Segundo Parcial de Tratamiento de Señales

Fuente: https://scipy-lectures.org/advanced/image_processing/

Realice los siguientes ejercicios. Conteste las preguntas y haga las modificaciones pedidas en algunos de los ejercicios**. Las preguntas y modificaciones están resaltadas en negritas en el documento.**

1. Abrir y escribir una imagen a un archivo

```
from scipy import misc
f = misc.face()
misc.imsave('face.png', f)

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

Nota: Si tiene problemas con `misc.imsave` use: `io.imwrite('face.png', f)`. Para usar `io.imwrite` debe importar imageio: `import imageio as io`.

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

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

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

face.shape, face.dtype
```

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

¿De qué tipo es la variable `face`?

¿Qué resultado arroja `face.shape`?

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

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)
```

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

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`:

```
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.

4. Despliegue de imágenes

Use `matplotlib` e `imshow` para desplegar una imagen dentro de una figura de `matplotlib`:

```
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`?

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

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

Se puede incrementar 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')
```

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

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

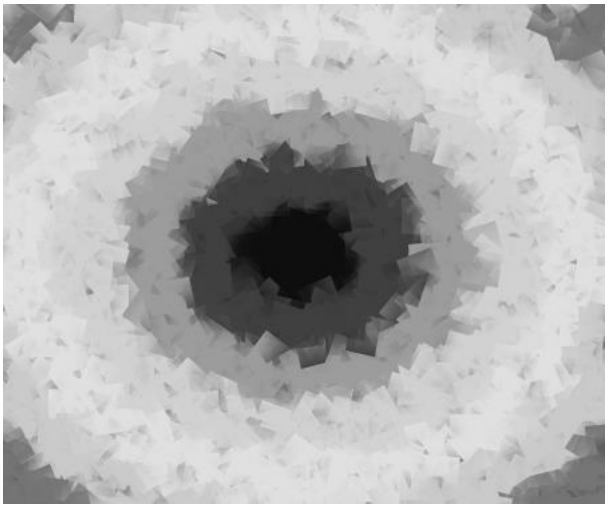
En la siguiente figura se mejora el contraste de la imagen de enmedio. La imagen de la derecha muestra el resultado de `plt.contour`.

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

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`):

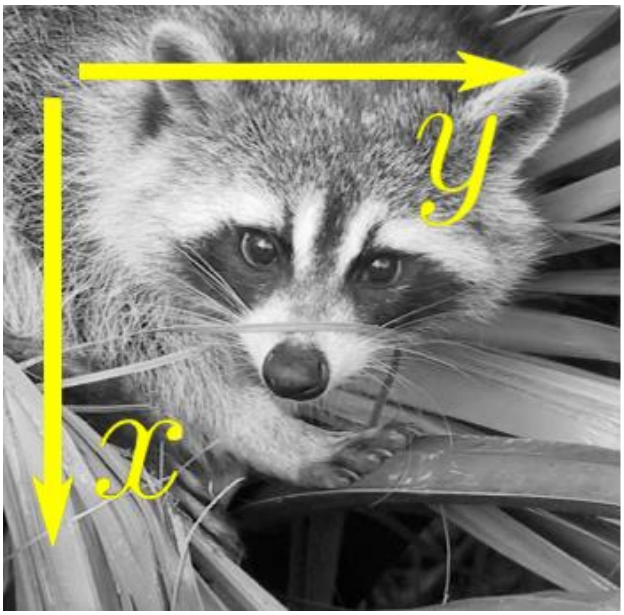
```
plt.contour(f, 5)
plt.contour(f, [50, 100, 200])
```

Imagen contour_gray.png



5. Manipulaciones básicas

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



0	1	2
3	4	5
6	7	8

```
# Obtiene el valor de un pixel de la imagen
face = misc.face(gray=True)
face[0, 40]
```

¿Cuánto vale el pixel `face[0, 40]`?

```
# Accesando secciones de la imagen
face[10:13, 20:23]
face[100:120] = 255
```

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

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.

```
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`?

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

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

Información Estadística

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

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

face.max(), face.min()
```

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:

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

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

Transformaciones geométricas

```
face = misc.face(gray=True)
lx, ly = face.shape
# Recorte
crop_face = face[lx // 4: - lx // 4, ly // 4: - ly // 4]
# up <-> down Voltear verticalmente
flip_ud_face = np.flipud(face)
# Rotación
rotate_face = ndimage.rotate(face, 45)
rotate_face_noreshape = ndimage.rotate(face, 45, reshape=False)
```

En la siguiente figura 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.

¿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.