

Práctica 1.

Manejo Básico de Imágenes.

31508811-1 Martiñón Luna Jonathan José.

31506842-1 Ortega Ibarra Jaime Jesus.

41800471-9 Tapia López José de Jesús.

I. INTRODUCCIÓN.

COMO parte de la materia "Reconocimiento de Patrones" se llevan a cabo diversas prácticas, las cuales tienen como principal objetivo que el alumno pueda comprender de mejor manera los conceptos aprendidos en clase. En la presente práctica se llevará a cabo con librerías desarrolladas en *Python* la manipulación de diversas imágenes seleccionadas con anterioridad para que fuesen óptimas en el transcurso del presente trabajo. Asimismo, se llevó a cabo una introducción con ejercicios básicos que pudiesen otorgar al alumno las bases para el desarrollo de prácticas posteriores.

Esta práctica pretende mostrar imágenes en diferentes formatos, con el objetivo de que a futuro podamos procesarlas y extraer información. En el ámbito que actualmente desempeñamos, las imágenes juegan un papel muy importante a través de: transferencia de conocimientos avanzados en álgebra lineal, estadísticas, probabilidad, procesamiento de señales, geometría, algoritmos, aprendizaje automático, aprendizaje, de máquina, ciencia de datos y desarrollo de software. La imagen puede mostrar cualquier situación con mayor precisión. El diagnóstico médico se realiza con demasiada precisión para el estudio y tratamiento de muchas enfermedades.

Por lo tanto, en esta práctica se pretenden mostrar algunas manipulaciones básicas con las imágenes.

A. Ambiente

La práctica, al igual que los ejercicios básicos fueron desarrollados en "*Jupyter Notebook*", haciendo uso de Python 3.

II. EJERCICIOS BÁSICOS

A. Abrir y escribir una imagen a un archivo

```
1 from scipy import misc
2 import imageio as io
3
4 f = misc.face()
5 io.imwrite('face.png', f)
6
7 import matplotlib.pyplot as plt
8 plt.imshow(f)
9 plt.show()
```



B. Creación de un arreglo numpy de un archivo de imagen

```
1 from scipy import misc
2 face = misc.face()
3 io.imwrite('face.png', face) # Se guarda la imagen
4     como png
5 face = io.imread('face.png') # Se lee la imagen png
6 type(face)
7 #Almacenamos el shape de la imagen en color para el
8 #ejercicio 4
9 Color_Shape = face.shape
10 face.shape, face.dtype
11 ((768, 1024, 3), dtype('uint8'))
12 print(type(face))
13 <class 'imageio.core.util.Array'>
14 type(face)
15 imageio.core.util.Array
```

Notamos que `dtype` es `uint8` para imágenes de 8 bits (0-255 niveles de gris).

¿De qué tipo es la variable `face`?

La variable es de tipo `imageio.core.util.Array` (de esa clase).

¿Qué resultado arroja `face.shape`?

La dimensión de la imagen; la primera entrada indica que la imagen tiene una altura de 768 píxeles, la segunda entrada indica que tiene una anchura de 1024 píxeles y la tercera entrada indica los 3 canales de la imagen. (La imagen puede contener un solo canal, esto es con aquellas que son a escala de grises o blanco y negro, así mismo múltiples canales, tal es el caso de las imágenes a color).

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

El resultado sería (768, 1024); indicando solamente la altura y anchura en pixeles.

C. Abrir archivos raw

```
1 import numpy as np
3 face.tofile('face.raw') # Se crea el archivo raw
  binario
face_from_raw = np.fromfile('face.raw', dtype=np.
  uint8)
5 face_from_raw.shape
```

¿Qué resultado arroja la primera instrucción `face_from_raw.shape`?

(786432,); lo cual indica que el archivo raw binario es de una dimensión (número de renglones).

Note que al leer archivos raw, es necesario conocer la forma de los datos, especificada con la instrucción `face_from_raw.shape = (...)`, así como el tipo de la imagen, especificado con `dtype = np.uint8`. Para datos grandes, use `np.memmap`:

```
1 face_memmap = np.memmap('face.raw', dtype=np.uint8,
  — shape=(768, 1024, 3))
```

Con `memmap`, los datos se leen del archivo y no se cargan en la memoria.

D. Despliegue de imágenes

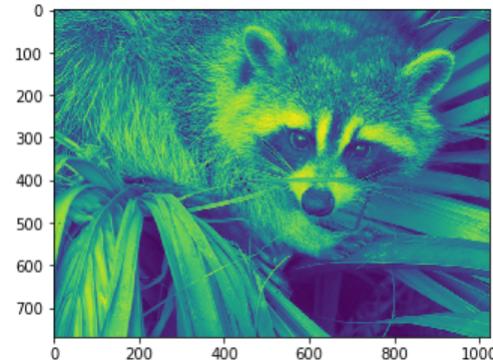
Como se vio anteriormente, importamos `"misc"`, el cual es elemento de la librería `"scipy"` para posteriormente importar la imagen `"face"`, quien finalmente mostramos por medio de `"matplotlib"`.

```
2 from scipy import misc
f = misc.face(gray=True)
import matplotlib.pyplot as plt
4 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`?

`cm.gray` nos indicará que se tomará la escala de grises, sin embargo, al no hacer uso de dicha instrucción, plt utilizará por defecto la escala de colores `viridis`, dando un resultado diferente al esperado, tal como se muestra en la imagen de abajo.



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

Como se mencionó anteriormente, el mapa de color por defecto de `"imshow"` es **"Viridis"**.

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

Con anterioridad se guardaron las dimensiones de la imagen a color para poder hacer un comparativo con la imagen en blanco y negro, posteriormente se procedió a comparar obteniendo las dimensiones de nuestra imagen en blanco y negro. Lo que nos devuelve los siguientes valores:

Color: (768, 1024, 3)

Grises: (768, 1024)

```
Gray_Shape = f.shape
print("Color:", Color_Shape, "\nGrises:", Gray_Shape)
```

Lo que podemos interpretar como que el color es una matriz con 3 dimensiones, las cuales corresponden a Green, Red, Blue (RGB), a diferencia de la imagen en blanco y negro, la cual sólamente necesita una dimensión.

Se puede incrementar el contraste especificando los valores mínimo y máximo en el despliegue:

```
1 plt.imshow(f, cmap=plt.cm.gray, vmin=30, vmax=200)
3 plt.axis('off')
```



Se pueden dibujar líneas de contorno con la instrucción `contour`.

```
1 plt.imshow(f, cmap=plt.cm.gray, vmin=30, vmax=200)
plt.axis('off')
3 plt.contour(f, [50, 200])
```



Investigue y ponga una breve descripción sobre la instrucción `contour`.

De acuerdo con:

<https://numython.github.io/posts/2016/02/graficas-de-contorno-en-matplotlib/>

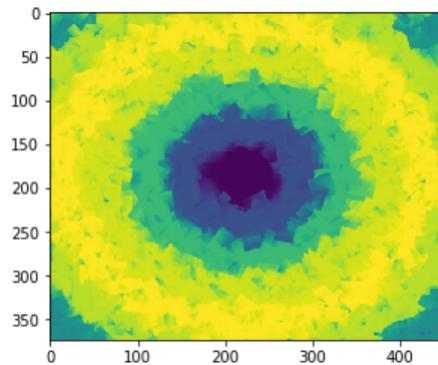
Sirve para marcar nuestras curvas de nivel, ya sea en 2D o en 3D, básicamente el comando es el siguiente:

`contour(X, Y, Z, 20)`

Donde cómo podemos observar, indicamos nuestras coordenadas en X, Y y Z, el número 20 será la cantidad de niveles a trabajar.

Obtenga y despliegue los contornos de la imagen `contour_gray.png` con las siguientes instrucciones (antes debe cargar la imagen `contour_gray.png` en la variable `f`):
`plt.contour(f, 5)`
`plt.contour(f, [50, 100, 200])`

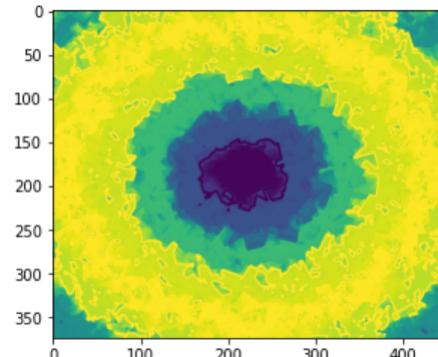
```
1 f = io.imread('contour_gray.png')
plt.imshow(f)
```



Como podemos observar la imagen que se nos fue otorgada, a pesar de ser en apariencia en blanco negro, al ser desplegada se muestra automáticamente como a color, al no pedirse que se mapeara a blanco y negro, se dejó así.

Una vez que cargamos la imagen, procedemos a desplegar los contornos.

```
2 plt.imshow(f)
plt.contour(f, 5)
plt.contour(f, [50, 100, 200])
```



E. Manipulaciones básicas

Las imágenes son arreglos, por lo que podemos usar la maquinaria de `numpy`.

En este caso, podemos obtener el valor específico de un pixel, mediante las siguientes líneas de código:

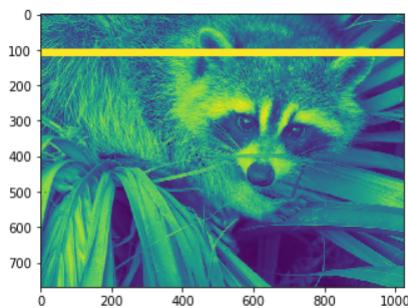
```
1 face = misc.face(gray=True)
face[0, 40]
```

127

¿Cuanto Vale el pixel `Face[0, 40]`? Tiene un valor de: 127

Ahora accesaremos a diversas secciones de una imagen, modificándola de la siguiente manera:

```
1 # Accesando secciones de la imagen
face[10:13, 20:23]
3 face[100:120] = 255
plt.imshow(face)
```



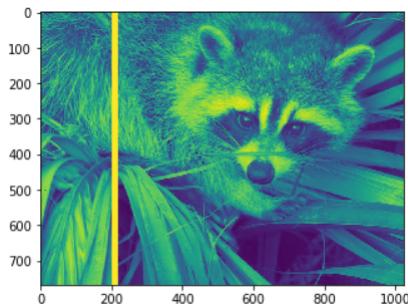
¿Qué efecto tiene la instrucción `face[100 : 120] = 255` en la imagen de abajo?

Entre 100 y 120 genera una franja horizontal de color amarillo

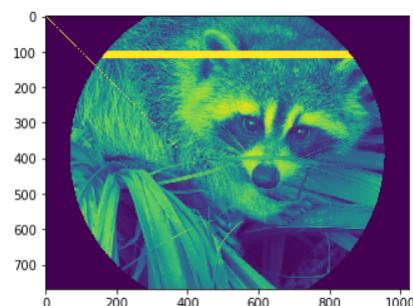
Pinte una franja vertical gris en la imagen que vaya de la columna 200 a la columna 220.

Tip : en los índices tiene que elegir todas las filas con `:` y luego indicar las columnas deseadas.

```
2 face[10:13, 20:23]
face[:,200:220] = 255
plt.imshow(face)
```



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



¿Cuánto vale `lx, ly` ?

`lx = 768`

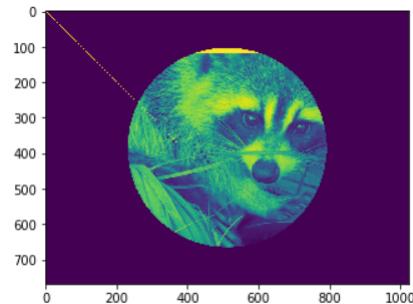
`ly = 1024`

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

Toma una pequeña linea de la imagen y la pinta de un tono naranja, desde la parte superior izquierda de la imagen, hasta el centro.

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

```
2 face2 = face
lx, ly = face2.shape
3 X, Y = np.ogrid[0:lx, 0:ly]
mask = (X - lx / 2) ** 2 + (Y - ly / 2) ** 2 > lx *
    ly / 10
# Masks
5 face2[mask] = 0
# Indexado con rangos
6 face[range(400), range(400)] = 255
7 plt.imshow(face2)
```



Información Estadística

Se puede obtener información estadística de la imagen usando el módulo misc.

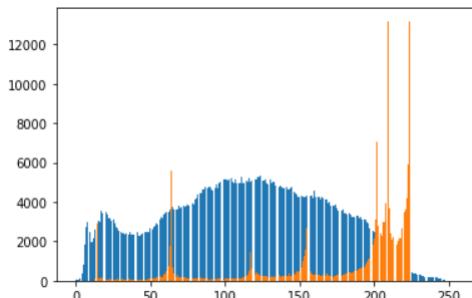
```
1 face = misc.face(gray=True)
print(face.mean())
3 face.max(), face.min()
```

113.48026784261067
(250, 0)

Una manera de obtener el histograma es con la instrucción `histogram` de numpy.

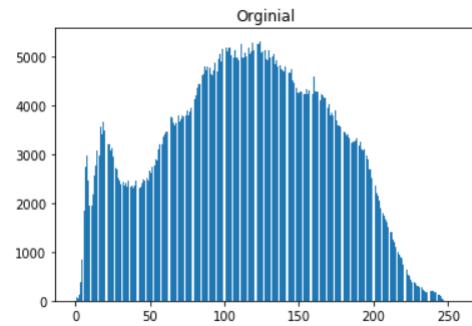
Despliegue el histograma de la imagen en grises del mapache original (es decir, antes de poner las franjas y de aplicar la máscara) usando:

```
1 hist, bins = np.histogram(face, bins=256, range
    =(0,256))
2 hist2, bins2 = np.histogram(f, bins=256, range
    =(0,256))
3 plt.bar(bins[0:-1], hist)
4 plt.bar(bins2[0:-1], hist2)
5 plt.show()
```



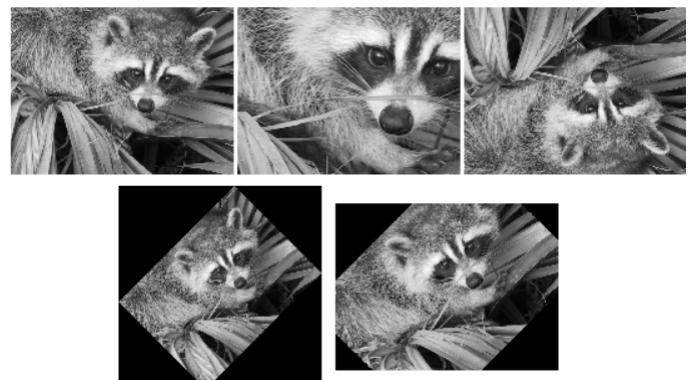
```
2 hist, bins = np.histogram(face, bins=256, range
   =(0,256))
plt.bar(bins[0:-1], hist)
4 plt.title('Orginial')
```

```
14 plt.axis('off')
15 plt.subplot(152)
16 plt.imshow(crop_face, cmap=plt.cm.gray)
17 plt.axis('off')
18 plt.subplot(153)
19 plt.imshow(flip_ud_face, cmap=plt.cm.gray)
20 plt.axis('off')
21 plt.subplot(154)
22 plt.imshow(rotate_face, cmap=plt.cm.gray)
23 plt.axis('off')
24 plt.subplot(155)
25 plt.imshow(rotate_face_noreshape, cmap=plt.cm.gray)
26 plt.axis('off')
27 plt.subplots_adjust(wspace=0.02, hspace=0.3, top=1,
   bottom=0.1, left=0,
   right=1)
28
30 plt.show()
```



Modifique el código para que sólo se tengan 64 bins en el histograma y muestre el histograma resultante.

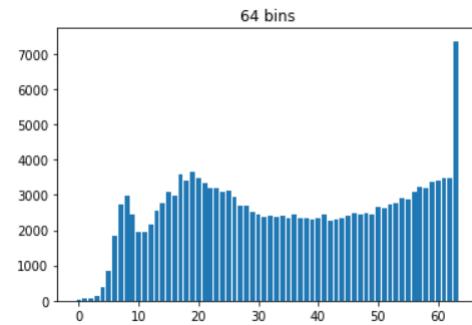
```
1 hist, bins = np.histogram(face, bins=64, range
   =(0,64))
2 plt.bar(bins[0:-1], hist)
3 plt.title('64 bins')
```



En la figura anterior se muestran: la imagen original del mapache, el recorte, la imagen volteada verticalmente, la imagen rotada y la imagen rotada conservando la forma original.

Investigue el operador // y ponga una breve descripción.
EL operador // Realiza una división, devolviendo como resultado un número entero, es decir $3//2 = 1$.

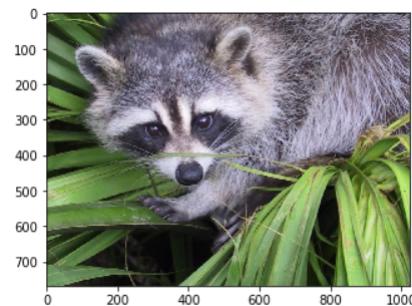
¿Qué efecto tiene el signo negativo en $cropface = face[lx // 4 : -lx // 4, ly // 4 : -ly // 4]$?
Corta la Imagen sin deformarla.



Transformaciones geométricas.

```
1 face = misc.face(gray=True)
2 lx, ly = face.shape
# Recorte
4 crop_face = face[lx // 4: -lx // 4, ly // 4: -ly // 4]
# up <-> down Voltear verticalmente
6 flip_ud_face = np.flipud(face)
# Rotaci n
8 rotate_face = ndimage.rotate(face, 45)
rotate_face_noreshape = ndimage.rotate(face, 45,
   reshape=False)
10 plt.figure(figsize=(12.5, 2.5))
11 plt.subplot(151)
12 plt.imshow(face, cmap=plt.cm.gray)
```

```
1 #nicamente cambiamos flipud por fliplr
2 flip_lr_face = np.fliplr(f)
3 plt.imshow(flip_lr_face)
```

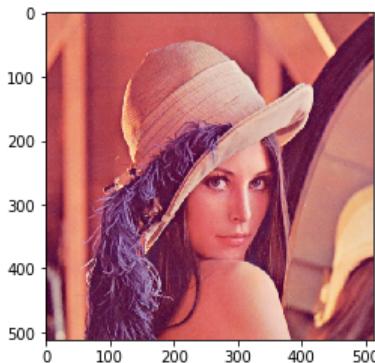


III. PRÁCTICA 1

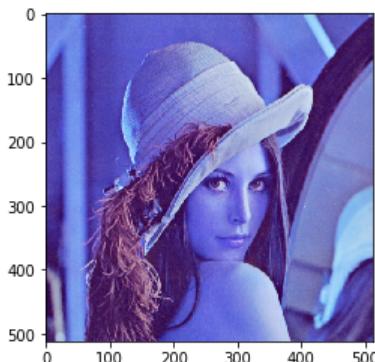
A. 4.1. De la carpeta de imágenes: realiza las siguientes actividades.

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

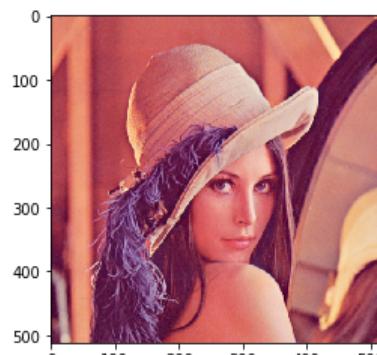
```
# importamos las bibliotecas que utilizaremos en la práctica
1
2 import matplotlib.image as mpimg
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import cv2
6 from skimage import data
7 from skimage import color
8 from skimage import img_as_float
9 from skimage import io
10 from PIL import Image
11 from scipy import misc
12 from pydicom import dcmread
13
14 # Leemos la imagen de lena con matplotlib.image (que se encuentra en la carpeta de la práctica)
15 lena_image = mpimg.imread('lena_color_512.tif')
# Desplegamos la imagen
16 lena_image_plot = plt.imshow(lena_image)
```



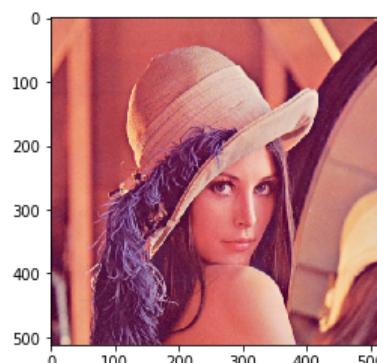
```
# Leemos la imagen de lena con OpenCV (que se encuentra en la carpeta de la práctica)
1 lena_opencv = cv2.imread('lena_color_512.tif')
# Desplegamos la imagen
2 lena_opencv_plot = plt.imshow(lena_opencv)
```



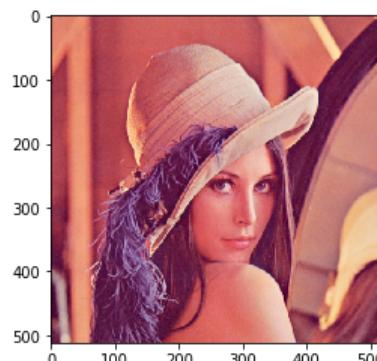
```
# Leemos la imagen de lena con scikit-image (que se encuentra en la carpeta de la práctica)
1 lena_skimage = io.imread('lena_color_512.tif')
# Desplegamos la imagen
2 io.imshow(lena_skimage)
io.show()
```



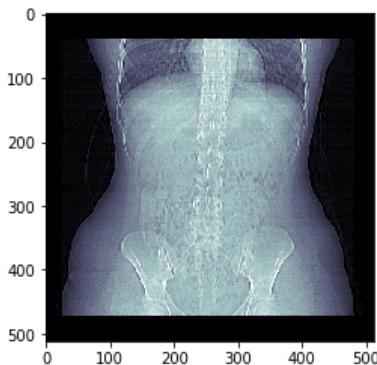
```
# Abrimos la imagen con PIL para desplegarla en una nueva ventana
1 lena_pil = Image.open('lena_color_512.tif')
2 lena_pil.show()
```



```
# Leemos y desplegamos la imagen de Lena con scipy:
1 lena_scipy = misc.imread('lena_color_512.tif')
2 lena_scipy_plot = plt.imshow(lena_scipy)
```



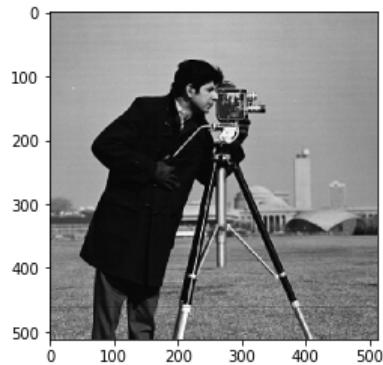
```
# Para el caso de imágenes DICOM:
1
2 #pip install pydicom
3 %matplotlib inline
4
5 Anonymized_dcm = dcmread('Anonymized20200210.dcm') #
6 Lee y carga la imagen en formato DICOM
7 Anonymized_dicom = Anonymized_dcm.pixel_array # Lo convierte a un arreglo de pixeles
8
9 # Graficamos la imagen usando matplotlib
10 plt.axis('on') #Habilita los ejes de escala y etiquetas
11 plt.imshow(Anonymized_dicom, cmap = plt.cm.bone)
12 <matplotlib.image.AxesImage at 0x7f57effe8198>
```



```

1 IM_0001_0007_dcm = dcmread('IM-0001-0007.dcm') # Lee
   y carga la imagen en formato DICOM
2 IM_0001_0007_dicom = IM_0001_0007_dcm.pixel_array #
   Lo convierte a un arreglo de pixeles
3 # Graficamos la imagen usando matplotlib
4 plt.axis('on') # Habilita los ejes de escala y
   etiquetas
5 plt.imshow(IM_0001_0007_dicom, cmap = plt.cm.bone) #
   Desplegamos la imagen

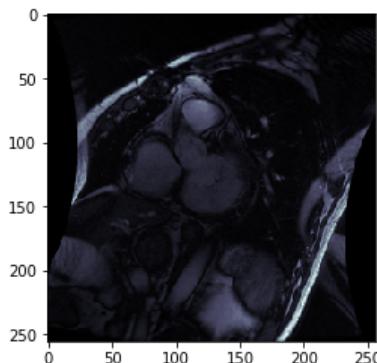
```



```

1 # Leemos y desplegamos la imagen 'cameraman' con
   scikit-image:
2 cameraman_skimage = io.imread('cameraman.tif')
3 io.imshow(cameraman_skimage)
4 io.show()

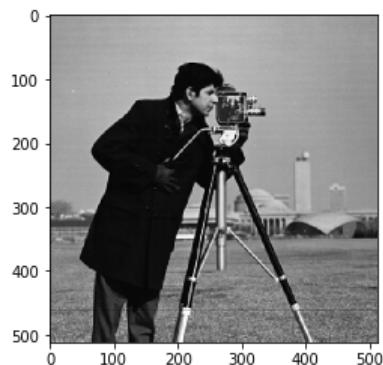
```



```

1 # Leemos y desplegamos la imagen 'cameraman' con
   matplotlib.image:
2 cameraman_image = mpimg.imread('cameraman.tif')
3 cameraman_image_plot = plt.imshow(cameraman_image)
4

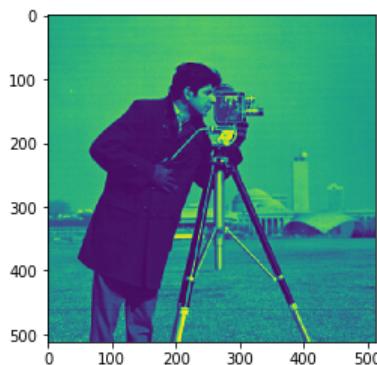
```



```

1 # Leemos y desplegamos en una nueva ventana la
   imagen 'cameraman' con PIL:
2 # Abrimos la imagen para desplegarla en una nueva
   ventana
3 cameraman_pil = Image.open('cameraman.tif')
4 cameraman_pil.show()

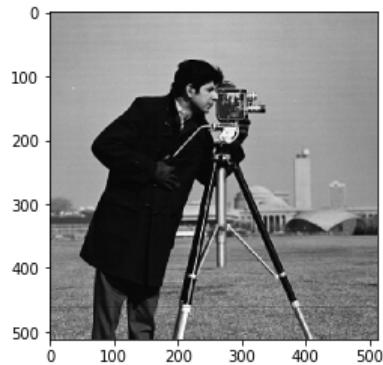
```



```

1 # Leemos y desplegamos la imagen 'cameraman' con
   OpenCV:
2 cameraman_opencv = cv2.imread('cameraman.tif')
3 cameraman_opencv_plot = plt.imshow(cameraman_opencv)
4

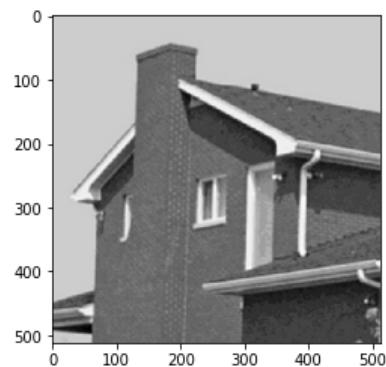
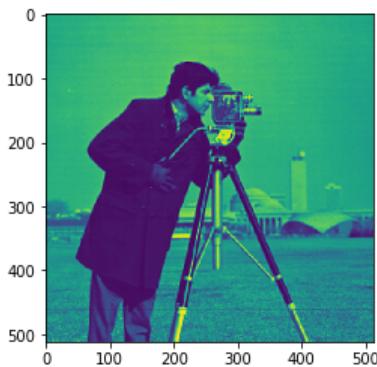
```



```

1 # Leemos y desplegamos la imagen 'cameraman' con
   scipy:
2 cameraman_scipy = misc.imread('cameraman.tif')
3 cameraman_scipy_plot = plt.imshow(cameraman_scipy)
4

```



```

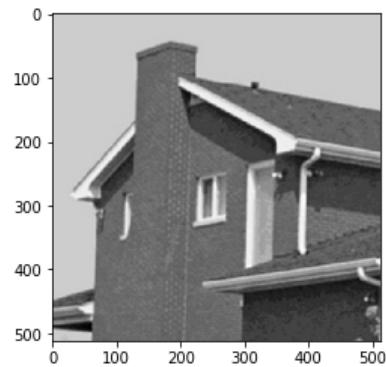
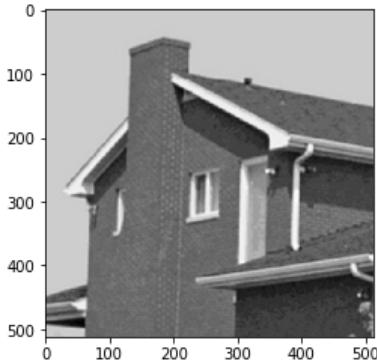
1 # Leemos y desplegamos la imagen 'house' con
   matplotlib.image
2
3 house_image = mpimg.imread('house.tif')
house_image_plot = plt.imshow(house_image)

```

```

# Leemos y desplegamos en una nueva ventana la
  imagen 'house', con PIL:
2
# Abrimos la imagen para desplegarla en una nueva
  ventana
3 house_pil = Image.open('house.tif')
house_pil.show()

```

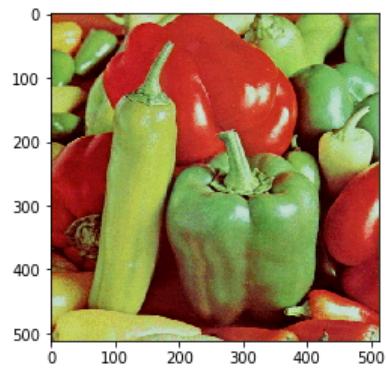
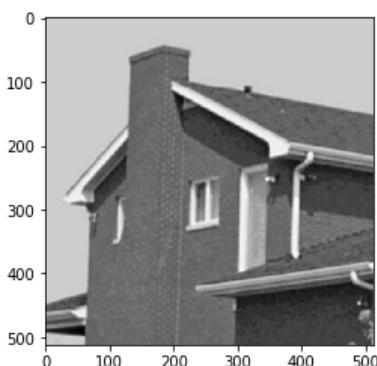


```

1 # Leemos y desplegamos la imagen 'house' con OpenCV:
2
3 house_opencv = cv2.imread('house.tif')
house_opencv_plot = plt.imshow(house_opencv)

```

Leer y desplegar la imagen 'house' con scipy NO SE PUEDE.



```

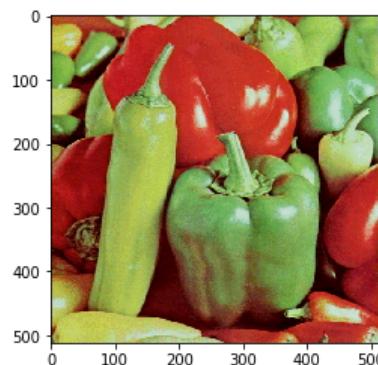
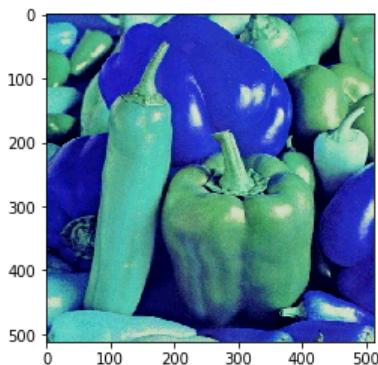
1 # Leemos y desplegamos la imagen 'house' con scikit-
   image:
2
3 #house_skimage = io.imread('house.tif')
#io.imshow(house_skimage)
5 #io.show()
7
7 # img_as_float convierte una imagen a formato de
   coma flotante, con valores en [0, 1]
9
9 house_skimage = img_as_float(house_image)
house_skimage_plot = plt.imshow(house_skimage)

```

```

# Leemos y desplegamos la imagen de peppers_color
  con OpenCV:
2
peppers_color_opencv = cv2.imread('peppers_color.tif')
4
peppers_color_opencv_plot = plt.imshow(
  peppers_color_opencv)

```



```

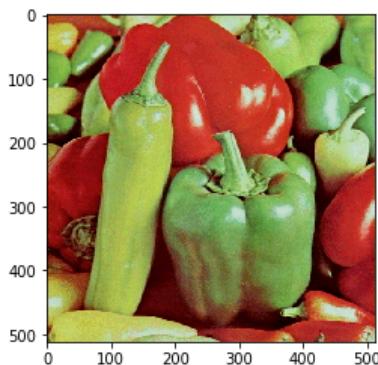
1 # Leemos y desplegamos la imagen de peppers_color
2   con scikit-image:
3
4 peppers_color_skimage = img_as_float(
5     peppers_color_image)
6 peppers_color_skimage_plot = plt.imshow(
7     peppers_color_skimage)

```

```

1 # Para el caso de la imagen RAW
2
3 rosa = np.fromfile('rosa800x600.raw', dtype=np.uint8)
4 rosa.shape = (800,600)
5 #Dibujando tal cual
6 plt.imshow(rosa)

```



```

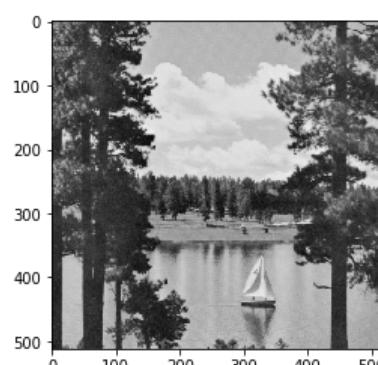
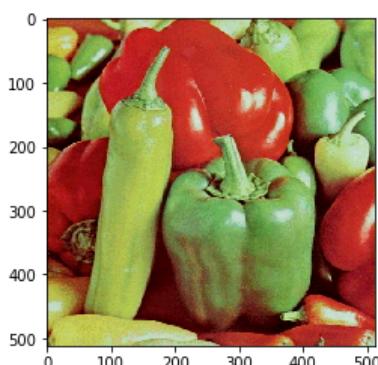
1 # Leemos y desplegamos en una nueva ventana la
2   imagen de peppers_color con PIL:
3
4 # Abrimos la imagen para desplegarla en una nueva
5   ventana
6 peppers_color_pil = Image.open('peppers_color.tif')
7 peppers_color_pil.show()

```

```

1 # Leemos y desplegamos la imagen 'lake' con
2   matplotlib.image:
3
4 lake_image = mpimg.imread('lake.tif')
5 lake_image_plot = plt.imshow(lake_image)

```



```

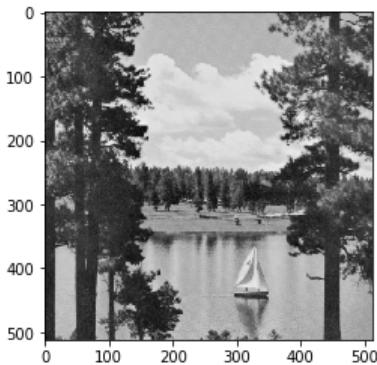
1 # Leemos y desplegamos la imagen de peppers_color
2   con scipy:
3
4 peppers_color_scipy = misc.imread('peppers_color.tif')
5 peppers_color_scipy_plot = plt.imshow(
6     peppers_color_scipy)

```

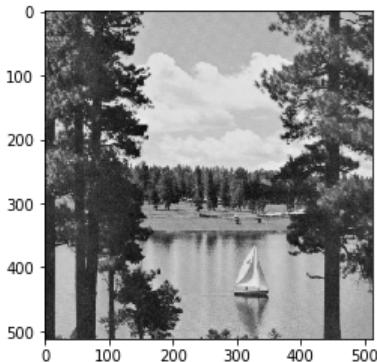
```

1 # Leemos y desplegamos la imagen 'lake' con OpenCV:
2
3 lake_opencv = cv2.imread('lake.tif')
4 lake_opencv_plot = plt.imshow(lake_opencv)

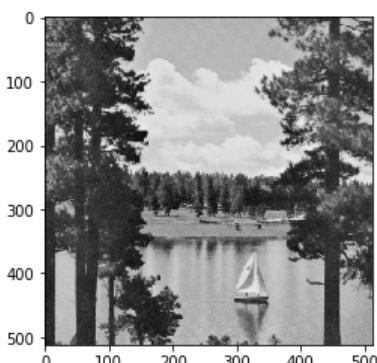
```



```
1 # Leemos y desplegamos la imagen 'lake' con scikit-image:
2
3 lake_skimage = img_as_float(lake_image)
4 lake_skimage_plot = plt.imshow(lake_skimage)
```



```
1 # Leemos y desplegamos en una nueva ventana la
2 # imagen 'lake' con PIL:
3
4 # Abrimos la imagen para desplegarla en una nueva
5 # ventana
6 lake_pil = Image.open('lake.tif')
7 lake_pil.show()
```



Leer y desplegar la imagen 'lake' con scipy NO SE PUEDE.

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

```
1 print("De la imagen lena, el tipo de imagen es: {0}
      , su tamaño es: {1} y su tipo de dato es: {2}")
      .format(type(lena_image),lena_image.shape,
              lena_image.dtype))
2
3 print("De la imagen cameraman, el tipo de imagen es:
      {0} , su tamaño es: {1} y su tipo de dato es:
      {2}")
```

```
5 .format(type(cameraman_image),cameraman_image.
      shape,cameraman_image.dtype))
6 print("De la imagen Anonymized, el tipo de imagen es
      : {0} , su tamaño es: {1} y su tipo de dato es:
      {2}")
      .format(type(Anonymized_dicom),
              Anonymized_dicom.shape,Anonymized_dicom.dtype))
7 print("De la imagen house, el tipo de imagen es: {0}
      , su tamaño es: {1} y su tipo de dato es: {2}")
      .format(type(house_image),house_image.shape,
              house_image.dtype))
8 print("De la imagen IM_0001_0007, el tipo de imagen
      es: {0} , su tamaño es: {1} y su tipo de dato
      es: {2}")
      .format(type(IM_0001_0007_dicom),
              IM_0001_0007_dicom.shape,IM_0001_0007_dicom.
              dtype))
9 print("De la imagen peppers_color, el tipo de imagen
      es: {0} , su tamaño es: {1} y su tipo de dato
      es: {2}")
      .format(type(peppers_color_image),
              peppers_color_image.shape,peppers_color_image.
              dtype))
10 print("De la imagen rosa, el tipo de imagen es: {0}
      , su tamaño es: {1} y su tipo de dato es: {2}")
      .format(type(rosa),rosa.shape,rosa.dtype))
11 print("De la imagen lake, el tipo de imagen es: {0}
      , su tamaño es: {1} y su tipo de dato es: {2}")
      .format(type(lake_image),lake_image.shape,
              lake_image.dtype))
```

De la imagen lena, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512, 3) y su tipo de dato es: uint8

De la imagen cameraman, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512) y su tipo de dato es: uint8

De la imagen Anonymized, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512) y su tipo de dato es: uint8

De la imagen house, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512, 4) y su tipo de dato es: uint8

De la imagen IM_0001_0007, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (256, 256) y su tipo de dato es: int16

De la imagen peppers_color, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512, 4) y su tipo de dato es: uint8

De la imagen rosa, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (800, 600) y su tipo de dato es: uint8

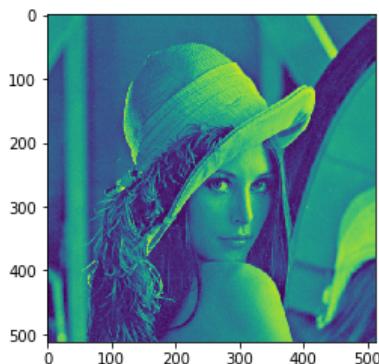
De la imagen lake, el tipo de imagen es: <class 'numpy.ndarray'>, su tamaño es: (512, 512, 4) y su tipo de dato es: uint8

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:

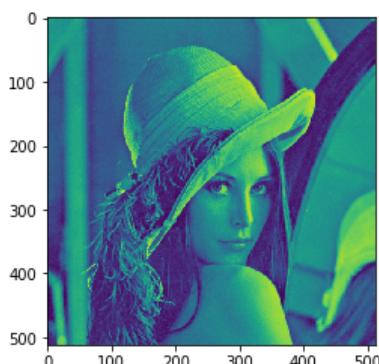
4.1.3.1. RGB a Escala de grises

```
# Para el caso de la imagen lena_color_512.tif
# usando OpenCV:
2
3 # Convertir RGB a escala de grises
4 lena_gray = cv2.cvtColor(lena_image, cv2.
                           COLOR_BGR2GRAY)
```

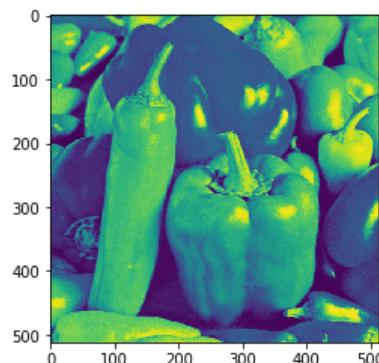
```
1 # Desplegamos la imagen
2 lena_gray_plot = plt.imshow(lena_gray)
```



```
1 # Para el caso de la imagen lena_color_512.tif
2 usando scikit-image:
3
4 lena_grayscale_image = img_as_float(lena_image)
5 # Convertimos la imagen rgb a escala de grises
6 lena_skimage_gray = color.rgb2grey(
    lena_grayscale_image)
7 # Desplegamos la imagen
8 lena_skimage_gray_plot = plt.imshow(
    lena_skimage_gray)
```

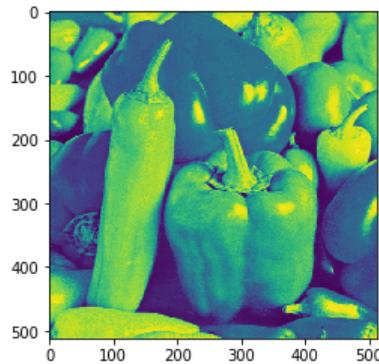


```
1 # Para el caso de la imagen lena_color_512.tif
2 usando OpenCV:
3
4 peppers_color_gray = cv2.cvtColor(
    peppers_color_image, cv2.COLOR_BGR2GRAY)
5 peppers_color_gray_plot = plt.imshow(
    peppers_color_gray)
```



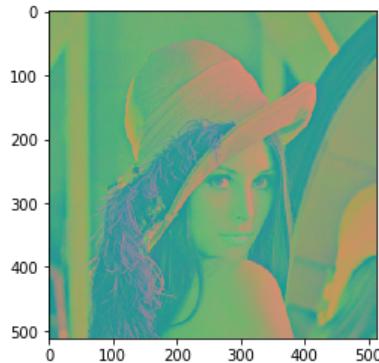
```
1 # Para el caso de la imagen lena_color_512.tif
2 usando scikit-image:
```

```
3 peppers_color_grayscale_image = img_as_float(
    peppers_color_image)
4 peppers_color_skimage_gray = color.rgb2grey(
    peppers_color_grayscale_image)
5 peppers_color_skimage_gray_plot = plt.imshow(
    peppers_color_skimage_gray)
```

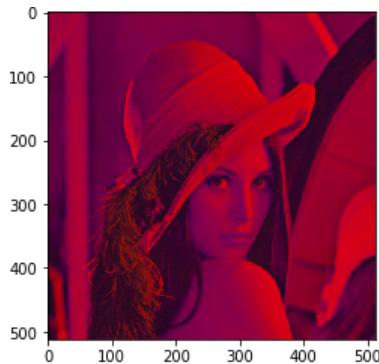


4.1.3.2. RGB a YUV

```
1 # Para el caso de la imagen lena_color_512.tif
2 usando OpenCV:
3
4 lena_yuv = cv2.cvtColor(lena_image, cv2.
    COLOR_BGR2YUV)
5 # Desplegamos la imagen
6 lena_yuv_plot = plt.imshow(lena_yuv)
```



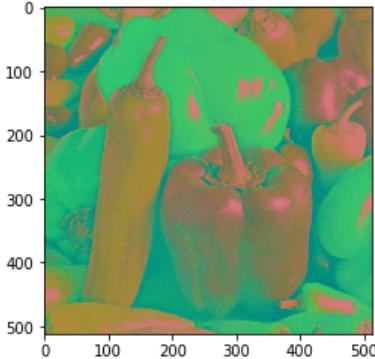
```
1 # Para el caso de la imagen lena_color_512.tif
2 usando Scikit-Image:
3
4 lena_yuv_image = img_as_float(lena_image)
5 # Convertimos de rgb a yuv
6 lena_skimage_yuv = color.rgb2yuv(lena_yuv_image)
7 # Desplegamos la imagen
8 lena_skimage_yuv_plot = plt.imshow(lena_skimage_yuv)
```



```

1 # Para el caso de la imagen peppers_color.tif usando
  OpenCV:
2
3 peppers_color_yuv = cv2.cvtColor(peppers_color_image
  , cv2.COLOR_BGR2YUV)
peppers_color_yuv_plot = plt.imshow(
  peppers_color_yuv)

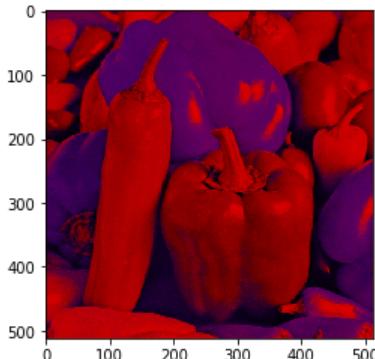
```



```

1 # Para el caso de la imagen peppers_color.tif usando
  Scikit-Image:
2
3 from skimage.io import imread
4 pc = imread("peppers_color.tif")
pc_yuv = img_as_float(pc)
6 pc_skimage_yuv = color.rgb2yuv(pc_yuv)
pc_skimage_yuv_plot = plt.imshow(pc_skimage_yuv)

```

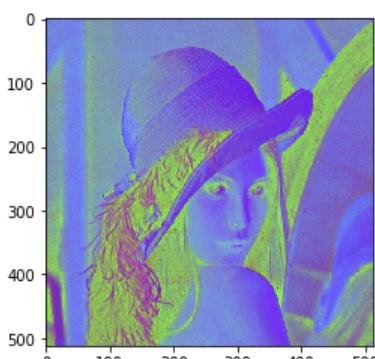


4.1.3.3. RGB a HSV

```

1 # Para el caso de la imagen lena_color_512.tif
  usando OpenCV:
3
4 lena_hsv = cv2.cvtColor(lena_image, cv2.
  COLOR_BGR2HSV)
lena_hsv_plot = plt.imshow(lena_hsv)

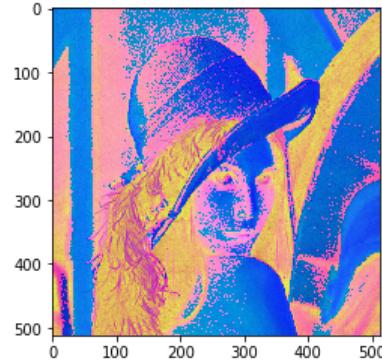
```



```

2 # Para el caso de la imagen lena_color_512.tif
  usando scikit-image:
4
5 lena_hsv_image = img_as_float(lena_image)
lena_skimage_hsv = color.rgb2hsv(lena_hsv_image)
lena_skimage_hsv_plot = plt.imshow(lena_skimage_hsv)

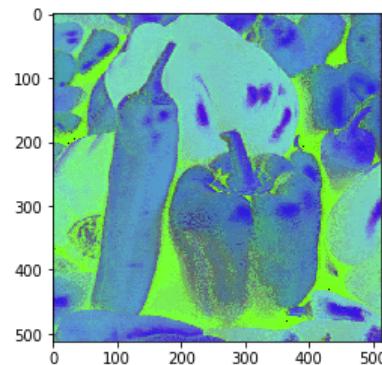
```



```

1 # Para el caso de la imagen peppers_color.tif usando
  OpenCV:
3
4 peppers_color_hsv = cv2.cvtColor(peppers_color_image
  , cv2.COLOR_BGR2HSV)
peppers_color_hsv_plot = plt.imshow(
  peppers_color_hsv)

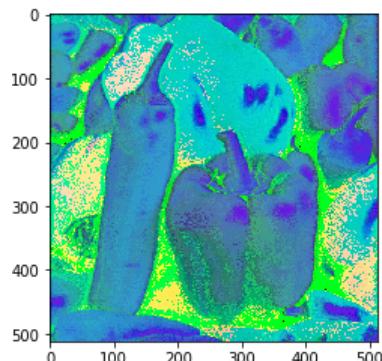
```



```

2 # Para el caso de la imagen peppers_color.tif usando
  scikit-image:
4
5 pc_hsv = img_as_float(pc)
pc_skimage_hsv = color.rgb2hsv(pc_hsv)
pc_skimage_hsv_plot = plt.imshow(pc_skimage_hsv)

```



4.1.3.4. Despliega la paleta de colores de RGB por separado

```

1 import skimage.io as io
import matplotlib.pyplot as plt

```

```

3 # Leemos la imagen
5 lena = io.imread('lena_color_512.tif')

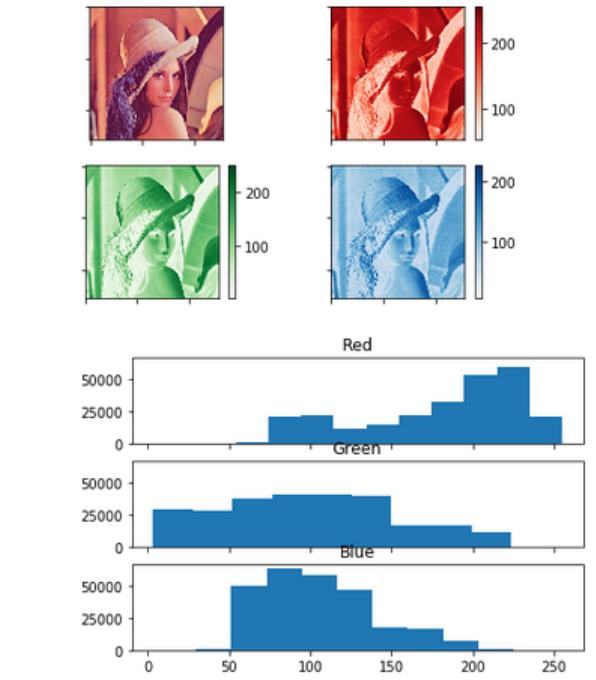
7 # Dividimos
red = lena[:, :, 0]
green = lena[:, :, 1]
blue = lena[:, :, 2]

11 # 'Graficas' de las im genes
13 fig, axs = plt.subplots(2,2)

15 cax_00 = axs[0,0].imshow(lena)
16 axs[0,0].xaxis.set_major_formatter(plt.NullFormatter())
17 axs[0,0].yaxis.set_major_formatter(plt.NullFormatter())
18
19 cax_01 = axs[0,1].imshow(red, cmap='Reds')
20 fig.colorbar(cax_01, ax=axs[0,1])
21 axs[0,1].xaxis.set_major_formatter(plt.NullFormatter())
22 axs[0,1].yaxis.set_major_formatter(plt.NullFormatter())
23
24 cax_10 = axs[1,0].imshow(green, cmap='Greens')
25 fig.colorbar(cax_10, ax=axs[1,0])
26 axs[1,0].xaxis.set_major_formatter(plt.NullFormatter())
27 axs[1,0].yaxis.set_major_formatter(plt.NullFormatter())
28
29 cax_11 = axs[1,1].imshow(blue, cmap='Blues')
30 fig.colorbar(cax_11, ax=axs[1,1])
31 axs[1,1].xaxis.set_major_formatter(plt.NullFormatter())
32 axs[1,1].yaxis.set_major_formatter(plt.NullFormatter())
33 plt.show()

35 # Graficamos los histogramas
37 fig, axs = plt.subplots(3, sharex=True, sharey=True)
38
39 axs[0].hist(red.ravel(), bins=10)
40 axs[0].set_title('Red')
41 axs[1].hist(green.ravel(), bins=10)
42 axs[1].set_title('Green')
43 axs[2].hist(blue.ravel(), bins=10)
44 axs[2].set_title('Blue')
45 plt.show()

```



```

import skimage.io as io
2
# Leemos la imagen
4 pc = io.imread('peppers_color.tif')
pc_skimage = img_as_float(pc)
6
# Dividimos
8 red = pc[:, :, 0]
green = pc[:, :, 1]
blue = pc[:, :, 2]
10
12 # Desplegamos de las im genes
13 fig, axs = plt.subplots(2,2)
14 fig.tight_layout()
15 cax_00 = axs[0,0].imshow(pc_skimage)
16 axs[0,0].xaxis.set_major_formatter(plt.NullFormatter())
17 axs[0,0].yaxis.set_major_formatter(plt.NullFormatter())
18
19 cax_01 = axs[0,1].imshow(red, cmap='Reds')
20 fig.colorbar(cax_01, ax=axs[0,1])
21 axs[0,1].xaxis.set_major_formatter(plt.NullFormatter())
22 axs[0,1].yaxis.set_major_formatter(plt.NullFormatter())
23
24 cax_10 = axs[1,0].imshow(green, cmap='Greens')
25 fig.colorbar(cax_10, ax=axs[1,0])
26 axs[1,0].xaxis.set_major_formatter(plt.NullFormatter())
27 axs[1,0].yaxis.set_major_formatter(plt.NullFormatter())
28
29 cax_11 = axs[1,1].imshow(blue, cmap='Blues')
30 fig.colorbar(cax_11, ax=axs[1,1])
31 axs[1,1].xaxis.set_major_formatter(plt.NullFormatter())
32 axs[1,1].yaxis.set_major_formatter(plt.NullFormatter())
33 plt.show()

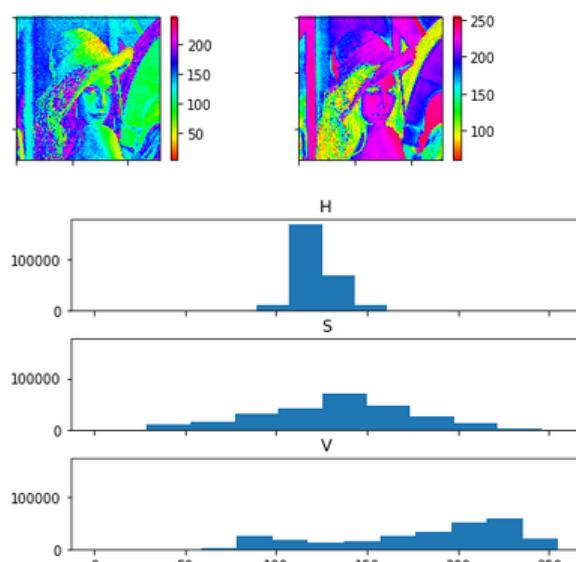
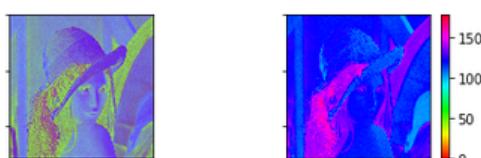
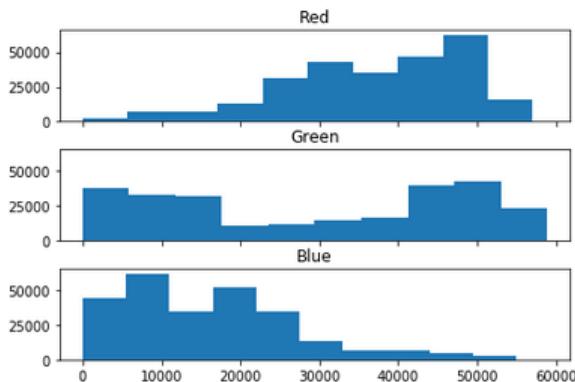
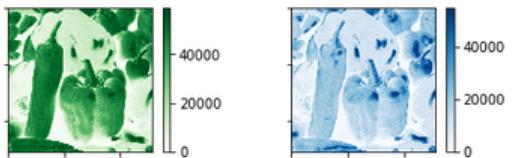
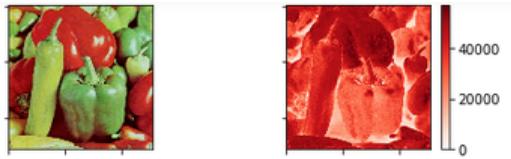
34 # Graficamos los histogramas
36 fig, axs = plt.subplots(3, sharex=True, sharey=True)
38 fig.tight_layout()
39 axs[0].hist(red.ravel(), bins=10)
40 axs[0].set_title('Red')

```

```

42 axs[1].hist(green.ravel(), bins=10)
43 axs[1].set_title('Green')
44 axs[2].hist(blue.ravel(), bins=10)
45 axs[2].set_title('Blue')
46 plt.show()
47
48 plt.show()
49 # Graficamos los histogramas
50 fig, axs = plt.subplots(3, sharex=True, sharey=True)
51 fig.tight_layout()
52 axs[0].hist(h.ravel(), bins=10)
53 axs[0].set_title('H')
54 axs[1].hist(s.ravel(), bins=10)
55 axs[1].set_title('S')
56 axs[2].hist(v.ravel(), bins=10)
57 axs[2].set_title('V')
58 plt.show()
59
60

```



4.1.3.5. Despliega la paleta de colores HSV por separado

```

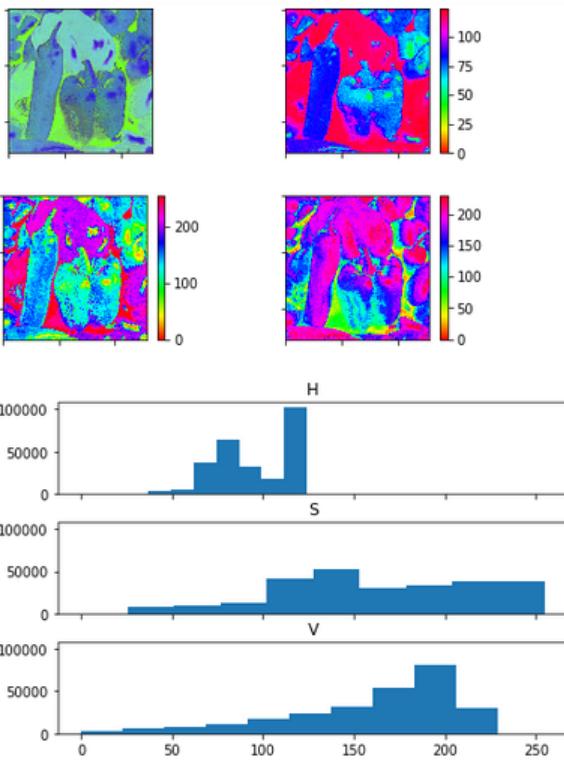
# Recordemos que ya las habamos convertido en hsv
# Dividimos
h = lena_hsv[:, :, 0]
s = lena_hsv[:, :, 1]
v = lena_hsv[:, :, 2]
6
# Desplegamos de las imágenes
7 fig, axs = plt.subplots(2,2)
8 fig.tight_layout()
9 cax_00 = axs[0,0].imshow(lena_hsv)
10 axs[0,0].xaxis.set_major_formatter(plt.NullFormatter())
11 axs[0,0].yaxis.set_major_formatter(plt.NullFormatter())
12 axs[0,0].xaxis.set_label('')
13 axs[0,0].yaxis.set_label('')
14 cax_01 = axs[0,1].imshow(h, cmap='hsv')
15 fig.colorbar(cax_01, ax=axs[0,1])
16 axs[0,1].xaxis.set_major_formatter(plt.NullFormatter())
17 axs[0,1].yaxis.set_major_formatter(plt.NullFormatter())
18 cax_10 = axs[1,0].imshow(s, cmap='hsv')
19 fig.colorbar(cax_10, ax=axs[1,0])
20 axs[1,0].xaxis.set_major_formatter(plt.NullFormatter())
21 axs[1,0].yaxis.set_major_formatter(plt.NullFormatter())
22 cax_11 = axs[1,1].imshow(v, cmap='hsv')
23 fig.colorbar(cax_11, ax=axs[1,1])
24 axs[1,1].xaxis.set_major_formatter(plt.NullFormatter())
25 axs[1,1].yaxis.set_major_formatter(plt.NullFormatter())
26
# Recordemos que ya las habamos convertido en hsv
# Dividimos
h = peppers_color_hsv[:, :, 0]
s = peppers_color_hsv[:, :, 1]
v = peppers_color_hsv[:, :, 2]
7
# Desplegamos de las imágenes
9 fig, axs = plt.subplots(2,2)
10 fig.tight_layout()
11 cax_00 = axs[0,0].imshow(peppers_color_hsv)
12 axs[0,0].xaxis.set_major_formatter(plt.NullFormatter())
13 axs[0,0].yaxis.set_major_formatter(plt.NullFormatter())
14 cax_01 = axs[0,1].imshow(h, cmap='hsv')
15 fig.colorbar(cax_01, ax=axs[0,1])
16 axs[0,1].xaxis.set_major_formatter(plt.NullFormatter())
17 axs[0,1].yaxis.set_major_formatter(plt.NullFormatter())
18 cax_10 = axs[1,0].imshow(s, cmap='hsv')
19 fig.colorbar(cax_10, ax=axs[1,0])
20 axs[1,0].xaxis.set_major_formatter(plt.NullFormatter())
21 axs[1,0].yaxis.set_major_formatter(plt.NullFormatter())
22

```

```

25 cax_11 = axs[1,1].imshow(v, cmap=' hsv ')
fig.colorbar(cax_11, ax=axs[1,1])
axs[1,1].xaxis.set_major_formatter(plt.NullFormatter())
27 axs[1,1].yaxis.set_major_formatter(plt.NullFormatter())
() plt.show()
29 # Graficamos los histogramas
31
33 fig, axs = plt.subplots(3, sharex=True, sharey=True)
fig.tight_layout()
35 axs[0].hist(h.ravel(), bins=10)
axs[0].set_title('H')
37 axs[1].hist(s.ravel(), bins=10)
axs[1].set_title('S')
39 axs[2].hist(v.ravel(), bins=10)
axs[2].set_title('V')
41 plt.show()

```



B. 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.

Inicialmente cambiamos a escala de grises, pues a color nos devuelve una Matriz tridimensional, por lo que será muy difícil de modificar, mientras que en escala de grises es una matriz unidimensional.

```

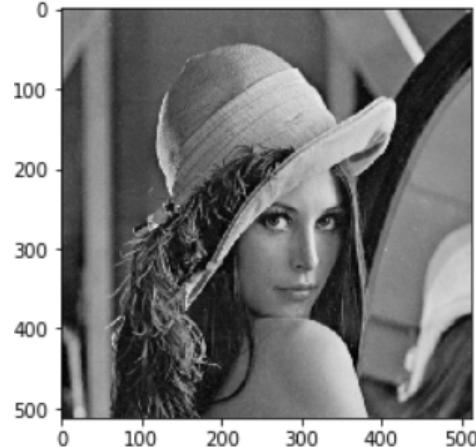
1 # Convertir RGB a escala de grises
2 imagen = mpimg.imread('lena_color_512.tif')
3 Imagen_gris = cv2.cvtColor(imagen, cv2.
    COLOR_BGR2GRAY)
# Desplegamos la imagen

```

```

5 plt.imshow(Imagen_gris, cmap=plt.get_cmap('gray'))
# Debido a que mostramos mediante imshow, tiene tonos
# verdes, por ello especificamos
7 # Gray, pero con la siguiente linea, podemos
# desplegar en tonos grises.
# Imagen_gris.show()
9 Imagen_gris.shape

```



Creamos una función que haga el promedio de los 4 píxeles y sustituya ese valor como uno nuevo, dentro de otra matriz, posteriormente se eliminan filas y columnas con valores = 0.

```

1 def decimacion(Imagen_gris):
2     a=np.array(Imagen_gris,dtype=np.float32)
3     print('Matriz Inicial:\n', a)
4     lx, ly = a.shape
5     New_mat = np.zeros((lx,ly))
6     for i in range(0,len(a),2):
7         for j in range(0,len(a),2):
8             New_mat[i,j]=((a[i,j]+a[i,j+1]+a[i+1,j]+a[i+1,j+1])/4)
9
10    good_cols = np.any(New_mat.T != 0, axis = 1)
11    matriz_Image = New_mat[:, good_cols][good_cols, :]
12    print('Matriz Nueva:\n', matriz_Image)
13    New_Image = Image.fromarray(matriz_Image.astype(
14        np.uint8))
15    print('Tamaño de imagen reducida:\n',New_Image.size)
16    print('Tamaño de imagen original:\n',
17        Imagen_gris.shape)
18    plt.figure(figsize=(20, 20))
19    plt.subplot(151)
20    plt.title('Original')
21    plt.imshow(Imagen_gris, cmap=plt.cm.gray)
22    plt.axis('off')
23    plt.subplot(152)
24    plt.title('Reducida')
25    plt.imshow(New_Image, cmap=plt.cm.gray)
26    plt.axis('off')
27    plt.show()
28 New_Image.show()

```

Mandamos a llamar a nuestra función.

```
decimacion(Imagen_gris)
```

Dando como resultado, la siguiente Imagen, por la izquierda la imagen original y por la derecha la imagen reducida, como podemos observar, la resolución espacial es distinta.



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

De igual manera, mandaremos a llamar a la función pero con distinto valor, en este caso 3.

Reajuste('cameraman','tif',10)

Lo cual nos generará nuevamente la misma imagen pero con 3 veces su tamaño.



C. 4.3.

Usando funciones de Python ya existentes y la imagen que usted escoja. Para este ejercicio, realizamos una función utilizando la Librería Cv2.resize, nuestra función recibe la imagen y las veces que se desea hacer el reajuste en tamaño, con el fin de poder ocupar dicha función para la actividad 4.3.1 y 4.3.2

```
1 import cv2
2 def Reajuste(imagen_name, formato, reajuste):
3     imagen_name = str(imagen_name)
4     formato = str(formato)
5     valor = str(reajuste)
6     imagen = mpimg.imread(imagen_name+'. '+formato)
7     lx, ly, d = imagen.shape
8     Nw_Imagen = cv2.resize(imagen, (int(lx*reajuste),
9         int(ly*reajuste)), interpolation=cv2.INTER_AREA)
10    cv2.imwrite(imagen_name+'x'+valor+'.'+formato,
11    Nw_Imagen)
```

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

Posteriormente Llamaremos a la función creada anteriormente,¹² especificando el valor 10.

1 Reajuste('lena_color_512','tif',10)

Lo cual nos generará nuevamente la misma imagen pero con 10 veces su tamaño.²⁰

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

De igual manera para realizar esta tarea, construimos una función que mediante la biblioteca Cv2, nos facilita la rotación de las fotografías.

```
1 import cv2
2 import matplotlib.pyplot as plt
3 def rotacion(imagen,tipo):
4     imagen = str(imagen)
5     tipo = str(tipo)
6     img = cv2.imread(imagen+tipo)
7     # obtiene la anchura y altura
8     (h, w) = img.shape[:2]
9     # calcula el centro de la imagen
10    center = (w / 2, h / 2)
11    # 90 grados
12    M = cv2.getRotationMatrix2D(center, 90, 1.0)
13    rotated90 = cv2.warpAffine(img, M, (h, w))
14    # 180 grados
15    M = cv2.getRotationMatrix2D(center, 180, 1.0)
16    rotated180 = cv2.warpAffine(img, M, (w, h))
17    # 45 grados
18    M = cv2.getRotationMatrix2D(center, 45, 1.0)
19    rotated45 = cv2.warpAffine(img, M, (h, w))
20    # guarda las nuevas imágenes
21    #cv2.imwrite(imagen+'_90.png',rotated90)
```

```

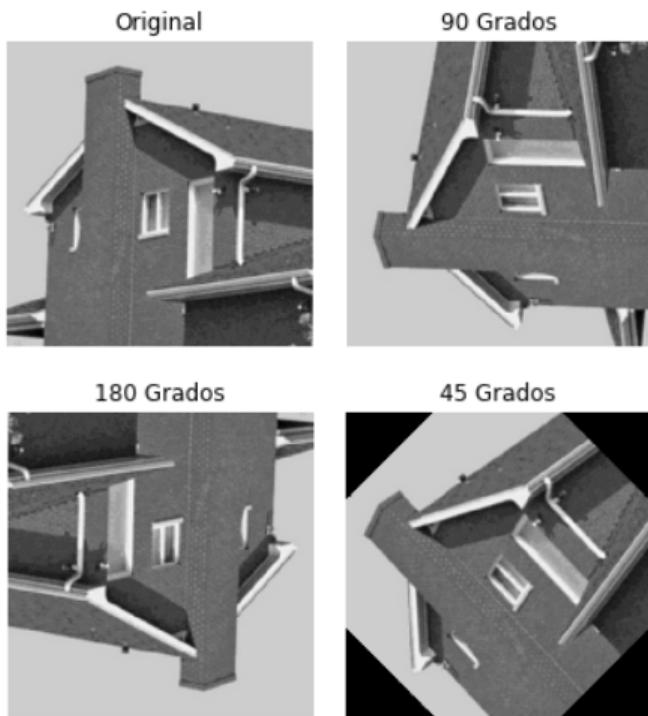
22 #cv2.imwrite(imagen+'_180.png',rotated180)
23 #cv2.imwrite(imagen+'_45.png',rotated45)
24 plt.figure(figsize=(12.5, 2.5))
25 plt.subplot(151)
26 plt.title('Original')
27 plt.imshow(img, cmap=plt.cm.gray)
28 plt.axis('off')
29 plt.subplot(152)
30 plt.title('90 Grados')
31 plt.imshow(rotated90, cmap=plt.cm.gray)
32 plt.axis('off')
33 plt.subplot(153)
34 plt.title('180 Grados')
35 plt.imshow(rotated180, cmap=plt.cm.gray)
36 plt.axis('off')
37 plt.subplot(154)
38 plt.title('45 Grados')
39 plt.imshow(rotated45, cmap=plt.cm.gray)
40 plt.axis('off')
41 plt.subplots_adjust(wspace=0.02, hspace=0.3, top=1, bottom=0.1, left=0,
42                                         right=1)
43 plt.show()

```

Una vez realizado esto, mandamos llamar a la función, dando el nombre del archivo, seguido del formato.

```
rotacion('house','.tif')
```

Dando el Siguiente resultado:



D. 4.4. Convierte la imagen peppers_color.tif a escala de grises.

Para poder realizar una conversión a escalas de grises usamos la fórmula

$$Y = R * 0.3 + G * 0.59 + B * 0.11$$

Puesto que es la cantidad de color que se necesita para conseguir formar el color gris es: Rojo 30%, Verde 59% y Azul 11%. Es decir, para cada pixel necesitamos su equivalente en Rojo, Verde y Azul, aplicamos la fórmula a cada pixel y obtenemos su equivalente en Gris.

```

1 import PIL
2
3 img_pepers = PIL.Image.open('peppers_color.tif')
4 img_pepers.show()
5
6 img_pepers = img_pepers.convert('RGB')
7
8 img_pepers_gray = PIL.Image.new('RGB',(512,512))
9
10 for i in range(512):
11     for j in range(512):
12         r, g, b = img_pepers.getpixel((i,j))
13
14         y = int(r * 0.3 + g * 0.59 + b * 0.11)
15
16         RGB = tuple([y, y, y])
17
18         img_pepers_gray.putpixel((i,j), RGB)
19
20 img_pepers_gray.save('peppers_gray.tif')
21 img_peppers_gray.show()
22
23

```



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

Nos apoyaremos en `.crop(left, top, right, bottom)` El cual recibe los parámetros:

1. Left: Límite mínimo en Y
2. Right: Límite máximo Y
3. Top: Límite mínimo en X
4. Bottom: Límite máximo en x

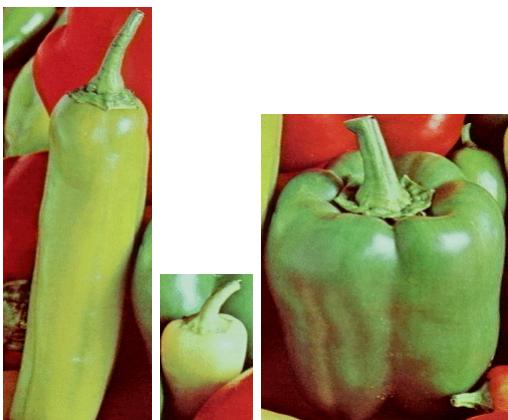
Es decir, si queremos el área (40:460, 52:200) sabemos que va de 40 (Top) a 460 (Bottom) en x Y de 52 (Left) a 200

(Right) en Y, por lo que la instrucción queda:

```
.crop( (52,40,200,460) )
```

```

1 left = 52
2 top = 40
3 right = 200
4 bottom = 460
5 crop_1 = img_pepers.crop((left , top , right , bottom))
6 crop_1.show()
7
8 left = 175
9 top = 180
10 right = 425
11 bottom = 490
12 crop_2 = img_pepers.crop((left , top , right , bottom))
13 crop_2.show()
14
15 left = 415
16 top = 140
17 right = 510
18 bottom = 290
19 crop_3 = img_peppers.crop((left , top , right , bottom))
20 crop_3.show()
```



Guárdela en formato .jpg

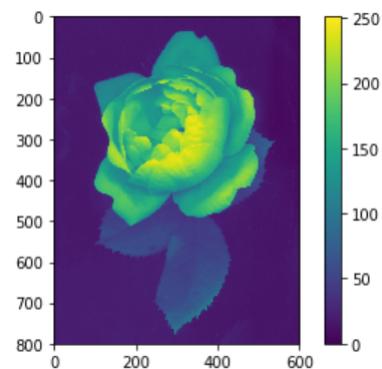
```

2 crop_1.save('peppers_recorte_1.jpg')
3 crop_2.save('peppers_recorte_2.jpg')
4 crop_3.save('peppers_recorte_3.jpg')
```

E. 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 pixels.

```

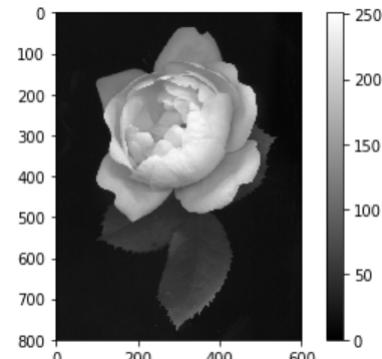
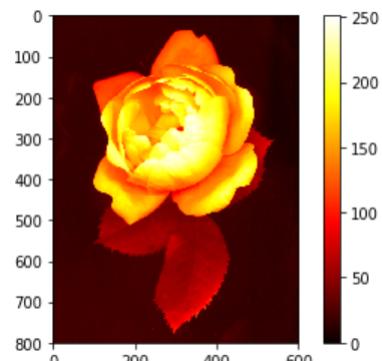
1 import numpy as np
2 import imageio
3
4 Rosa = np.fromfile('rosa800x600.raw', dtype=np.uint8)
5 Rosa.shape = (800,600)
6
7 plt.imshow(Rosa)
8 plt.colorbar()
```

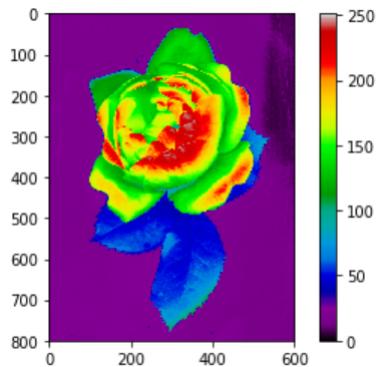


Finalmente, se probó la visualización de la imagen "rosa800x600.raw" con diferentes mapeos:

```

#En escala de grises
2 plt.imshow(Rosa , cmap = 'gray')
3 plt.colorbar()
4
#Escala de colores espectral
5 plt.imshow(Rosa , cmap = 'nipy_spectral')
6 plt.colorbar()
7
#Escala 'hot'
8 plt.imshow(Rosa , cmap = 'hot')
9 plt.colorbar()
```





IV. CONCLUSIÓN

EXISTEN diferentes tipos de formatos de imágenes, los cuales han sido diseñados con diferentes objetivos, tal es el caso de las imágenes *.dcm*, las cuales han sido desarrolladas para trabajos del área médica, *.jpg* se refiere al estándar de imágenes, que en general reduce tamaños de imagen (con pérdidas), *.png* que se refiere a un formato de compresión en el cual no se tienen presentes las pérdidas, entre otros.

En esta práctica se logró la familiarización e introducción a las diversas librerías de *python* que nos permiten y facilitan la manipulación de diversos tipos de imágenes.

Finalmente, podemos concluir que pudimos mejorar el aspecto de las imágenes y hacer más evidentes en ellas ciertos detalles que se desean hacer notar. El aprovechar las librerías que ofrece Python para trabajar con imágenes así como el alto desempeño de graficación nos permitirá implementar sistemas y funciones para el reconocimiento de patrones. De esta manera, la práctica estuvo diseñada para facilitar la tarea labor del análisis de imágenes.