

Práctica 1. Manejo básico de Imágenes

(septiembre, 2021)

1st Guerra Silva Erick Iván
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
erickivanguerra0.0@gmail.com

2nd Lázaro Martínez Abraham Josué
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
abrahamlazaro@comunidad.unam.mx

3rd Martínez Gutiérrez Carlos Giovanni
Ingeniería en Computación
Universidad Nacional Autónoma de México
 Ciudad de México, México
cgiovanni@comunidad.unam.mx

Resumen—Este documento presenta los resultados obtenidos para los ejercicios planteados en la práctica para la manipulación básica de imágenes. Se utilizaron diferentes librerías para el procesamiento de imágenes al igual que se utilizaron diferentes funciones para el despliegue, redimensionamiento, recorte, cambio de colores y rotación en imágenes. Toda la práctica se realizó en lenguaje Python utilizando Google Collaboratory y se utilizaron librerías como Matplotlib, OpenCV, Scikit-Image, PIL y Sci-Py.

Índice de Términos—Procesamiento de Imagenes, Python, paleta de colores, decimación, formato, renglón, columna, escala.

I. INTRODUCCIÓN

El formato de una imagen describe cómo se almacenarán los datos relacionados con la imagen. Los datos se pueden almacenar en formato comprimido, sin comprimir o vectorial. Cada formato de imagen tiene una ventaja y una desventaja diferentes.

Aunque hay muchos formatos de archivo de imagen y extensiones de archivo, todos se dividen en una de dos categorías: rásteres y vectores.

Las imágenes ráster son las más comunes de los dos tipos de imágenes. Son archivos complejos de alta calidad creados con millones de píxeles minúsculos. Debido a esto, no se pueden cambiar drásticamente de tamaño sin que aparezcan granulosos y de baja calidad.

Las imágenes vectoriales se crean utilizando polígonos en lugar de píxeles. A diferencia de las imágenes rasterizadas, las imágenes vectoriales conservan la calidad cuando se redimensionan. Un inconveniente de las imágenes vectoriales grandes es que requieren más polígonos para conservar la calidad, lo que puede crear archivos increíblemente grandes.

Las imágenes gráficas se almacenan digitalmente utilizando una pequeña cantidad de formatos de archivos gráficos estandarizados, incluidos mapas de bits, TIFF, JPEG, GIF, PNG; también se pueden almacenar como datos sin procesar (RAW).[1]

El formato RAW en imágenes, se trata del negativo digital generado por una cámara fotográfica. El sensor de la cámara capta toda la información y la almacena en ese archivo, que contiene todos los datos acerca de la imagen y lo convierte, por lo tanto, en el más polivalente a la hora de permitirnos jugar con el resultado final y la edición de la fotografía.[2]

El espacio de color es la gama de colores que puede representar tu ordenador, es una lista estándar de colores codificados.[3]

1. HSV

El espacio de color HSV es una representación tridimensional del color basado en los componentes de tinte, matiz o tonalidad (hue, en inglés), saturación (saturation) y brillo o valor (value); A diferencia del modelo RGB que sigue un modelo de coordenadas euclidianas, este modelo sigue una representación más parecida a las coordenadas cilíndricas.[4]

2. YUV

El espacio de color YUV en las imágenes se divide entre diversos componentes. En concordancia con el nombre de la extensión, se llaman Y, U y V, respectivamente. La información sobre el brillo se almacena como Y, mientras que U y V contienen la información sobre el color. Los archivos YUV pueden codificarse en diversos rangos de bits, como 12, 16 o 24 bits. [5]

II. OBJETIVO

Al término de esta práctica, el alumno conocerá las distintas formas de desplegar una imagen en distintos formatos, así como algunas manipulaciones básicas, para el procesamiento de imágenes.

III. RESULTADOS

A continuación, se presentan los ejercicios desarrollados en esta práctica. Antes de realizar los ejercicios se realizó la descarga y descompresión de las imágenes para poder trabajar con ellas. Una vez descargadas las imágenes se procedió a iterar sobre el directorio para crear una lista de rutas con todas las imágenes en las carpetas y después poder

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

ser manipuladas.

3.1. De la carpeta de imágenes: realiza las siguientes actividades.

3.1.1. Desarrolla un script para leer y desplegar cada imagen con los paquetes de Matplotlib, OpenCV, Scikit-Image, PIL y Sci-Py.

A. Matplotlib

Para la lectura y despliegue de imágenes con *Matplotlib* se importaron los módulos *image* y *pyplot* de *Matplotlib*, al igual que *pydicom* para el manejo de imágenes con formato .dcm.

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
# se usa para formatos dcm
import pydicom as dicom
```

Una vez importados los módulos se programó un ciclo para la lectura y el despliegue de todas las imágenes en la carpeta. Para la lectura de imágenes se utiliza la función *imread* y para el despliegue se utiliza *imshow* de *pyplot*.

La función *imshow* es una función nativa de *Matplotlib*, y nos permite realizar gráficas y figuras, por lo cual la manipulación de los objetos que conforman a la gráfica (como el título y los ejes) son personalizables.

```
img = mpimg.imread(imagePath)
ax1.imshow(img)
```

Para el despliegue de imágenes tipo DICOM se utilizó la biblioteca *pydicom*, con la función *dcmread* para leer las imágenes. Para su despliegue se utilizó el atributo *pixel_array* del objeto tipo imagen DICOM generado por *pydicom*. Se ocupa la misma función de *Matplotlib* para poder desplegar la imagen.

```
ds = dicom.dcmread(imagePath)
ax1.imshow(img)
```

A continuación, se presentan ejemplos de lectura y despliegue de imágenes con *Matplotlib* y con *pydicom*.

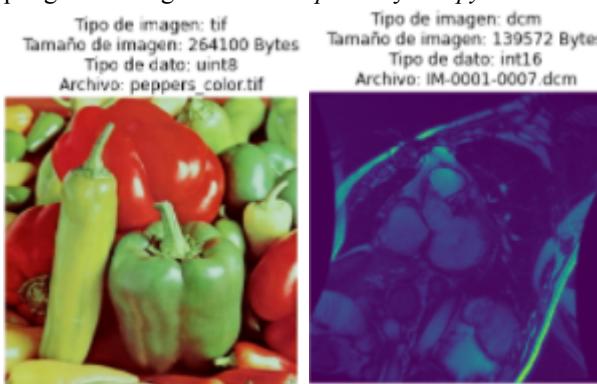


Fig. 1. Lectura y despliegue de imágenes con el módulo *Matplotlib*

B. OpenCV

Para la lectura y despliegue de imágenes con *OpenCV*

se importó el módulo *cv2*, al igual que se importó la función *cv2_imshow* de la biblioteca de parches de Google colab ya que la función *imshow* propia de *cv2* provoca que las sesiones sean interrumpidas. Trabajamos en Google Colab, y la función original *imshow* de la biblioteca *OpenCV* genera problemas, por lo cual el equipo de soporte de Google desarrolló una función *imshow* para trabajo específico en Colab.

```
import cv2
from google.colab.patches import cv2_imshow
```

Una vez importados los módulos se programó un ciclo para la lectura y el despliegue de todas las imágenes en la carpeta. Dentro del ciclo *for* se agregó un condicional para evitar la lectura de imágenes con formato .raw y .dcm ya que para la lectura y despliegue de imágenes con estos formatos deben utilizarse funciones y métodos adicionales.

Para la lectura de imágenes se utiliza la función *imread* del módulo *cv2* y para el despliegue se utiliza la función parche *cv2_imshow*.

```
img = cv2.imread(imagePath)
cv2_imshow(img)
```

Para la impresión de formato, tamaño, tipo y nombre se utiliza el mismo método mencionado anteriormente.

A continuación, se presentan ejemplos de lectura y despliegue de imágenes con *OpenCV*.

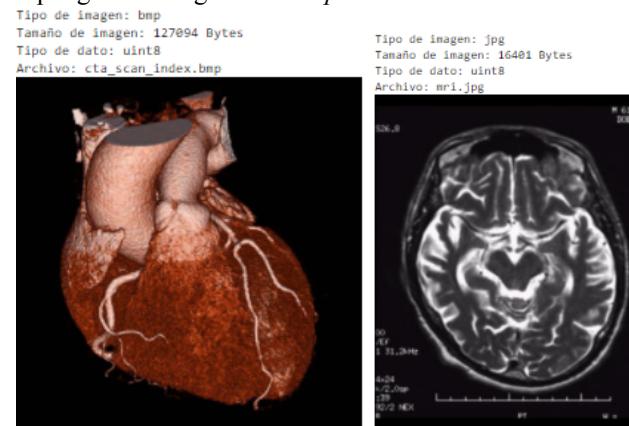


Fig. 2. Lectura y despliegue de imágenes con el módulo *OpenCV*

C. Scikit-Image

Para la lectura y despliegue de imágenes con *Scikit-Image* se importó el módulo *io* de la biblioteca *skimage*.

```
from skimage import io
```

Una vez cargado el módulo se realizó la lectura y despliegue de las imágenes contenidas en la carpeta. La lectura se realizó con la función *imread* del módulo *io* mientras que el despliegue se realizó utilizando las funciones *imshow* y *show*, la función *imshow* prepara la imagen en un gráfico para su despliegue y la función *show* despliega el gráfico.

Al utilizar la función *imread*, con imágenes de tipo

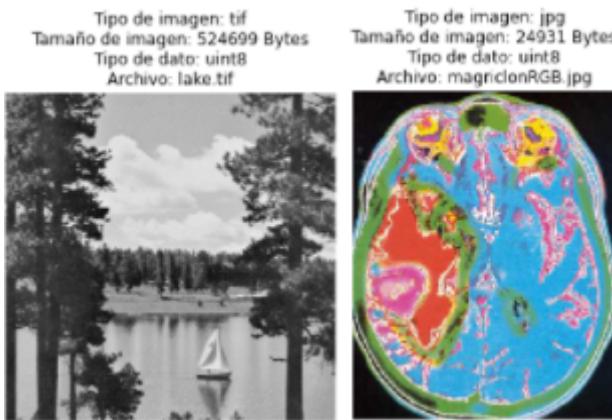
RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

TIFF, es necesario especificar un plugin para la lectura de imágenes. Esto se define como segundo parámetro, después de la dirección de la imagen a cargar, `plugin='pil'`

```
img = io.imread(imagePath)
io.imshow(img)
io.show()
```

Para la impresión de formato, tamaño, tipo y nombre se utiliza el mismo método mencionado anteriormente.

A continuación, se presentan ejemplos de lectura y despliegue de imágenes con *Scikit-Image*.



D. PIL

Para la lectura y despliegue de imágenes con *PIL* se importa el módulo *image* de la biblioteca *PIL*.

```
from PIL import Image
```

Para la lectura se utilizó la función *open* y para el despliegue, en Google Colab la imagen se imprime solo poniendo la referencia a la variable.

```
img=Image.open(imagePath)
img
```

Para la lectura y despliegue con este módulo no se utilizó un ciclo for debido a que al realizarlo de esta manera únicamente se despliega la última imagen en la lista de imágenes ya que no se mantiene la impresión de las imágenes que habían sido desplegadas anteriormente, por lo que para este caso, las imágenes debían ser leídas y mostradas una a una.



Fig. 4. Lectura y despliegue de imágenes con el módulo *PIL*

E. Sci-Py

Para el caso de la lectura de imágenes con el módulo *misc* de *Sci-Py* está descontinuada y se prefiere utilizar *imageio*.

Se ocupa la función *imread* del paquete *imageio*. También se ocupa *matplotlib* para el despliegue de la imagen.

```
img = imageio.imread(imagePath)
plt.imshow(img)
plt.show()
```

3.1.2. Imprimir el tipo de imagen, el tamaño y el tipo de dato.

A continuación, se explica la manera en la que se imprimieron los atributos tipo, tamaño, tipo de dato y nombre de la imagen. Para desplegar la información del tipo de imagen y nombre del archivo, solo es necesario obtener el nombre o la ruta de la imagen. Para el tamaño de la imagen fue necesario usar la función *stat* del módulo *os*, el cual nos retornará un conjunto de estadísticas del archivo. Entre dichas características, se encuentra el tamaño en Bytes. Para el tipo de dato, se emplea el atributo *dtype* de los arreglos de *numpy* (recordemos que la imagen se guarda en un arreglo de *numpy*).

```
Formato: imagePath[-3:]
Tamaño: os.stat(imagePath).st_size
Tipo: img.dtype
Nombre: imagePath.split("/")[-1]
```

3.1.3. De las imágenes “lena_color_512.tif”, “peppers_color.tif”. Desarrolla un script con OpenCV y Scikit-Image para cambiar el espacio de color de:

3.1.3.1. RGB a Escala de grises.

A. OpenCV

Para la conversión de imagen RGB a imagen en escala de grises con *OpenCV* se utilizó la función *imread* en la que se agregó un 0 como segundo parámetro, este 0 indica que la lectura de la imagen debe ser en escala de grises.

```
imgGray=cv2.imread(imagePath,0)
```

A continuación se agrega el prototipo de la función y

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

los valores que puede tomar.

```
cv2.imread(path, flag)
```

El parámetro *flag* puede tomar 3 valores:

- 1 o `cv2.IMREAD_COLOR`: lectura de imagen a color en formato RGB.
- 0 o `cv2.IMREAD_GRAYSCALE`: lectura de imagen en escala de grises.
- -1 o `cv2.IMREAD_UNCHANGED`: lectura de imagen a color en formato ARGB.

A continuación, se presentan ejemplos de conversión a escala de grises con *OpenCV*.



Fig. 5. Conversión RGB a escala de grises con *OpenCV*

B. Scikit-Image

Para la conversión de imagen RGB a imagen en escala de grises con *Scikit-Image* se utilizó la función `rgb2gray` y se mandó como parámetro la imagen leída con la función `imread`. El resultado de la imagen en escala de grises se almacena en una nueva variable para después ser mostrada con `imshow` y `show`.

```
img=io.imread(imagePath,plugin='pil')
imgGray = color.rgb2gray(img)
```

A continuación, se presentan ejemplos de conversión a escala de grises con *Scikit-Image*.



Fig. 6. Conversión RGB a escala de grises con *Scikit-Image*

3.1.3.2. RGB a YUV.

A. OpenCV

Para la conversión de imagen RGB a imagen YUV con *OpenCV* se utilizó la función `cvtColor` y se mandó como parámetro la imagen leída con la función `imread` y en el parámetro *flag* el valor `cv2.COLOR_BGR2YUV` para indicar una conversión de BGR a YUV. El resultado de la imagen en YUV se almacena en una nueva variable para después ser

mostrada `cv2_imshow`.

```
img=cv2.imread(imagePath)
img_yuv=cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
```

A continuación, se presentan ejemplos de conversión a YUV con *OpenCV*.



Fig. 7. Conversión RGB a escala YUV con *OpenCV*

B. Scikit-Image

Para la conversión de imagen RGB a imagen YUV con *Scikit-Image* se utilizó la función `rgb2yuv` y se mandó como parámetro la imagen leída con la función `imread`. El resultado de la imagen en escala yuv se almacena en una nueva variable para después ser mostrada con `imshow` y `show`.

```
img=io.imread(imagePath,plugin='pil')
img_yuv = color.rgb2yuv(img)
```

A continuación, se presentan ejemplos de conversión a YUV con *Scikit-Image*.

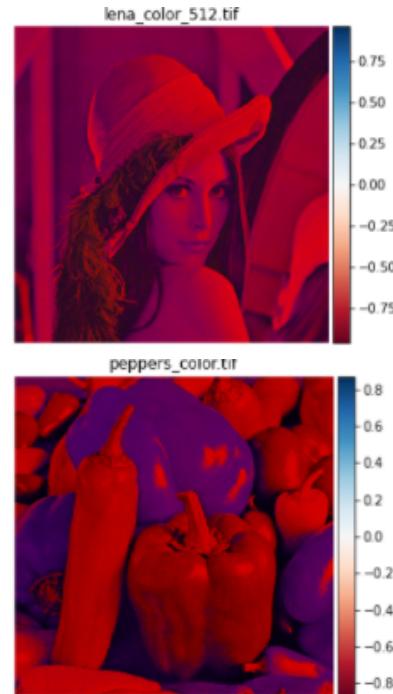


Fig. 8. Conversión RGB a YUV con *Scikit-Image*

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

3.1.3.3. RGB a HSV

A. OpenCV

Para la conversión de imagen RGB a imagen HSV con *OpenCV* se utilizó la función *cvtColor* y se mandó como parámetro la imagen leída con la función *imread* y en el parámetro *flag* el valor `cv2.COLOR_BGR2HSV` para indicar una conversión de BGR a HSV. El resultado de la imagen en HSV se almacena en una nueva variable para después ser mostrada con *cv2_imshow*.

```
img=cv2.imread(imagePath)
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
```

A continuación, se presentan ejemplos de conversión a HSV con *OpenCV*.



Fig. 9. Conversión RGB a HSV con *OpenCV*

B. Scikit-Image

Para la conversión de imagen RGB a imagen HSV con *Scikit-Image* se utilizó la función *rgb2hsv* y se mandó como parámetro la imagen leída con la función *imread*. El resultado de la imagen en escala hsv se almacena en una nueva variable para después ser mostrada con *imshow* y *show*.

```
img=io.imread(imagePath, plugin='pil')
img_hsv = color.rgb2hsv(img)
```

A continuación, se presentan ejemplos de conversión a HSV con *Scikit-Image*.

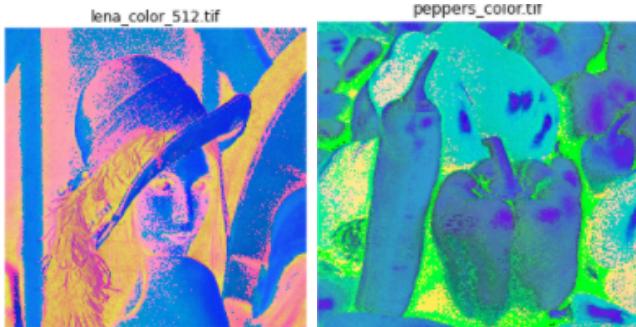


Fig. 10. Conversión RGB a HSV con *Scikit-Image*

3.1.3.4. Despliega la paleta de colores de RGB por separado

A. OpenCV

Para la descomposición de colores RGB de una

imagen se utilizó la función *split* de *cv2* y el resultado se asignó a una tupla en la que se almacenaron las componentes RGB por separado. Una vez teniendo las tuplas separadas, procedimos a crear una lista de ceros que después combinaremos con cada componente para obtener una imagen por componente. El resultado de la combinación entre el arreglo de ceros y la componente RGB se pasaba como parámetro a la función *cv2_imshow* para mostrar la imagen asociada a la componente RGB.

```
image=cv2.imread(imagePath)
(B, G, R) = cv2.split(image)
zeros = np.zeros(image.shape[:2], dtype="uint8")
cv2_imshow(cv2.merge([zeros, zeros, R]))
cv2_imshow(cv2.merge([zeros, G, zeros]))
cv2_imshow(cv2.merge([B, zeros, zeros]))
```

A continuación se muestran las componentes RGB por separado.

R



Fig. 11. Canal Rojo (R) generado con *OpenCV*

G



Fig. 12. Canal Verde (G) generado con *OpenCV*

B



Fig. 13. Canal Azul (B) generado con *OpenCV*

B. Scikit-Image

Para la descomposición de colores RGB de una

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

Imagen en *Scikit-Image* primero se realizó una copia de la imagen original utilizando la función *copy* asociada a la imagen leída. Una vez teniendo la copia, la componente deseada se deja como estaba y el resto de las componentes se llenan con ceros. Una vez hecho esto, se muestra la imagen con la función *imshow*. Esto se realiza para las 3 componentes (R,G,B).

A continuación se muestra un ejemplo para la separación del canal rojo de una imagen.

```
red_img = img.copy()
red_img[:, :, 1] = 0
red_img[:, :, 2] = 0
fig, ax1 = plt.subplots(1, 1)
io.imshow(red_img, axes=ax1)
```

R.



Fig. 14. Canal Rojo generado con *Scikit-Image*

G.



Fig. 15. Canal Verde generado con *Scikit-Image*

B.

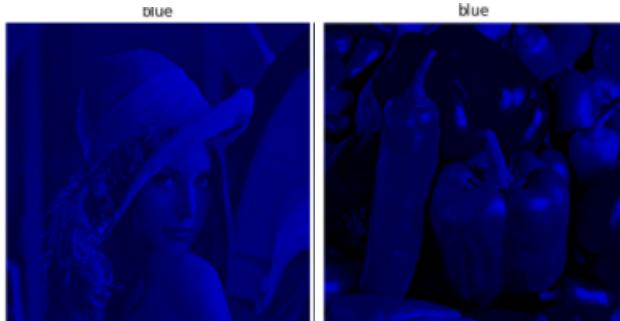


Fig. 16. Canal Azul generado con *Scikit-Image*

3.1.3.5. Despliega la paleta de colores HSV por separado

A. OpenCV.

Para la descomposición de colores HSV de una imagen primero se realiza la conversión a imagen HSV y posteriormente se realiza un split de la imagen para obtener los canales hue (H), saturation (S) y value (V).

Una vez teniendo los canales HSV se imprime cada uno utilizando la función *imshow* especificando el parámetro *cmap* como *hsv* para imprimir en formato HSV en vez de RGB que es el default.

```
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
(hue_img, s_img, value_img) = cv2.split(img_hsv)
```

A continuación, se agrega un ejemplo de impresión del canal “HUE” de una imagen.

```
ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
```

A continuación se muestran las componentes HSV de una imagen por separado.



Fig. 17. Despliega la paleta de colores HSV generado con OpenCV

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1



Fig. 18. Despliega la paleta de colores HSV generado con OpenCV

B. Scikit-Image.

Para la descomposición de colores HSV de una imagen en *Scikit-Image* se utilizó la función *rgb2hsv* con la que se obtuvo una conversión de imagen RGB a HSV. Una vez con la imagen en colores HSV, se creó una lista por cada componente asignando cada componente.

Una vez teniendo los canales HSV por separado, se imprime cada uno utilizando la función *imshow* especificando el parámetro *cmap* como *hsv* para imprimir en formato HSV en vez de RGB que es el default.

```
hsv_img = rgb2hsv(rgb_img)
hue_img = hsv_img[:, :, 0]
s_img = hsv_img[:, :, 1]
value_img = hsv_img[:, :, 2]
```

A continuación, se agrega un ejemplo de impresión del canal “HUE” de una imagen.

```
ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
```

A continuación se muestran las componentes HSV de una imagen por separado.

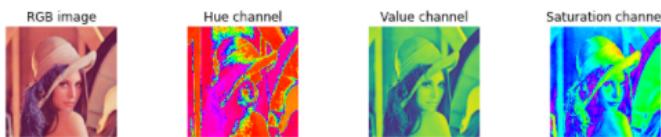


Fig. 19. Despliega la paleta de colores HSV generado con Scikit-Image

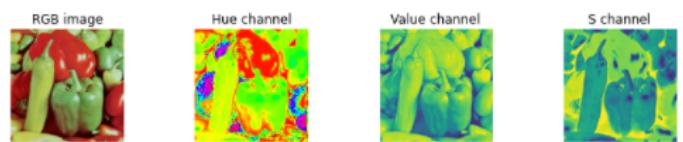


Fig. 20. Despliega la paleta de colores HSV generado con Scikit-Image

3.2. De una imagen que usted escoja, dejarla en escala de grises y procure que sea igual en renglones y columnas. Programe una función que realice decimación de una imagen, reduciéndola a la mitad de su tamaño original. Y promediando en grupos de 4 píxeles. Pruebe con su imagen.

Para poder leer la imagen en escala de grises se utilizó *OpenCV*, y se lee la imagen con el parámetro 0, como vimos en los resultados anteriores.

Para mantener a la imagen con el mismo número de columnas y renglones, se realizó una validación con las dimensiones del arreglo que corresponde a la imagen. Si el número de columnas y renglones es diferente, se realiza un redimensionamiento con la función *resize* de *cv2*.

La función definida para realizar la decimación, toma en cuenta los índices hasta el penúltimo índice tanto de columnas como renglones. Para realizar la decimación, se realizó el promedio de los valores adyacentes formando un cuadrado de 4 píxeles del índice en cuestión, los cuales se mapean a una estructura de datos (lista) nueva para poder guardar los nuevos valores.

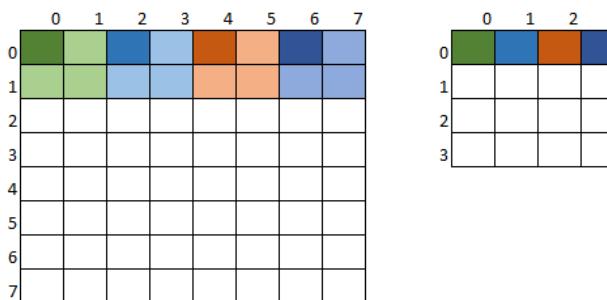


Fig. 20. Diagrama para ejemplificar la decimación

La parte más importante de la función es el siguiente par de ciclos anidados que realizan el mapeo anteriormente mencionado.

```
nuevoArreglo = list()
for renglon in range(0,height-1,2):
    nuevoRenglon = list()
    for columna in range(0,width-1,2):
        suma = int(img[renglon,columna]) +
               int(img[renglon,columna+1]) +
               int(img[renglon+1,columna]) +
               int(img[renglon+1,columna+1])
        nuevoRenglon.append(int(suma/4))
    nuevoArreglo.append(nuevoRenglon)
```

Podemos ver el resultado de utilizar la función anteriormente descrita en la siguiente imagen.

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1



Fig. 21. Decimación

3.3. Usando funciones de Python ya existentes y la imagen que usted escoja.

3.3.1. Reajusta el tamaño de la carpeta imágenes a 10 veces su tamaño original

Para el escalamiento de una imagen se utilizó la función `resize` en la que se pasó como primer parámetro la imagen a escalar, una tupla (0,0) que indica que no se desea especificar un valor determinado para el tamaño de la imagen y los parámetros fx y fy para indicar el factor de escala en x y y. En este caso con un valor de 10 para obtener imágenes 10 veces más grandes que la original. Esto se realizó en un ciclo `for` para poder escalar todas las imágenes en la carpeta.

```
img = cv2.imread(imagePath)
height = np.size(img, 0)
width = np.size(img, 1)
rszImg = cv2.resize(img, (0,0), fx=10, fy=10)
cv2_imshow(rszImg)
```

A continuación, se muestra una imagen original y una porción del escalamiento.



Fig. 22. Aumentando el tamaño original 10 veces

3.3.2. Reajusta el tamaño de la carpeta imágenes a 3 veces el tamaño original.

Para este ejercicio se realizó el código utilizado en el ejercicio anterior, la única diferencia es que para este ejercicio fx y fy es 3 el cual indica un valor de escalamiento a 3 veces el valor original de la imagen.

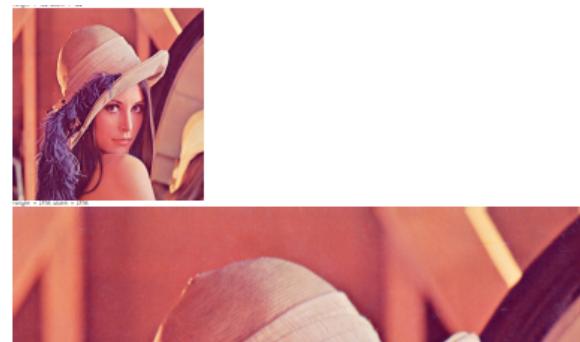


Fig. 23. Aumentando el tamaño original 3 veces

3.3.3. Rote la imagen a 45, 90 y 180 grados, guárdalas en formato png.

Para poder rotar la imagen, ocupamos la librería `cv2` y `numpy`, la primera para poder leer y manipular la imagen, y `numpy` para trabajar con las columnas y las filas.

```
image=cv2.imread('/content/imágenes2/retinaRGB.jpg')
')
ancho = image.shape[1] #columnas
alto = image.shape[0] # filas
```

Posteriormente obtenemos nuestra matriz de rotación, para poder girar nuestra imagen.

```
M45=cv2.getRotationMatrix2D((ancho//2,alto//2),45,
1) # 45 grados
```

Transformamos la imagen y la imprimimos en pantalla.

```
image45 = cv2.warpAffine(image,M45,(ancho,alto))
print("45 grados")
```

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

```
cv2_imshow(image45)
```

Para finalizar, guardamos la imagen en formato **PNG**.

```
cv2.imwrite("/content/image45.png",image45)
```

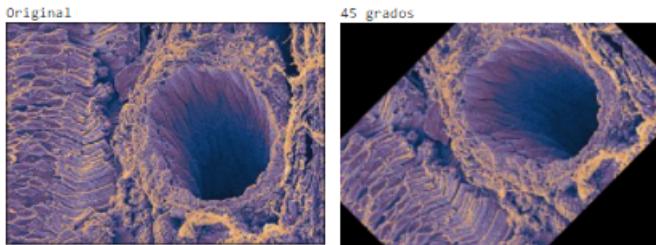


Fig. 24. Imagen original e imagen rotada 45 grados

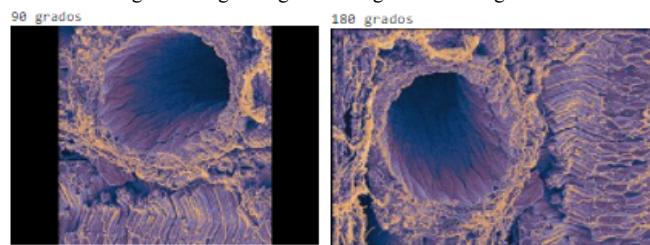


Fig. 25. Imagen rotada 90 grados e imagen rotada 180 grados

3.4. Convierte la imagen peppers_color.tif a escala de grises.

3.4.1. Recortela de manera que solo quede uno de los pimientos verdes en ese recorte

3.4.2. Guárdela en formato .jpg.

Para poder realizar la actividad 3.4, 3.4.1, 3.4.2, primero convertimos nuestra imagen original a escala de grises con ayuda de la librería cv2, esto se realizó de la siguiente manera.

```
image=cv2.imread("/content/Imagenes/peppers_color.tif",0)
```

Para poder recortar la imagen y solo quedara un pimiento, nos apoyamos en las columnas y filas.

```
imageOut = image[180:500,183:420]#Columnas #Filas
```

Por último, guardamos la imagen.

```
cv2.imwrite("/content/pimientoBlancoNegro.jpg",ima
```



Fig. 26. Imagen convertida a escala de grises.



Fig. 27. Imagen recortada

3.5. Un formato de imágenes sin ningún tipo de codificación se conoce como formato crudo (RAW). De la imagen “rosa800x600.raw” lea y despliegue la imagen. Tome en cuenta que esta imagen maneja la precisión de integer8 y el tamaño es de 600 x 800 píxeles.

Para poder procesar este formato, primero definimos las filas y las columnas. Esta información debe ser proporcionada en algún lado, en este caso, en el nombre de la imagen.

```
rows = 800
columns = 600
```

Posteriormente leímos el archivo y lo convertimos en un arreglo *numpy*, para poder trabajar con sus dimensiones. Se ocupa la función *fromfile* para crear un ndarray con la información de un archivo, especificando la cantidad de elementos a leer y el tipo de dato a leer.

```
file = open("/content/Imagenes/rosa800x600.raw")
img = np.fromfile(file, dtype = np.uint8, count = rows * columns)
```

Las imágenes son un arreglo bidimensional, y el arreglo leído de una imagen tipo raw solo es un arreglo unidimensional, por lo cual es necesario redimensionar el arreglo para poder desplegarla como una imagen. Para ello se modificó el atributo shape de la imagen leída como se muestra a continuación.

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

```
img.shape = (img.size // columns, columns)
```

La imagen resultante del proceso anterior se puede observar en la siguiente imagen.



Fig. 28. Formato .RAW, procesado

Funciones

Como parte de nuestros resultados, generamos un formulario de funciones que nos permitirá tener un sitio donde buscar funciones que aprendimos a utilizar durante la práctica. Dicho formulario lo encontraremos al final del documento en el anexo II.

IV. CONCLUSIONES

De manera general, la práctica nos sirvió mucho para aprender a utilizar funciones de manipulación de imágenes en Python. El uso y el entendimiento de funciones de OpenCV, por ejemplo, nos permitió realizar la función de decimación sin preocuparnos por la manipulación de la imagen, y nos permitió concentrarnos en la lógica de la función.

En un punto en específico donde tuvimos problemas fue en el despliegue de la imagen tipo RAW. Para esta parte, el despliegue de la imagen depende del redimensionamiento de la información contenida en el archivo. En la primera iteración, el resultado arrojado era una imagen aberrante sin mucho sentido. Durante la sesión de dudas con el profesor Boris Escalante, nos dimos cuenta de que el problema es que invertimos el número de columnas y renglones, por lo que el redimensionamiento no arrojó un resultado correcto.

Si bien, el uso de los paquetes nos facilita el trabajo de manipulación de imágenes, también podemos manipularlas a mano alterando o modificando el arreglo que contiene la información de la imagen. Esto es bueno y problemático, ya que como vimos, somos libres de realizar nuestras propias funciones y operaciones sobre la imagen, pero esto también puede arrojarnos resultados erróneos.

A. Paquetes

Durante el desarrollo de la práctica realizamos varias veces la lectura y despliegue de imágenes con diferentes

paquetes de python. Realmente no cambia mucho leer una imagen con uno u otro paquete. Llegamos a esta conclusión ya que prácticamente todos los paquetes utilizan internamente el paquete PIL para lectura de imágenes, y prácticamente todos los paquetes, con su respectiva función imread u open, retornan un objeto tipo ndarray.

Algo que nos llamó la atención es, dado esto, ¿Por qué existe una gran variedad de paquetes para la lectura de imágenes? Pensamos que la respuesta va más allá de las funciones de lectura y despliegue, que, si bien OpenCV y Matplotlib fueron nuestras favoritas, la utilización de uno u otro paquete depende de las funciones extras de dichos paquetes.

Por ejemplo, nos fue mucho más sencillo transformar las imágenes con OpenCV, con las funciones como resize o warpAffine. Pero también observamos que la transformación a diferentes espacios de colores (ya sea YUV o HSV) son más sencillas con Scikit-Image, ya que ocupa funciones con nombres muy ilustrativos para dicha tarea.

B. Formatos HSV y YUV

Durante una sesión con el Dr. Boris Escalante, pudimos observar de manera más clara las transformaciones a estos formatos. Algo que no comprendimos era por qué las imágenes, al desplegarlas tanto en HSV como en YUV, se ven de colores verdes o colores muy oscuros.

Durante la sesión de dudas entendimos cómo entender estos canales. Para HSV, el canal Hue se puede desplegar en blanco y negro o con un mapa de color HSV para poder ver los colores. Pero los canales Saturation y Value son más visibles a escala de grises.

Para YUV, es más ilustrativo mostrar el canal Y en blanco y negro, ya que este canal está representado en escalas de gris. Los canales U y V son más visibles en colores, ya que estos tienen información especial sobre los colores que componen la imagen.

Algo que entendimos después de esta explicación, es que, aunque no podamos visualizar de manera correcta estos canales, por su naturaleza y dado que la mayor parte de las funciones de despliegue de imágenes imprimen las imágenes en formato RGB, con las funciones que utilizamos ya tenemos acceso a esos formatos y a sus canales, y podemos utilizar la información de los canales para poder realizar diferentes aplicaciones.

Por ejemplo, si nos importa solo el color de las imágenes, podemos tomar el canal Hue, y si necesitamos información del contorno o definición de las figuras, podríamos utilizar el canal Y del formato YUV.

REFERENCIAS

- [1] Y. González, “Tipos de formato de imagen más usados,” *Grupo Atico* 34, 2020.
https://protecciondatos-lodp.com/empresas/tipos-formato-imagen/#Que_es_el_formato_de_una_imagen (accessed Sep. 06, 2021).
- [2] Lucialonzo, “¿Qué es y para qué sirve el formato raw?,”

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

- DOMESTIKA, 2020.
<https://www.domestika.org/es/blog/2741-que-es-y-para-que-sirve-el-formato-raw> (accessed Sep. 06, 2021).
- [3] C. Musso, “¿QUÉ ES Y PARA QUÉ SIRVE EL ESPACIO DE COLOR?,” *Blog del fotógrafo*, 2018.
<https://www.blogdelfotografo.com/espacio-color/> (accessed Sep. 06, 2021).
- [4] “Modelo HSV,” *EcuRed*. https://www.ecured.cu/Modelo_HSV (accessed Sep. 06, 2021).
- [5] Online-convert, “YUV (YUV Encoded Image File),” *Formatos de archivo*. <https://www.online-convert.com/es/formato-de-archivo/yuv> (accessed Sep. 06, 2021).

ANEXO I

4. Ejercicios básicos de imágenes.**4.1. Abrir y escribir una imagen a un archivo.**

Abrimos la imagen con ayuda de la librería scipy.

```
from scipy import misc
import imageio
f = misc.face()
```

Escribimos la imagen.

```
imageio.imwrite('face.png', f) # uses the Image module (PIL)
```

Imprimimos la imagen.

```
import matplotlib.pyplot as plt
plt.imshow(f)
plt.show()
```

**4.2. Creación de un arreglo numpy de un archivo de imagen.**

```
from scipy import misc
import imageio
face = misc.face()
imageio.imwrite('face.png', face) # Se salva la imagen como png

face = imageio.imread('face.png') # Se lee la imagen png
type(face)

face.shape, face.dtype

((768, 1024, 3), dtype('uint8'))
```

-¿De qué tipo es la variable face? ndarray

La variable face es un arreglo n dimensional de Numpy (ndarray), y el arreglo se compone de números enteros sin signo de 8 bits.

-¿Qué resultado arroja face.shape?

(768, 1024, 3)

Este arreglo representa las dimensiones de la imagen,

renglones, columnas y profundidad de color.

-Si fuera una imagen en tonos de gris, ¿cuál sería el resultado esperado de face.shape?

(768, 1024)

En este caso NumPy elimina la última dimensión del arreglo ya que al ser en blanco y negro se tiene 0 para esta componente.

4.3. Abrir archivos raw

```
face.tofile('face.raw') # Se crea el archivo raw binario
face_from_raw = np.fromfile('face.raw',
dtype=np.uint8)
face_from_raw.shape

face_from_raw.shape = (768, 1024, 3)
```

**4.4. Despliegue de imágenes.**

```
f = misc.face(gray=True) # Lee la imagen en escala de grises
import matplotlib.pyplot as plt
plt.imshow(f, cmap=plt.cm.gray)
```



-¿Qué pasa si al desplegar la imagen con plt.imshow no se especifica el mapa de color plt.cm.gray?

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

La imagen se despliega en tonos verdes.

-¿Cuál es el mapa de color default de imshow?

El valor por default del mapa de color (cmap) en la función imshow de Matplotlib es None.

-Imprima la forma de f (propiedad shape de f) y compare contra la forma de la misma imagen a color.

Solo cambia el tercer dígito el cual representa la dimensión del color. Cuando se lee a color tiene un valor de 3 y cuando se lee en escala de grises tiene un valor de 0 por lo que se elimina del arreglo de numpy.

Incrementando el contraste especificando los valores mínimo y máximo en el despliegue:

```
plt.imshow(f, cmap=plt.cm.gray, vmin=30, vmax=200)
```

```
# Remueve los ejes y las marcas (ticks)
```

```
plt.axis('off')
```

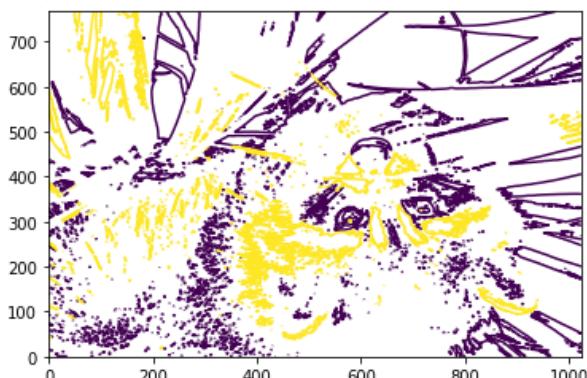
```
(-0.5, 1023.5, 767.5, -0.5)
```



Dibujando las líneas de contorno con la instrucción contour.

```
plt.contour(f, [50, 200])
```

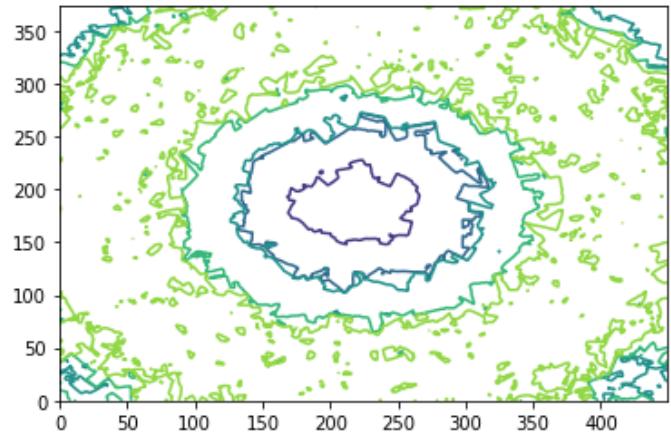
```
<matplotlib.contour.QuadContourSet at 0x7fe17b68bcd0>
```



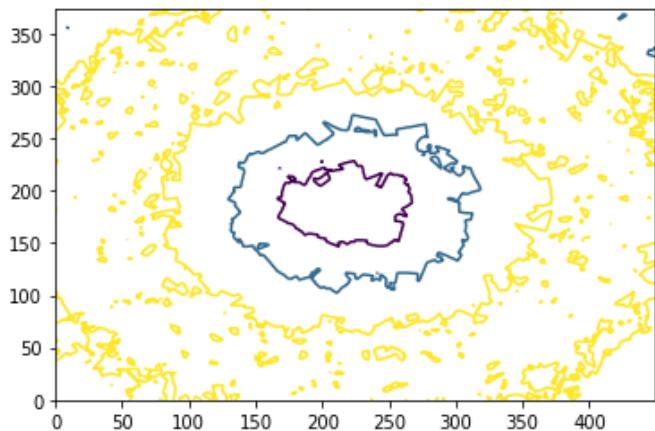
Despliegue los contornos de la imagen contour_gray.png.

```
f = imageio.imread('contour_gray.png')
```

```
plt.contour(f, 5)
```



```
plt.contour(f, [50, 100, 200])
```



4.5. Manipulaciones básicas.

Obteniendo el valor de un píxel en una imagen.

```
face = misc.face(gray=True)
```

```
face[0, 40]
```

-¿Cuánto vale el pixel face[0, 40]?

```
127
```

Accediendo a secciones de la imagen.

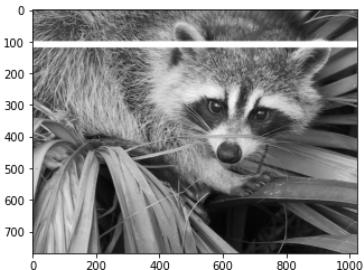
```
face[10:13, 20:23]
```

```
array([[141, 153, 145],
       [133, 134, 125],
       [ 96,  92,  94]], dtype=uint8)
```

```
face[100:120] = 255
```

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

```
# Accesando secciones de la imagen
face[10:13, 20:23]
#¿Qué efecto tiene la instrucción face[100:120] = 255 en la imagen de abajo?
face[100:120] = 255
plt.imshow(face,cmap="gray")
plt.show()
```



-¿Qué efecto tiene la instrucción `face[100:120] = 255` en la imagen de abajo?

[100:120]-> dibuja la franja en la imagen.

255->La franja se pinta de blanco.

Pinte una franja vertical gris en la imagen que vaya de la columna 200 a la columna 220.

[:,200:220]-> dibuja la franja en la imagen.

150>La franja se pinta de gris.



```
lx, ly = face.shape
X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx *
ly / 4
# Masks
face[mask] = 0
# Indexado con rangos
face[range(400), range(400)] = 255
```

-¿Cuánto vale lx, ly?

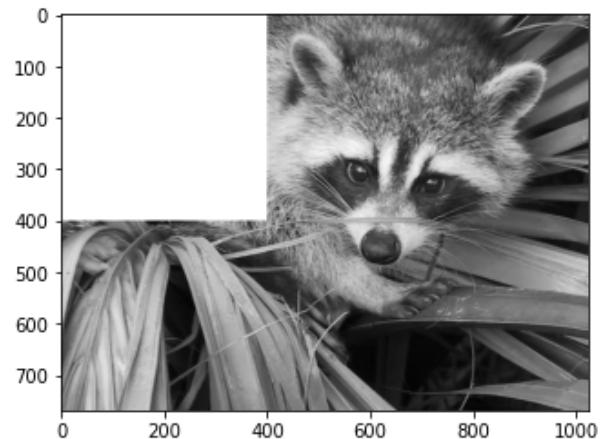
lx = 768, ly = 1024

-¿Qué efecto tiene en la imagen la instrucción
`face[range(400), range(400)] = 255`?

`face[range(400), range(400)] = 255` -> Al ocupar este comando, no se pinta la franja.

Al ocupar [0:400, 0:400]=255 -> Se pinta un cuadro de la siguiente manera.

```
face[0:400, 0:400] = 255
plt.imshow(face,cmap="gray")
plt.show()
```

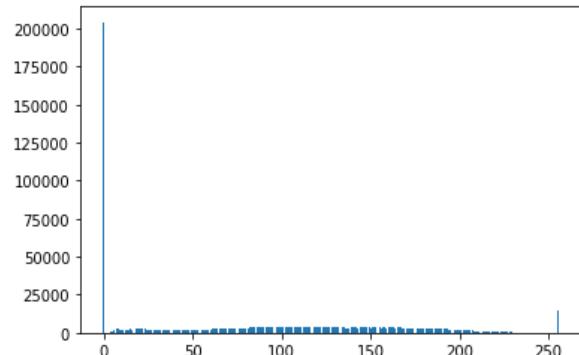


-Modifique el código para que la máscara sea un círculo más pequeño y despliegue el resultado.

Desplegando el histograma de la imagen en grises del mapache original.

```
hist, bins = np.histogram(face,
bins=256,range=(0,256))
plt.bar(bins[0:-1], hist)
```

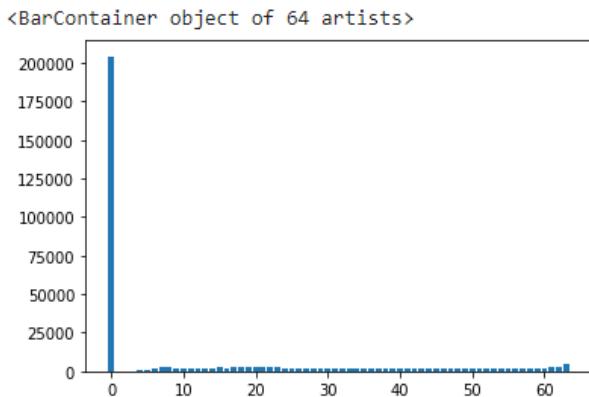
<BarContainer object of 256 artists>



Modificando el código para que sólo se tengan 64 bins en el histograma.

```
hist, bins      = np.histogram(face,      bins=64,
range=(0,64))
plt.bar(bins[0:-1], hist)
```

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

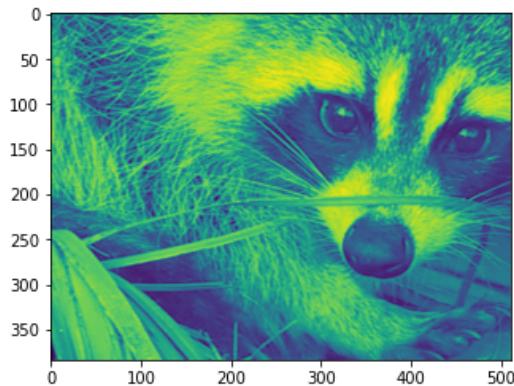


Transformaciones geométricas.

```
from scipy import ndimage
face = misc.face(gray=True)
lx, ly = face.shape
```

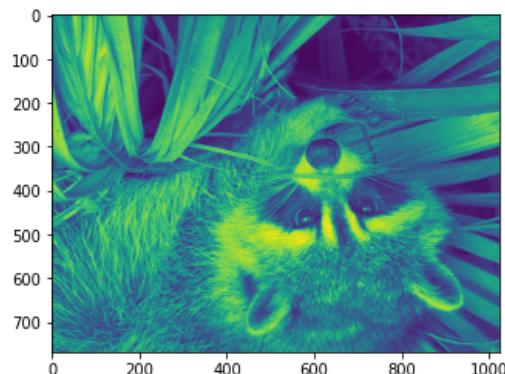
Recortando la imagen.

```
crop_face = face[lx // 4: - lx // 4, ly // 4: - ly // 4]
plt.imshow(crop_face)
```



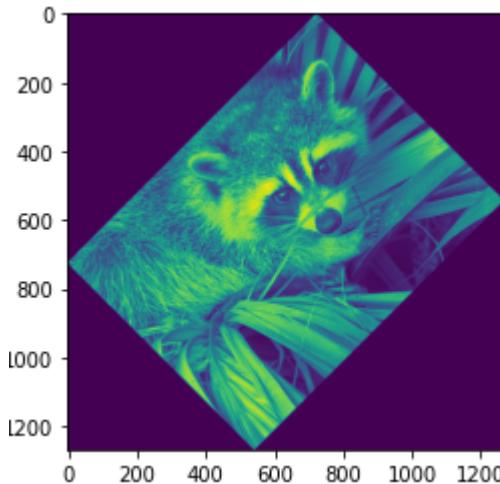
-Voltear verticalmente.

```
flip_ud_face = np.fliplr(face)
plt.imshow(flip_ud_face)
```

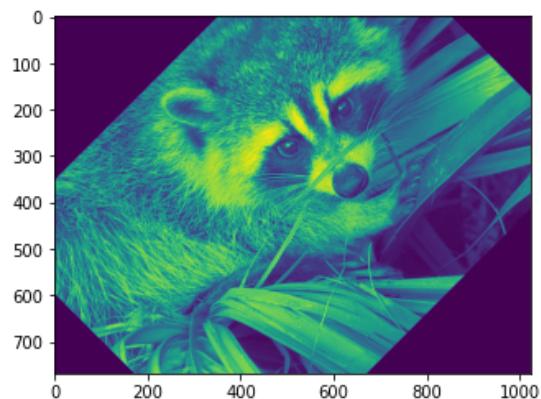


Rotacion.

```
rotate_face = ndimage.rotate(face, 45)
plt.imshow(rotate_face)
```



```
rotate_face_noreshape = ndimage.rotate(face, 45,
reshape=False)
plt.imshow(rotate_face_noreshape)
```



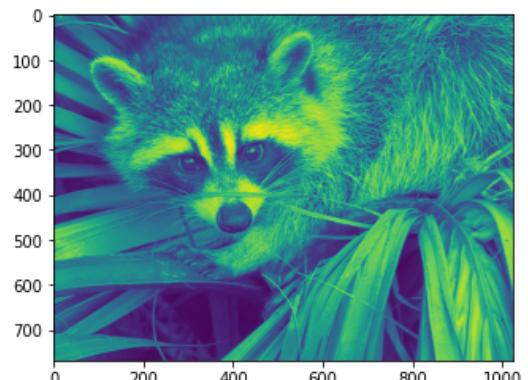
-Investigue el operador //

Es la división entera.

-¿Qué efecto tiene el signo negativo en `crop_face = face[lx // 4: - lx // 4, ly // 4:- ly // 4]`?

Voltee la imagen original del mapache, pero esta vez horizontalmente, de izquierda a derecha.

```
flip_ud_face = np.fliplr(face)
plt.imshow(flip_ud_face)
```



RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

ANEXO II

Biblioteca	Paquete	Función	Parámetros	Descripción
Matplotlib	image	imread	Dirección del archivo	Lee una imagen, regresa un ndarray de numpy
	pyplot	imshow	Arreglo de bits tipo ndarray	Despliega una imagen como gráfica
	pyplot	axis	Opción, "off" para quitar los ejes	Manipula los ejes de una gráfica
Pydicom		dcmread	Dirección del archivo	Lee una imagen formado dcm. La imagen se guarda en el atributo pixel_array del objeto que retorna
OpenCV		imread	Dirección del archivo 0 para escala de grises	Regresa un objeto de tipo imagen
		imshow	Imagen de entrada	Muestra la imagen
		cvtColor	Imagen de entrada (array) Constante de cambio	Convierte los colores cv2.COLOR_BGR2YUV para YUV cv2.COLOR_BGR2HSV para HSV
		split	Imagen de entrada (array)	Separa los canales del arreglo
		merge	Lista de arreglos lineales de bits	Junta los arreglos para tener un arreglo 3D
		resize	Imagen de entrada (array) Dimensiones deseadas Factor de escala en x Factor de escala en y	Redimensiona una imagen
		getRotationMatrix2D	Centro Angulo Escala	Calcula una matriz de rotación tomando en cuenta un centro.
		warpAffine	Imagen de entrada (array) Matriz de transformación Tamaño deseado	Transforma una imagen tomando en cuenta una matriz de transformación. Se define el tamaño deseado para el resultado.
		imwrite	Dirección de la imagen nuevo Imagen de entrada (array)	Guarda un arreglo como una imagen en la dirección y con el nombre especificados

RECONOCIMIENTO DE PATRONES. PRÁCTICA 1

Scikit-Image	io	imread	Dirección del archivo Plugging	Lee una imagen. Ocupar plugin="PIL" en caso de ser archivo .tif
	io	show	Imagen de entrada Elemento tipo Axes de Matplotlib	Muestra una imagen, se puede especificar a la gráfica en la cual se mostrará
	color	rgb2gray	Imagen de entrada (array)	Cambia una imagen de RGB a escala de grises
	color	rgb2yuv	Imagen de entrada (array)	Cambia una imagen de RGB a YUV
	color	rgb2hsv	Imagen de entrada (array)	Cambia una imagen de RGB a HSV
PIL	Image	open	Dirección del archivo	Lee una imagen
Sci-Py	Se descontinua, se ocupa imageio			
imageio		imread	Dirección del archivo	Lee una imagen
os		stat	Dirección del archivo	Nos regresa información del archivo. Ocupamos el parámetro st_size para obtener el tamaño de la imagen
numpy		fromfile	Archivo, objeto tipo File Tipo de dato a leer Número de elementos a leer	Nos permite leer un arreglo desde un archivo

▼ Anexo III

<https://colab.research.google.com/drive/12bG77xofR6068WCcwwAInA3H5YDFPxI#scrollTo=a-Lgs9Lq12-i>

► Pre-ejercicio

Celdas para descargar las imágenes

[] ↓ 1 cell hidden

▼ 4.1. De la carpeta de imágenes: realiza las siguientes actividades.

4.1.1. Desarrolla un script para leer y desplegar cada imagen con los paquetes de Matplotlib, OpenCV, Scikit-Image, PIL y Sci-Py.

4.1.2.. Imprimir el tipo de imagen, el tamaño y el tipo de dato

```
# Se ocupará os para obtener la ruta de los archivos
import os

listaImagenes = []
for carpeta in ["Imagenes", "imagenes2"]:
    for name in os.listdir("/content/" + carpeta):
        listaImagenes.append("/content/" + carpeta + "/" + name)
```

▼ Matplotlib

```
import matplotlib.image as mpimg
import matplotlib.pyplot as plt
# se usa para formados dcm
import pydicom as dicom

# para cada imagen
for imagePath in listaImagenes:
    # siempre que no sea una imagen tipo dcm o raw
    # PREGUNTA: Se puede leer una imagen .dcm o .raw con matplotlib?
    if imagePath.endswith(".dcm"):
        fig, ax1 = plt.subplots(1, 1)
        ds = dicom.dcmread(imagePath)
```

```

Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:],os.stat(imagePath).st_size,
                      ds.pixel_array.dtype,imagePath.split("/")[-1])
ax1.imshow(ds.pixel_array)
ax1.set_title(Texto)
ax1.axis("off")
elif not imagePath.endswith(".raw"):
    fig, ax1 = plt.subplots(1,1)
    img = mpimg.imread(imagePath)
    Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:],os.stat(imagePath).st_size,
                      img.dtype,imagePath.split("/")[-1])
    ax1.imshow(img)
    ax1.set_title(Texto)
    ax1.axis("off")

```

```

# para formato raw
import numpy as np

# tamaño de la imagen
rows = 600
columns = 800

# leer el archivo
file = open("/content/Imagenes/rosa800x600.raw")

# convirtiendo a un arreglo numpy
img = np.fromfile(file, dtype = np.uint8, count = rows * columns)
print("shape original:",img.shape)
print("size original:",img.size)
# cambiamos las dimensiones de la imagen
img.shape = (img.size // columns, columns)
print("new shape:",img.shape)
print("new size:",img.size)

# Mostrando la imagen
fig, ax1 = plt.subplots(1,1)
ax1.imshow(img)
Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format("raw",
                      os.stat("/content/Imagenes/rosa800x600.raw").st_size,
                      img.dtype,"rosa800x600.raw")
ax1.set_title(Texto)
ax1.axis("off")

comentario = """

```

```
Matplotlib can only read PNGs natively. Further image formats are
supported via the optional dependency on Pillow. Note, URL strings
are not compatible with Pillow. Check the Pillow documentation_
for more information.
```

```
"""
```

```
# https://towardsdatascience.com/reading-an-image-in-python-without-using-special-libraries-534
```

▼ OpenCV

```
import cv2
# Se ocupa, Colab tiene deshabilitado el imshow de opencv, crashea las sesiones
from google.colab.patches import cv2_imshow

# para cada imagen
for imagePath in listaImagenes:
    if not (imagePath.endswith(".dcm") or imagePath.endswith(".raw")):
        img = cv2.imread(imagePath)
        Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:],
                           os.stat(imagePath).st_size,
                           img.dtype,imagePath.split("/")[-1])
        print(Texto)
        cv2_imshow(img)
        # cv2.imshow(img)
comentario = """
. - Windows bitmaps - \*.bmp, \*.dib (always supported)
. - JPEG files - \*.jpeg, \*.jpg, \*.jpe (see the *Note* section)
. - JPEG 2000 files - \*.jp2 (see the *Note* section)
. - Portable Network Graphics - \*.png (see the *Note* section)
. - WebP - \*.webp (see the *Note* section)
. - Portable image format - \*.pbm, \*.pgm, \*.ppm \*.pxm, \*.pnm (always supported)
. - PFM files - \*.pfm (see the *Note* section)
. - Sun rasters - \*.sr, \*.ras (always supported)
. - TIFF files - \*.tiff, \*.tif (see the *Note* section)
. - OpenEXR Image files - \*.exr (see the *Note* section)
. - Radiance HDR - \*.hdr, \*.pic (always supported)
. - Raster and Vector geospatial data supported by GDAL (see the *Note* section)
"""
# https://www.geeksforgeeks.org/python-opencv-cv2-imread-method/
```

▼ Scikit-Image

```
from skimage import io

# para cada imagen
for imagePath in listaImagenes:
```

```

# Pone restricciones de tamaño, no específicamente del formato
if imagePath.endswith(".png") or imagePath.endswith(".jpg"):
    img = io.imread(imagePath)
    Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:], os.stat(imagePath).st_size, img.dtype, imagePath.split("/")[-1])
    fig, ax1 = plt.subplots(1,1)
    io.imshow(img,axes=ax1)
    ax1.set_title(Texto)
    ax1.axis("off")
    io.show()
elif imagePath.endswith(".tif") or imagePath.endswith(".bmp"):
    if imagePath not in ["/content/imagenes2/rxpie-rodilla.tif"]:
        img = io.imread(imagePath,plugin='pil')
        Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:], os.stat(imagePath).st_size, img.dtype, imagePath.split("/")[-1])
        fig, ax1 = plt.subplots(1,1)
        io.imshow(img,axes=ax1)
        ax1.set_title(Texto)
        ax1.axis("off")
        io.show()

img = io.imread("/content/imagenes2/rxpie-rodilla.tif")
Texto = """Tipo de imagen: tif
Tamaño de imagen: {0} Bytes
Tipo de dato: {1}
Archivo: rxpie-rodilla.tif""".format(os.stat("/content/imagenes2/rxpie-rodilla.tif").st_size, img.dtype)
fig, ax1 = plt.subplots(1,1)
io.imshow(img,axes=ax1)
ax1.set_title(Texto)
ax1.axis("off")
io.show()
https://numython.github.io/posts/2016/01/introduccion-scikit-image-procesamiento/

```

▼ PIL

```

from PIL import Image

# lee todas bien menos el formato .raw
imag = Image.open("/content/imagenes2/mri.jpg")
imag
imag = Image.open("/content/imagenes2/abdomen.png")
.
```

```
imag
imag = Image.open("/content/imagenes2/rxpie-rodilla.tif")
imag
imag = Image.open("/content/imagenes2/cta_scan_index.bmp")
imag

# https://pythonexamples.org/python-pillow-read-image/
# https://www.geeksforgeeks.org/python-pil-image-open-method/#:~:text=PIL.-,Image.,call%20the%2
```

▼ Sci-Py

La lectura de imagenes con misc esta descontinuada, se prefiere utilizar imageio

<https://docs.scipy.org/doc/scipy-1.2.1/reference/generated/scipy.misc.imread.html>

```
import matplotlib.pyplot as plt
from scipy import misc
import imageio

for imagePath in listaImagenes:
    # solo jpg o png
    if imagePath.endswith(".jpg") or imagePath.endswith(".png"):
        img = imageio.imread(imagePath)
        Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format(imagePath[-3:],os.stat(imagePath).st_size,
                           img.dtype,imagePath.split("/")[-1])
        plt.imshow(img)
        plt.title(Texto)
        plt.axis("off")
        plt.show()
```

▼ **4.1.3.**De las imágenes “lena_color_512.tif”, “peppers_color.tif”. Desarrolla un script con OpenCV y Scikit-Image para cambiar el espacio de color de:

▼ **4.1.3.1.**RGB a Escala de grises

▼ Open CV

```
#OpenCV
#https://www.delftstack.com/es/howto/python/convert-image-to-grayscale-python/#convertir-una-im
import cv2
from google.colab.patches import cv2_imshow
```

```
# se ocupa el parametro 0 para leerlo en escala de grises
# Imagen 1
print("lena_color_512.tif")
imgGray = cv2.imread("/content/Imagenes/lena_color_512.tif",0)
cv2_imshow(imgGray)

#Imagen 2
print("peppers_color.tif")
imgGray = cv2.imread("/content/Imagenes/peppers_color.tif",0)
cv2_imshow(imgGray)
```

▼ Scikit-Image

```
#Scikit-Image
from skimage import color
from skimage import io
from skimage.external.tifffile import imshow as skimshow

# Imagen 1
img = io.imread("/content/Imagenes/lena_color_512.tif",plugin='pil')
imgGray = color.rgb2gray(img)
fig, ax1 = plt.subplots(1,1)
io.imshow(imgGray,axes=ax1)
ax1.set_title("peppers_color.tif")
ax1.axis("off")
io.show()

# Imagen 2
img = io.imread("/content/Imagenes/peppers_color.tif",plugin='pil')
imgGray = color.rgb2gray(img)
fig, ax1 = plt.subplots(1,1)
io.imshow(imgGray,axes=ax1)
ax1.set_title("peppers_color.tif")
ax1.axis("off")
io.show()
```

▼ 4.1.3.2.RGB a YUV

▼ OpenCV

```
#OpenCV
import cv2
from google.colab.patches import cv2_imshow

print("lena_color_512.tif")
img = cv2.imread("/content/Imagenes/lena_color_512.tif")
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
```

```
cv2_imshow(img_yuv)

print("peppers_color.tif")
img = cv2.imread("/content/Imagenes/peppers_color.tif")
img_yuv = cv2.cvtColor(img, cv2.COLOR_BGR2YUV)
cv2_imshow(img_yuv)
```

▼ Scikit-Image

```
#Scikit-Image
#https://scikit-image.org/docs/dev/api/skimage.color.html#skimage.color.rgb2yuv
from skimage import color
from skimage import io

img = io.imread("/content/Imagenes/lena_color_512.tif", plugin='pil')
img_yuv = color.rgb2yuv(img)

#img_yuv = ((img_yuv+1)/2)*255
#img_yuv = img_yuv.astype('uint32')

fig, ax1 = plt.subplots(1,1)
io.imshow(img_yuv, axes=ax1)
ax1.set_title("lena_color_512.tif")
ax1.axis("off")
io.show()

img = io.imread("/content/Imagenes/peppers_color.tif", plugin='pil')
img_yuv = color.rgb2yuv(img)
fig, ax1 = plt.subplots(1,1)
io.imshow(img_yuv, axes=ax1)
ax1.set_title("peppers_color.tif")
ax1.axis("off")
io.show()
```

▼ 4.1.3.3.RGB a HSV

▼ OpenCV

```
#OpenCV
#https://unipython.com/cambiando-espacios-color/
import cv2
from google.colab.patches import cv2_imshow

print("lena_color_512.tif")
img = cv2.imread("/content/Imagenes/lena_color_512.tif")

img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2_imshow(img_hsv)
```

```
#img_hsv = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)
#cv2_imshow(img_hsv)

print("peppers_color.tif")
img = cv2.imread("/content/Imagenes/peppers_color.tif")
img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
cv2_imshow(img_hsv)
```

▼ Scikit-Image

```
#Scikit-Image
#https://scikit-image.org/docs/dev/api/skimage.color.html#skimage.color.rgb2hsv
from skimage import color
from skimage import io

img = io.imread("/content/Imagenes/lena_color_512.tif", plugin='pil')
img_hsv = color.rgb2hsv(img)
fig, ax1 = plt.subplots(1,1)
io.imshow(img_hsv, axes=ax1)
ax1.set_title("lena_color_512.tif")
ax1.axis("off")
io.show()

img = io.imread("/content/Imagenes/peppers_color.tif", plugin='pil')
img_hsv = color.rgb2hsv(img)
fig, ax1 = plt.subplots(1,1)
io.imshow(img_hsv, axes=ax1)
ax1.set_title("peppers_color.tif")
ax1.axis("off")
io.show()

# Sale muy diferente openCV con Scikit Image, y el YUV sigue sin salir
```

▼ 4.1.3.4. Despliega la paleta de colores de RGB por separado

▼ OpenCV

```
#OpenCV
import cv2
from google.colab.patches import cv2_imshow

print("lena_color_512.tif")
image = cv2.imread("/content/Imagenes/lena_color_512.tif")
(B, G, R) = cv2.split(image)
zeros = np.zeros(image.shape[:2], dtype="uint8")
cv2_imshow(cv2.merge([zeros, zeros, R]))
cv2_imshow(cv2.merge([zeros, G, zeros]))
cv2_imshow(cv2.merge([B, zeros, zeros]))
```

```
print("peppers_color.tif")
image = cv2.imread("/content/Imagenes/peppers_color.tif")
(B, G, R) = cv2.split(image)
zeros = np.zeros(image.shape[:2], dtype="uint8")
cv2_imshow(cv2.merge([zeros, zeros, R]))
cv2_imshow(cv2.merge([zeros, G, zeros]))
cv2_imshow(cv2.merge([B, zeros, zeros]))
```

▼ Scikit-Image

```
#Scikit-Image
from skimage import color
from skimage import io
import numpy as np

img = io.imread("/content/Imagenes/lena_color_512.tif",plugin='pil')
fig, ax1 = plt.subplots(1,1)
io.imshow(img,axes=ax1)
ax1.set_title("lena_color_512.tif")
ax1.axis("off")
io.show()

red_img = img.copy()
red_img[:, :, 1] = 0
red_img[:, :, 2] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(red_img,axes=ax1)
ax1.set_title("red")
ax1.axis("off")
io.show()

green_img = img.copy()
green_img[:, :, 0] = 0
green_img[:, :, 2] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(green_img,axes=ax1)
ax1.set_title("green")
ax1.axis("off")
io.show()

blue_img = img.copy()
blue_img[:, :, 0] = 0
blue_img[:, :, 1] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(blue_img,axes=ax1,cmap="Blues")
ax1.set_title("blue")
ax1.axis("off")
io.show()

img = io.imread("/content/Imagenes/peppers_color.tif",plugin='pil')
```

```

fig, ax1 = plt.subplots(1,1)
io.imshow(img,axes=ax1)
ax1.set_title("peppers_color.tif")
ax1.axis("off")
io.show()

red_img = img.copy()
red_img[:, :, 1] = 0
red_img[:, :, 2] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(red_img,axes=ax1)
ax1.set_title("red")
ax1.axis("off")
io.show()

green_img = img.copy()
green_img[:, :, 0] = 0
green_img[:, :, 2] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(green_img,axes=ax1)
ax1.set_title("green")
ax1.axis("off")
io.show()

blue_img = img.copy()
blue_img[:, :, 0] = 0
blue_img[:, :, 1] = 0
fig, ax1 = plt.subplots(1,1)
io.imshow(blue_img,axes=ax1,cmap="Blues")
ax1.set_title("blue")
ax1.axis("off")
io.show()

```

▼ 4.1.3.5. Despliega la paleta de colores HSV por separado

▼ OpenCV

```

#OpenCV
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

print("lena_color_512.tif")
img = cv2.imread("/content/Imagenes/lena_color_512.tif")

img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
(hue_img, s_img, value_img) = cv2.split(img_hsv)

fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(8, 2))
cv2.imshow(img)

```

```
ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
ax2.imshow(value_img)
ax2.set_title("Value channel")
ax2.axis('off')
ax3.imshow(s_img)
ax3.set_title("S channel")
ax3.axis('off')

fig.tight_layout()
```

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow

print("peppers_color.tif")
img = cv2.imread("/content/Imagenes/peppers_color.tif")

img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
(hue_img, s_img, value_img) = cv2.split(img_hsv)

fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(8, 2))
cv2_imshow(img)

ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
ax2.imshow(value_img, cmap='hsv')
ax2.set_title("Value channel")
ax2.axis('off')
ax3.imshow(s_img, cmap='hsv')
ax3.set_title("S channel")
ax3.axis('off')

fig.tight_layout()
```

▼ Scikit-Image

```
#Scikit-Image
#https://scikit-image.org/docs/dev/auto_examples/color_exposure/plot_rgb_to_hsv.html
import matplotlib.pyplot as plt

from skimage.color import rgb2hsv
from skimage import io

print("/content/Imagenes/lena_color_512.tif")
rgb_img = io.imread("/content/Imagenes/lena_color_512.tif", plugin='pil')
hsv_img = rgb2hsv(rgb_img)
```

```

hue_img = hsv_img[:, :, 0]
s_img = hsv_img[:, :, 1]
value_img = hsv_img[:, :, 2]

fig, (ax0, ax1, ax2, ax3) = plt.subplots(ncols=4, figsize=(10, 2))

ax0.imshow(rgb_img)
ax0.set_title("RGB image")
ax0.axis('off')
ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
ax2.imshow(value_img)
ax2.set_title("Value channel")
ax2.axis('off')
ax3.imshow(s_img, cmap='hsv')
ax3.set_title("Saturation channel")
ax3.axis('off')

fig.tight_layout()

```

```

#Scikit-Image
https://scikit-image.org/docs/dev/auto\_examples/color\_exposure/plot\_rgb\_to\_hsv.html
import matplotlib.pyplot as plt

from skimage.color import rgb2hsv
from skimage import io

print("/content/Imagenes/peppers_color.tif")
rgb_img = io.imread("/content/Imagenes/peppers_color.tif", plugin='pil')
hsv_img = rgb2hsv(rgb_img)
hue_img = hsv_img[:, :, 0]
s_img = hsv_img[:, :, 1]
value_img = hsv_img[:, :, 2]

fig, (ax0, ax1, ax2, ax3) = plt.subplots(ncols=4, figsize=(10, 2))

ax0.imshow(rgb_img)
ax0.set_title("RGB image")
ax0.axis('off')
ax1.imshow(hue_img, cmap='hsv')
ax1.set_title("Hue channel")
ax1.axis('off')
ax2.imshow(value_img)
ax2.set_title("Value channel")
ax2.axis('off')
ax3.imshow(s_img)
ax3.set_title("S channel")
ax3.axis('off')

fig.tight_layout()

```

4.2. De una imagen que usted escoja, dejarla en escala de grises y procure que sea igual en renglones y en columnas. Programe una función que realice decimación de una imagen, reduciéndola a la mitad de su tamaño original. Y promediando en grupos de 4 píxeles.

Pruebe con su imagen.

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from skimage import color

# se lee a blanco y negro
image = cv2.imread("/content/imagenes2/abdomen.png",0)
print(image.shape)

height, width = image.shape
if height != width:
    image = cv2.resize(image, (height,height))
    print("Redimensionado a ",image.shape)

image = np.array(image,dtype="uint8")

def decimacion(img):
    """Función Decimación
    Reduce a la mitad una imagen, promediando sus píxeles en grupos de 4

    Parámetros
    img: ndarray de numpy, representación de una imagen en blanco y negro

    return nuevo ndarray con la imagen reducida
    """
    # Encontramos las dimensiones de la imagen, deben ser las mismas
    height, width = img.shape
    print("Tamaño original",height,width)

    nuevoArreglo = list()
    for renglon in range(0,height-1,2):
        nuevoRenglon = list()
        for columna in range(0,width-1,2):
            suma = int(img[renglon,columna]) + int(img[renglon,columna+1]) \
                + int(img[renglon+1,columna]) + int(img[renglon+1,columna+1])
            nuevoRenglon.append(int(suma/4))
        nuevoArreglo.append(nuevoRenglon)

    return np.array(nuevoArreglo)

# Usando la función
```

```

# Usando la función
nuevo = decimacion(image)
print("Tamaño nuevo",nuevo.shape)

# Imprimiendo las imágenes
print("Nueva")
cv2_imshow(image)
print("Decimación")
cv2_imshow(nuevo)

```

4.3. Usando funciones de Python ya existentes y la imagen que usted escoja.

4.3.1. Reajusta el tamaño de la carpeta imágenes a 10 veces su tamaño original

```

import numpy as np
import cv2

# Se ocupa, Colab tiene deshabilitado el imshow de opencv, crashea las sesiones
from google.colab.patches import cv2_imshow

# para cada imagen
for imagePath in listaImagenes:
    if not (imagePath.endswith(".dcm") or imagePath.endswith(".raw")):
        img = cv2.imread(imagePath)
        print(imagePath)
        height = np.size(img, 0)
        width = np.size(img, 1)
        print("Height = " + str(height) + " Width = " + str(width))
        cv2_imshow(img)
        ##Redimensionamos
        rszImg = cv2.resize(img, (0,0), fx=10, fy=10)
        height = np.size(rszImg, 0)
        width = np.size(rszImg, 1)
        print("Height = " + str(height) + " Width = " + str(width))
        # Guardamos imagen
        #cv2.imwrite("/content/image.png",rszImg)
        #Mostramos imagen
        cv2_imshow(rszImg)

https://www.delftstack.com/es/howto/python/resize-image-python/#:~:text=Para%20cambiar%20el%20
https://pythonexamples.org/python-opencv-cv2-resize-image/

```

4.3.2. Reajusta el tamaño de la carpeta imágenes a 3 veces el tamaño original.

```
import numpy as np
```

```

import cv2

# Se ocupa, Colab tiene deshabilitado el imshow de opencv, crashea las sesiones
from google.colab.patches import cv2_imshow

# para cada imagen
for imagePath in listaImagenes:
    if not (imagePath.endswith(".dcm") or imagePath.endswith(".raw")):
        img = cv2.imread(imagePath,)
        print(imagePath)
        height = np.size(img, 0)
        width = np.size(img, 1)
        print("Height = " + str(height) + " Width = " + str(width))
        cv2_imshow(img)
        ##Redimensionamos
        rszImg = cv2.resize(img, (0,0), fx=3, fy=3)
        height = np.size(rszImg, 0)
        width = np.size(rszImg, 1)
        print("Height = " + str(height) + " Width = " + str(width))
        # Guardamos imagen
        #cv2.imwrite("/content/image.png",rszImg)
        #Mostramos imagen
        cv2_imshow(rszImg)

```

▼ 4.3.3.Rote la imagen a 45, 90 y 180 grados, guárdelas en formato png.

```

#https://omes-va.com/trasladar-rotar-escalar-recortar-una-imagen-opencv/
#Rotar en spyder, solo falata el guardado
import cv2
import numpy as np

image = cv2.imread('/content/imagenes2/retinaRGB.jpg')
ancho = image.shape[1] #columnas
alto = image.shape[0] # filas

# Matrices de Rotación
M45 = cv2.getRotationMatrix2D((ancho//2,alto//2),45,1)# 45 grados
M90 = cv2.getRotationMatrix2D((ancho//2,alto//2),90,1)# 90 grados
M180 = cv2.getRotationMatrix2D((ancho//2,alto//2),180,1)# 180 grados

# Transformación a la imagen
image45 = cv2.warpAffine(image,M45,(ancho,alto))
image90 = cv2.warpAffine(image,M90,(ancho,alto))
image180 = cv2.warpAffine(image,M180,(ancho,alto))

# impresion
print("Original")
cv2_imshow(image)
print("45 grados")
cv2_imshow(image45)

```

```

print("90 grados")
cv2_imshow(image90)
print("180 grados")
cv2_imshow(image180)

# guardar imagenes
cv2.imwrite("/content/image45.png",image45)
cv2.imwrite("/content/image90.png",image90)
cv2.imwrite("/content/image180.png",image180)

```

▼ **4.4.**Convierte la imagen peppers_color.tif a escala de grises,

4.4.1.Recortela de manera que solo quede uno de los pimientos verdes en ese recorte

4.4.2.Guárdela en formato .jpg.

```

#https://omes-va.com/trasladar-rotar-escalar-recortar-una-imagen-opencv/
import cv2
from google.colab.patches import cv2_imshow

# se lee a blanco y negro
image = cv2.imread("/content/Imagenes/peppers_color.tif",0)

#Recortar una imagen
imageOut = image[180:500,183:420]#Columnas #Filas
print("Original")
cv2_imshow(image)
print("Recortada")
cv2_imshow(imageOut)

# guardando
cv2.imwrite("/content/pimientoBlancoNegro.jpg",imageOut)

```

4.5.Un formato de imágenes sin ningún tipo de codificación se conoce como formato crudo (RAW). De la imagen “rosa800x600.raw” lea y despliegue la imagen. Tome en cuenta que esta imagen maneja la precisión de integer8 y el tamaño es de 600x800 pixeles.

```

# para formato raw
import matplotlib.pyplot as plt
import numpy as np

```

```
# tamaño de la imagen
rows = 800
columns = 600

# leer el archivo
file = open("/content/Imagenes/rosa800x600.raw")

# convirtiendo a un arreglo numpy
img = np.fromfile(file, dtype = np.uint8, count = rows * columns)
print("shape original:",img.shape)
print("size original:",img.size)
# cambiamos las dimensiones de la imagen
img.shape = (img.size // columns, columns)
print("new shape:",img.shape)
print("new size:",img.size)

# Mostrando la imagen
fig, ax1 = plt.subplots(1,1)
ax1.imshow(img, cmap = "gray")
Texto = """Tipo de imagen: {0}
Tamaño de imagen: {1} Bytes
Tipo de dato: {2}
Archivo: {3}""".format("raw",
                      os.stat("/content/Imagenes/rosa800x600.raw").st_size,
                      img.dtype, "rosa800x600.raw")
ax1.set_title(Texto)
ax1.axis("off")
```