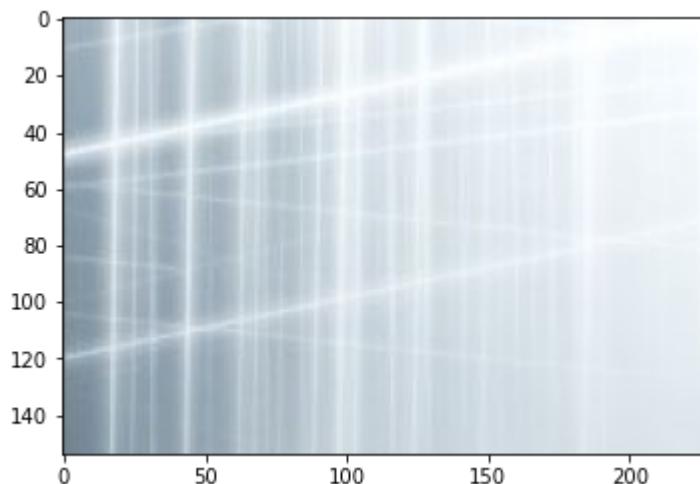
```
In [58]: # cambiamos el tamano de imagen--Logo  
fil,col,_ = AC.shape  
fil2,col2,_ = logo.shape  
logo = cv.resize(logo,(col2//2,fil2//2))  
plt.imshow(logo)
```

Out[58]: <matplotlib.image.AxesImage at 0x1c862af0fa0>



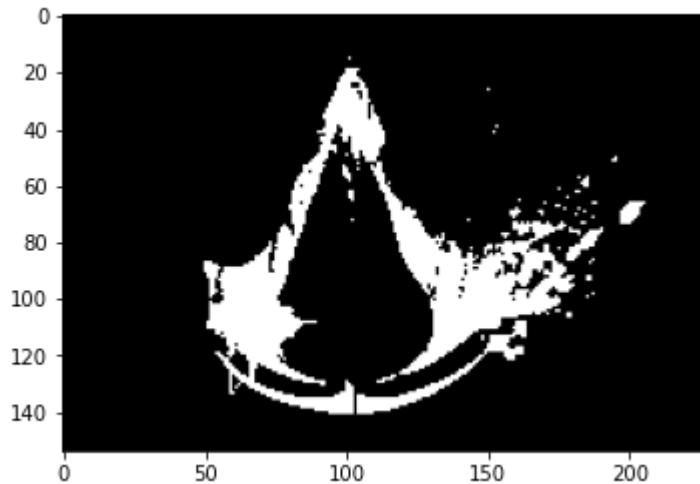
```
In [59]: fil, col,_ = logo.shape  
roi = AC[0:fil,0:col]  
plt.imshow(roi)
```

Out[59]: <matplotlib.image.AxesImage at 0x1c862b4fa60>



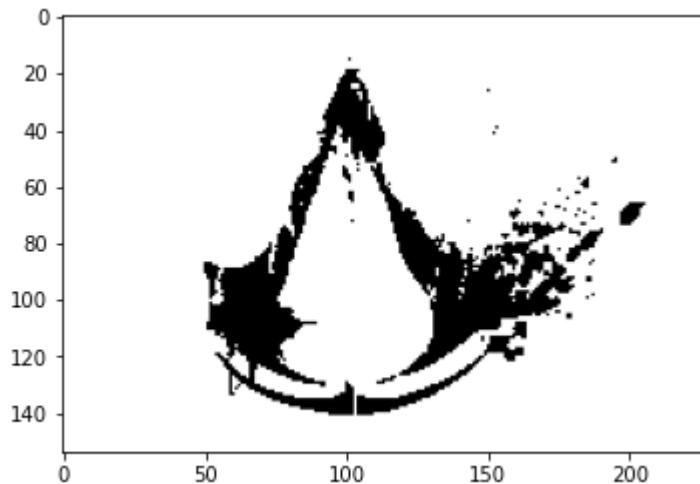
```
In [60]: #Crear una máscara del Logo opencv  
logo_gray = cv.cvtColor(logo, cv.COLOR_RGB2GRAY)  
ret,mask = cv.threshold(logo_gray, 15, 255, cv.THRESH_BINARY)  
plt.imshow(mask, cmap="gray")
```

Out[60]: <matplotlib.image.AxesImage at 0x1c862ba1cd0>



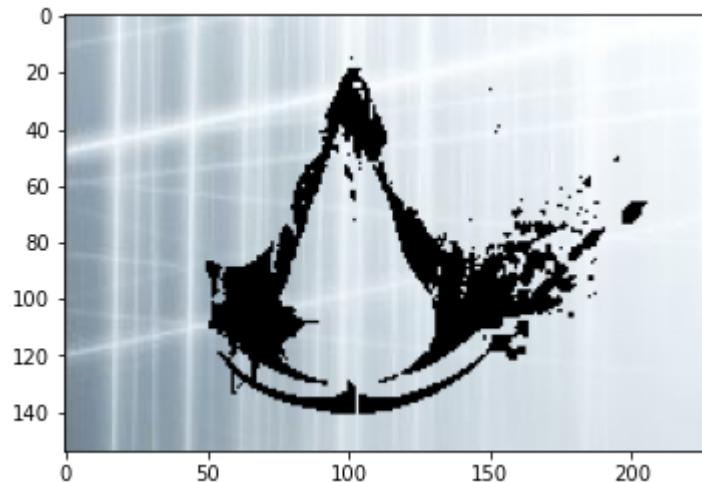
```
In [61]: # Creamos una mascara invertida  
mask_inv = cv.bitwise_not(mask)  
plt.imshow(mask_inv, cmap="gray")
```

Out[61]: <matplotlib.image.AxesImage at 0x1c862bf2e20>



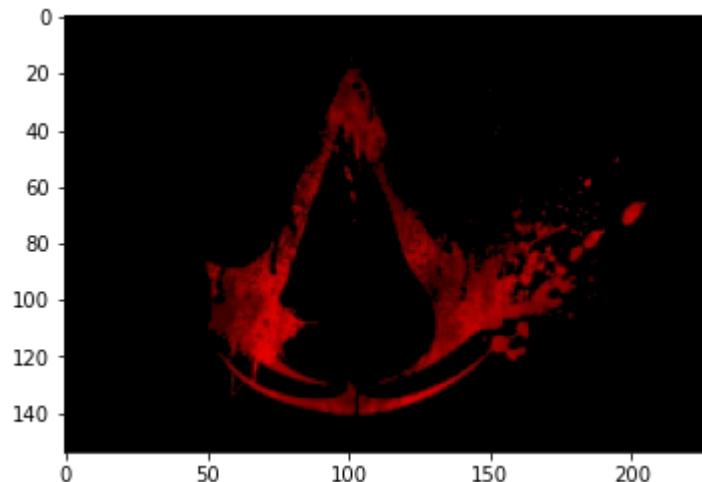
```
In [62]: # tomamos el roi menos la mascara  
img1_bg = cv.bitwise_and(roi,roi,mask = mask_inv)  
plt.imshow(img1_bg)
```

Out[62]: <matplotlib.image.AxesImage at 0x1c862c50100>



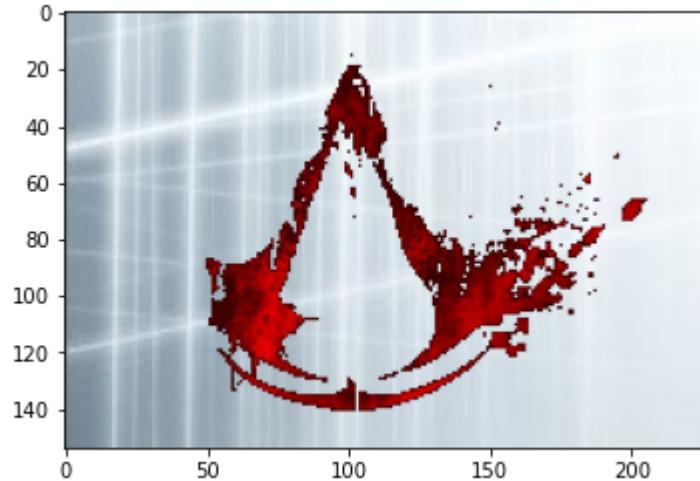
```
In [63]: #tomamos region de intere del logotipo opencv  
img2_bg = cv.bitwise_and(logo,logo,mask=mask)  
plt.imshow(img2_bg)
```

Out[63]: <matplotlib.image.AxesImage at 0x1c862ca03a0>



```
In [64]: img = img1_bg + img2_bg  
plt.imshow(img)
```

```
Out[64]: <matplotlib.image.AxesImage at 0x1c864392550>
```



```
In [65]: AC[0:fil,0:col]=img  
plt.imshow(AC)
```

```
Out[65]: <matplotlib.image.AxesImage at 0x1c8643e2a00>
```



DÍA 4

Detección de Colores

Ahora que sabemos cómo convertir una imagen BGR a HSV, podemos usar esto para extraer un objeto coloreado. En HSV, es más fácil representar un color que en el espacio de color BGR. En nuestra aplicación, intentaremos extraer un objeto de color azul. Así que aquí está el método:

Toma cada fotograma del video

Convertir de espacio de color BGR a HSV

Umbralizamos la imagen HSV para un rango de color azul

Ahora extraiga el objeto azul solo, podemos hacer lo que queramos en esa imagen

```
In [7]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

cap = cv.VideoCapture(0)

#HSV Azul
azul_bajo = np.array([80,50,50],np.uint8)
azul_alto = np.array([120,255,255],np.uint8)

while cap.isOpened():
    ret,frame = cap.read()

    if ret:

        img_hsv = cv.cvtColor(frame, cv.COLOR_BGR2HSV)
        mask = cv.inRange(img_hsv, azul_bajo, azul_alto)
        res = cv.bitwise_and(frame, frame, mask = mask)

        cv.imshow("ORIGINAL", frame)
        cv.imshow("MASCARA", mask)
        cv.imshow("RESULTADO", res)

        if cv.waitKey(10) & 0xFF == ord('q'):
            break
    else:
        break

cap.release()
cv.destroyAllWindows()
```

4.2 Transformaciones Geométricas con Imágenes

Escalado

```
In [11]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread("Imagen2.jpg")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)

plt.imshow(img)
```

Out[11]: <matplotlib.image.AxesImage at 0x2a3a4074910>



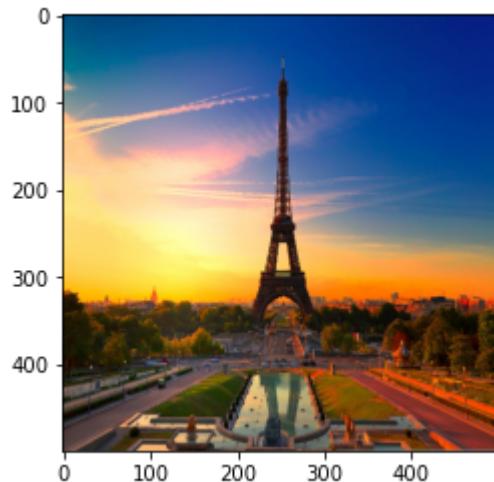
```
In [16]: #escalado
img_escalada = cv.resize(img, None, fx = 2, fy = 2, interpolation = cv.INTER_CUBIC)
plt.imshow(img_escalada)
```

Out[16]: <matplotlib.image.AxesImage at 0x2a3a3da83d0>



```
In [20]: #Cambio en dimensiones de la ventana  
result = cv.resize(img,(500,500))  
plt.imshow(result)
```

Out[20]: <matplotlib.image.AxesImage at 0x2a3a8156fd0>



Rotacion

```
In [23]: import numpy as np  
import cv2 as cv  
import matplotlib.pyplot as plt  
  
img = cv.imread("imagenes/assassinsCreed.jpg")  
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)  
  
plt.imshow(img)
```

Out[23]: <matplotlib.image.AxesImage at 0x2a3a92a3820>



```
In [31]: # Girar la imagen
fil,col,_ = img.shape
#matriz de rotacion
mat = cv.getRotationMatrix2D(((col-1)/2.0,(fil-1)/2.0),-30,1)
#transformacion de la imagen
result = cv.warpAffine(img, mat, (col,fil))
plt.imshow(result)
```

Out[31]: <matplotlib.image.AxesImage at 0x2a3b1f41c40>



Traslación de una imagen

```
In [69]: import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt

img = cv.imread("imagenes/assassinsCreed.jpg")
#plt.imshow(img)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
fil,col,_ = img.shape

M = np.float32([[1,0,100],[0,1,150]])
result = cv.warpAffine(img, M, (col,fil))
plt.imshow(result)
```

Out[69]: <matplotlib.image.AxesImage at 0x1c864439f70>



Transformación Afine

```
In [71]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.cvtColor(cv.imread("imagenes/assassinsCreed.jpg"), cv.COLOR_BGR2RGB)

plt.figure(1)
plt.imshow(img), plt.title("Imagen original")

fil,col,_ = img.shape

#puntos de entrada
scr_point = np.float32([[0,0],[col-1,0],[0,fil-1]])
#puntos de salida
dts_points = np.float32([[0,0],[int(0.6*(col-1)),0],[int(0.4*(col-1)),fil-1]])
#matriz de transformacion afín
matris_afin = cv.getAffineTransform(scr_point, dts_points)
#Transformación
output = cv.warpAffine(img,matris_afin,(col,fil))

plt.figure(2)
plt.imshow(output), plt.title("Salida")
```

Out[71]: (<matplotlib.image.AxesImage at 0x1c86abceb50>, Text(0.5, 1.0, 'Salida'))



Espejo de una imagen

```
In [37]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.cvtColor(cv.imread("imagenes/assassinsCreed.jpg"), cv.COLOR_BGR2RGB)

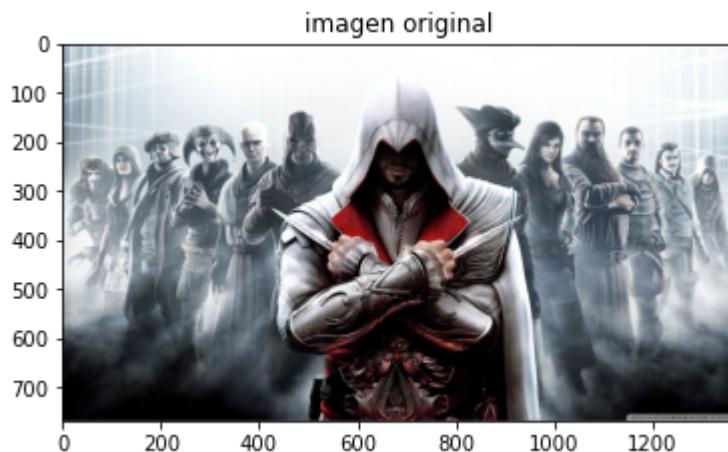
plt.figure(1)
plt.imshow(img), plt.title("Imagen original")

fil,col,_ = img.shape

#puntos de entrada
scr_point = np.float32([[0,0],[col-1,0],[0,fil-1]])
#puntos de salida
dts_points = np.float32([[col-1,0],[0,0],[col-1,fil-1]])
#matriz de transformacion afín
matris_afin = cv.getAffineTransform(scr_point, dts_points)
#Transformación
output = cv.warpAffine(img,matris_afin,(col,fil))

plt.figure(2)
plt.imshow(output), plt.title("Salida")
```

Out[37]: (<matplotlib.image.AxesImage at 0x2a3a95ee370>, Text(0.5, 1.0, 'Salida'))



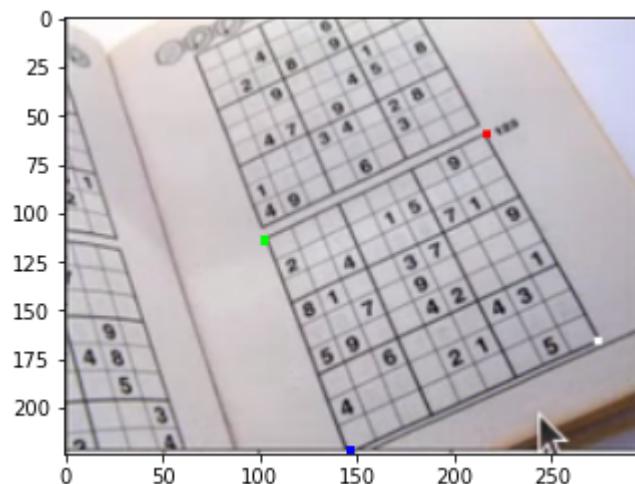
Transformaciones de perspectiva

matriz de transformacion de 3x3, 4 puntos de la imagen, 3 de ellos no colineales.

```
In [39]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread("imagenes/sudoku.png")
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
#hallar puntos
#arriba iz
img[112:117,101:105] = [0,255,0]
#arriba derecha
img[58:62,215:219] = [255,0,0]
#abajo iz
img[220:224,145:149]=[0,0,255]
#abajo der
img[164:168,272:276] = [255,255,255]
plt.imshow(img)
```

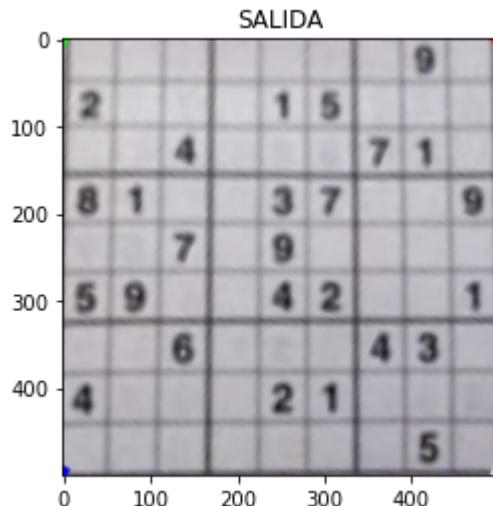
Out[39]: <matplotlib.image.AxesImage at 0x2a3b1376880>



```
In [40]: pts1 = np.float32([[103,115],[147,222],[217,60],[274,166]])
pts2 = np.float32([[0,0],[0,500-1],[500-1,0],[500-1,500-1]])

matriz = cv.getPerspectiveTransform(pts1,pts2)
output = cv.warpPerspective(img, matriz, (500,500))
plt.imshow(output),plt.title("SALIDA")
```

Out[40]: (<matplotlib.image.AxesImage at 0x2a3b4f2f850>, Text(0.5, 1.0, 'SALIDA'))



4.3 Umbralización de imágenes

Umbralización simple

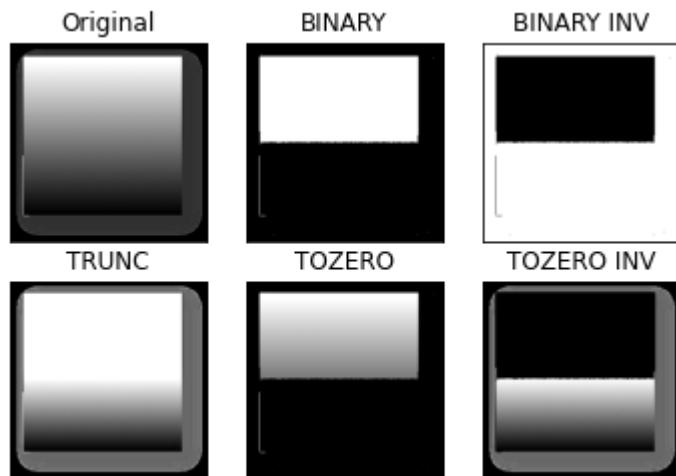
```
In [43]: import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread("imagenes/degradadoGris.png")
img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

ret, thresh1 = cv.threshold(img,120,255,cv.THRESH_BINARY)
ret, thresh2 = cv.threshold(img,120,255,cv.THRESH_BINARY_INV)
ret, thresh3 = cv.threshold(img,120,255,cv.THRESH_TRUNC)
ret, thresh4 = cv.threshold(img,120,255,cv.THRESH_TOZERO)
ret, thresh5 = cv.threshold(img,120,255,cv.THRESH_TOZERO_INV)

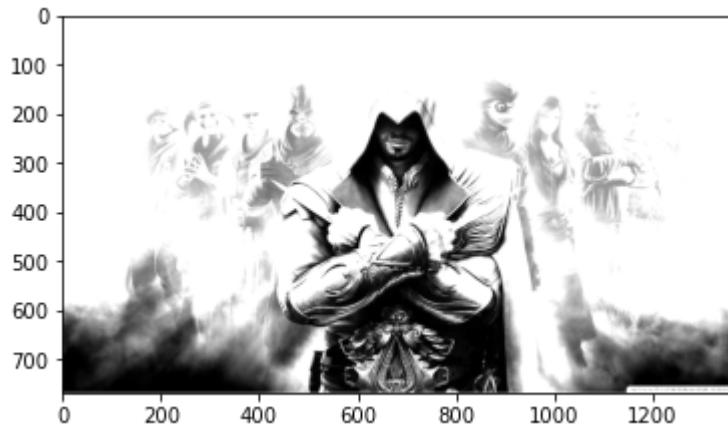
titles = ['Original','BINARY','BINARY INV','TRUNC','TOZERO','TOZERO INV']
images = [img,thresh1,thresh2,thresh3,thresh4,thresh5]

for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i], 'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```



```
In [113]: import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt
img = cv.imread('imagenes/assassinsCreed.jpg')
img2 = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
img = cv.cvtColor(img, cv.COLOR_BGR2RGB)
ret,th1 = cv.threshold(img2,100,255, cv.THRESH_TRUNC)
plt.figure(1),plt.imshow(img)
plt.figure(2),plt.imshow(th1, cmap="gray")
```

Out[113]: (<Figure size 432x288 with 1 Axes>,
<matplotlib.image.AxesImage at 0x1c86ef69d90>)



In []: