Compte Rendu de TP 02

Analyse et Traitement des images

Nom: BOUARICHE

Prénom: Iheb

Année : 2023/2024

Spécialité : Instrumentation an2

```python
import cv2
from matplotlib import pyplot as plt
from scipy import signal
from scipy import misc
from mpl_toolkits import mplot3d
from skimage import io
import numpy as np
```

A. Filtrage fréquentiel:

```python
image = io.imread("flowers.tif",True)
```

```python
image.dtype
```

```
dtype('float64')
```

```python
io.imshow(image)
```

```
<matplotlib.image.AxesImage at 0x7980ae980130>
```


```python
TF_image = cv2.dft(np.float64(image),flags = cv2.DFT_COMPLEX_OUTPUT)
```

```python
# La dimension de l'image
image.shape
```

```
(362, 500)
```

```python
# La dimension de la transformé de Fourier
TF_image.shape
```

```
(362, 500, 2)
```

Commentaire: On remarque qu'on a deux couches, ces deux couches representent la partie reel et la partie imaginaire, car les valeurs de pixels sont définie au domaine complexe.

```python
# La transfomé de fourier inverse
TFI_image = cv2.idft(TF_image, flags=cv2.DFT_SCALE | cv2.DFT_REAL_OUTPUT)

io.imshow(TFI_image,cmap="gray")
```

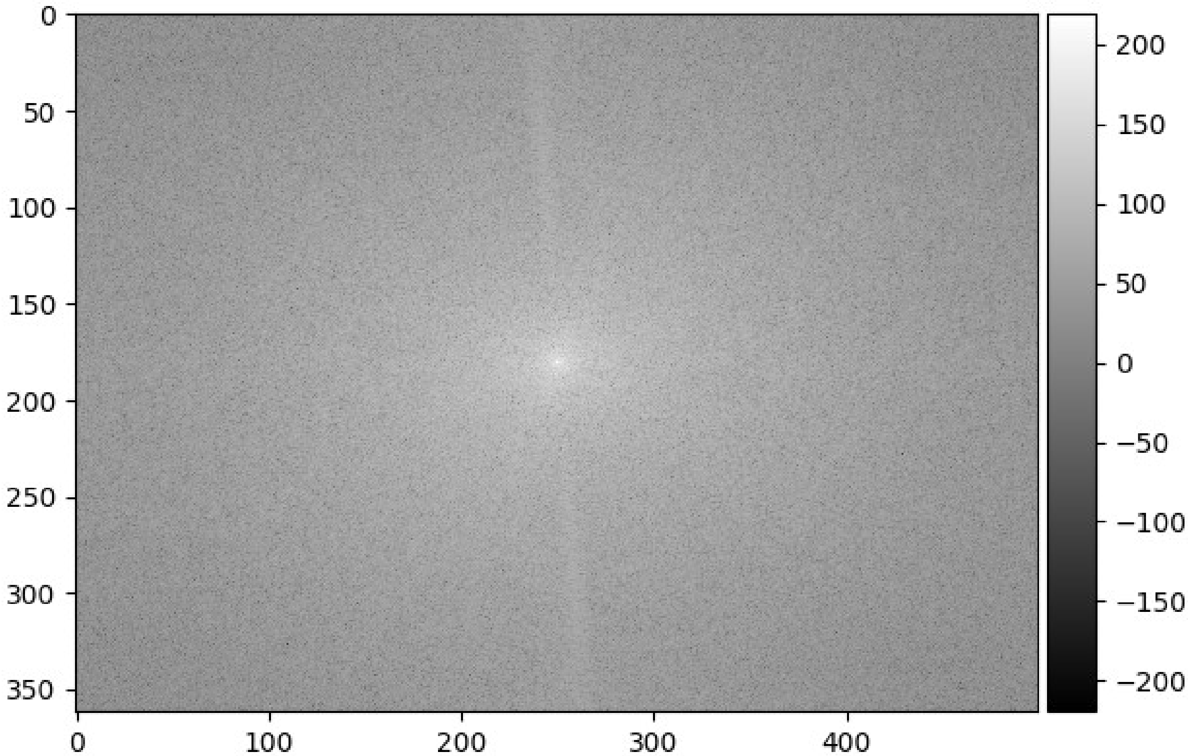
```
<matplotlib.image.AxesImage at 0x7980ae9cfe80>
```

Commentaire: on a activer l'option "cv2.DFT_REAL_OUTPUT" pour avoir seulement les sorties en réels, donc on a eu que les valeurs réels de l'image. C'est la meme image que l'original.

```python
TF_image = cv2.dft(np.float64(image), flags= cv2.DFT_COMPLEX_OUTPUT)
```

```
dft_shift = np.fft.fftshift(TF_image)

magnitude_spectrum =
20*np.log(cv2.magnitude(dft_shift[:,:,0]+1,dft_shift[:,:,1]+1))

io.imshow(magnitude_spectrum,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7980ae8ab2e0>
```



Commentaire: on remarque que le spectre a des valeurs élevés autout du centre, ça veut dire que le maximum d'informations fréquentiels sont autour du centre (0,0).

L ' application du mask:

```
magnitude_spectrum.shape
```

```
(362, 500)
```

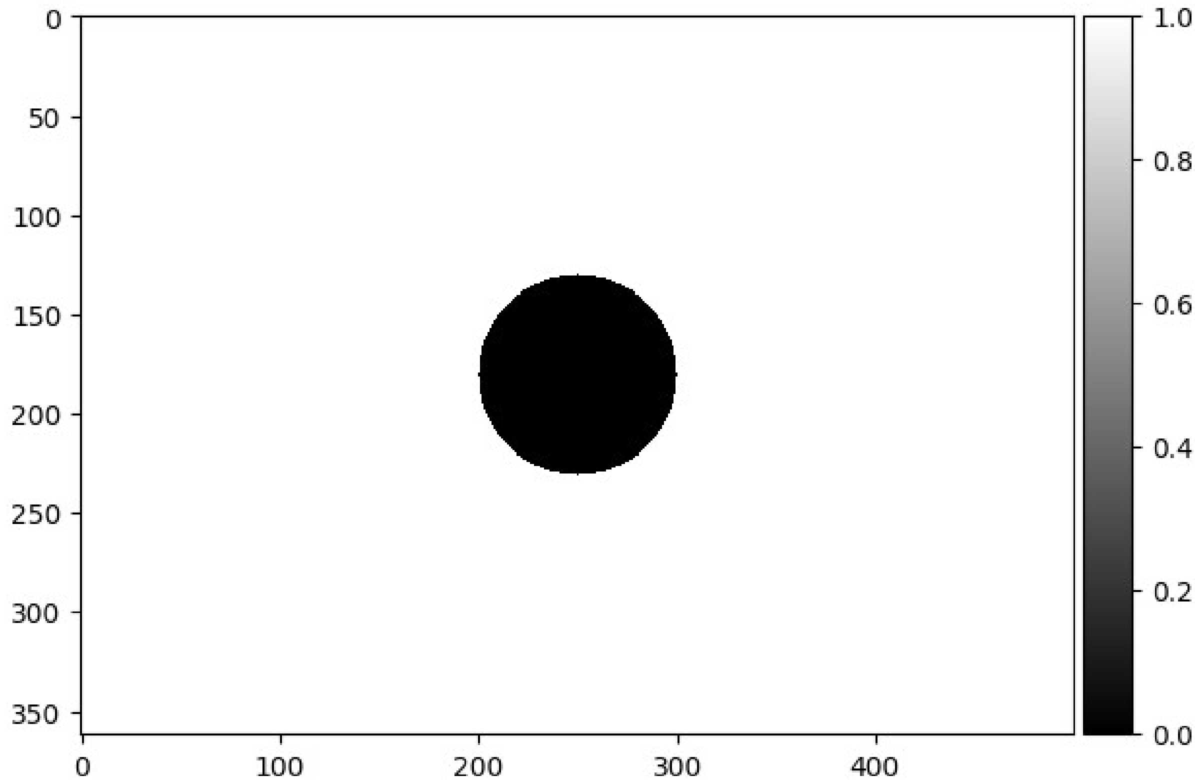
```
print("le centre du cercle du filtre est:
",magnitude_spectrum[181,250] )
```

```
le centre du cercle du filtre est:  219.5059038839273
```

```
#Mask 01
r = 50
center = [181,250]
mask = np.ones((362,500,2),np.uint8)
```

```
x,y = np.ogrid[:362,:500]
mask_area = (x-center[0])**2 + (y-center[1])**2 <= r*r
mask[mask_area] = 0

io.imshow(mask[:,:,0],cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7980ae6ac9d0>
```



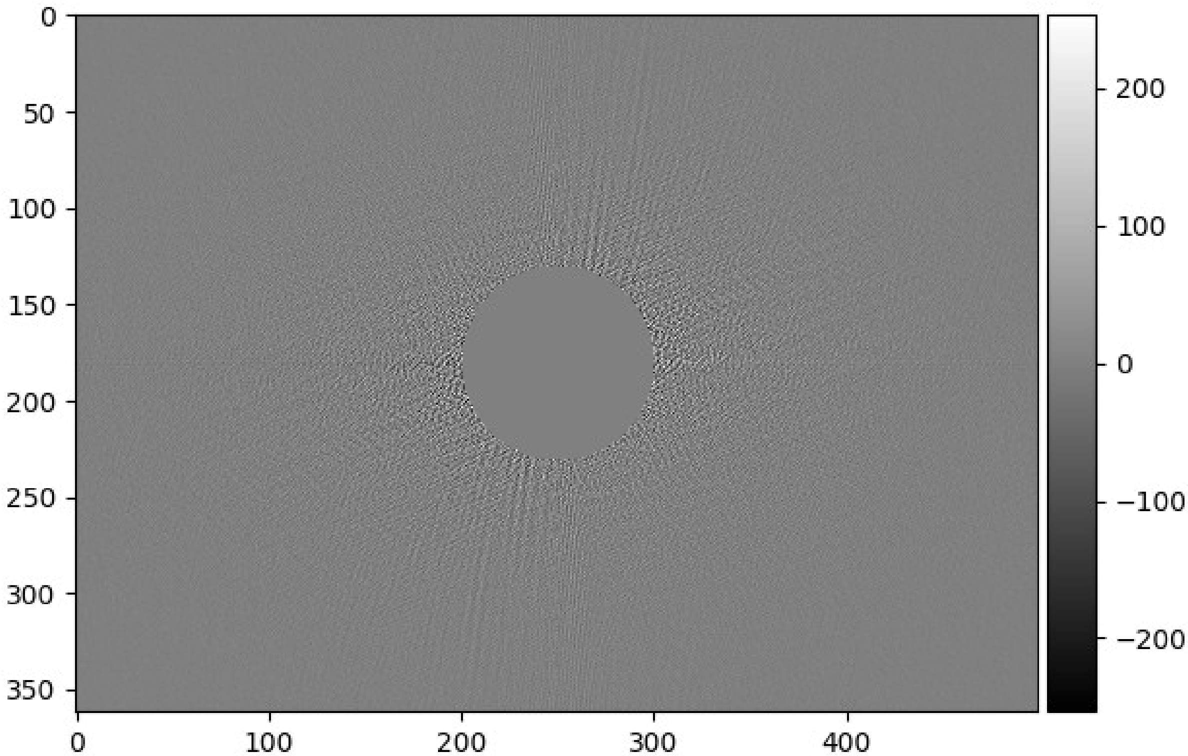
C'est un cercle centré au centre de l'image et de rayon r.

```
mask.shape
```

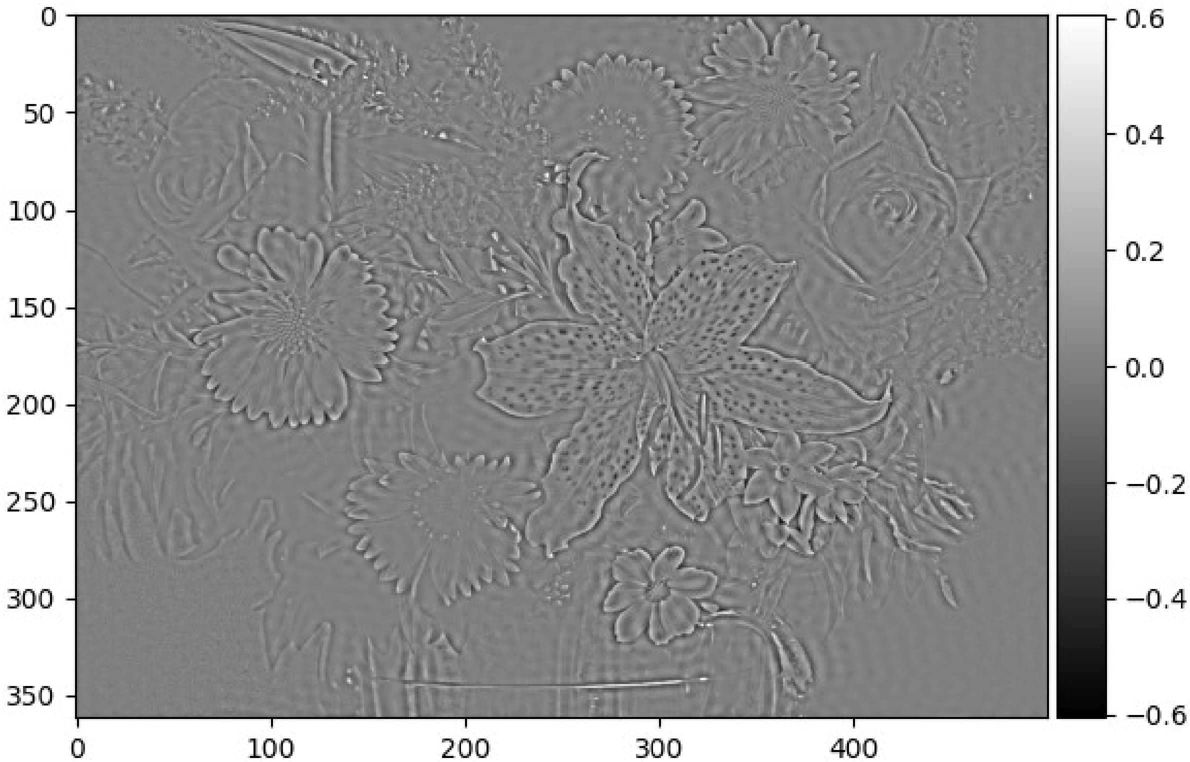
```
(362, 500, 2)
```

```
dft_image_filtré = dft_shift*mask

io.imshow(dft_image_filtré[:,:,0],cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7980ae58c190>
```



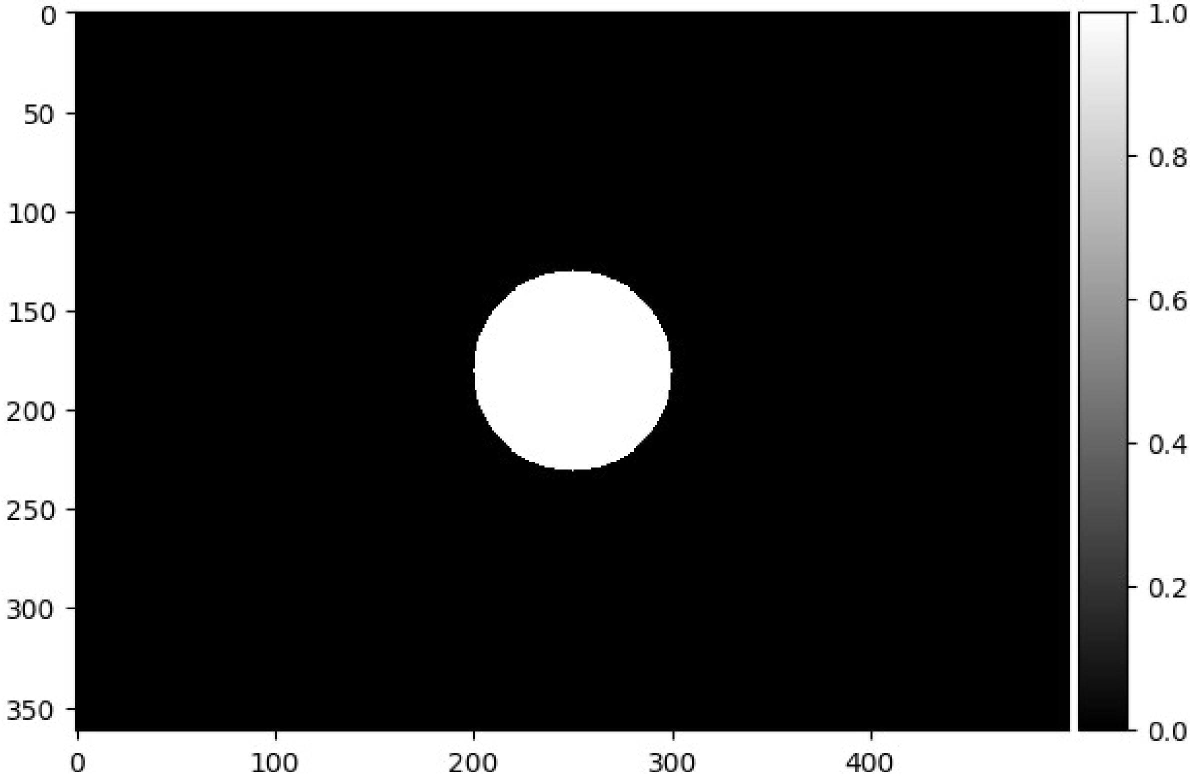

```
inverse_dft_shift = np.fft.ifftshift(dft_image_filtré)

# La transfomé de fourier inverse
TFI_image = cv2.idft(inverse_dft_shift, flags=cv2.DFT_SCALE |
cv2.DFT_REAL_OUTPUT)

io.imshow(TFI_image, cmap='gray')

<matplotlib.image.AxesImage at 0x7980ad16eec0>
```



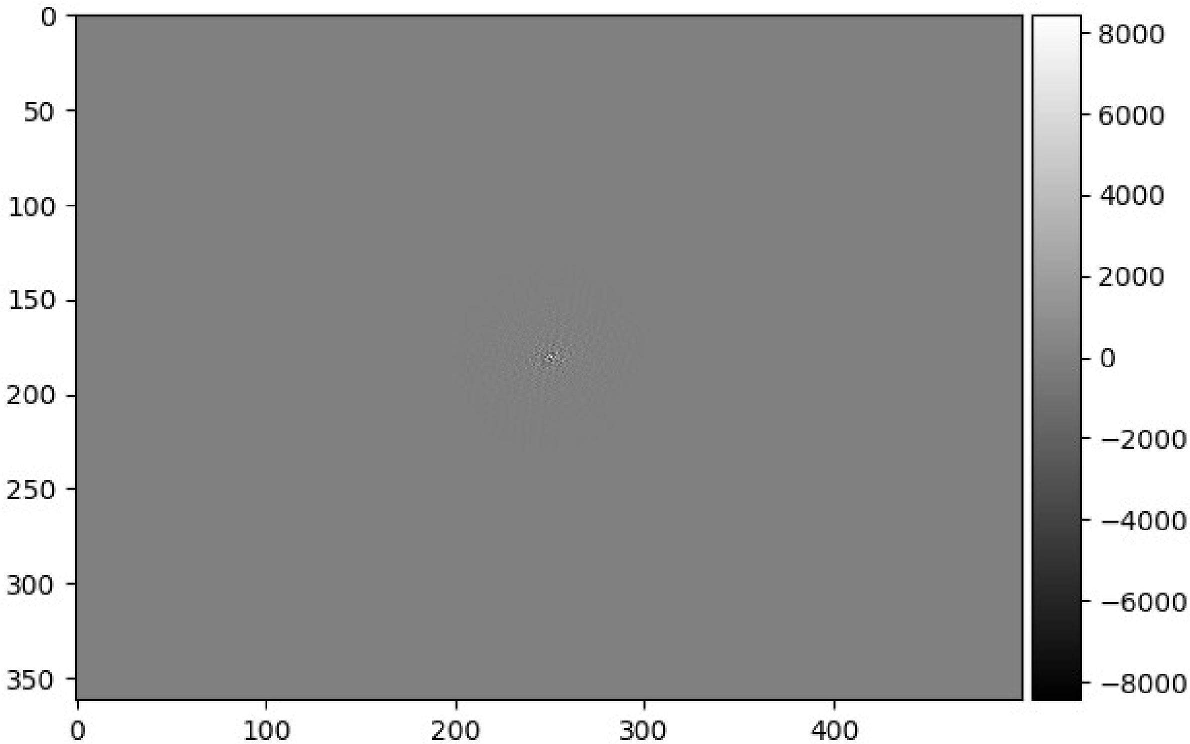
L'impact du rayon r: Si on augmente le rayon plus d'informations et details sur l'image vont étre perdus.

```
# Mask 02
r = 50
center = [181,250]
mask = np.zeros((362,500,2),np.uint8)
x,y = np.ogrid[:362,:500]
mask_area = (x-center[0])**2 + (y-center[1])**2 <= r*r
mask[mask_area] = 1

io.imshow(mask[:,:,1],cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7980ae308c70>
```

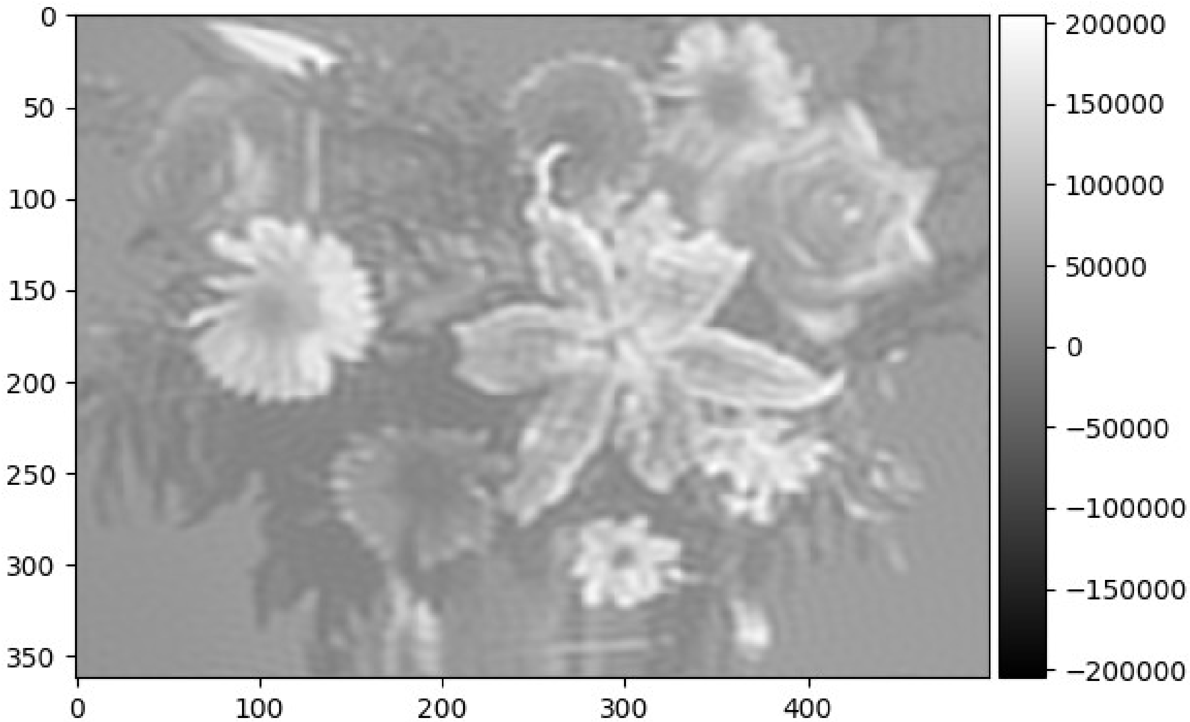



```
dft_image_filtré = dft_shift*mask

io.imshow(dft_image_filtré[:,:,0],cmap="gray")

<matplotlib.image.AxesImage at 0x7980addd6aa0>
```




```
inverse_dft_shift = np.fft.ifftshift(dft_image_filtré)

# La transfomé de fourier inverse
TFI_image = cv2.idft(inverse_dft_shift, flags= cv2.DFT_REAL_OUTPUT)

io.imshow(TFI_image, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x7980adcb3460>
```



Interpretation: On remarque que si on supprime les informations fréquentielles au centre du spectre, vont supprimer une grande partie de details dans l'image. au contraire si on garde seulement le cercle d'informations fréquentielles au centre du spectre, les details de l'image ne vont pas etre perdu.

B. Filtrage Spatial:

Exercice 01:

```
S1 = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]])

S2 = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])
```

1. On constate que S2 represente une rotation de S1 par 90 degrée.

- S1(u) est un filtre passe bas tandis que S1(v) est un filtre passe haut.
- S2(u) est un filtre passe haut tandis que S2(v) est un filtre passe bas.

```
S11 = [[1], [2], [1]]
S12 = [[1], [0], [-1]]
S21 = [[1], [0], [-1]]
S22 = [[1], [2], [1]]

_,TFS11 = signal.freqz(S11)
_,TFS12 = signal.freqz(S12)
```

On commence par la séparation du premier filtre :

$$S1 = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Apres on fait la transformé de fourrier :

$$TF\left(\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}\right) = e^{j2\pi u} + 2 - e^{-j2\pi u}$$

$$TF(\begin{bmatrix} 1 & 0 & -1 \end{bmatrix}) = e^{j2\pi v} - e^{-j2\pi v}$$

$$\boldsymbol{TF(S1) \ = 4jsin(2\pi v)(1 + \cos(2\pi u)}$$

La même chose pour la séparation du deuxième filtre :

$$S1 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Apres on fait la transformé de fourrier :

$$TF\left(\begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}\right) = e^{j2\pi v} - e^{-j2\pi v}$$

$$TF(\begin{bmatrix} 1 & 2 & 1 \end{bmatrix}) = e^{j2\pi u} + 2 - e^{-j2\pi u}$$

$$\boldsymbol{TF(S2) \ = 4jsin(2\pi v)(1 + \cos(2\pi u)}$$

```
_,TFS21 = signal.freqz(S21)
_,TFS22 = signal.freqz(S22)
```

Le cas de filtre S1

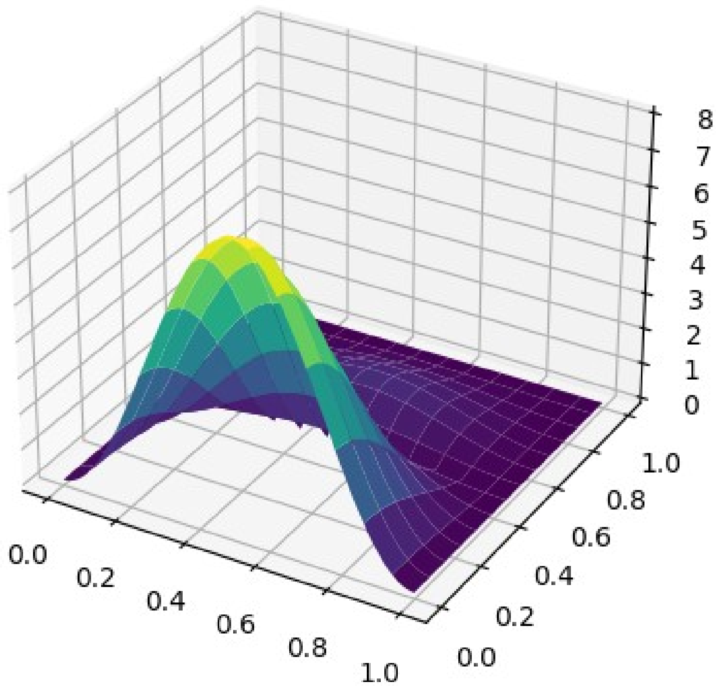
```
Z1 = abs(np.float64(TFS11).reshape(-1,1)*np.float64(TFS12))
```

```
<ipython-input-276-3185a0a62a4d>:1: ComplexWarning: Casting complex
values to real discards the imaginary part
  Z1 = abs(np.float64(TFS11).reshape(-1,1)*np.float64(TFS12))
```

```
Z1.shape
```

```
(512, 512)
```

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming your matrix is called 'matrix'
matrix = np.random.rand(512, 512)  # Replace this with your actual
data

# Create mesh grid
x, y = np.meshgrid(np.arange(512)/512, np.arange(512)/512)

# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Flatten the matrix to 1D arrays and plot
ax.plot_surface(x, y, Z1, rstride = 40, cstride = 40, cmap =
"viridis", edgecolor = "none")

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



Le filtre est un filtre passe haut suivant l'axe x et un filtre passe haut suivant l'axe y.

Le cas de filtre S2

```
Z2 = abs(np.float64(TFS21).reshape(-1,1)*np.float64(TFS22))
```

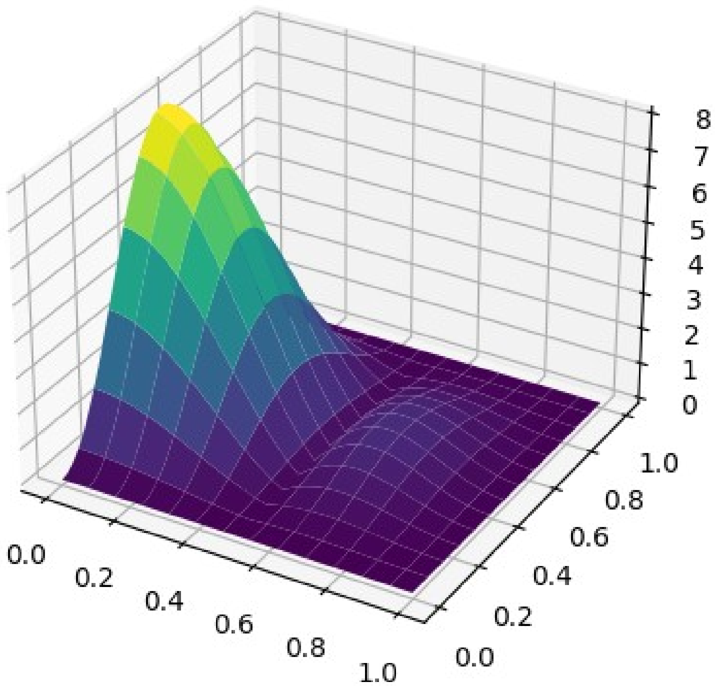
```
<ipython-input-293-dcb0cce2cc39>:1: ComplexWarning: Casting complex
values to real discards the imaginary part
  Z2 = abs(np.float64(TFS21).reshape(-1,1)*np.float64(TFS22))
```

```
Z2.shape
```

```
(512, 512)
```

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming your matrix is called 'matrix'
matrix = np.random.rand(512, 512)  # Replace this with your actual
data

# Create mesh grid
x, y = np.meshgrid(np.arange(512)/512, np.arange(512)/512)

# Create a figure and a 3D axis
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
```

```
# Flatten the matrix to 1D arrays and plot
ax.plot_surface(x, y, Z2, rstride = 40, cstride = 40, cmap =
"viridis", edgecolor = "none")

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



Le filtre est un filtre passe haut suivant l'axe y et un filtre passe haut suivant l'axe x.

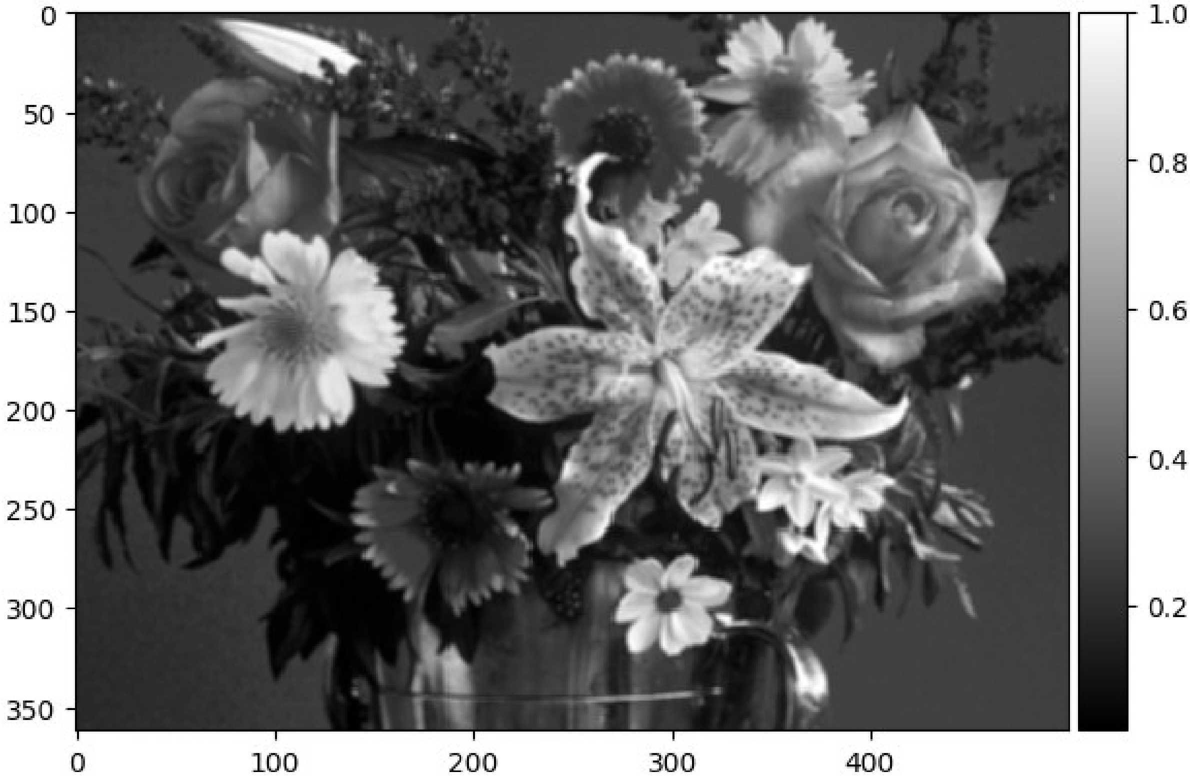
Ce filtre represente une rotation de 90 degrée par rapport au filtre précédent. Ce qui confime la réponse de la question 3.

Exercice 02

```
image = io.imread("flowers.tif",True)

K1 = np.array([[0,0,0],[0,1,0],[0,0,0]])

Filtred_image = cv2.filter2D(src = image,ddepth=-1,kernel=K1)

io.imshow(Filtred_image)

<matplotlib.image.AxesImage at 0x7980aae5a590>
```


```
K2 = (1/9)*(np.array([[1,1,1],[1,1,1],[1,1,1]]))

Filtred_image = cv2.filter2D(src = image,ddepth=-1,kernel=K2)

io.imshow(Filtred_image,cmap="gray")
```

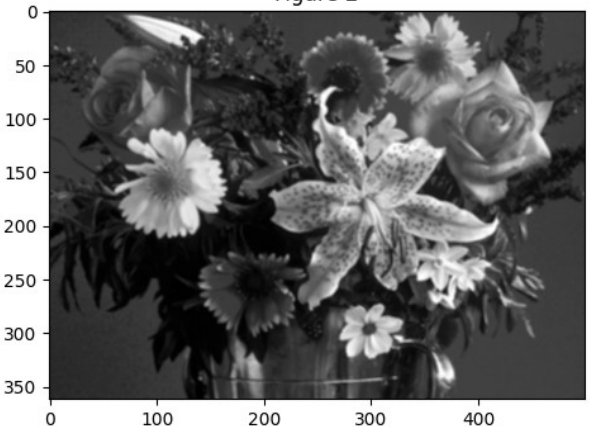
```
/usr/local/lib/python3.10/dist-packages/skimage/io/_plugins/
matplotlib_plugin.py:150: UserWarning: Float image out of standard
range; displaying image with stretched contrast.
  lo, hi, cmap = _get_display_range(image)

<matplotlib.image.AxesImage at 0x7980aab90d00>
```




```
plt.figure(figsize=(12, 6))

# Première sous-fenêtre
plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Figure 1')

# Deuxième sous-fenêtre
plt.subplot(1, 2, 2)
plt.imshow(Filtred_image, cmap='gray')
plt.title('Figure 2')

plt.show()
```

Figure 1

Figure 2



Comentaire: On remarque que l'image filtré est un peu flou par rapport a l'image original.

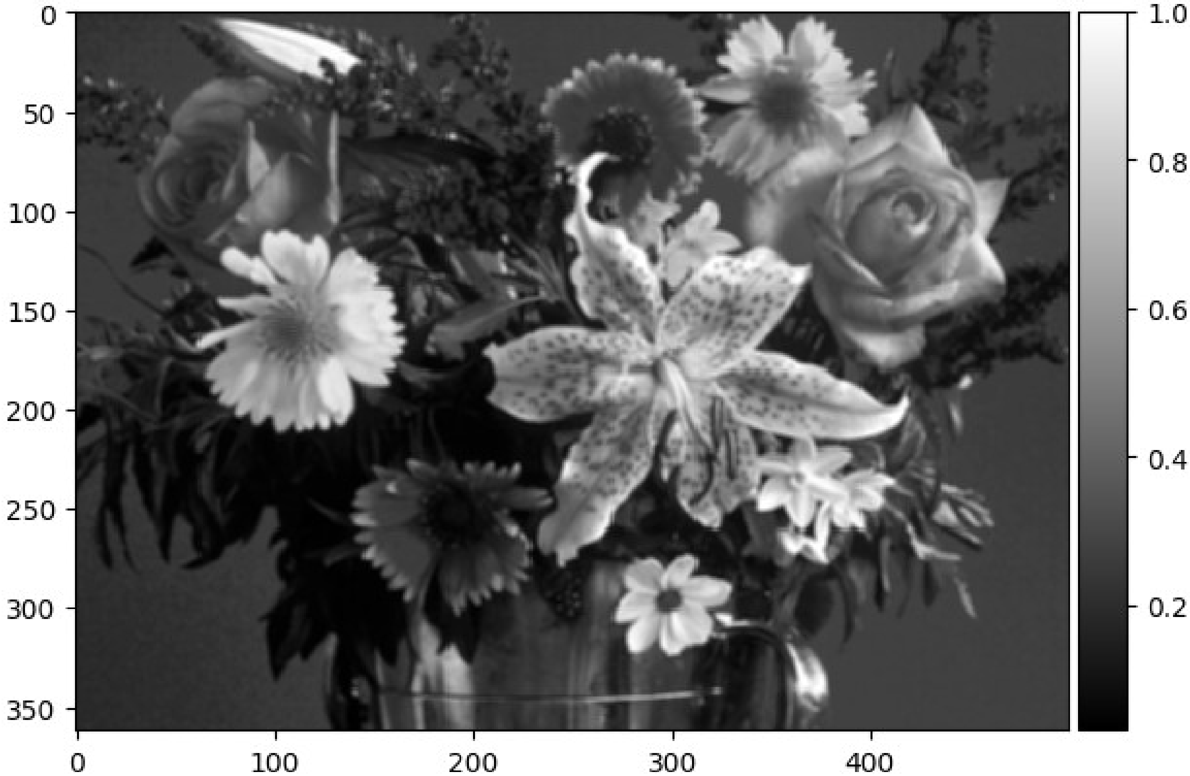
```
K2 = (1/9)*(np.array([[1,1,1,1,1],[1,1,1,1,1],[1,1,1,1,1],[1,1,1,1,1],
[1,1,1,1,1]]))
Filtred_image1 = cv2.filter2D(src = image,ddepth=-1,kernel=K2)
io.imshow(Filtred_image,cmap="gray")
```

```
/usr/local/lib/python3.10/dist-packages/skimage/io/_plugins/
matplotlib_plugin.py:150: UserWarning: Float image out of standard
range; displaying image with stretched contrast.
  lo, hi, cmap = _get_display_range(image)
```

```
<matplotlib.image.AxesImage at 0x7980aab1df00>
```


```
K2 = (1/9)*(np.ones(7))
Filtred_image2 = cv2.filter2D(src = image,ddepth=-1,kernel=K2)
io.imshow(Filtred_image,cmap="gray")
```

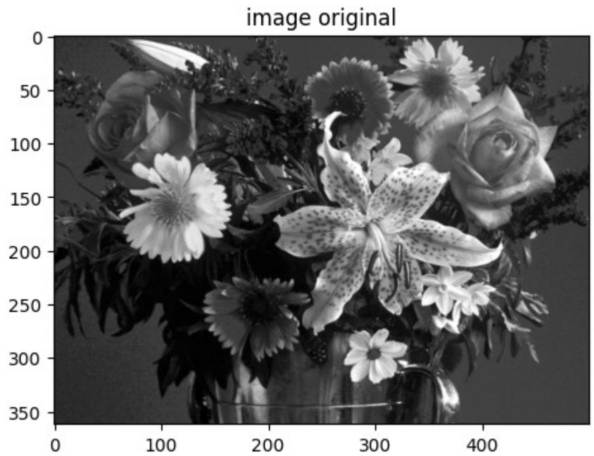
```
<matplotlib.image.AxesImage at 0x7980aaa0c0a0>
```

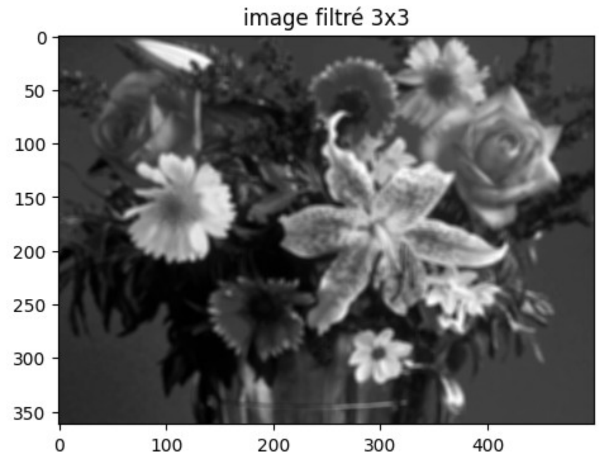


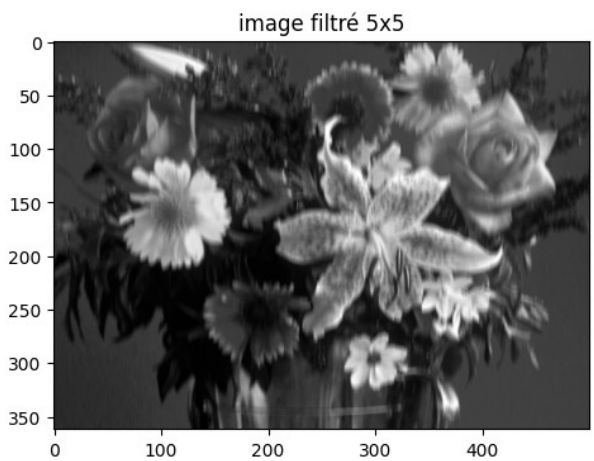

```
K2 = (1/9)*(np.ones(9))
Filtred_image3 = cv2.filter2D(src = image,ddepth=-1,kernel=K2)
io.imshow(Filtred_image,cmap="gray")
```

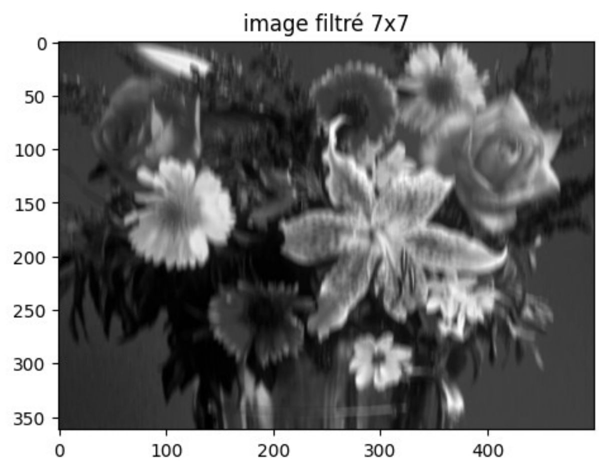
```
<matplotlib.image.AxesImage at 0x7980aaab5000>
```


```
plt.figure(figsize=(12, 12))

# Première sous-fenêtre
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('image original')

# Deuxième sous-fenêtre
plt.subplot(2, 2, 2)
plt.imshow(Filtred_image1, cmap='gray')
plt.title('image filtré 3x3')

# Troisieme sous-fenêtre
plt.subplot(2, 2, 3)
plt.imshow(Filtred_image2, cmap='gray')
plt.title('image filtré 5x5')

# Quatrieme sous-fenêtre
plt.subplot(2, 2, 4)
plt.imshow(Filtred_image3, cmap='gray')
plt.title('image filtré 7x7')
plt.show()
```





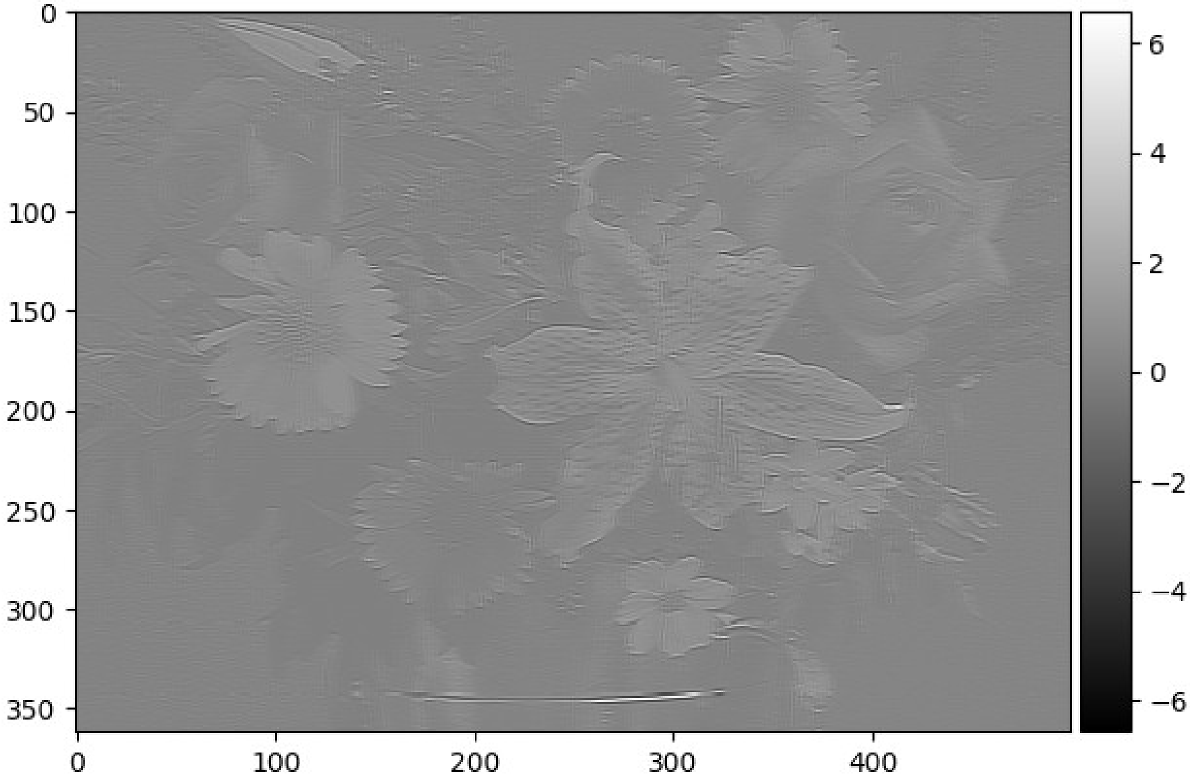




On remarque que lorsqu'on augmente la taille de ce filtre, l'image devient plus flou et perdre des details.

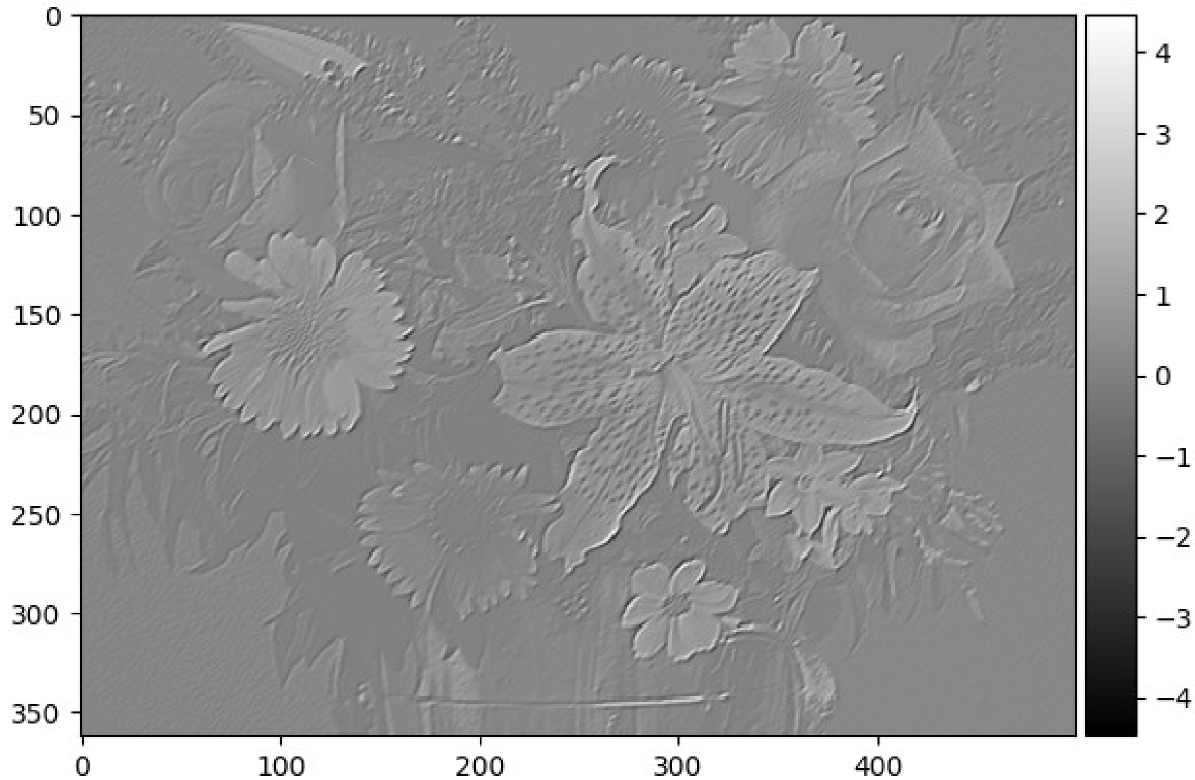
```
K3 = np.array([[0,-1,0],[1,1,1],[0,-1,0]])
K4 = np.array([[0,-1,0],[-1,1,1],[0,1,0]])

Filtred_image = cv2.filter2D(src = image,ddepth=-1,kernel=K3)
Filtred_image =
signal.convolve2d(np.float32(Filtred_image),K3,mode="same",boundary="f
ill")
io.imshow(Filtred_image,cmap="gray")
```

```
<matplotlib.image.AxesImage at 0x7980ad7d96f0>
```




```
Filtred_image = cv2.filter2D(src = image,ddepth=-1,kernel=K4)
Filtred_image =
signal.convolve2d(np.float32(Filtred_image),K4,mode="same",boundary="f
ill")
io.imshow(Filtred_image,cmap="gray")
```

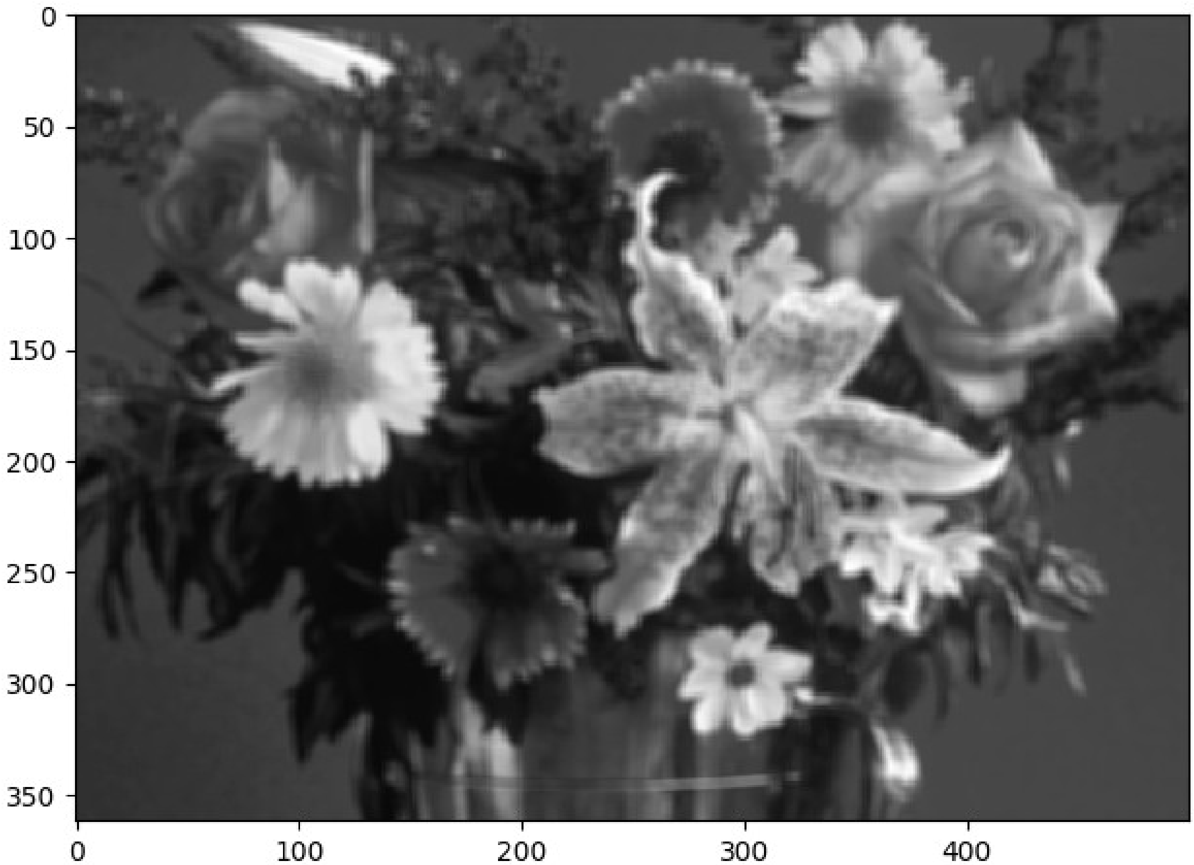
```
<matplotlib.image.AxesImage at 0x7980ad6b6110>
```



Les deux filtres ont un effet sur l'image et l'image filtré perds des details. Mais on peut dire que le deuxieme filtre et mieux que le premier car on peut mieux regarder l'image.

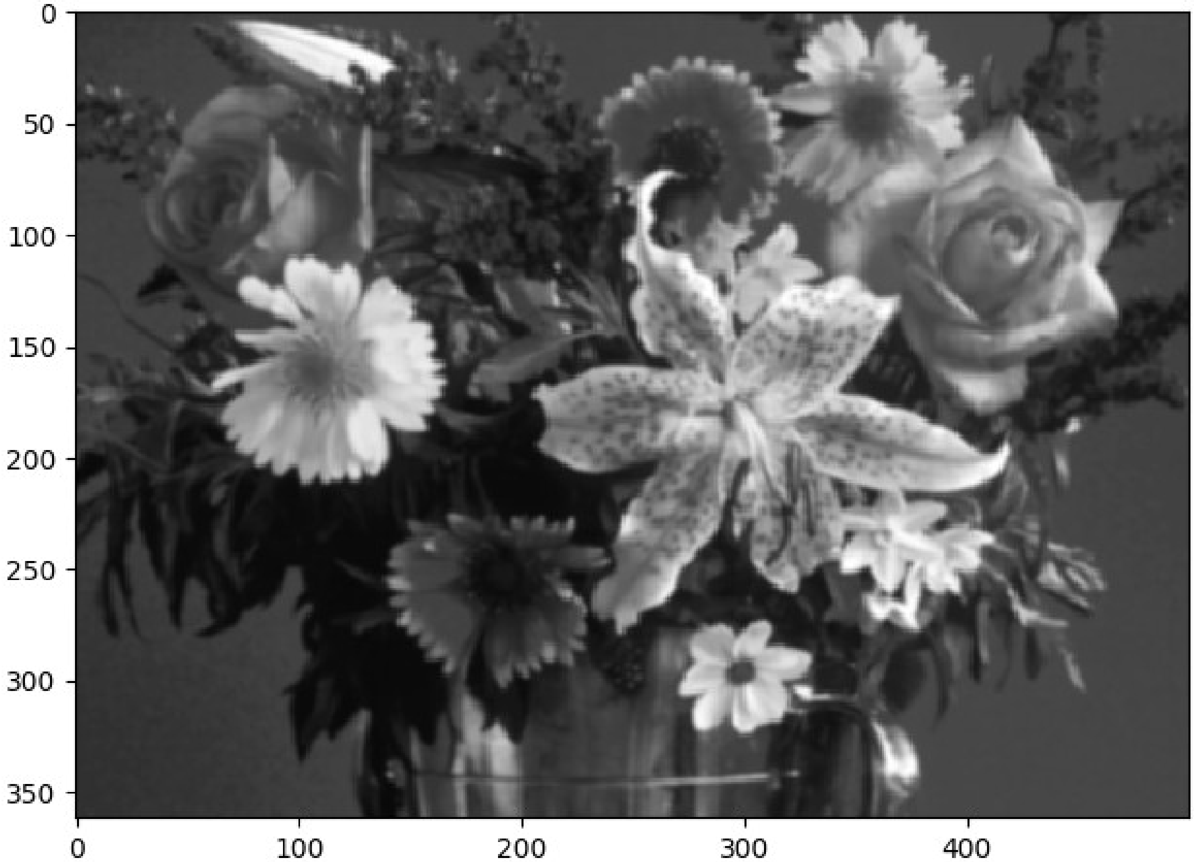
```
image1 = cv2.blur(src=image,ksize=(5,5))
image2 = cv2.GaussianBlur(src=image,ksize=(5,5),sigmaX=0,sigmaY=0)
image3 = cv2.medianBlur(src=np.float32(image), ksize=5)
image4 =
cv2.bilateralFilter(src=np.float32(image),d=9,sigmaColor=75,sigmaSpace
=75)

io.imshow(image1)
```

```
<matplotlib.image.AxesImage at 0x7980ad59a8f0>
```



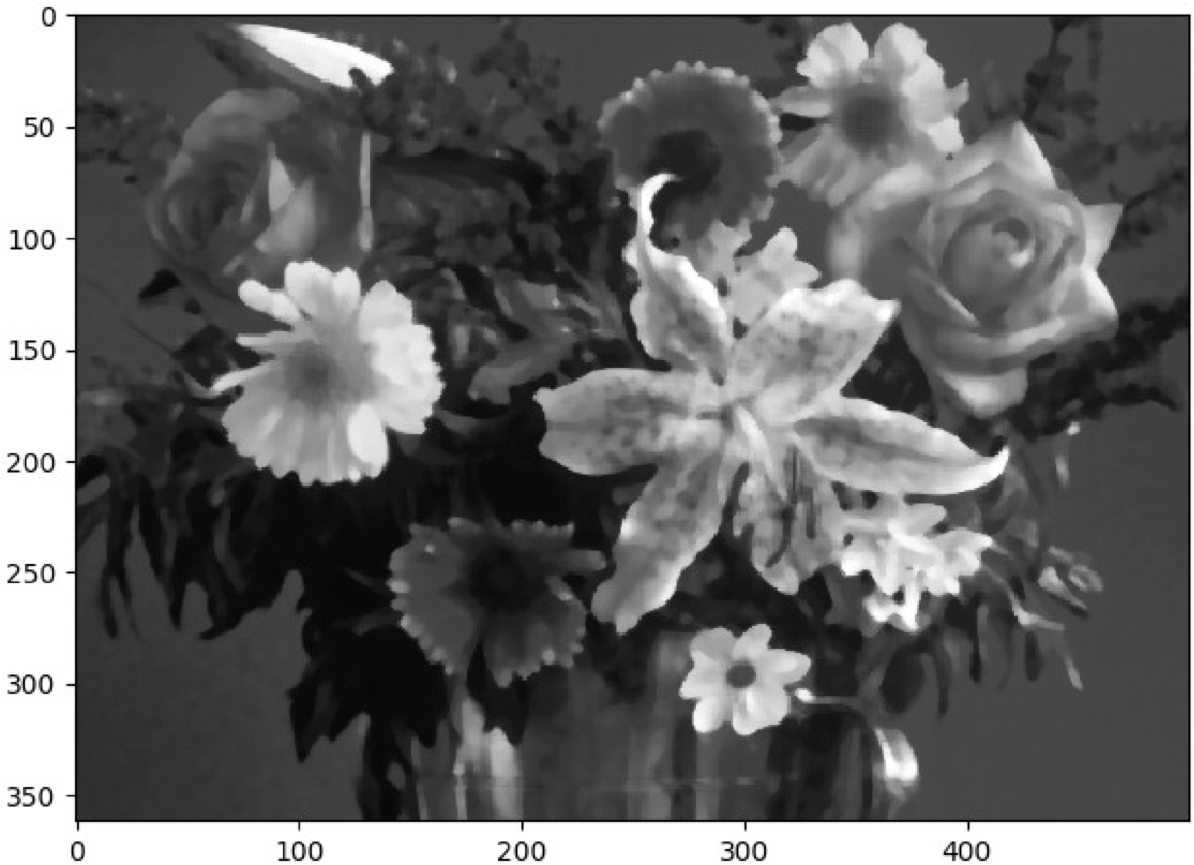

```
io.imshow(image2)
```

```
<matplotlib.image.AxesImage at 0x7980ad601de0>
```



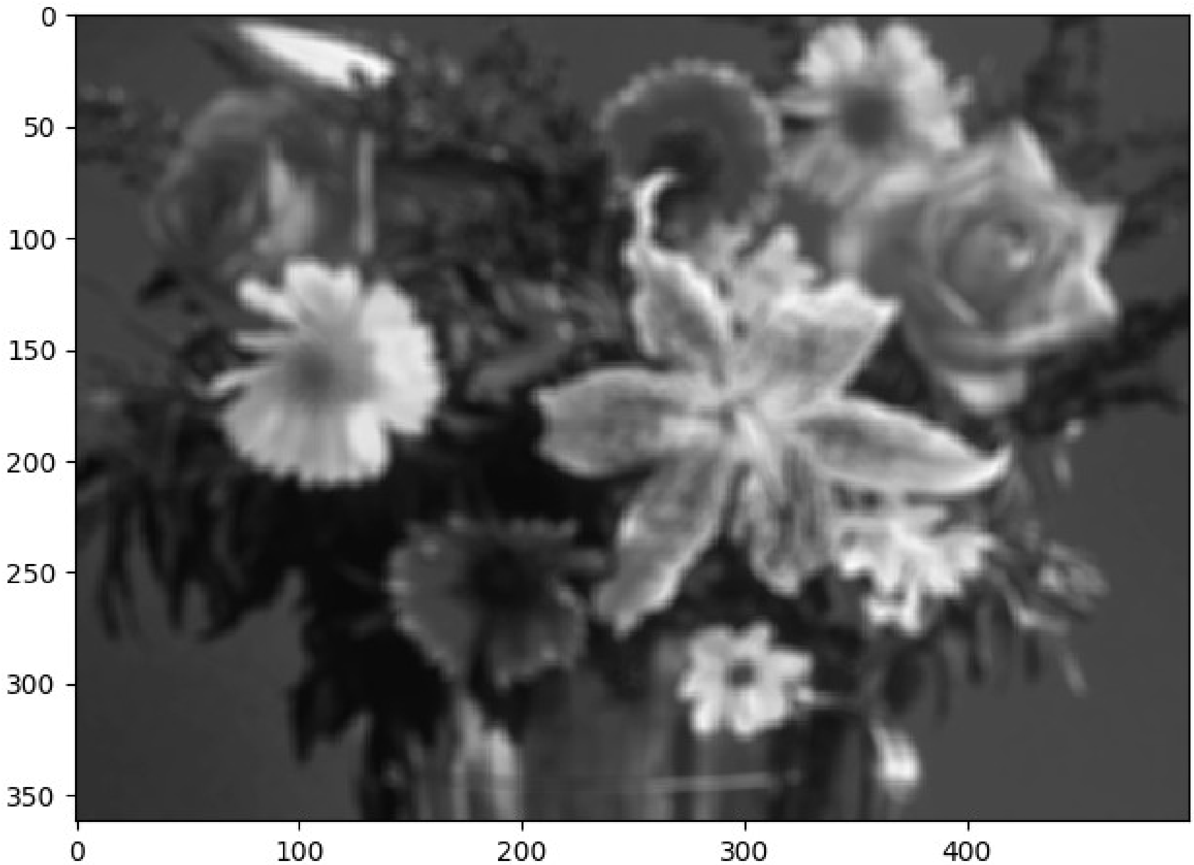

```
io.imshow(image3)
```

```
<matplotlib.image.AxesImage at 0x7980ad48db70>
```



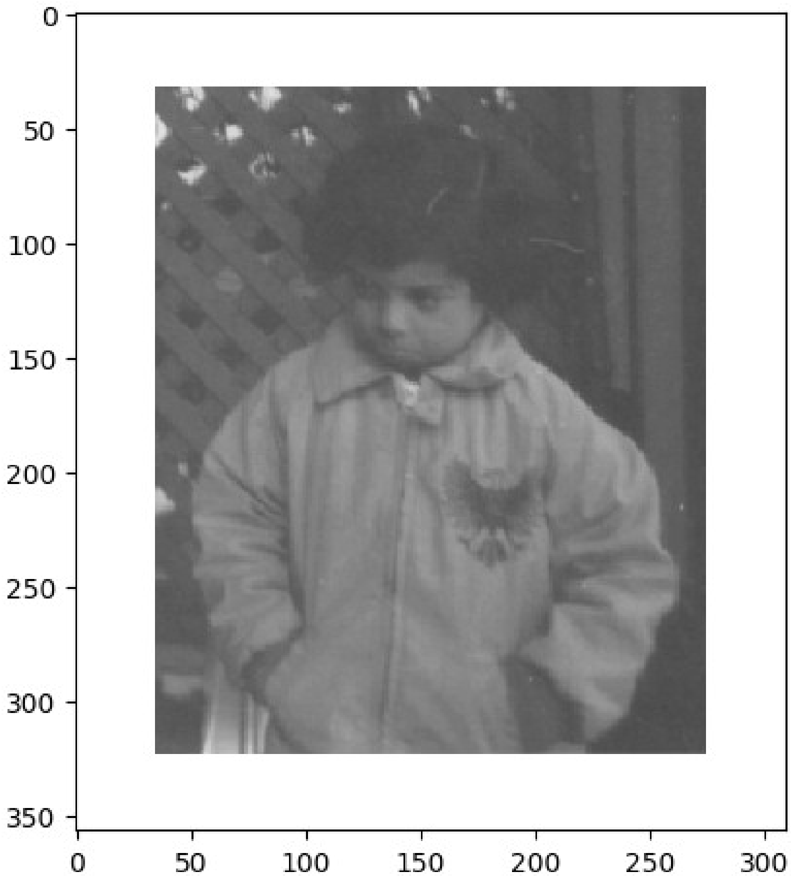

```
io.imshow(image4)
```

```
<matplotlib.image.AxesImage at 0x7980ad510e50>
```



C. Rehaussement de contrast:

```
poupee = cv2.imread("poupee.tif", cv2.IMREAD_GRAYSCALE)

io.imshow(poupee)

<matplotlib.image.AxesImage at 0x7980ad390f10>
```




```
#Calcul de contraste
import cv2
import numpy as np

def michelson_contrast(image):
    min_intensity = np.min(image)
    max_intensity = np.max(image)
    return (max_intensity - min_intensity) / (max_intensity +
min_intensity)

michelson = michelson_contrast(poupee)
print("Michelson Contrast:", michelson)

Michelson Contrast: 2.4794520547945207

<ipython-input-244-509d877fca37>:7: RuntimeWarning: overflow
encountered in ubyte_scalars
  return (max_intensity - min_intensity) / (max_intensity +
min_intensity)

import cv2
import numpy as np
```

```python
def global_contrast(image):
    mean_intensity = np.mean(image)
    std_deviation = np.std(image)
    return std_deviation / mean_intensity

global_contrast_val = global_contrast(poupee)
print("Global Contrast:", global_contrast_val)
```

```
Global Contrast: 0.4411218008780237
```

```python
import cv2
import numpy as np

def rms_contrast(image):
    mean_intensity = np.mean(image)
    squared_diff = np.square(image - mean_intensity)
    mean_squared_diff = np.mean(squared_diff)
    return np.sqrt(mean_squared_diff)

rms_contrast_val = rms_contrast(poupee)
print("Root Mean Squared Contrast:", rms_contrast_val)
```

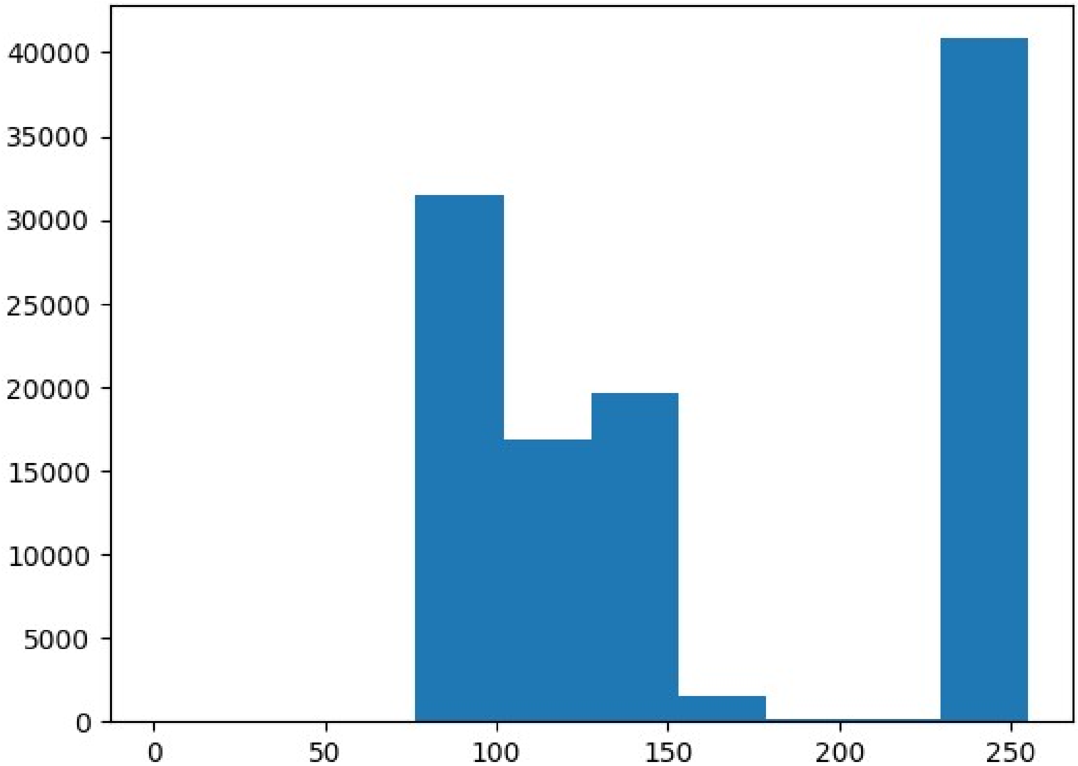
```
Root Mean Squared Contrast: 72.20598278281041
```

```python
import cv2
import numpy as np

def calculate_luminance(image):
    average_intensity = np.mean(image)
    return average_intensity

luminance = calculate_luminance(poupee)
print("Luminance:", luminance)
```

```
Luminance: 163.68717809704526
```

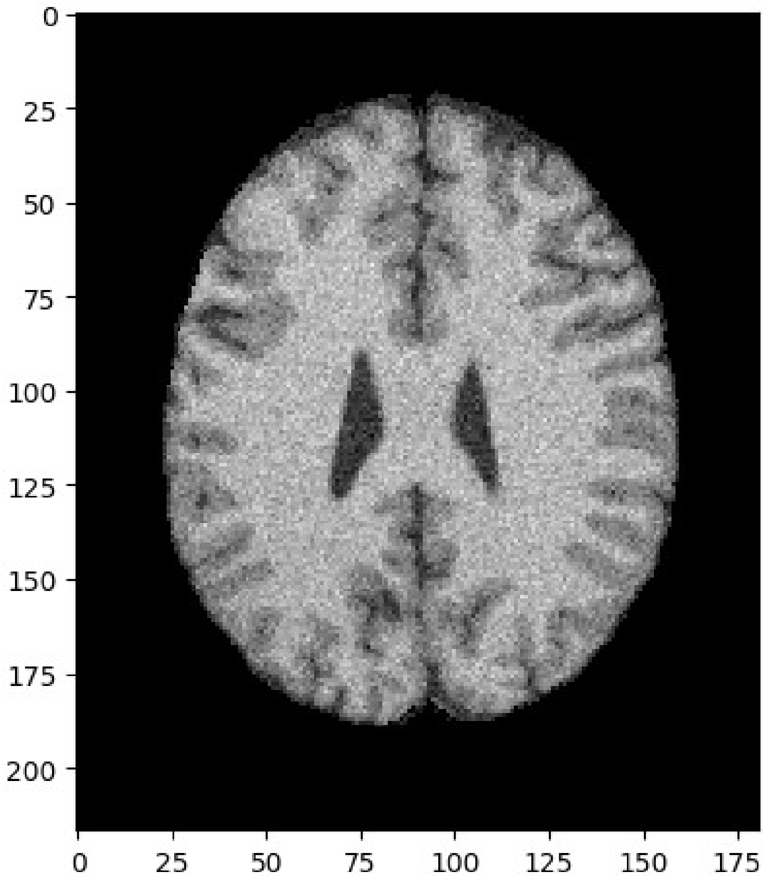
```python
plt.hist(poupee.flat, bins=10, range=(0,255))
```

```
(array([0.0000e+00, 0.0000e+00, 4.0000e+01, 3.1418e+04, 1.6905e+04,
        1.9593e+04, 1.5840e+03, 1.8800e+02, 1.1200e+02, 4.0830e+04]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <BarContainer object of 10 artists>)
```



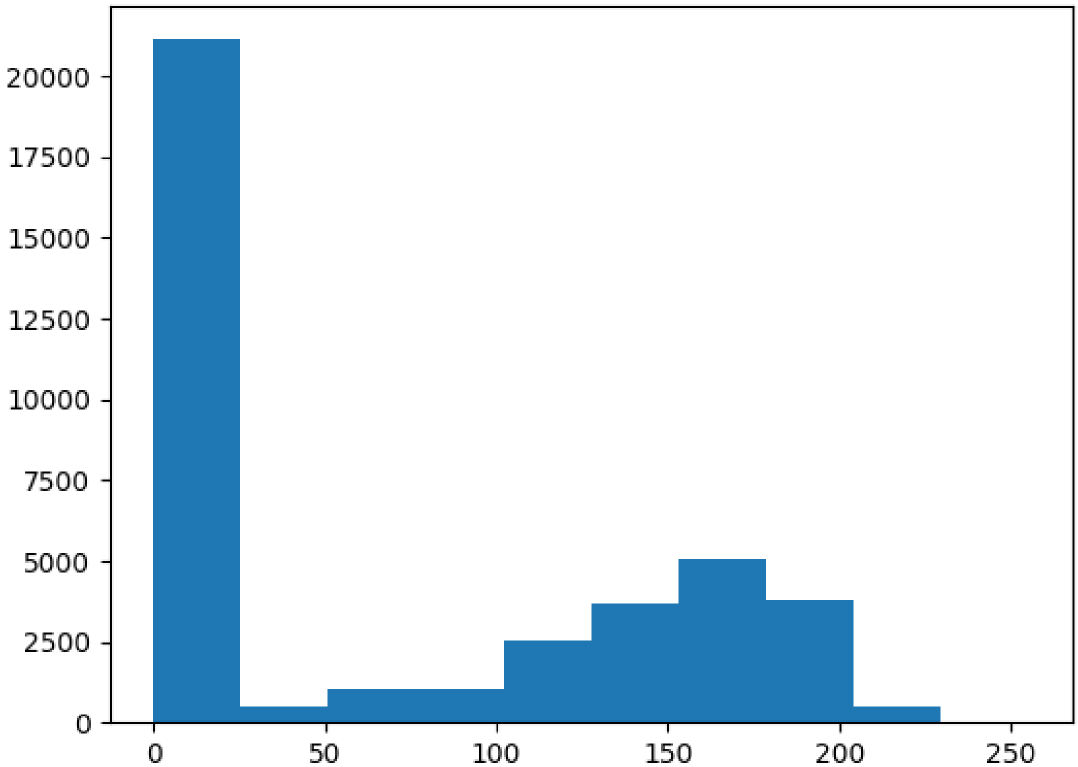

```
crane = io.imread("sanscranetest100bruit9%.png",True)
io.imshow(crane)
```

```
<matplotlib.image.AxesImage at 0x7980ac27b5b0>
```



```
michelson = michelson_contrast(crane)
print("Michelson Contrast:", michelson)

Michelson Contrast: 1.0

global_contrast_val = global_contrast(crane)
print("Global Contrast:", global_contrast_val)

Global Contrast: 1.1530750024227125

rms_contrast_val = rms_contrast(crane)
print("Root Mean Squared Contrast:", rms_contrast_val)

Root Mean Squared Contrast: 78.10431047420921

luminance = calculate_luminance(crane)
print("Luminance:", luminance)

Luminance: 67.73567227639586

plt.hist(crane.flat, bins=10, range=(0,255))

(array([2.1124e+04, 5.1000e+02, 1.0250e+03, 1.0070e+03, 2.5480e+03,
        3.6750e+03, 5.0870e+03, 3.7710e+03, 5.2400e+02, 6.0000e+00]),
```

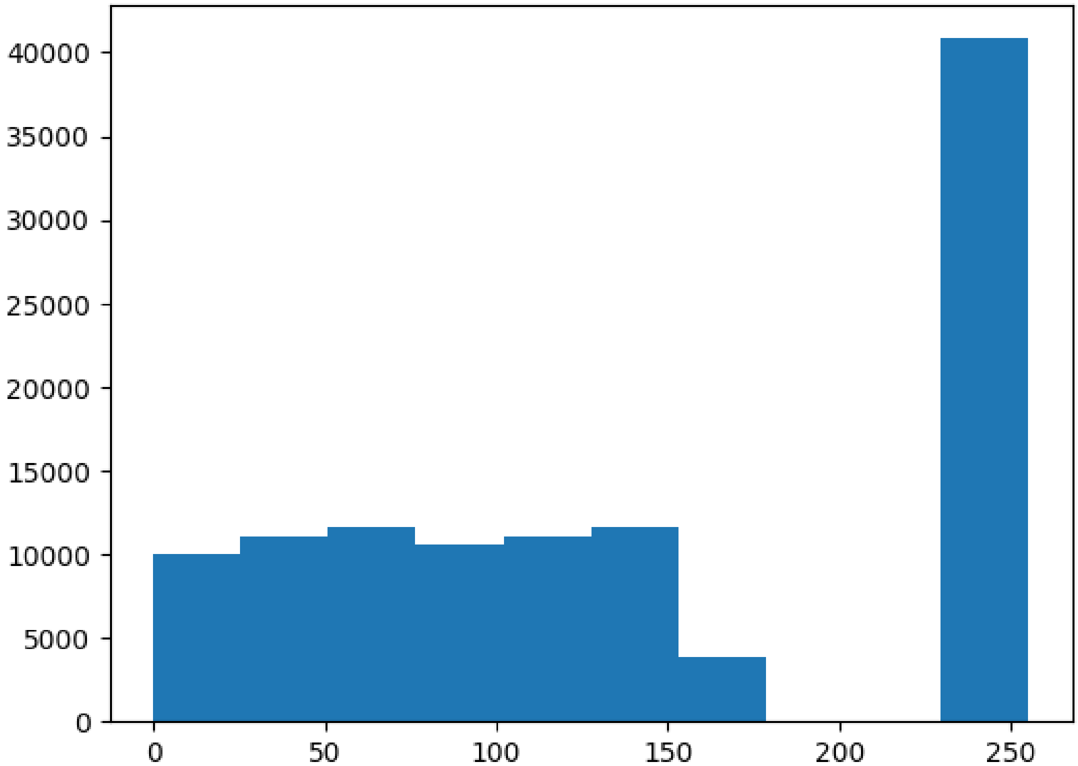
```
array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
       229.5, 255. ]),
 <BarContainer object of 10 artists>)
```



4. Egalisation d'histogramme

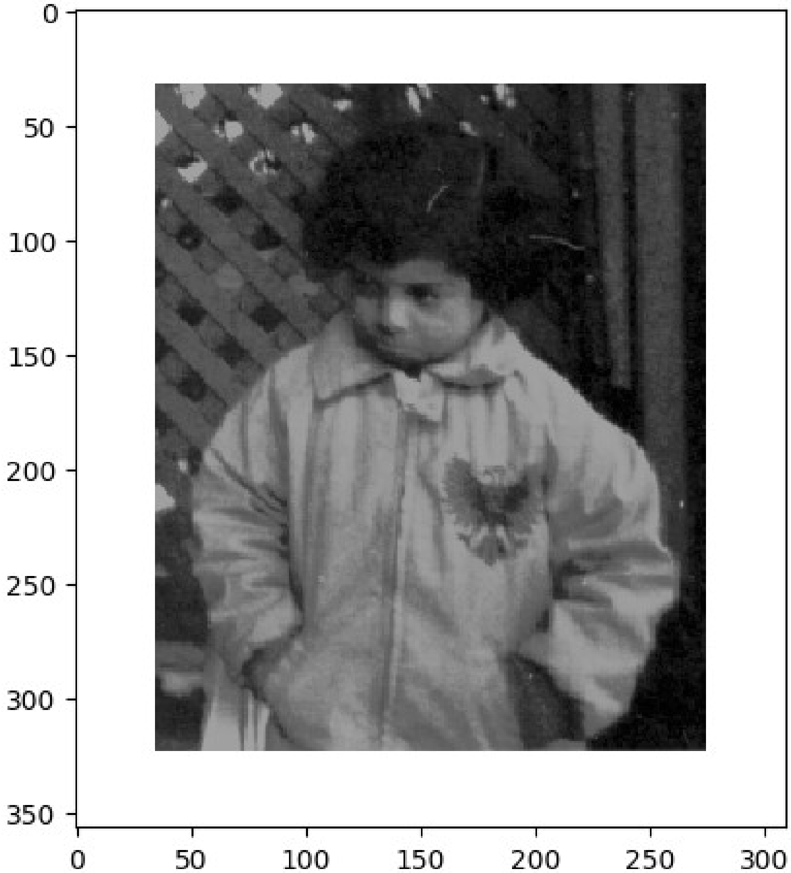
L'égalisation d'histogramme est une technique de traitement d'image qui ajuste la répartition des niveaux de gris dans une image pour améliorer son contraste et sa visualisation.

```
poupee_eq= cv2.equalizeHist(poupee)

plt.hist(poupee_eq.flat, bins=10, range=(0,255))

(array([ 9966., 11060., 11650., 10545., 11089., 11676.,  3854.,
0.,
            0., 40830.]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
       229.5, 255. ]),
 <BarContainer object of 10 artists>)
```

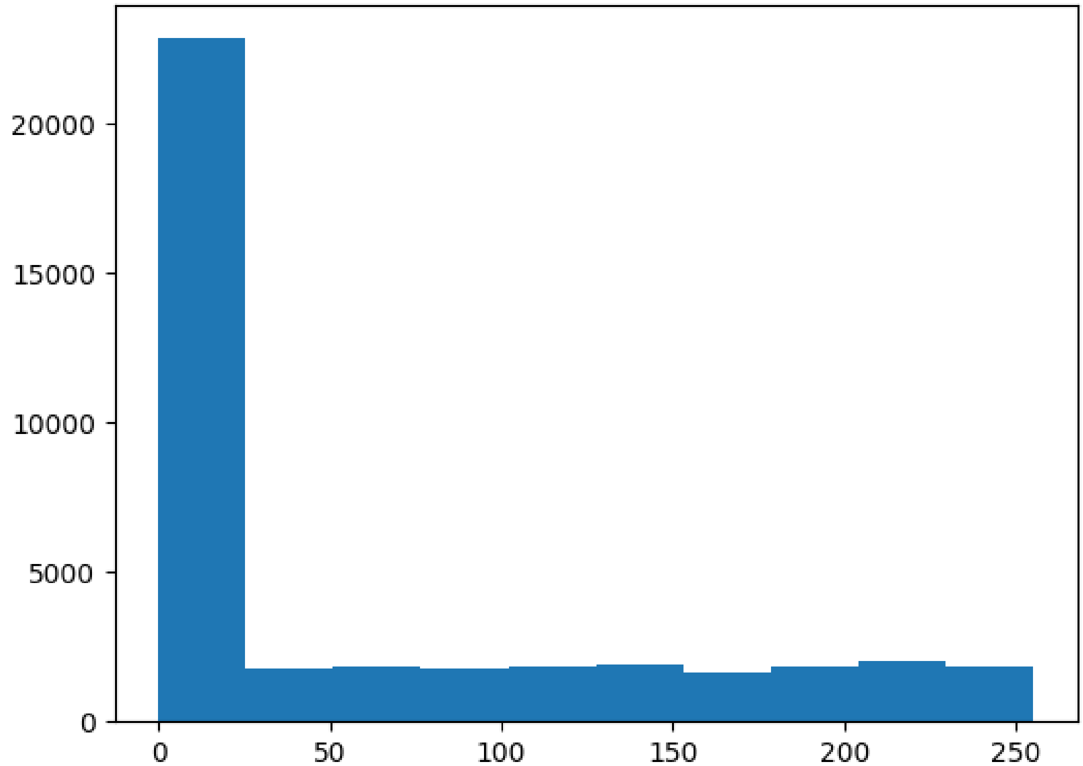



```
io.imshow(poupee_eq)
```

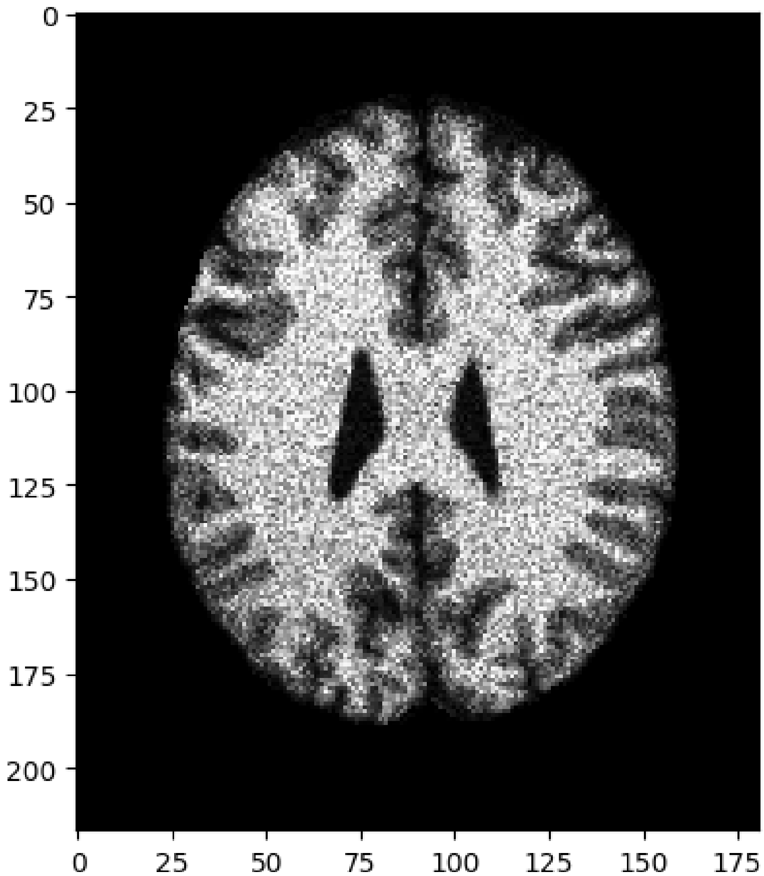
```
<matplotlib.image.AxesImage at 0x7980abc04430>
```




```
crane_eq = cv2.equalizeHist(crane)

plt.hist(crane_eq.flat, bins=10, range=(0,255))

(array([22843.,  1792.,  1810.,  1783.,  1816.,  1885.,  1662.,
1820.,
         2024.,  1842.]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <BarContainer object of 10 artists>)
```




```
io.imshow(crane_eq)

<matplotlib.image.AxesImage at 0x7980acd44130>
```



La mesure du contraste des images égalisés:

Pour la poupee

```
michelson = michelson_contrast(poupee_eq)
print("Michelson Contrast:", michelson)
```

```
Michelson Contrast: 1.0
```

```
global_contrast_val = global_contrast(poupee_eq)
print("Global Contrast:", global_contrast_val)
```

```
Global Contrast: 0.623291099864977
```

```
rms_contrast_val = rms_contrast(poupee_eq)
print("Root Mean Squared Contrast:", rms_contrast_val)
```

```
Root Mean Squared Contrast: 90.98287811844985
```

Pour la crane

```
michelson = michelson_contrast(crane_eq)
print("Michelson Contrast:", michelson)
```

```
Michelson Contrast: 1.0
```

```
global_contrast_val = global_contrast(crane_eq)
print("Global Contrast:", global_contrast_val)
```

```
Global Contrast: 1.3655103074971957
```

```
rms_contrast_val = rms_contrast(crane_eq)
print("Root Mean Squared Contrast:", rms_contrast_val)
```

```
Root Mean Squared Contrast: 81.48394287810467
```

Commentaire: On remarque que les valeurs de contraste sont élevés par rapport les images original a cause de l'égalisation d'hitogramme

Egalisation de CLAHE:

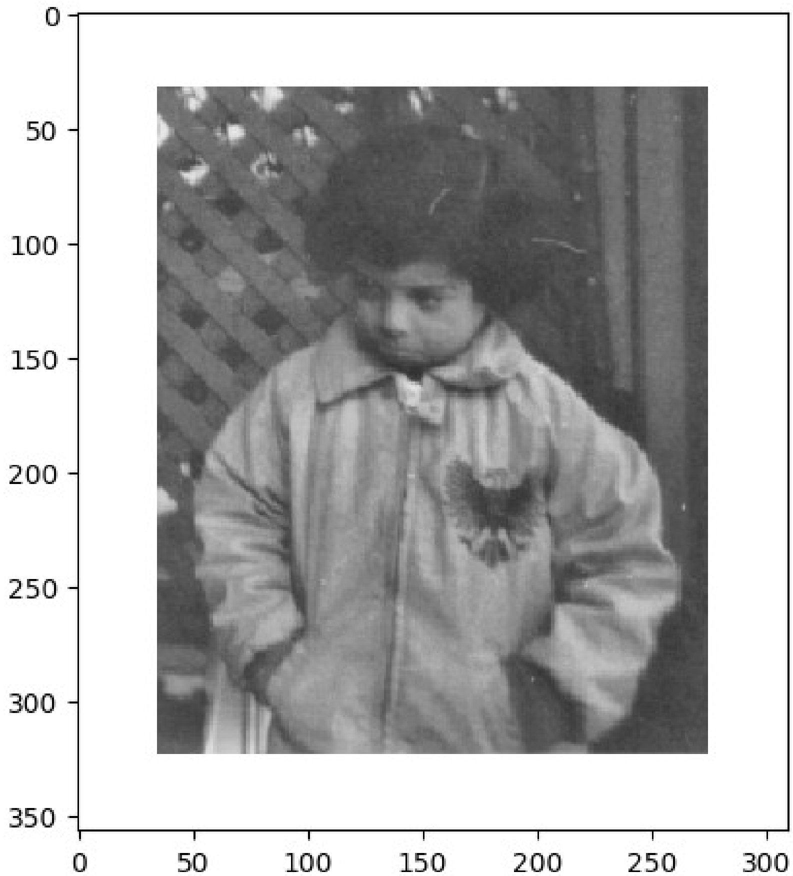
CLAHE (Contrast Limited Adaptive Histogram Equalization) est une technique de traitement d'image qui améliore le contraste local d'une image en utilisant une égalisation d'histogramme adaptative avec une limitation de contraste. Contrairement à l'égalisation d'histogramme classique, qui égalise l'histogramme de toute l'image, CLAHE divise l'image en petits blocs (généralement des carrés) et égalise l'histogramme de chaque bloc individuellement.

Pour la poupee

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
poupee_clahe = clahe.apply(poupee)
```

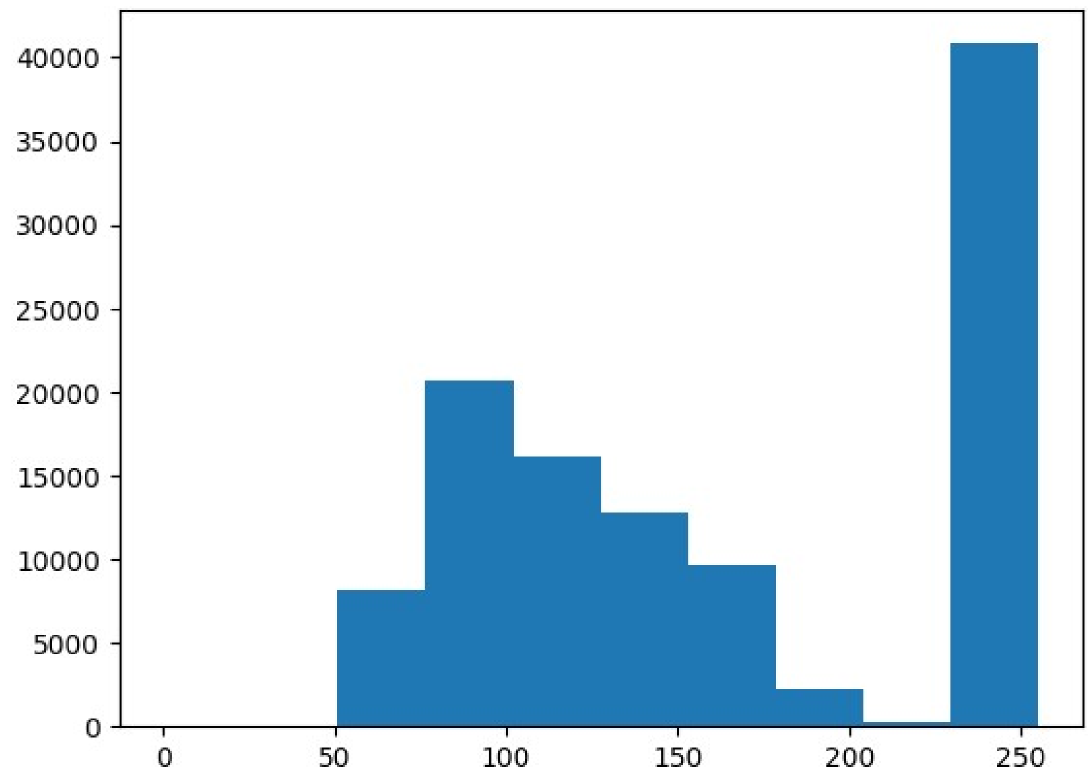
```
io.imshow(poupee_clahe)
```

```
<matplotlib.image.AxesImage at 0x7980ac9b64a0>
```




```
plt.hist(poupee_clahe.flat, bins=10, range=(0,255))
```

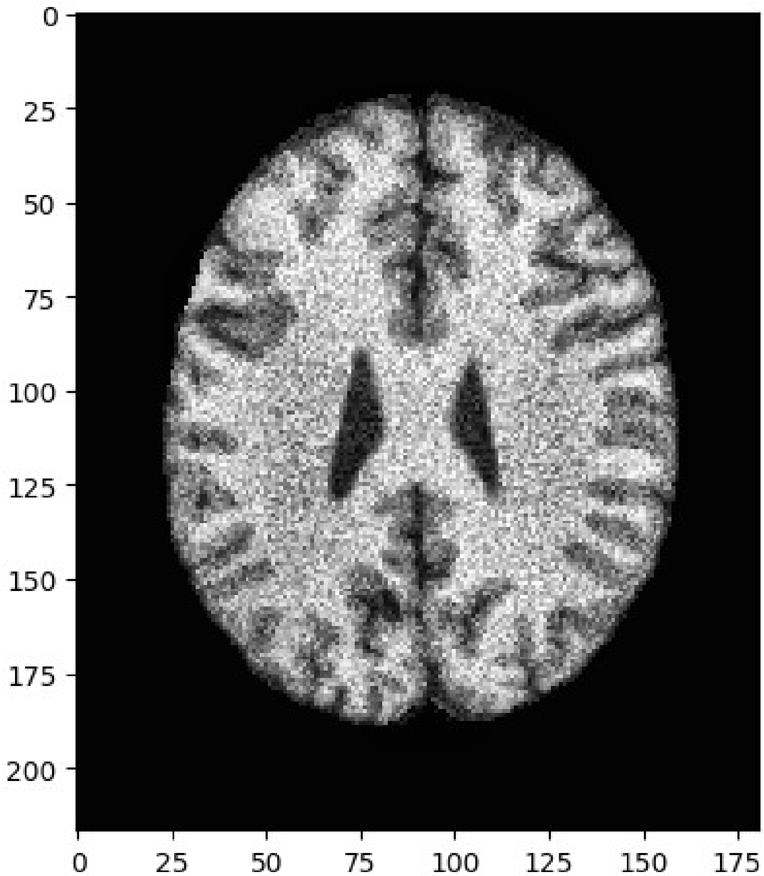
```
(array([0.0000e+00, 1.0000e+00, 8.1700e+03, 2.0675e+04, 1.6100e+04,
        1.2748e+04, 9.6560e+03, 2.2540e+03, 2.3400e+02, 4.0832e+04]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <BarContainer object of 10 artists>)
```




```
michelson = michelson_contrast(poupee_clahe)
print("Michelson Contrast:", michelson)
global_contrast_val = global_contrast(poupee_clahe)
print("Global Contrast:", global_contrast_val)
rms_contrast_val = rms_contrast(poupee_clahe)
print("Root Mean Squared Contrast:", rms_contrast_val)

Michelson Contrast: 4.291666666666667
Global Contrast: 0.435323895559744
Root Mean Squared Contrast: 72.60772249625568

<ipython-input-244-509d877fca37>:7: RuntimeWarning: overflow
encountered in ubyte_scalars
  return (max_intensity - min_intensity) / (max_intensity +
min_intensity)
```

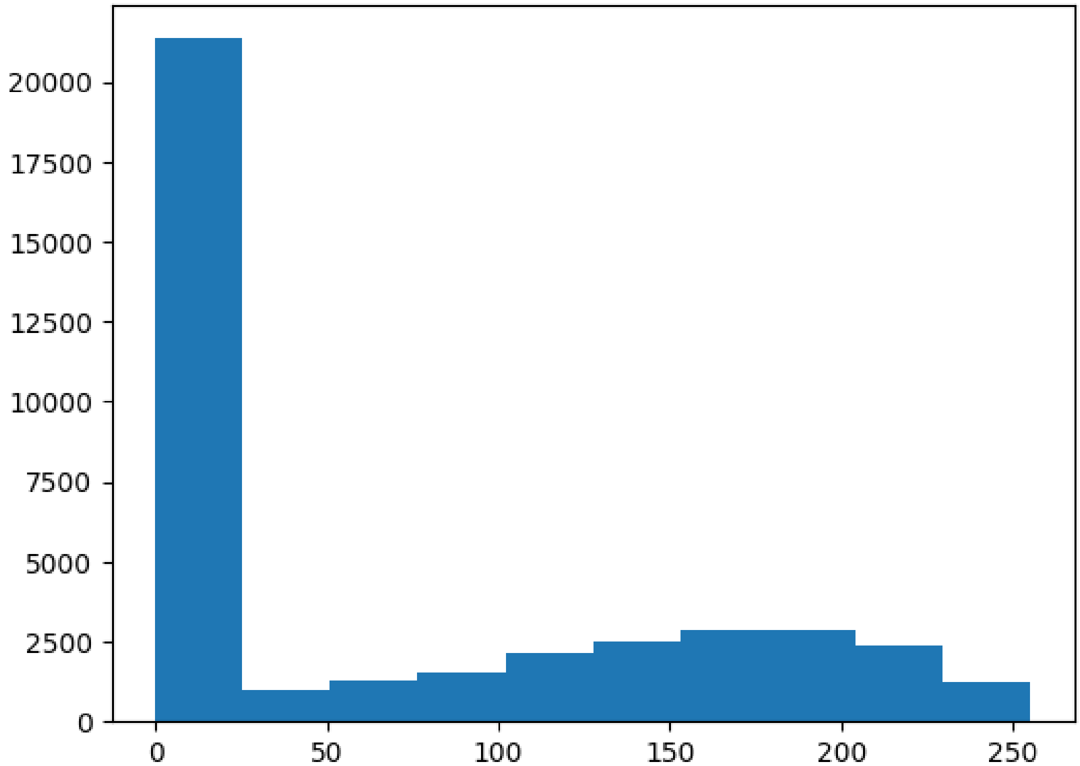
Pour la crane

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))  # Vous
pouvez ajuster les valeurs selon vos besoins
crane_clahe = clahe.apply(crane)

io.imshow(crane_clahe)

<matplotlib.image.AxesImage at 0x7980ac5eca30>
```




```
plt.hist(crane_clahe.flat, bins=10, range=(0,255))
```

```
(array([21359.,   990.,  1310.,  1562.,  2148.,  2523.,  2886.,
2879.,
         2388.,  1232.]),
 array([  0. ,  25.5,  51. ,  76.5, 102. , 127.5, 153. , 178.5, 204. ,
        229.5, 255. ]),
 <BarContainer object of 10 artists>)
```




```
michelson = michelson_contrast(crane_clahe)
print("Michelson Contrast:", michelson)
global_contrast_val = global_contrast(crane_clahe)
print("Global Contrast:", global_contrast_val)
rms_contrast_val = rms_contrast(crane_clahe)
print("Root Mean Squared Contrast:", rms_contrast_val)
```

```
Michelson Contrast: 253.0
Global Contrast: 1.1752102640899205
Root Mean Squared Contrast: 82.86173379803887
```

```
<ipython-input-244-509d877fca37>:7: RuntimeWarning: overflow
encountered in ubyte_scalars
  return (max_intensity - min_intensity) / (max_intensity +
min_intensity)
```

Comparaison des résultats: L'égalisation d'histogramme a des valeurs de contraste moins que de l'egalisation de CLAHE.

L'égalisation d'histogramme classique redistribue les niveaux de gris de l'ensemble de l'image de manière à égaliser l'histogramme global. Cela signifie qu'elle peut augmenter le contraste global de l'image, mais elle ne tient pas compte des variations locales de contraste. Dans certaines images, cela peut entraîner une amplification du bruit et des artefacts indésirables, ce qui peut réduire la qualité globale de l'image.

CLAHE, en revanche, divise l'image en petits blocs et égalise l'histogramme de chaque bloc individuellement. De plus, il limite le contraste dans chaque bloc, ce qui signifie qu'il évite d'amplifier le bruit dans les régions homogènes de l'image. Cette approche adaptative permet d'augmenter le contraste dans les zones de faible contraste tout en maintenant le contraste dans les zones de forte variation. Cela conduit souvent à une amélioration du contraste local sans les effets indésirables de l'égalisation d'histogramme classique.