TP3

Luciana Diaz Kralj
Agustina Sol Ortu
Paula Oseroff

18 de septiembre de 2024

Profesor: Daniel Jacoby

ex1

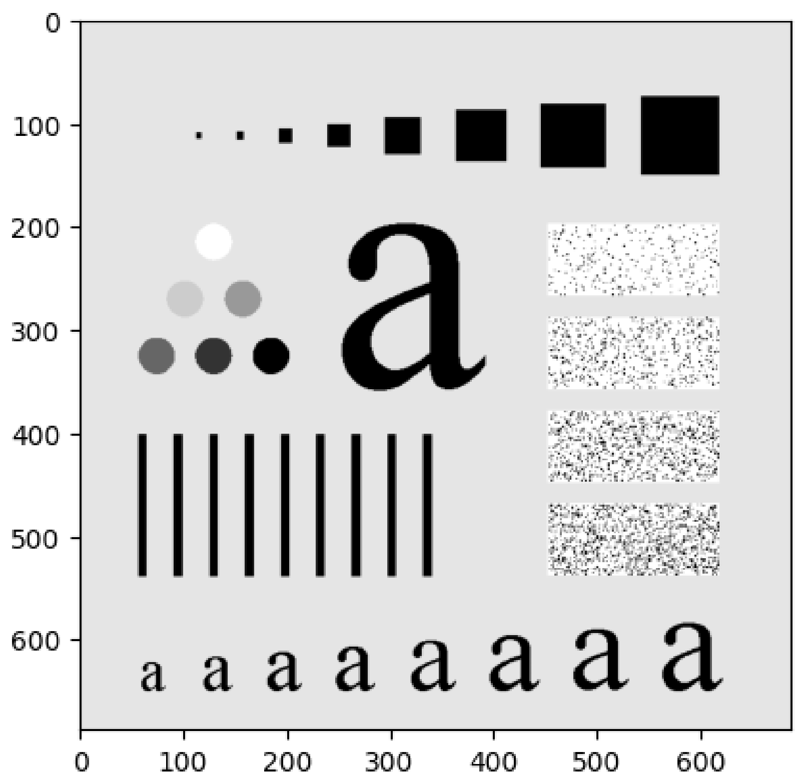
September 18, 2024

[14]:

```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from scipy.signal import convolve2d
```

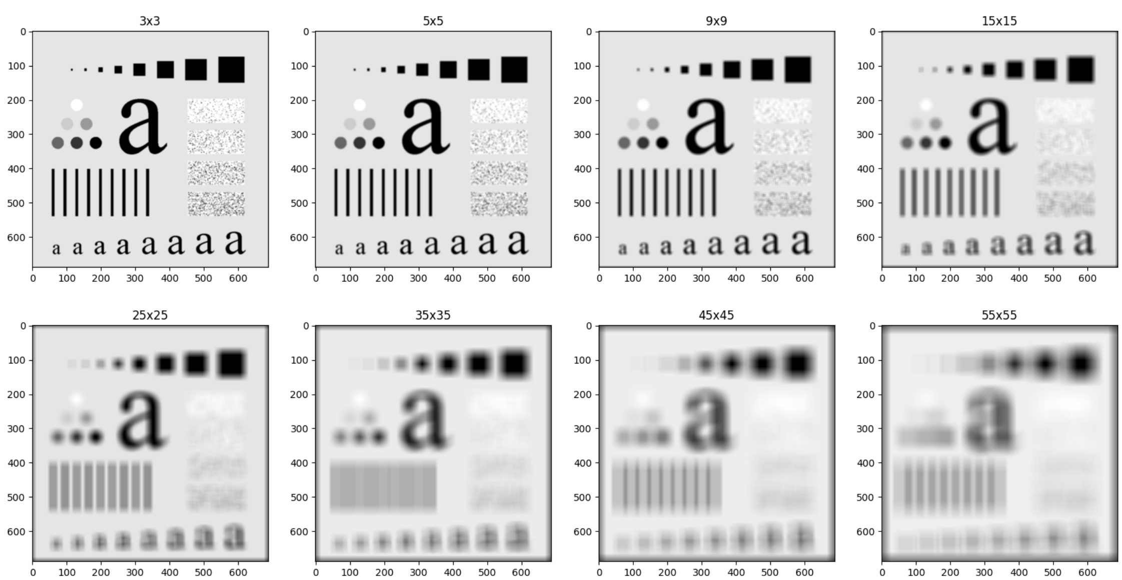
[19]:

```
f = np.array(Image.open('./assets/Fig0448(a)(characters_test_pattern).tif').
 ↪convert('L'))

# Plot the image
plt.imshow(f, cmap='gray')
plt.show()
```



[20]:
```
# Declare the sizes of kernel to be used when filtering
sizes = [3, 5, 9, 15, 25, 35, 45, 55]

# Apply each kernel and store the filtered image
filter = np.zeros((len(sizes), f.shape[0], f.shape[1]))

for i, size in enumerate(sizes):

    # Create the kernel with the appropriate size (n*n) and normalize it␣
↪(divide it for it size)
    kernel = np.ones((size,size), np.float32) / size**2

    # Apply the filter using a 2D convolution
    filter[i,:,:] = convolve2d(f, kernel, mode='same')
```

[21]:
```
# Plotting
fig, ax = plt.subplots(2, len(filter) // 2, figsize=(20, 10))
for i, filtered_image in enumerate(filter):
    ax.flatten()[i].set_title(f'{sizes[i]}x{sizes[i]}')
    ax.flatten()[i].imshow(filtered_image, cmap='gray')

# Show the plot
plt.show()
```



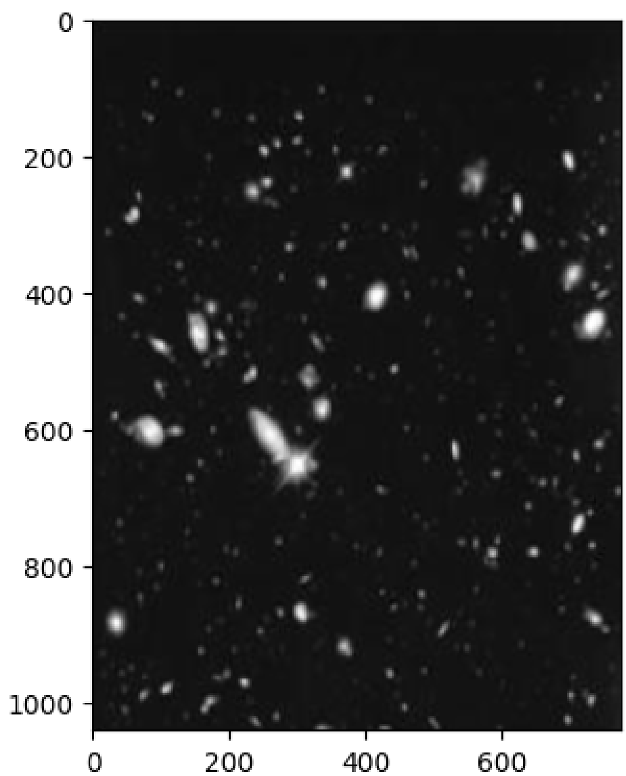
Esto demuestra que aplicar filtros pasabajos con diferentes tamaños a una imagen provoca un aumento en el efecto de difuminado o desenfoque a medida que incrementa justamente el tamanio. Esto se da porque los filtros eliminan las altas frecuencias espaciales, lo que suaviza los detalles finos y reduce la definición de la imagen. Este efecto es útil para reducir el ruido, ya que las áreas con patrones aleatorios se vuelven homogéneas, pero también puede causar una pérdida significativa de información espacial, como la distinción entre líneas y bordes. En conclusion mientras las imagenes pequeñas ofrecen un suavizado moderado, los más grandes pueden resultar en una imagen con menos detalles y más difusa, es esencial elegir el tamaño del kernel según el objetivo específico del procesamiento de la imagen.

ex2

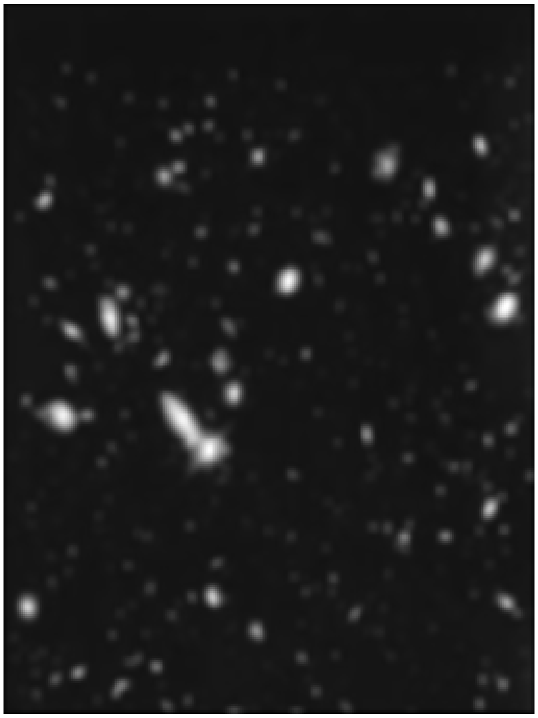
September 18, 2024

[5]:
```
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from scipy.signal import convolve2d
```

[6]:
```
f = np.array(Image.open('./assets/stars.png').convert('L'))

# Plot the image
plt.imshow(f, cmap='gray')
plt.show()
```



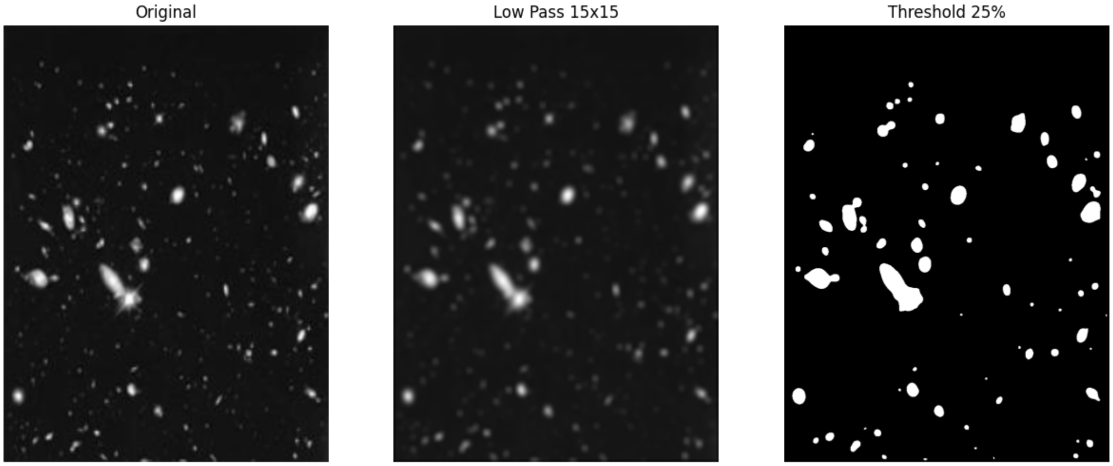
[20]:
```
# Low Pass 15x15
n = 15
kernel = np.ones((n,n), np.float32) / n**2

filter = convolve2d(f, kernel, mode='same')

plt.imshow(filter, cmap='gray')
plt.axis('off')
plt.show()
```



[21]:
```
# Threshold 25%
th = 0.25 * 255

# f > th -> each pixel will be True or False according if they are bigger or␣
↪lower than the threshold
# np.uint8 -> converts True and False into 1 and 0
# and then multiply all by 255, so finally:
#
# Pixels with intensity values greater than th will be set to the maximum value␣
↪(255),
# while pixels with intensity values less than or equal to th will be set to 0.
threshold = (filter > th).astype(np.uint8) * 255
```

```
plt.imshow(threshold, cmap='gray')
plt.axis('off')
plt.show()
```

[22]:
```
plt.figure(figsize=(15,8))
ax = plt.subplot(1,3,1)
ax.imshow(f, cmap='gray')
ax.axis('off')
ax.set_title('Original')

ax = plt.subplot(1,3,2)
ax.imshow(filter, cmap='gray')
ax.set_title(f'Low Pass {n}x{n}')
ax.axis('off')

ax = plt.subplot(1,3,3)
ax.imshow(threshold, cmap='gray')
ax.set_title(f'Threshold {th / 255 * 100:.0f}%')
ax.axis('off')

plt.show()
```


Original
Low Pass 15x15
Threshold 25%

ex3

September 18, 2024

[1]:
```python
import numpy as np
import cv2
from pathlib import Path
from skimage.util import random_noise
from scipy import signal
import skimage
```

[2]:
```python
ASSETS_FOLDER_PATH = "./assets"
OUTPUT_FOLDER_PATH = "."
```

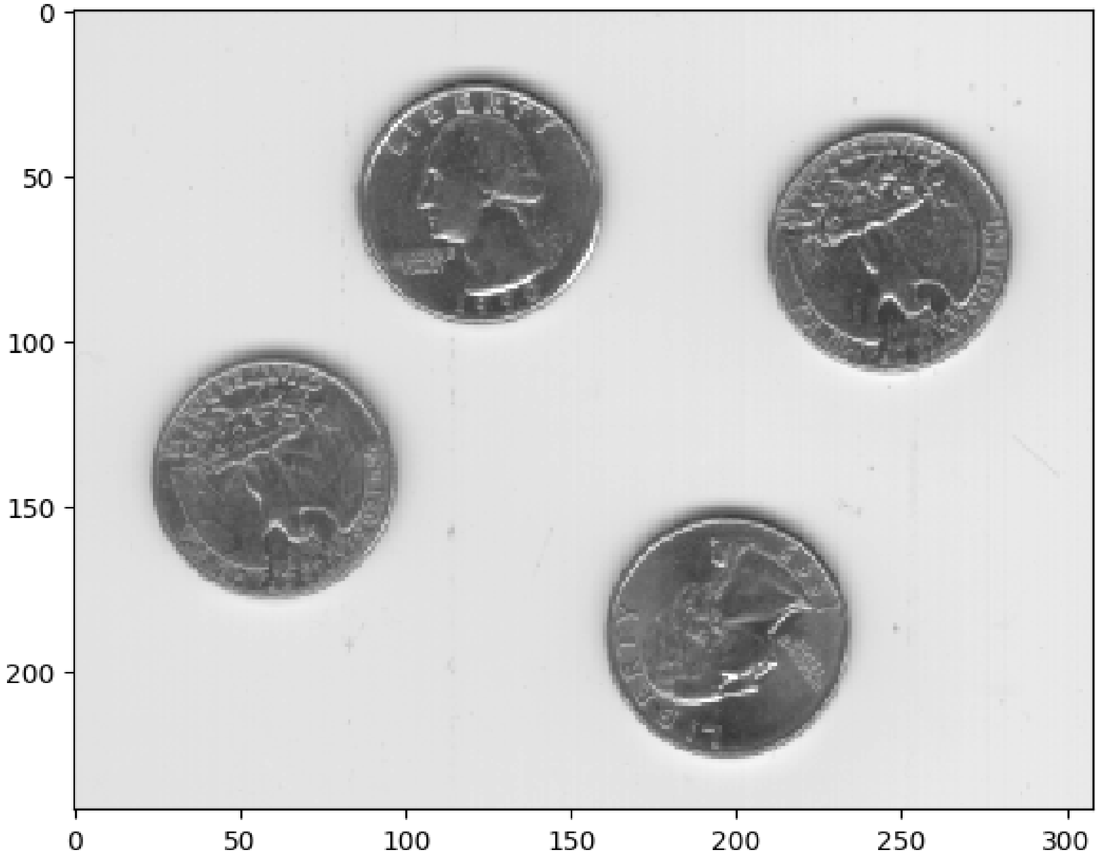
[3]:
```python
Path(OUTPUT_FOLDER_PATH).mkdir(parents=True, exist_ok=True)
```

Cargamos la imagen

[4]:
```python
eight = skimage.io.imread(fname=f"{ASSETS_FOLDER_PATH}/eight.tif")
```

[5]:
```python
skimage.io.imshow(eight)
```

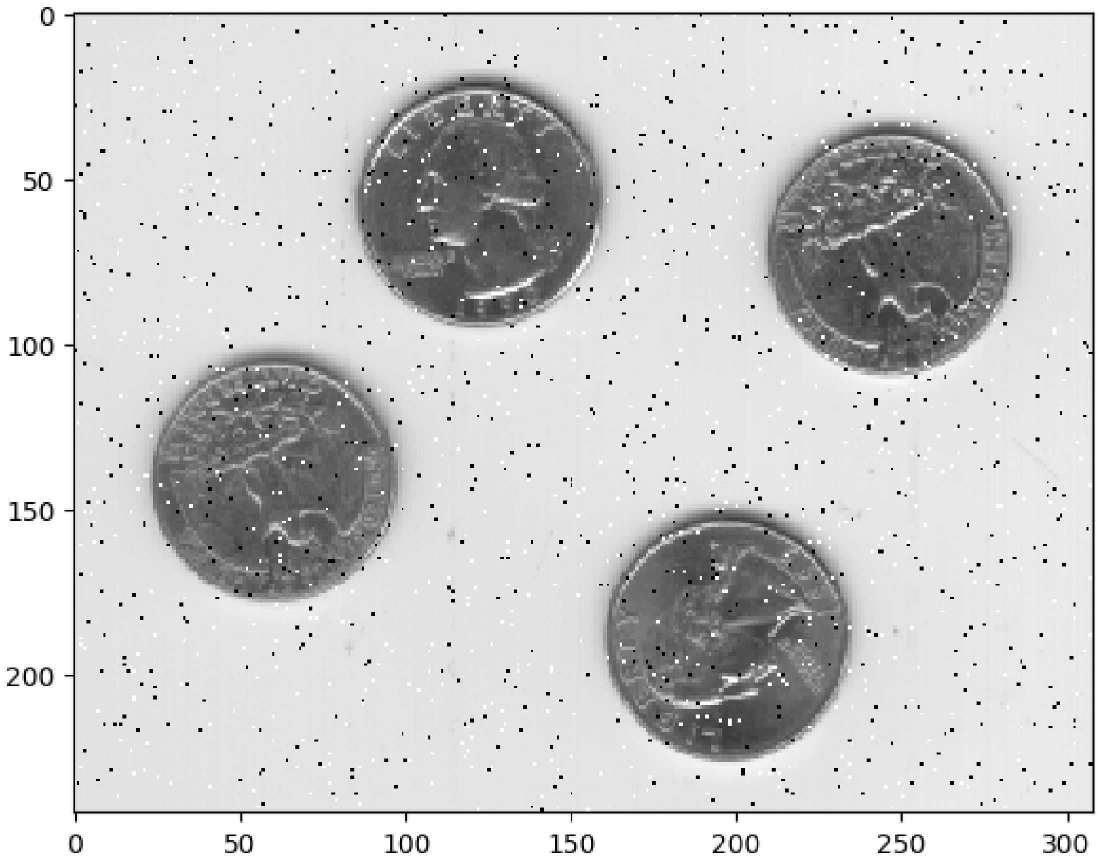
[5]:
```
<matplotlib.image.AxesImage at 0x7147602ad870>
```



```
[6]: sp_eight = random_noise(eight, mode='s&p', amount=0.02)
```

```
[7]: skimage.io.imshow(sp_eight)
```

```
[7]: <matplotlib.image.AxesImage at 0x7147601b0d00>
```



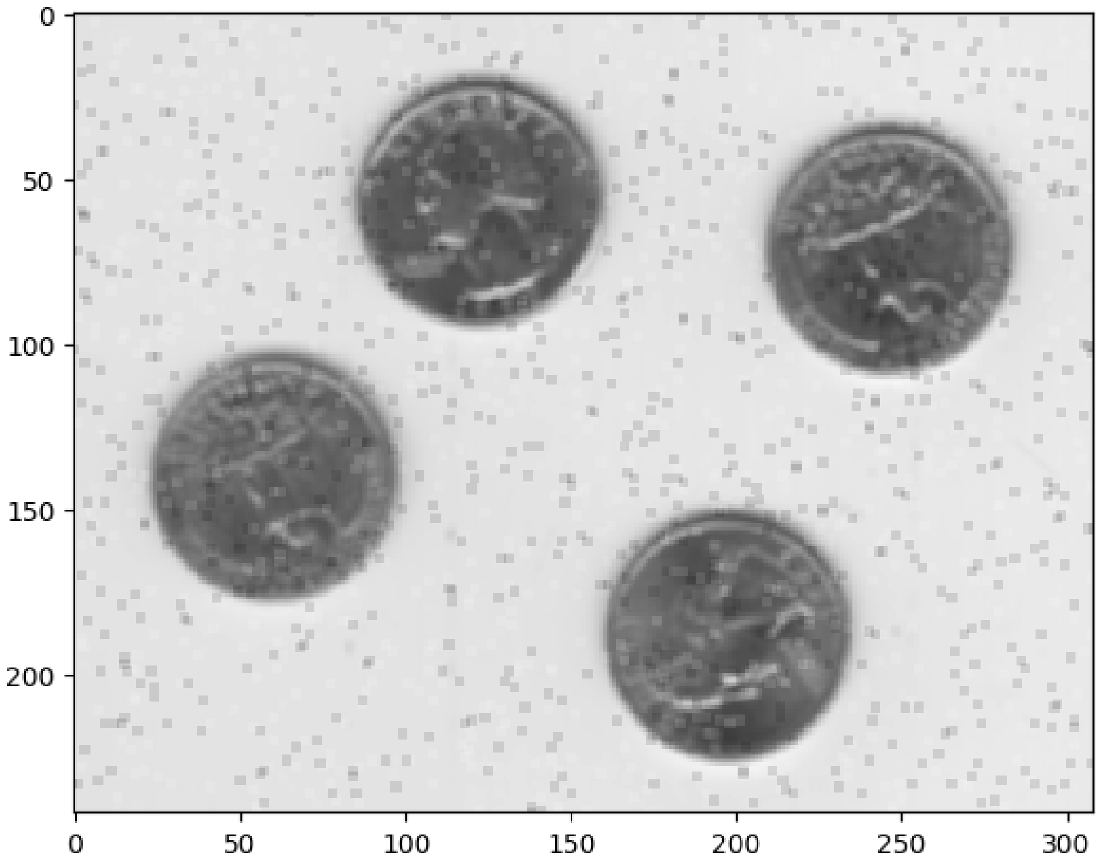
[8]:
```
average = np.ones((3, 3))
average = average / average.sum()
average
```

[8]:
```
array([[0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111],
       [0.11111111, 0.11111111, 0.11111111]])
```

[9]:
```
avg_eight = cv2.filter2D(src=sp_eight, ddepth=-1, kernel=average)
```

[10]:
```
skimage.io.imshow(avg_eight)
```

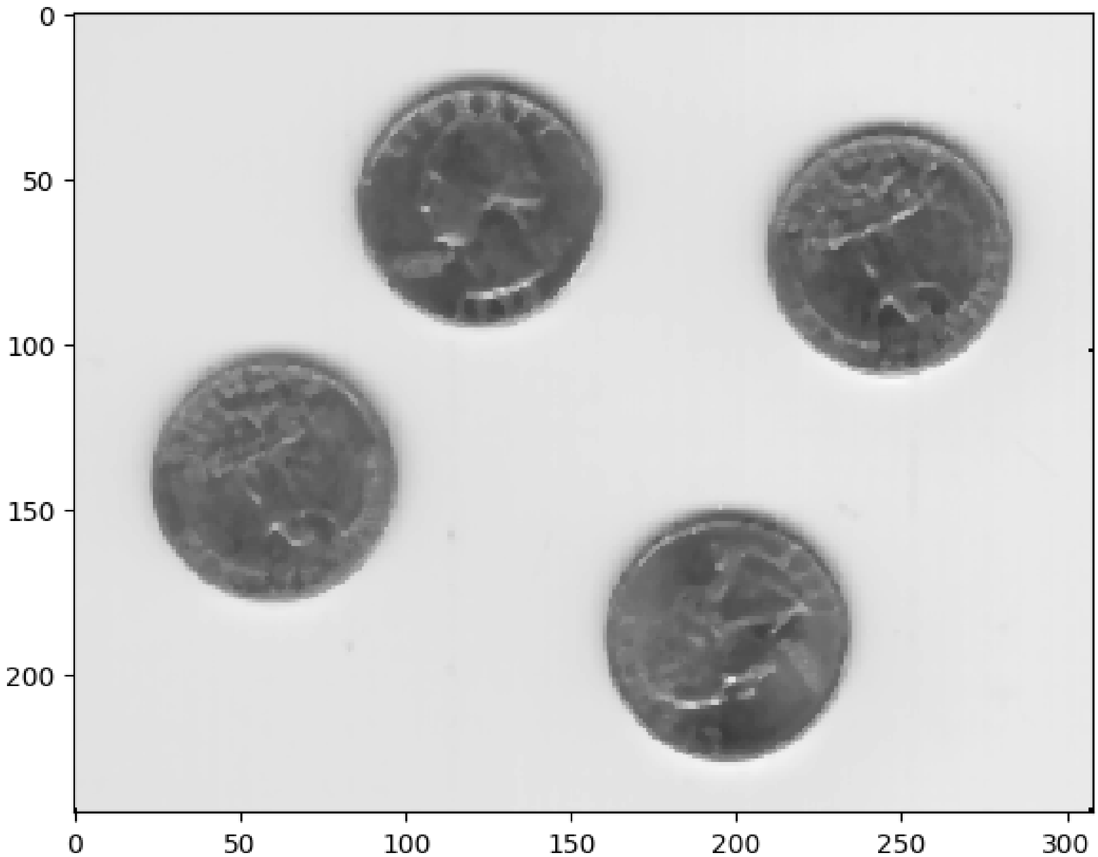
[10]:
```
<matplotlib.image.AxesImage at 0x71475dd26020>
```



[11]:
```
med_eight = signal.medfilt2d(sp_eight, kernel_size=[3, 3])
```

[12]:
```
skimage.io.imshow(med_eight)
```

[12]: <matplotlib.image.AxesImage at 0x71475dd96a40>

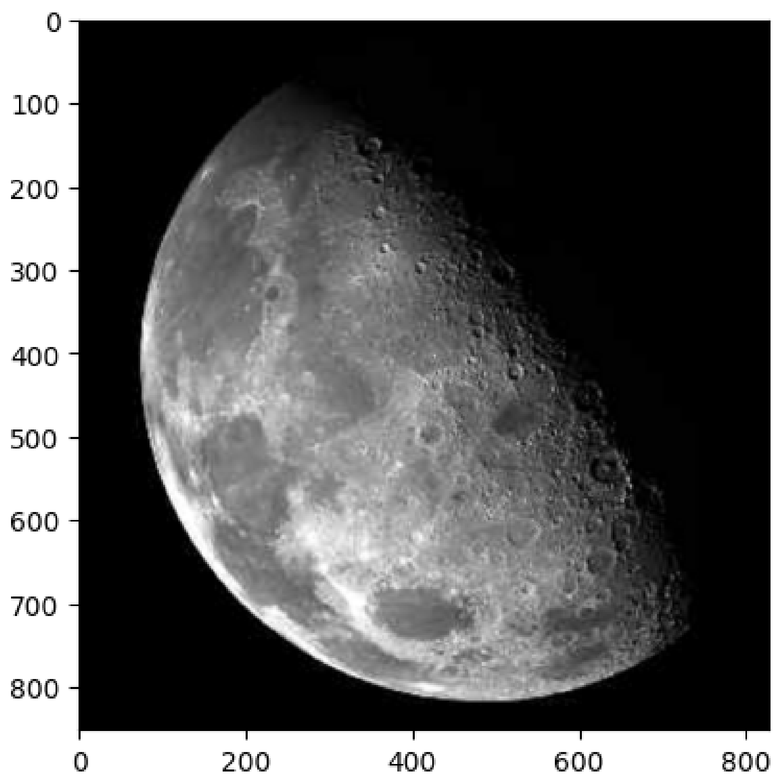

0
50
100
150
200
0
50
100
150
200
250
300

ex4

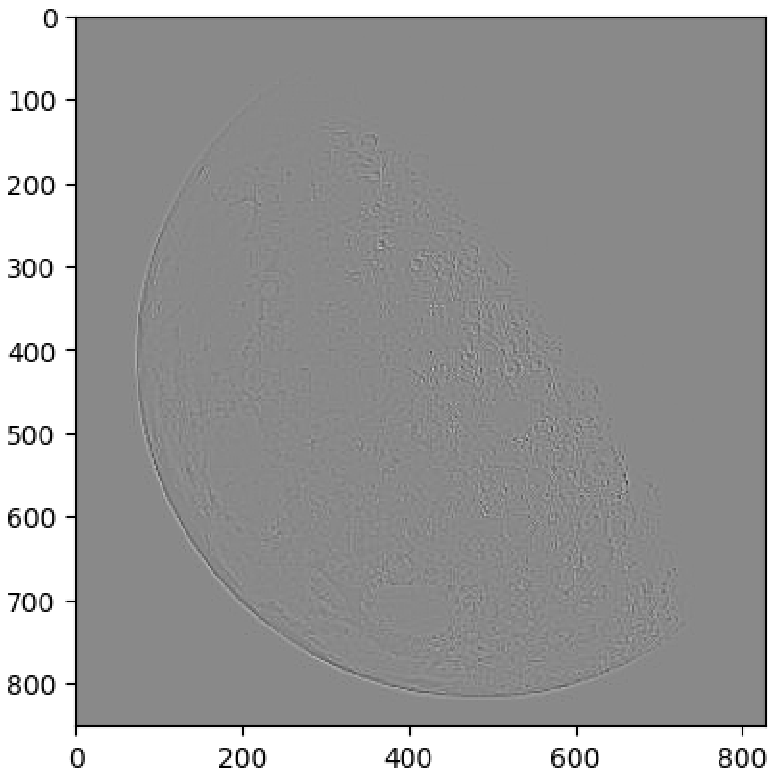
September 18, 2024

[5]:
```python
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from scipy.signal import convolve2d
```

[6]:
```python
f = np.array(Image.open('./assets/moon.png').convert('L'))

# Plot the image
plt.imshow(f, cmap='gray')
plt.show()
```



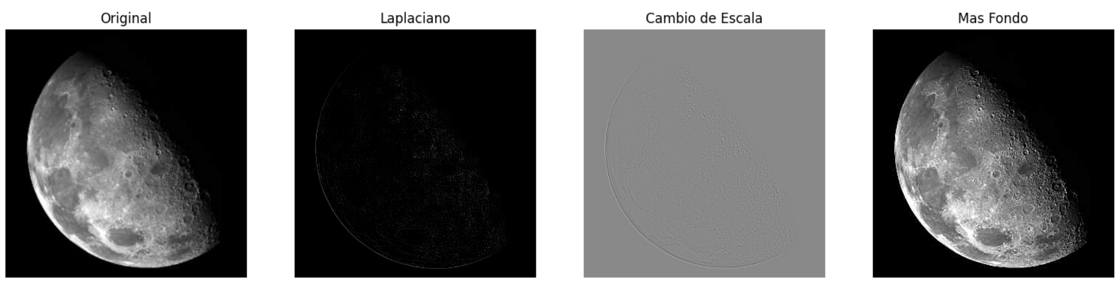
[16]:
```
kernel = np.ones((3,3))
kernel[1,1] = -8
# 1  1 1
# 1 -8 1
# 1  1 1
laplacian = convolve2d(f, kernel, mode='same')

# Plot the image
plt.imshow(laplacian, cmap='gray')
plt.show()
```



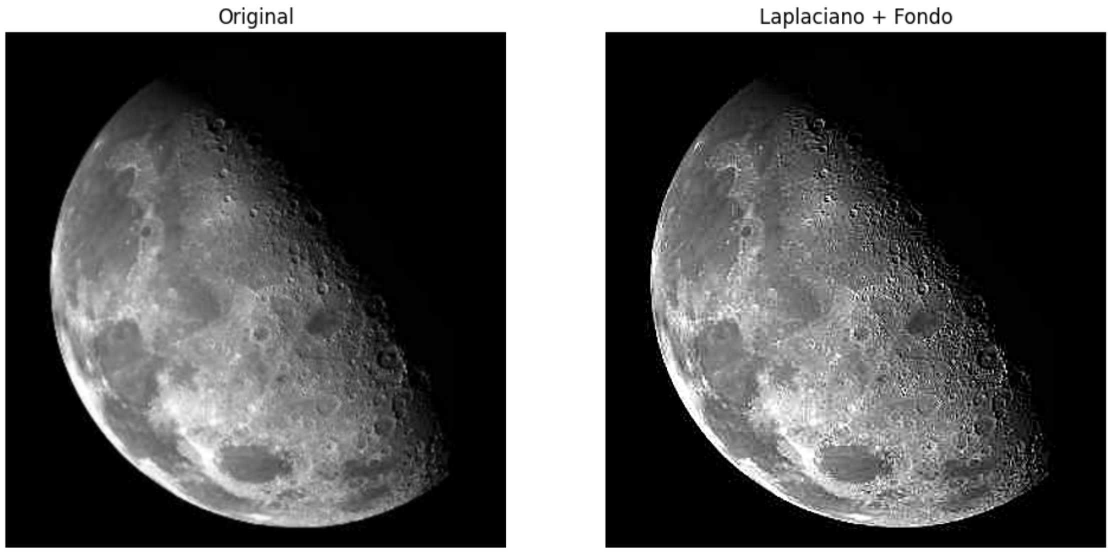
[27]:
```
kernel = -np.ones((3,3))
kernel[1,1] = 9
# -1 -1 -1
# -1  9 -1
# -1 -1 -1
laplacian_fondo = convolve2d(f, kernel, mode='same')
```

[28]:
```
plt.figure(figsize=(18,8))
ax = plt.subplot(1,4,1)
ax.imshow(f, cmap='gray', vmin=0, vmax=255)
ax.axis('off')
```

```
ax.set_title('Original')

ax = plt.subplot(1,4,2)
ax.imshow(laplacian, cmap='gray', vmin=0, vmax=255)
ax.set_title(f'Laplaciano')
ax.axis('off')

ax = plt.subplot(1,4,3)
ax.imshow(laplacian, cmap='gray')
ax.set_title(f'Cambio de Escala')
ax.axis('off')

ax = plt.subplot(1,4,4)
ax.imshow(laplacian_fondo, cmap='gray', vmin=0, vmax=255)
ax.set_title(f'Mas Fondo')
ax.axis('off')

plt.show()
```



[29]:
```
plt.figure(figsize=(12,8))
ax = plt.subplot(1,2,1)
ax.imshow(f, cmap='gray', vmin=0, vmax=255)
ax.axis('off')
ax.set_title('Original')

ax = plt.subplot(1,2,2)
ax.imshow(laplacian_fondo, cmap='gray', vmin=0, vmax=255)
ax.set_title(f'Laplaciano + Fondo')
ax.axis('off')

plt.show()
```


Original
Laplaciano + Fondo

ex5

September 18, 2024

1 ex5: High Boost

[81]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

[82]:
```
def plot_images(img1, title1, img2, title2, img3, title3, img4, title4):
    fig = plt.figure(figsize=(6, 7))

    fig.add_subplot(2, 2, 1)
    plt.imshow(img1)
    plt.axis('off')
    plt.title(title1)

    fig.add_subplot(2, 2, 2)
    plt.imshow(img2)
    plt.axis('off')
    plt.title(title2)

    fig.add_subplot(2, 2, 3)
    plt.imshow(img3)
    plt.axis('off')
    plt.title(title3)

    fig.add_subplot(2, 2, 4)
    plt.imshow(img4)
    plt.axis('off')
    plt.title(title4)

    plt.subplots_adjust(wspace=0, hspace=0.17)
    plt.show()
```

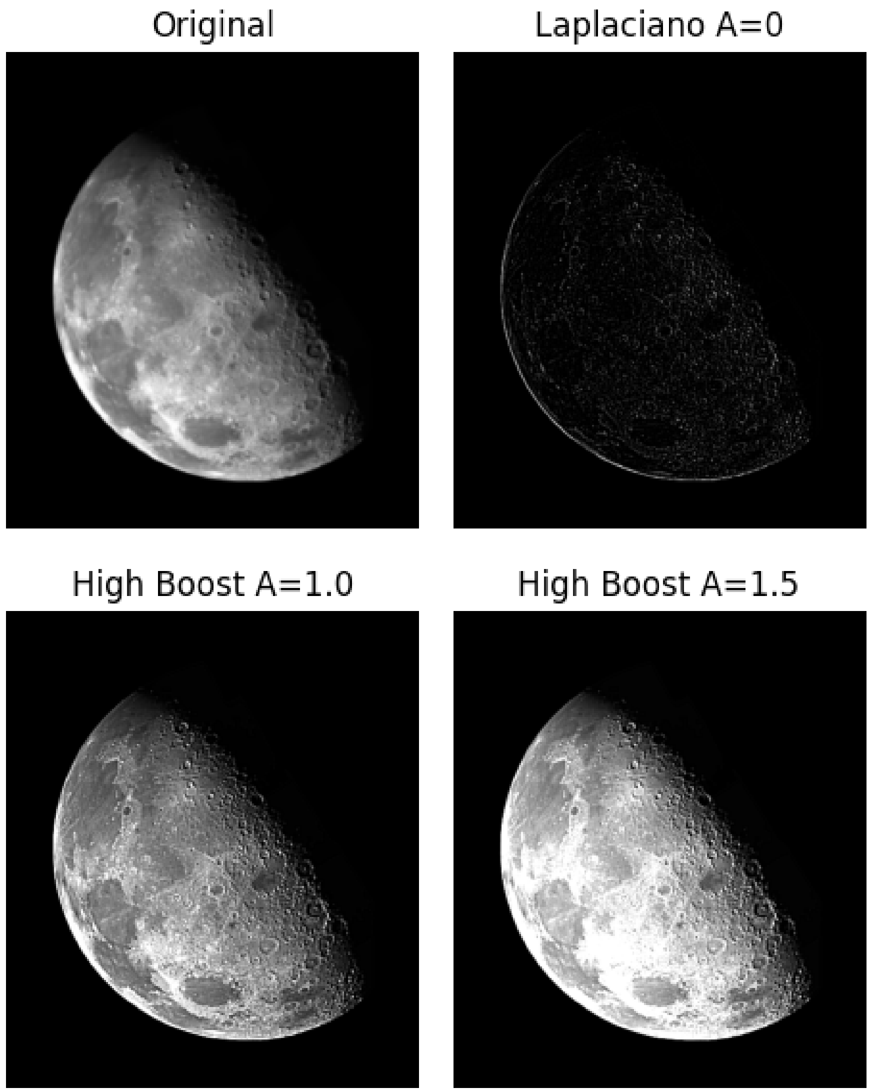
`high_boost_filter` aplica un filtro que realza los detalles de una imagen aumentando los componentes de la alta frecuencia de una imagen. El parámetro `A` controla la cantidad de dicha amplificación.

[83]:
```
def high_boost_filter(image, A):
    kernel = np.array([[-1,   -1,     -1],
                       [-1,   A + 8,  -1],
                       [-1,   -1,     -1]])
    return cv2.filter2D(image, ddepth=-1, kernel=kernel)
```

Si aumentamos el valor de `A`:

[86]:
```
def show_high_boost(image):
    A = 0
    filtered_image = high_boost_filter(image, A)
    A1 = 1.0
    filtered_image1 = high_boost_filter(image, A1)
    A2 = 1.5
    filtered_image2 = high_boost_filter(image, A2)
    plot_images(image, "Original",
                filtered_image, f"Laplaciano A={A}",
                filtered_image1, f"High Boost A={A1}",
                filtered_image2, f"High Boost A={A2}")
```

[87]:
```
image = cv2.imread('assets/blurry_moon.tif')
show_high_boost(image)
```



En principio, incrementar el valor de `A` resulta en una mejora pronunciada de los bordes y detalles ya que suelen ser los componentes con mayor valor de frecuencia espacial de una imagen, pero pasado cierto valor se vuelven a perder.

ex6

September 18, 2024

[1]:
```
import numpy as np
from scipy import fft
from PIL import Image
import matplotlib.pyplot as plt
```

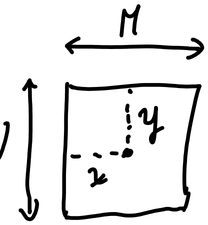
[3]:
```
f = np.array(Image.open('./assets/tun.jpg').convert('L'))

# Plot the image
plt.imshow(f, cmap='gray')
plt.show()
```

[17]:
```python
# f(x,y) -> ln -> DFT -> H(u,v) -> (DFT)^-1 -> exp -> g(x,y)
def homomorphic_filtering(f, y_H=0.4, y_L=1.6, c=1, D_0=0.1):
    # ln -> DFT
    X = fft.fft2(np.log(f + 1e-8))

    # H(u,v)
    x, y = X.shape
    d_center = np.array([x // 2, y // 2])
    # D(u,v): Distancia desde el origen
    D = np.array([np.sqrt((range(x) - d_center[0])**2 + (i - d_center[1])**2)␣
 ↪for i in range(y)]).T

    # Diseño del filtro
    # H = (yH - yL) * [1-e^(-c*(D(u,v)^2 / D0^2))] + yL
    H = (y_H - y_L) * (1 - np.exp(-c *(D**2 / (D_0 * x)**2))) + y_L

    # Normalización del filtro
    H = np.interp(H, (H.min(), H.max()), (H.min(), 1))

    # Aplicación del filtro en el dominio de la frecuencia
    Y = np.multiply(X, H)

    # (DFT)^-1 -> exp -> g(x,y)
    g = np.exp(fft.ifft2(Y).real)

    return g, H
```

[22]:
```python
g, H = homomorphic_filtering(f)

plt.figure(figsize=(12,8))
ax = plt.subplot(1,2,1)
ax.imshow(f, cmap='gray')
ax.axis('off')
ax.set_title('Original')

ax = plt.subplot(1,2,2)
ax.imshow(g, cmap='gray')
ax.set_title(f'Homomorphic Filtered')
ax.axis('off')

plt.show()
```


Original
Homomorphic Filtered

Dada una imágen $i(x,y)$ de dimensiones $M \times N$



Primero desplazamos la imágen al centro

$$i_1(x,y) = (-1)^{x+y} \cdot i(x,y)$$

Sabemos que la DFT de una función f

$$F(u,v) = DFT\{f(x,y)\} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\, e^{-2\pi j\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

Aplicando la DFT a nuestra i_1

$$I_1(u,v) = DFT\{i_1(x,y)\} = DFT\{(-1)^{x+y} i(x,y)\}$$

$$= \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} e^{\pi j x} e^{\pi j y}\, i(x,y)\, e^{-2\pi j\left(\frac{ux}{M} + \frac{vy}{N}\right)} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} i(x,y)\, e^{-2\pi j\left(\frac{u-\frac{M}{2}}{M}x + \frac{v-\frac{N}{2}}{N}y\right)}$$

$$= I\left(u-\frac{M}{2}, v-\frac{N}{2}\right)$$

Al conjugarlo

tomando $x' = -x$, $y' = -y$

$$I_2(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} i(x,y)\, e^{2\pi j\left(\frac{u-\frac{M}{2}}{M}x + \frac{v-\frac{N}{2}}{N}y\right)} = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} i(-x',-y')\, e^{-2\pi j\left(\frac{u-\frac{M}{2}}{M}x' + \frac{v-\frac{N}{2}}{N}y'\right)}$$

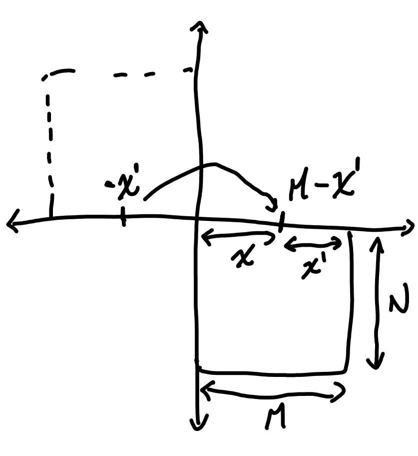
Luego, aplicamos la IDFT

$$i_2(x',y') = IDFT\{I_2(u,v)\} = i(-x',-y')\,(-1)^{x'+y'}$$

Entonces volvemos a trasladar la imágen

$$i'(x',y') = i(-x',-y')(-1)^{x'+y'}(-1)^{x+y} = i(-x',-y')\,(-1)^{x'+x+y'+y}$$

Ahora desplazamos la imágen al cuadrante original



$$\left.\begin{aligned} M-x' &= -x' = x \\ N-y' &= -y' = y \end{aligned}\right\} \quad \begin{aligned} x+x' &= M \\ y+y' &= N \end{aligned}$$

$$\Rightarrow i'(x',y') = i(-x',-y')(-1)^{M+N}$$

si $M+N$ es par $\Rightarrow i'(x',y') = i(-x',-y')$ la imagen queda invertida

si $M+N$ es impar $\Rightarrow i'(x',y') = -i(-x',-y')$ la imagen no solo queda invertida, si no que queda en su NEGATIVO

ex7bc

September 18, 2024

[1]:
```python
import matplotlib.pyplot as plt
import numpy as np
from PIL import Image
```

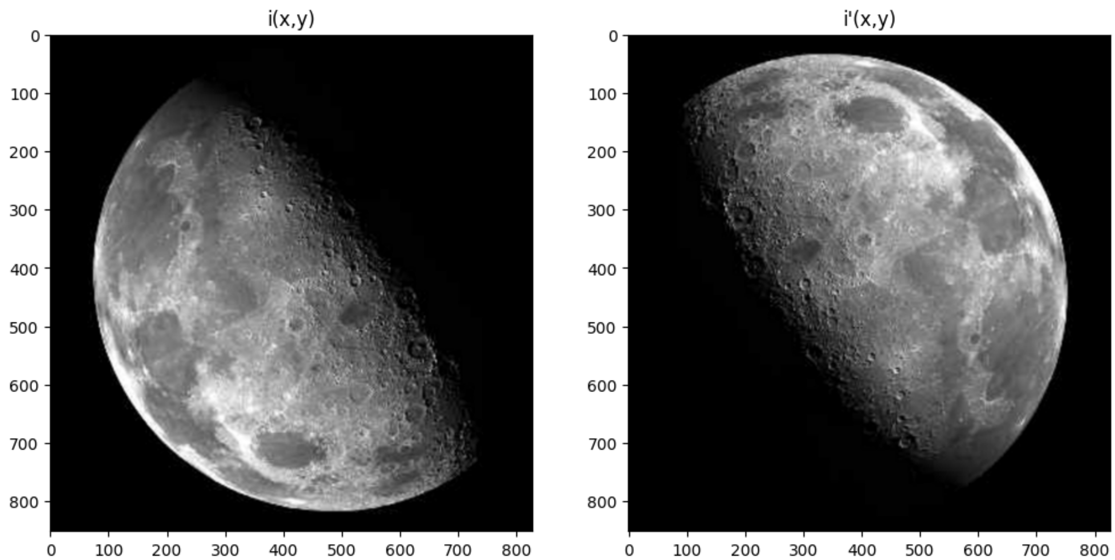
[7]:
```python
img = np.array(Image.open('./assets/moon.png').convert('L'))
height, width = img.shape
print(height, width)
```

852 827

[12]:
```python
# (-1)^(x + y)
neg_one_pow = np.array([[-1**(x + y) for x in range(width)] for y in␣
↪range(height)])

# image X (-1)^(x + y)
i_1 = img * neg_one_pow

# DFT
I_1 = np.fft.fft2(i_1)

# Conj
I_2 = np.conj(I_1)

# IDFT
i_2 = np.fft.ifft2(I_2)

# IDFT X (-1)^(x + y)
final_img = i_2 * neg_one_pow
```

[13]:
```python
plt.figure(figsize=(12,8))
ax = plt.subplot(1,2,1)
ax.imshow(img, cmap='gray')
ax.set_title('i(x,y)')

ax = plt.subplot(1,2,2)
ax.imshow(np.real(final_img), cmap='gray')
ax.set_title('i\'(x,y)')
```

```
plt.show()
```



La imagen final final_img es una versión reflejada y conjugada de la imagen original img. Específicamente, está invertida espacialmente y tiene los valores de intensidad reflejados en el sentido de conjugado complejo. Esto significa que si observas la imagen final, parecerá una versión “espejada” de la imagen original.