

# Cyfrowe przetwarzanie sygnałów i obrazów

## Laboratorium nr 2 - Przetwarzanie obrazów

Autorzy:

Imię i nazwisko	Numer indeksu
Maksymilian Tara	264000
Łukasz Gawron	264475

```
In [ ]: %matplotlib widget
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
import cv2 as cv

from skimage.util import img_as_ubyte
import skimage.morphology as morph
from skimage.filters import rank
from scipy.ndimage import uniform_filter
from scipy.ndimage import convolve
from scipy.ndimage import median_filter
from scipy.ndimage import minimum_filter
from scipy.ndimage import maximum_filter
from scipy.ndimage import gaussian_filter

def get_image(path: str):
    # convert("L") - konwersja zdjęcia do skali szarości
    return np.array(Image.open(path).convert("L"))

def show_image(imageArray: np.ndarray):
    img = Image.fromarray(imageArray)
    plt.figure(figsize=(8, 4))
    plt.imshow(img, cmap='gray', vmin=0, vmax=255)
    plt.show()
```

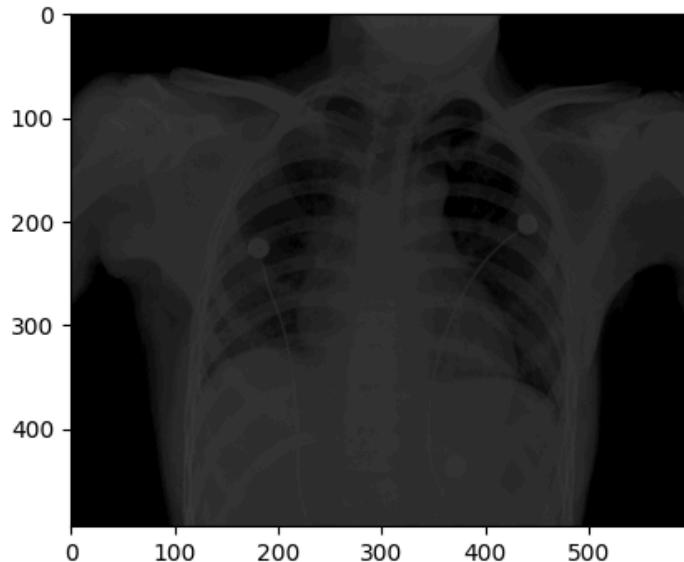
## Zadanie nr 1: Platforma testowa

Napisz skrypt w Pythonie/Matlabie umożliwiający wczytywanie i wizualizację badanych obrazów.  
Program powinien umożliwiać:

1. wyświetlanie obrazu wczytanego z pliku o podanej nazwie,
2. sporządzenie wykresów zmian poziomu szarości wzdłuż wybranej linii poziomej lub pionowej o zadanej współrzędnej,
3. wybór podobrazu (prostokątnego obszaru) o podanych współrzędnych oraz jego zapis do pliku o zadanej nazwie.

```
In [ ]: img_chest_xray = get_image('resources/chest-xray.tif')
show_image(img_chest_xray)
```

Figure



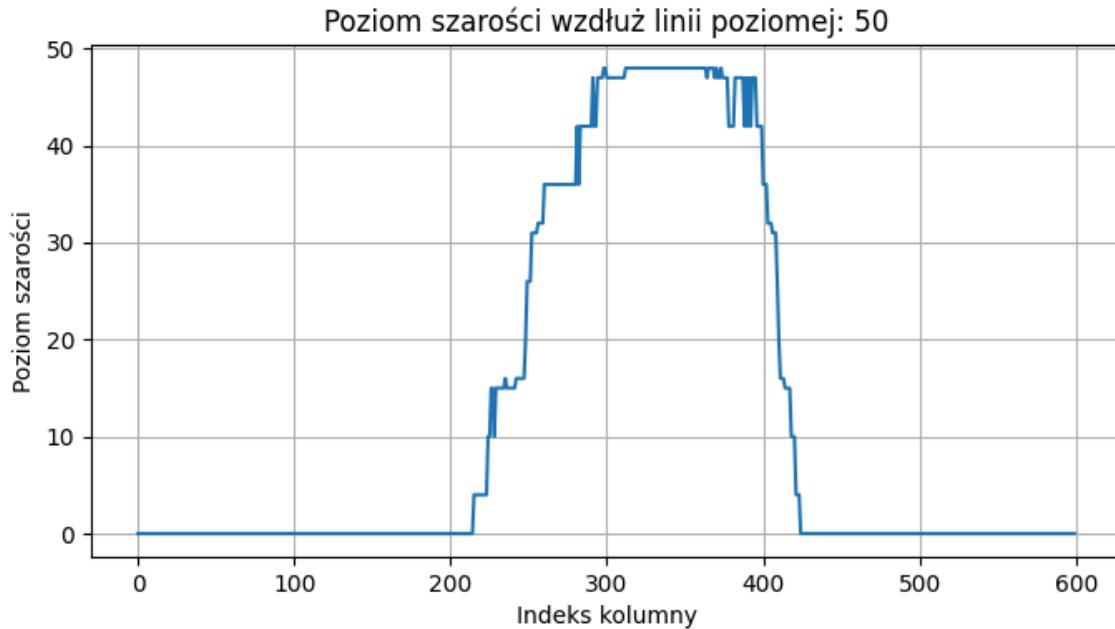
```
In [ ]: def plot_line_profile(image: np.ndarray, index: int, orientation: str):
    plt.figure(figsize=(8, 4))

    if orientation == 'horizontal':
        line_data = image[index, :] # Extract the row
        plt.title(f'Poziom szarości wzdłuż linii poziomej: {index}')
        plt.xlabel('Indeks kolumny')
    elif orientation == 'vertical':
        line_data = image[:, index] # Extract the column
        plt.title(f'Poziom szarości wzdłuż linii pionowej: {index}')
        plt.xlabel('Indeks wiersza')
    else:
        raise ValueError("Orientation must be 'horizontal' or 'vertical')

    plt.plot(line_data)
    plt.ylabel('Poziom szarości')
    plt.grid(True)
    plt.show()

global img_chest_xray
plot_line_profile(img_chest_xray, 50, 'horizontal')
```

Figure



## Zadanie nr 2: Przekształcenia punktowe

Przekształcenie punktowe  $T$  obrazu ma ogólną postać  $s = T(r)$ , gdzie  $r$ ,  $s$  oznaczają odpowiednio poziom szarości piksela obrazu wejściowego i wyjściowego.

Zaobserwuj działanie następujących przekształceń punktowych na przykładowych obrazach:

- a) Mnożenie obrazu przez stałą  $T(r) = c \cdot r$ , gdzie  $c$  jest stałą. Obrazy: chest\_xray.tif, pollen-dark.tif, spectrum.tif.
- b) Transformację logarytmiczną  $T(r) = c \cdot \log(1 + r)$ . Obraz: spectrum.tif.
- c) Zmianę dynamiki skali szarości (kontrastu). Możesz zastosować transformację o postaci  $T(r) = 1/(1 + (m/r)^e)$ , gdzie  $m$  i  $e$  są ustalonymi parametrami przekształcenia (np.  $m = 0,45$ ,  $e = 8$ ). Wykreśl  $T(r)$ , by lepiej uwidoczyć wpływ  $T$  na kontrast obrazu wyjściowego. Przeprowadź eksperymenty z różnymi wartościami parametrów  $m$  i  $e$ .
- d) Korekcję gamma, zdefiniowaną jako  $s = c \cdot r^\gamma$ , gdzie  $c > 0$  i  $\gamma > 0$  są stałymi we wzorze przekształcenia. Obraz: aerial\_view.tif.

```
In [ ]: def image_transform_iterative(image: np.ndarray, transform: callable):
    copy = image.copy()
    for i in range(copy.shape[0]):
        for j in range(copy.shape[1]):
            copy[i, j] = transform(copy[i, j])
    return copy

def image_transform(image: np.ndarray, transform: callable):
    vfunc = np.vectorize(transform)
    return vfunc(image.copy())
```

```
# UWAGA: funkcja transformująca musi przyjmować i zwracać wartości z zakresu [0, 255], jeśli nie, to otrzymamy błąd
def image_transform_optimized(image: np.ndarray, transform: callable):
    transformed_img = transform(image.copy())
    return np.clip(transformed_img, 0, 255).astype(image.dtype)
```

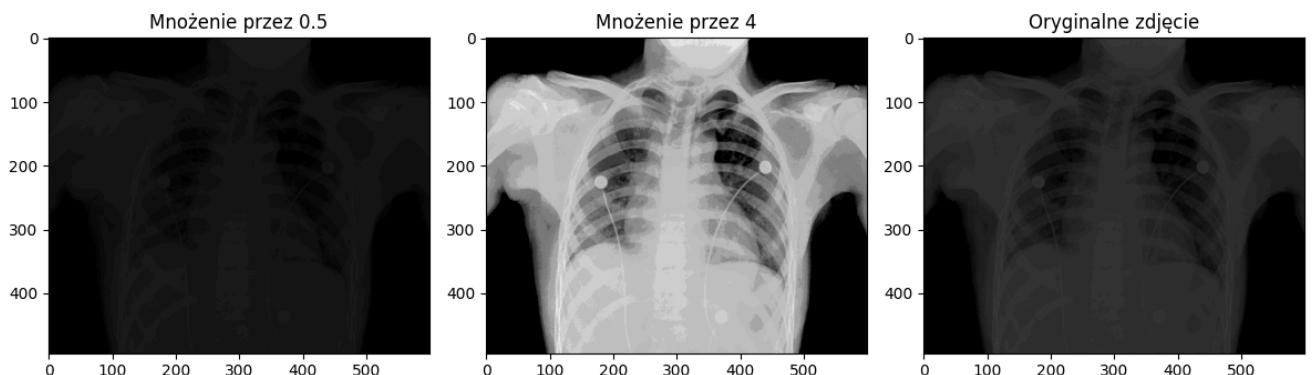
In [ ]:

```
# a) Mnożenie razy stałą
def const_transform1(x): return x*0.5
def const_transform2(x): return x*4

global img_chest_xray
img_chest_xray_transformed1 = image_transform_optimized(
    img_chest_xray, const_transform1)
img_chest_xray_transformed2 = image_transform_optimized(
    img_chest_xray, const_transform2)

fig, axes = plt.subplots(1, 3, figsize=(12, 4))
(ax1, ax2, ax3) = axes
ax1.imshow(Image.fromarray(img_chest_xray_transformed1),
            cmap='gray', vmin=0, vmax=255)
ax1.set_title('Mnożenie przez 0.5')
ax2.imshow(Image.fromarray(img_chest_xray_transformed2),
            cmap='gray', vmin=0, vmax=255)
ax2.set_title('Mnożenie przez 4')
ax3.imshow(Image.fromarray(img_chest_xray),
            cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginalne zdjęcie')
plt.tight_layout()
plt.show()
```

Figure



In [ ]:

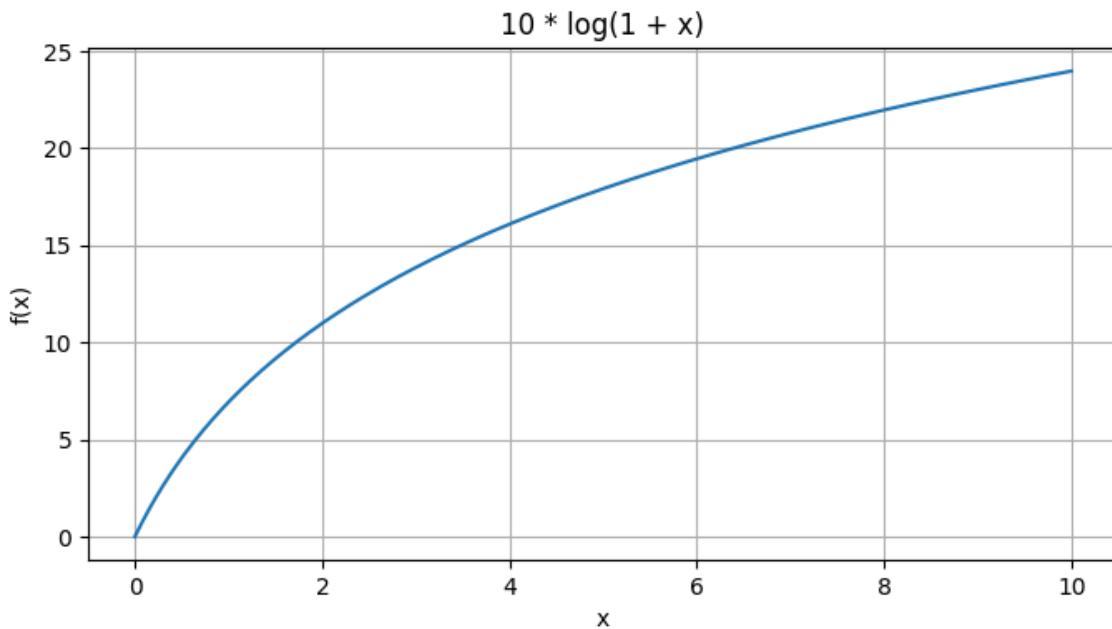
```
# FUNKCJA LOGARYTMICZNA f(x)=10·Log(1+x)
def f(x):
    return 10 * np.log(1 + x)

x = np.linspace(0, 10, 400)
y = f(x)

plt.figure(figsize=(8, 4))
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('f(x)')
plt.title('10 * log(1 + x)')
```

```
plt.grid(True)
plt.show()
```

Figure



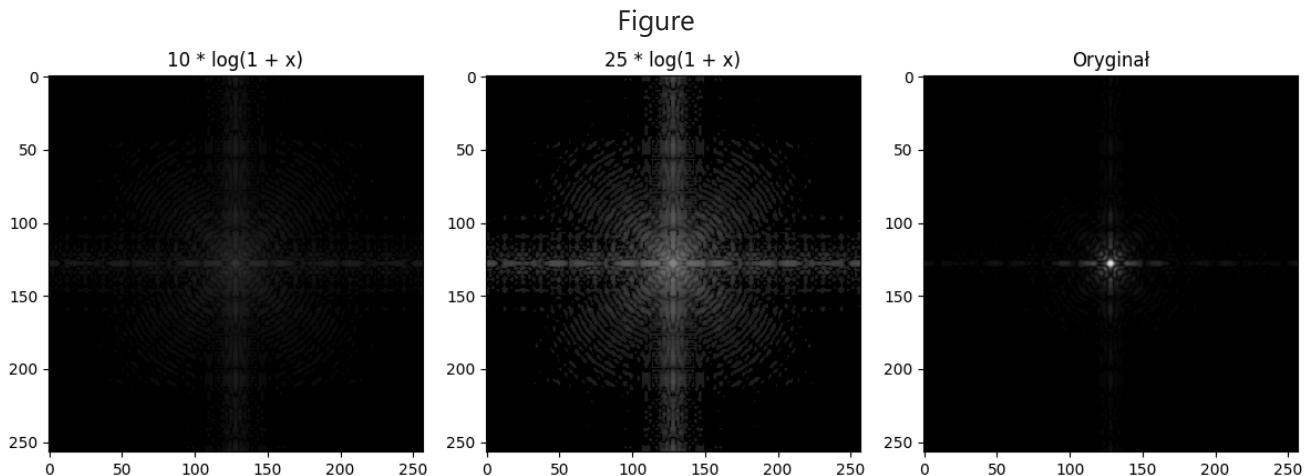
```
# b) Transformacja Logarytmiczna
epsilon = 1e-8

def logarithmic_transform(x): return 10 * np.log(1 + (x+epsilon))
def logarithmic_transform2(x): return 25 * np.log(1 + (x+epsilon))

img_spectrum = get_image('resources/spectrum.tif')

img_transformed1 = image_transform_optimized(
    img_spectrum, logarithmic_transform)
img_transformed2 = image_transform_optimized(
    img_spectrum, logarithmic_transform2)

fig, axes = plt.subplots(1, 3, figsize=(12, 4))
(ax1, ax2, ax3) = axes
ax1.imshow(Image.fromarray(img_transformed1),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('10 * log(1 + x)')
ax2.imshow(Image.fromarray(img_transformed2),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('25 * log(1 + x)')
ax3.imshow(Image.fromarray(img_spectrum),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')
plt.tight_layout()
plt.show()
```



```
In [ ]: # Dynamika szarości
def T(r, m, e):
    return 1 / (1 + (m / r)**e)

m1 = 0.45
e1 = 8
r1 = np.linspace(0.1, 10, 400)
T1_values = T(r1, m1, e1)

m2 = 0.45
e2 = 2
r2 = np.linspace(0.1, 10, 400)
T2_values = T(r2, m2, e2)

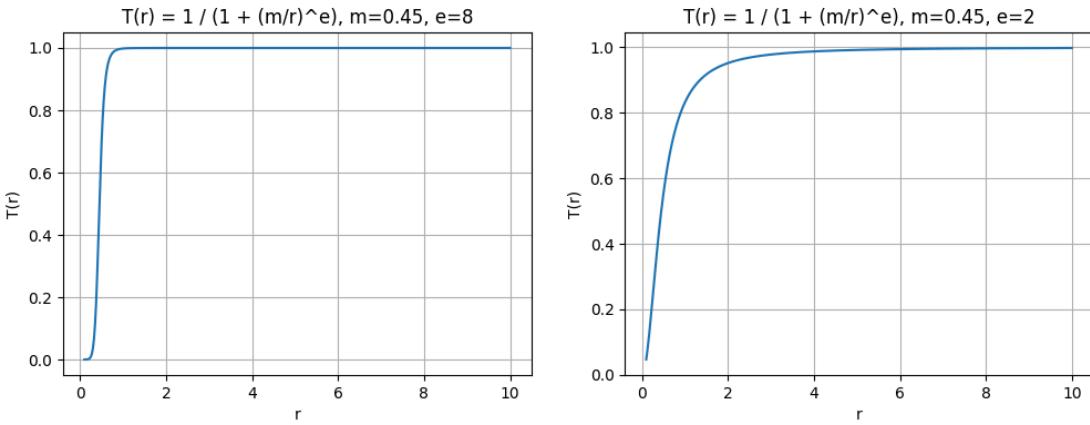
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(r1, T1_values)
plt.xlabel('r')
plt.ylabel('T(r)')
plt.title(f'T(r) = 1 / (1 + (m/r)^e), m={m1}, e={e1}')
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(r2, T2_values)
plt.xlabel('r')
plt.ylabel('T(r)')
plt.title(f'T(r) = 1 / (1 + (m/r)^e), m={m2}, e={e2}')
plt.grid(True)

plt.show()
```

## Figure



```
In [ ]: # c) Zmianę dynamiki skali szarości
epsilon = 1e-8

def contrast_stretching_transform(x): return (
    255/(1+np.power(0.45/((x+epsilon)/255.0), 8)))

def contrast_stretching_transform2(
    x): return (255/(1+np.power(0.45/((x+epsilon)/255.0), 2)))

def contrast_stretching_transform3(
    x): return (255/(1+np.power(0.45/((x+epsilon)/255.0), 1)))

def contrast_stretching_transform4(
    x): return (255/(1+np.power(0.3/((x+epsilon)/255.0), 8)))

def contrast_stretching_transform5(
    x): return (255/(1+np.power(0.2/((x+epsilon)/255.0), 8)))

def contrast_stretching_transform6(
    x): return (255/(1+np.power(0.1/((x+epsilon)/255.0), 8)))

def contrast_stretching_transform7(
    x): return (255/(1+np.power(0.2/((x+epsilon)/255.0), 2)))

global img_chest_xray

img_chest_xray_transformed1 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform)
img_chest_xray_transformed2 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform2)
img_chest_xray_transformed3 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform3)
img_chest_xray_transformed4 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform4)
img_chest_xray_transformed5 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform5)
```

```
img_chest_xray_transformed6 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform6)
img_chest_xray_transformed7 = image_transform_optimized(
    img_chest_xray, contrast_stretching_transform7)

plt.figure(figsize=(8, 12))
plt.subplot(4, 2, 1)

plt.subplot(4, 2, 1)
plt.title('Oryginal')
plt.imshow(ImageView.fromarray(img_chest_xray), cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 2)
plt.title('m = 0.45, e = 8')
plt.imshow(img_chest_xray_transformed1, cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 3)
plt.title('m = 0.45, e = 2')
plt.imshow(img_chest_xray_transformed2, cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 4)
plt.title('m = 0.45, e = 1')
plt.imshow(img_chest_xray_transformed3, cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 5)
plt.title('m = 0.3, e = 8')
plt.imshow(img_chest_xray_transformed4, cmap='gray', vmin=0, vmax=255)

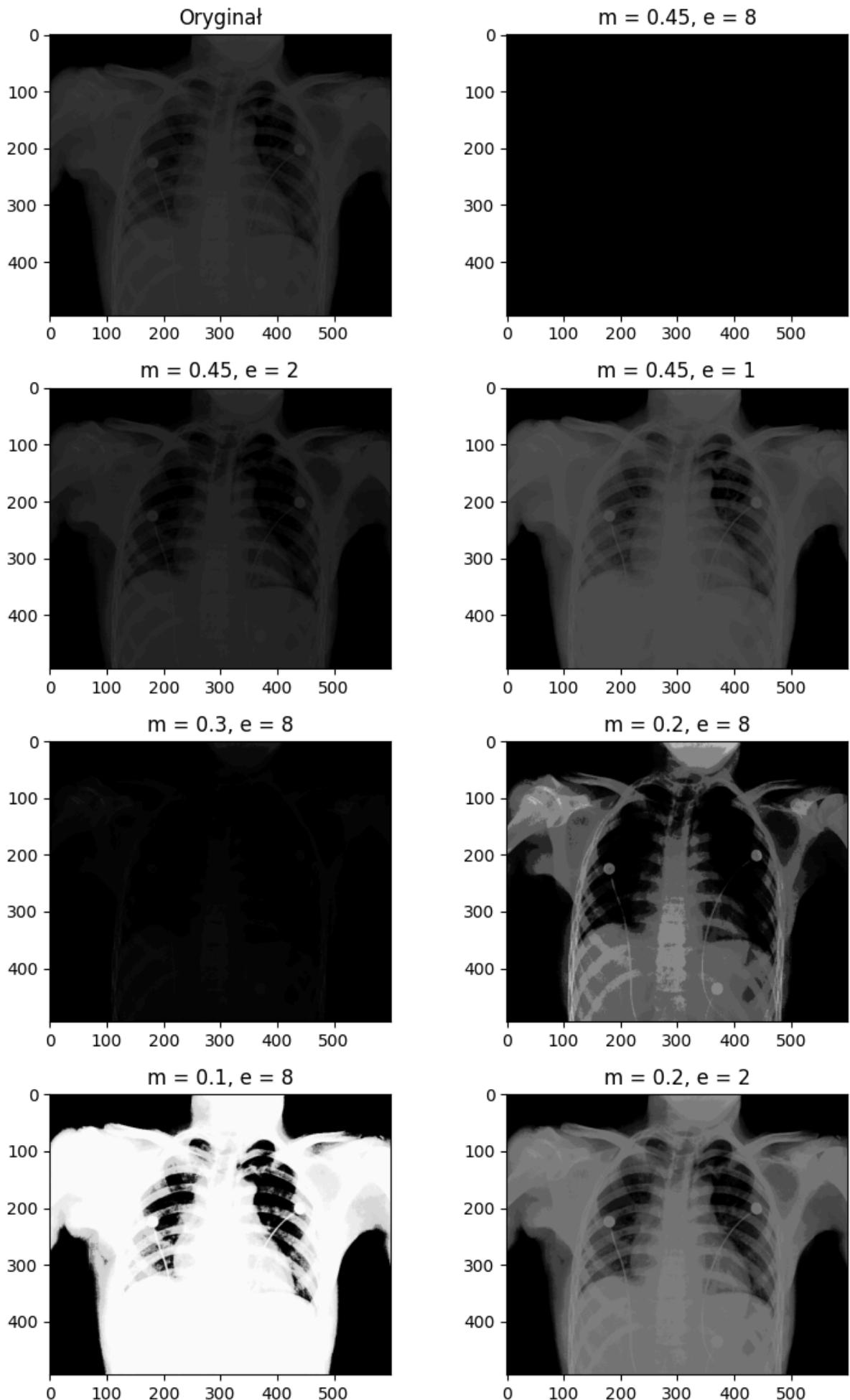
plt.subplot(4, 2, 6)
plt.title('m = 0.2, e = 8')
plt.imshow(img_chest_xray_transformed5, cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 7)
plt.title('m = 0.1, e = 8')
plt.imshow(img_chest_xray_transformed6, cmap='gray', vmin=0, vmax=255)

plt.subplot(4, 2, 8)
plt.title('m = 0.2, e = 2')
plt.imshow(img_chest_xray_transformed7, cmap='gray', vmin=0, vmax=255)

plt.tight_layout()
plt.show()
```

Figure

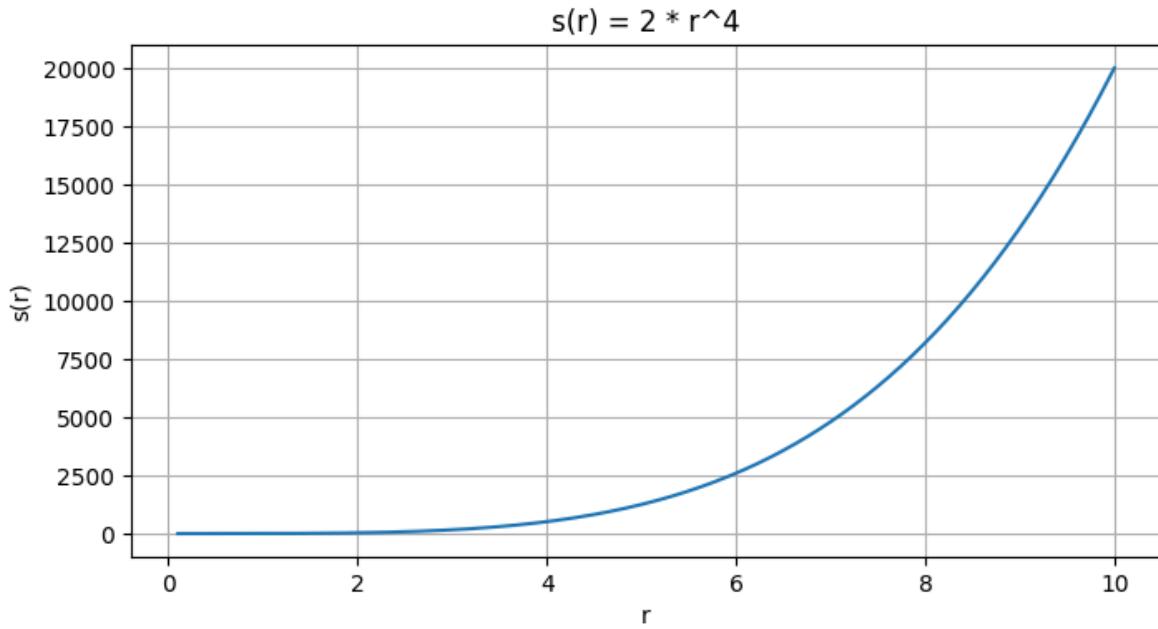


```
In [ ]: # Korekcja gamma
def s(r, c, gamma):
    return c * r**gamma

c = 2
gamma = 4
r = np.linspace(0.1, 10, 400)
s_values = s(r, c, gamma)

plt.figure(figsize=(8, 4))
plt.plot(r, s_values)
plt.xlabel('r')
plt.ylabel('s(r)')
plt.title(f's(r) = {c} * r^{gamma}')
plt.grid(True)
plt.show()
```

Figure



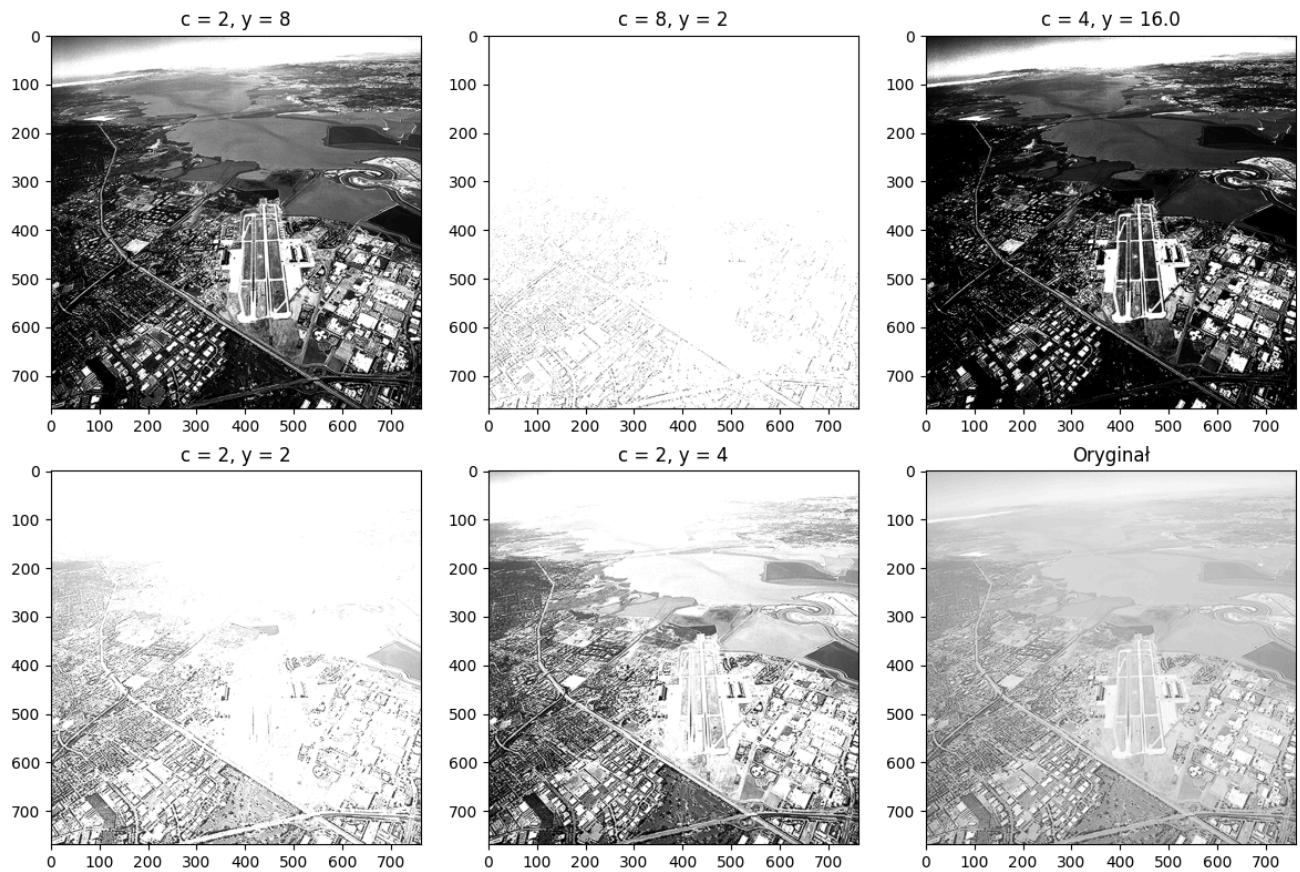
```
In [ ]: # d) Korekcje gamma
def contrast_stretching_transform(x): return 255*2*np.power(x/255, 8.0)
def contrast_stretching_transform1(x): return 255*8*np.power(x/255, 2)
def contrast_stretching_transform2(x): return 255*4*np.power(x/255, 16.0)
def contrast_stretching_transform3(x): return 255*2*np.power(x/255, 2.0)
def contrast_stretching_transform4(x): return 255*2*np.power(x/255, 4.0)

img_aerial_view = get_image('resources/aerial_view.tif')

img_transformed1 = image_transform_optimized(
    img_aerial_view, contrast_stretching_transform)
img_transformed2 = image_transform_optimized(
    img_aerial_view, contrast_stretching_transform1)
img_transformed3 = image_transform_optimized(
    img_aerial_view, contrast_stretching_transform2)
img_transformed4 = image_transform_optimized(
    img_aerial_view, contrast_stretching_transform3)
```

```
img_transformed5 = image_transform_optimized(  
    img_aerial_view, contrast_stretching_transform4)  
  
plt.figure(figsize=(12, 8))  
  
plt.subplot(2, 3, 1)  
plt.title('c = 2, y = 8')  
plt.imshow(img_transformed1, cmap='gray', vmin=0, vmax=255)  
  
plt.subplot(2, 3, 2)  
plt.title('c = 8, y = 2')  
plt.imshow(img_transformed2, cmap='gray', vmin=0, vmax=255)  
  
plt.subplot(2, 3, 3)  
plt.title('c = 4, y = 16.0')  
plt.imshow(img_transformed3, cmap='gray', vmin=0, vmax=255)  
  
plt.subplot(2, 3, 4)  
plt.title('c = 2, y = 2')  
plt.imshow(img_transformed4, cmap='gray', vmin=0, vmax=255)  
  
plt.subplot(2, 3, 5)  
plt.title('c = 2, y = 4')  
plt.imshow(img_transformed5, cmap='gray', vmin=0, vmax=255)  
  
plt.subplot(2, 3, 6)  
plt.title('Oryginal')  
plt.imshow(img_aerial_view, cmap='gray', vmin=0, vmax=255)  
  
plt.tight_layout()  
plt.show()
```

Figure



## Zadanie nr 3: Histogram obrazu

Wypróbuj działanie wyrównywania histogramu na przykładowych obrazach. By zaobserwować skuteczność procedury, poddaj wyrównywaniu obrazy zbyt ciemne i zbyt jasne. Narysować histogramy obrazów przed i po wyrównaniu.

Obrazy: chest\_xray.tif, pollen-dark.tif, pollen-ligt.tif, pollen-lowcontrast.tif, pout.tif, spectrum.tif.

In [ ]: `global img_chest_xray`

```
image_correction = cv.equalizeHist(img_chest_xray)

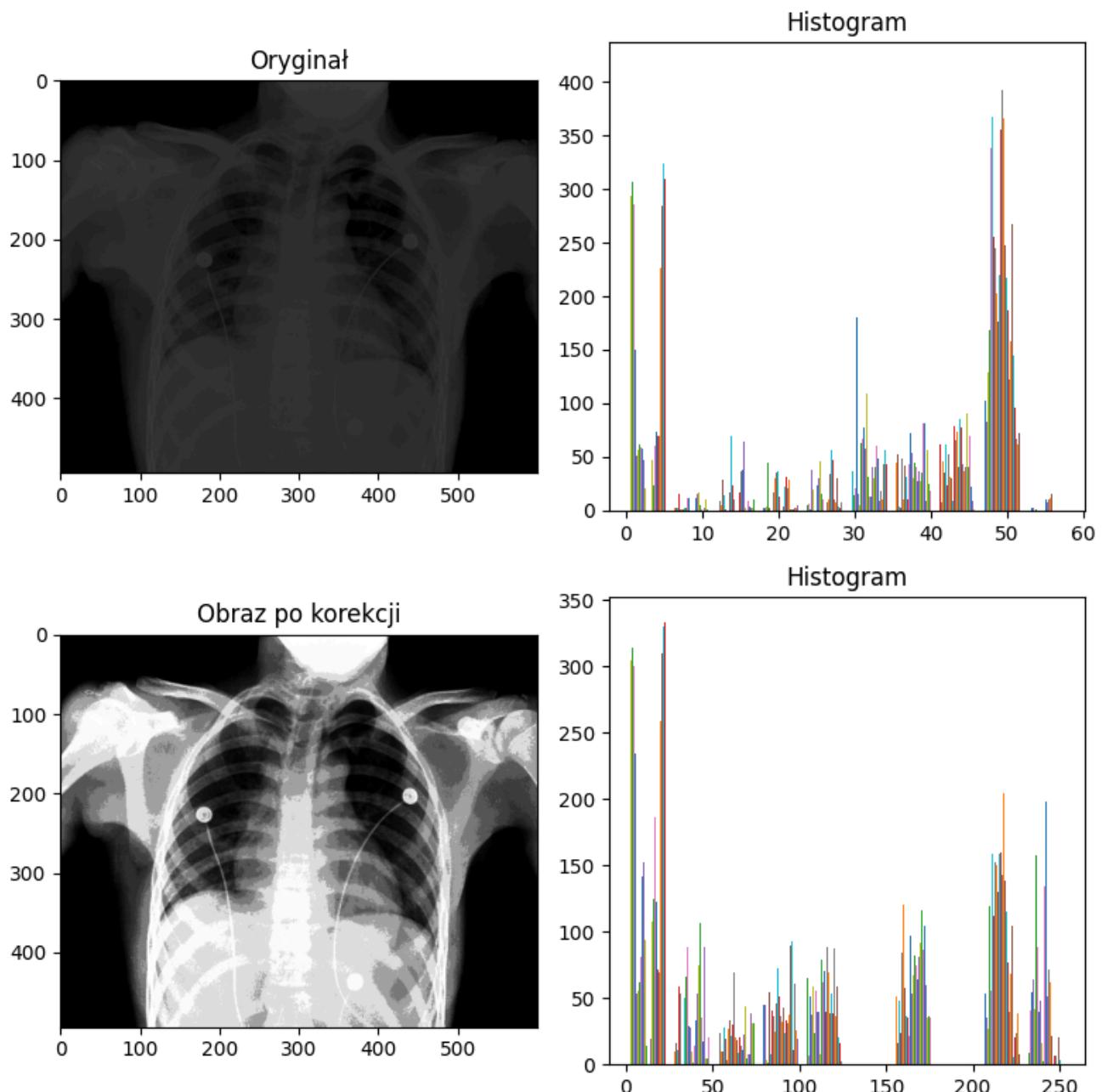
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(img_chest_xray),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Oryginal')
ax2.hist(img_chest_xray)
ax2.set_title('Histogram')

ax3.imshow(Image.fromarray(image_correction),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Obraz po korekcji')
ax4.hist(image_correction)
ax4.set_title('Histogram')
```

```
plt.tight_layout()
plt.show()
```

Figure



```
In [ ]: img_pollen_light = get_image('resources/pollen-ligt.tif')

image_correction = cv.equalizeHist(img_pollen_light)

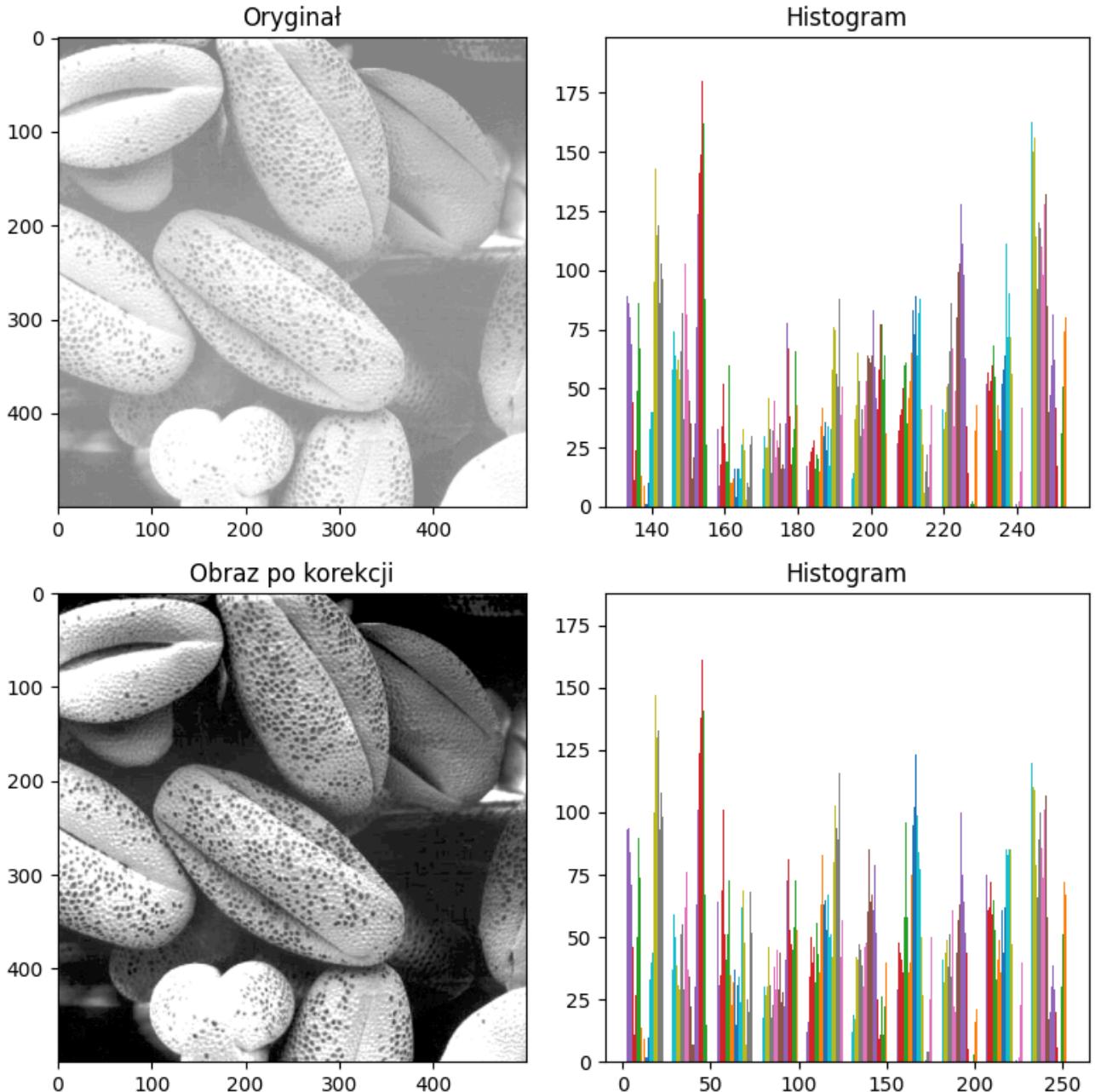
fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(img_pollen_light),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Oryginal')
ax2.hist(img_pollen_light)
ax2.set_title('Histogram')

ax3.imshow(Image.fromarray(image_correction),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Obraz po korekcji')
ax4.hist(image_correction)
ax4.set_title('Histogram')
```

```
cmap='gray', vmin=0, vmax=255)
ax3.set_title('Obraz po korekcji')
ax4.hist(image_correction)
ax4.set_title('Histogram')
plt.tight_layout()
plt.show()
```

Figure



```
In [ ]: img_pollen_dark = get_image('resources/pollen-dark.tif')

image_correction = cv.equalizeHist(img_pollen_dark)

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(img_pollen_dark),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Oryginal')
```

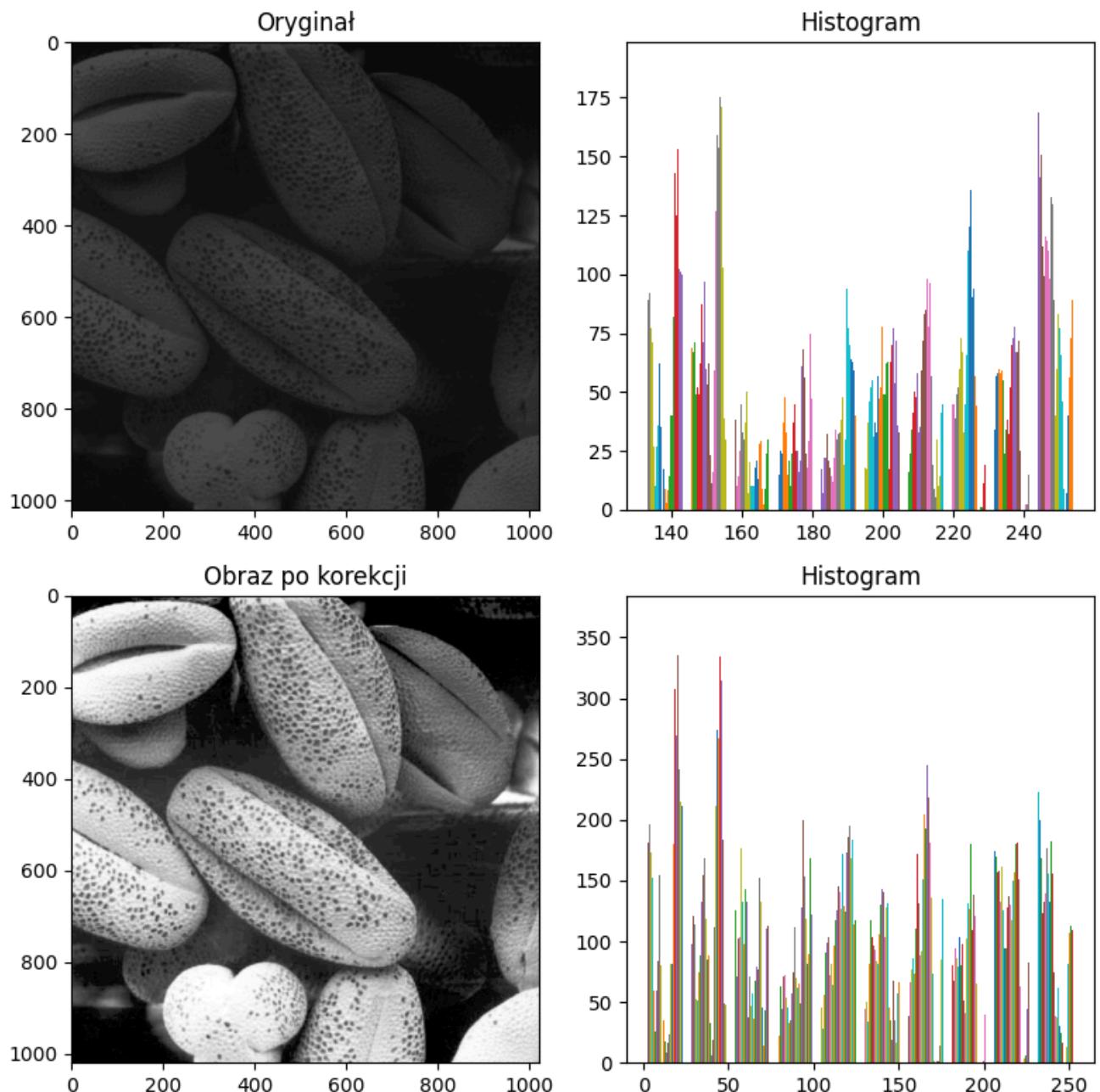
```

ax2.hist(img_pollen_light)
ax2.set_title('Histogram')

ax3.imshow( Image.fromarray(image_correction),
            cmap='gray', vmin=0, vmax=255)
ax3.set_title('Obraz po korekcji')
ax4.hist(image_correction)
ax4.set_title('Histogram')
plt.tight_layout()
plt.show()

```

Figure



Procedura wyrównywania histogramu działa skutecznie, zarówno dla zbyt ciemnych jak i zbyt jasnych obrazów.

## Zadanie nr 4: działanie lokalnych kontekstowych

Sprawdź działanie lokalnych kontekstowych omówionych na wykładzie pt. „Transformacje poziomu jasności” jako:

- a) lokalne wyrównywanie histogramu,
- b) poprawa jakości oparta na lokalnych statystykach.

Wykonaj eksperymenty dla różnych rozmiarów masek.

```
In [ ]: # funkcja służąca wyrównywaniu lokalnemu
def equalize_histogram_locally(input_img, radius: int):
    img = img_as_ubyte(input_img)
    kernel = morph.square(radius)
    img_local = rank.equalize(img, footprint=kernel)
    return img_local
```

```
In [ ]: global img_chest_xray
img_hidden_symbols = get_image('resources/hidden-symbols.tif')

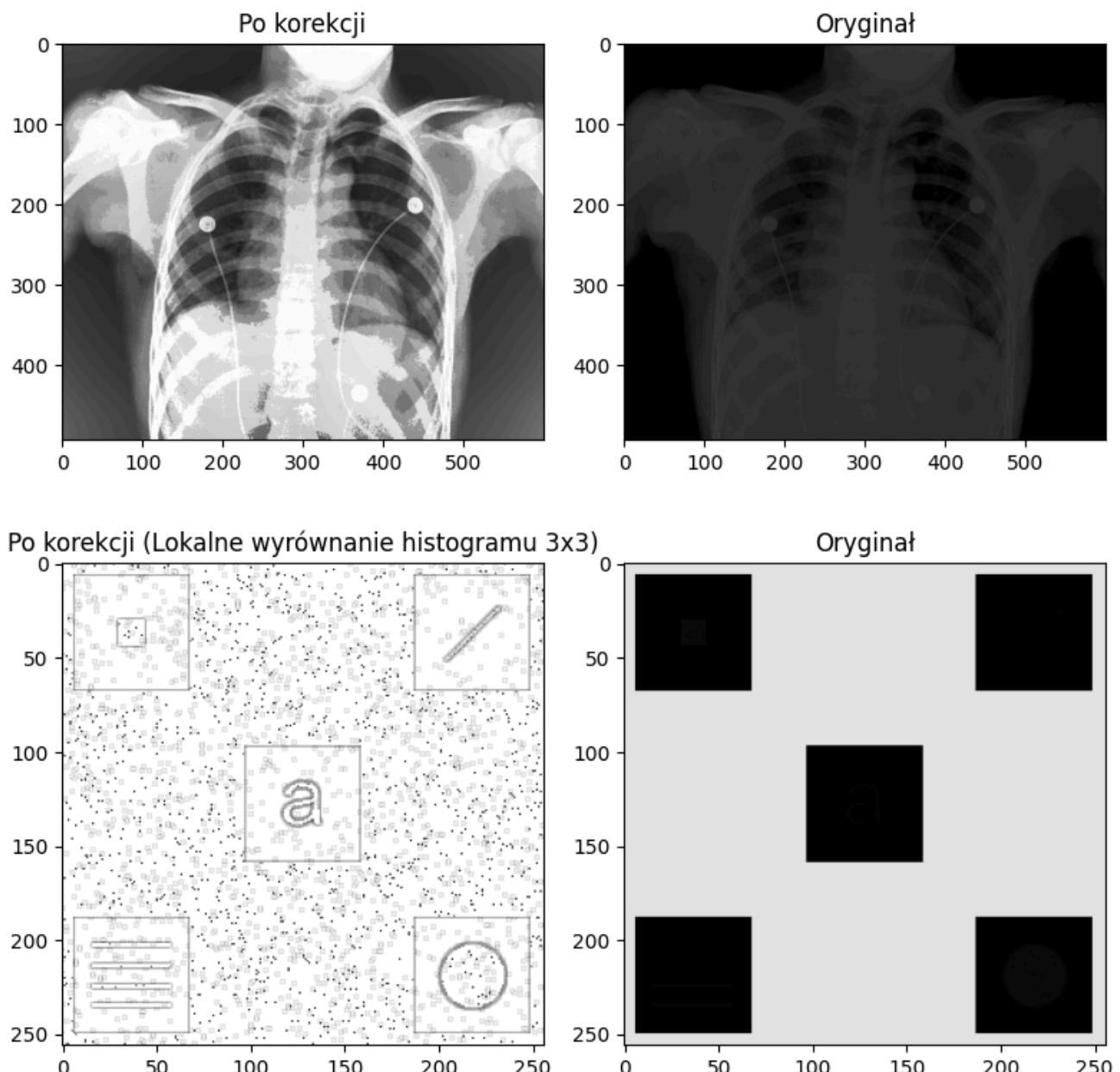
eq_local_xray = equalize_histogram_locally(img_chest_xray, 400)
eq_local_symbols = equalize_histogram_locally(img_hidden_symbols, 3)

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(eq_local_xray),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Po korekcji')
ax2.imshow(Image.fromarray(img_chest_xray),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Oryginał')

ax3.imshow(Image.fromarray(eq_local_symbols),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Po korekcji (Lokalne wyrównanie histogramu 3x3)')
ax4.imshow(Image.fromarray(img_hidden_symbols),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Oryginał')
plt.tight_layout()
plt.show()
```

Figure



```
In [ ]: # Funkcja wyrównująca histogram statystycznie - podejście ITERACYJNE
def equalize_histogram_statisticallyV1(image, C=22.8, k0=0, k1=0.1, k2=0, k3=0.1):
    # Konwertujemy obraz do odcieni szarości, jeśli jest w kolorze
    if len(image.shape) == 3:
        image_gray = cv.cvtColor(image, cv.COLOR_BGR2GRAY)
    else:
        image_gray = image

    # Obliczamy średnią i odchylenie standardowe
    mG = np.mean(image_gray)
    oG = np.std(image_gray)

    # Tworzymy kopię obrazu do modyfikacji
    improved_image = image_gray.copy()

    # Okno
    window_size = 3
    half_window = window_size // 2
```

```

# Iterujemy przez każdy piksel obrazu
for y in range(half_window, image_gray.shape[0] - half_window):
    for x in range(half_window, image_gray.shape[1] - half_window):
        # Okno Lokalne
        local_window = image_gray[y-half_window:y +
                                    half_window+1, x-half_window:x+half_window+1]

        # Obliczamy lokalną średnią i odchylenie standardowe
        mSxy = np.mean(local_window)
        oSxy = np.std(local_window)

        # Przetwarzanie
        if k0 * mG <= mSxy <= k1 * mG and k2 * oG <= oSxy <= k3 * oG:
            improved_image[y, x] = C * image_gray[y, x]
        else:
            improved_image[y, x] = image_gray[y, x]

return improved_image

# Funkcja wyrównująca histogram statystycznie - zoptymalizowana
def equalize_histogram_statistically(image, C=22.8, k0=0, k1=0.1, k2=0, k3=0.1):
    # Obliczamy średnią i odchylenie standardowe
    mG = np.mean(image)
    oG = np.std(image)

    # Rozmiar Lokalnego okna
    window_size = 3

    # Wyznaczanie lokalnej średniej i odchylenia standardowego za pomocą splotu
    local_mean = uniform_filter(image, size=window_size)
    local_sq_mean = uniform_filter(image**2, size=window_size)
    local_std = np.sqrt(local_sq_mean - local_mean**2)

    # Maska na bazie warunków przetwarzania
    mask = (k0 * mG <= local_mean) & (local_mean <= k1 *
                                         mG) & (k2 * oG <= local_std) & (local_std <= k3 * oG)

    # Nałożenie zmian na obraz
    improved_image = np.where(mask, C * image, image)

    # Obcięcie wartości do zakresu [0, 255]
    improved_image = np.clip(improved_image, 0, 255).astype(np.uint8)

return improved_image

```

```

In [ ]: global img_chest_xray
eq_stat_xray = equalize_histogram_statistically(img_chest_xray)

global img_hidden_symbols
eq_stat_symbols = equalize_histogram_statistically(img_hidden_symbols)

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(eq_stat_xray),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Po korekcji')

```

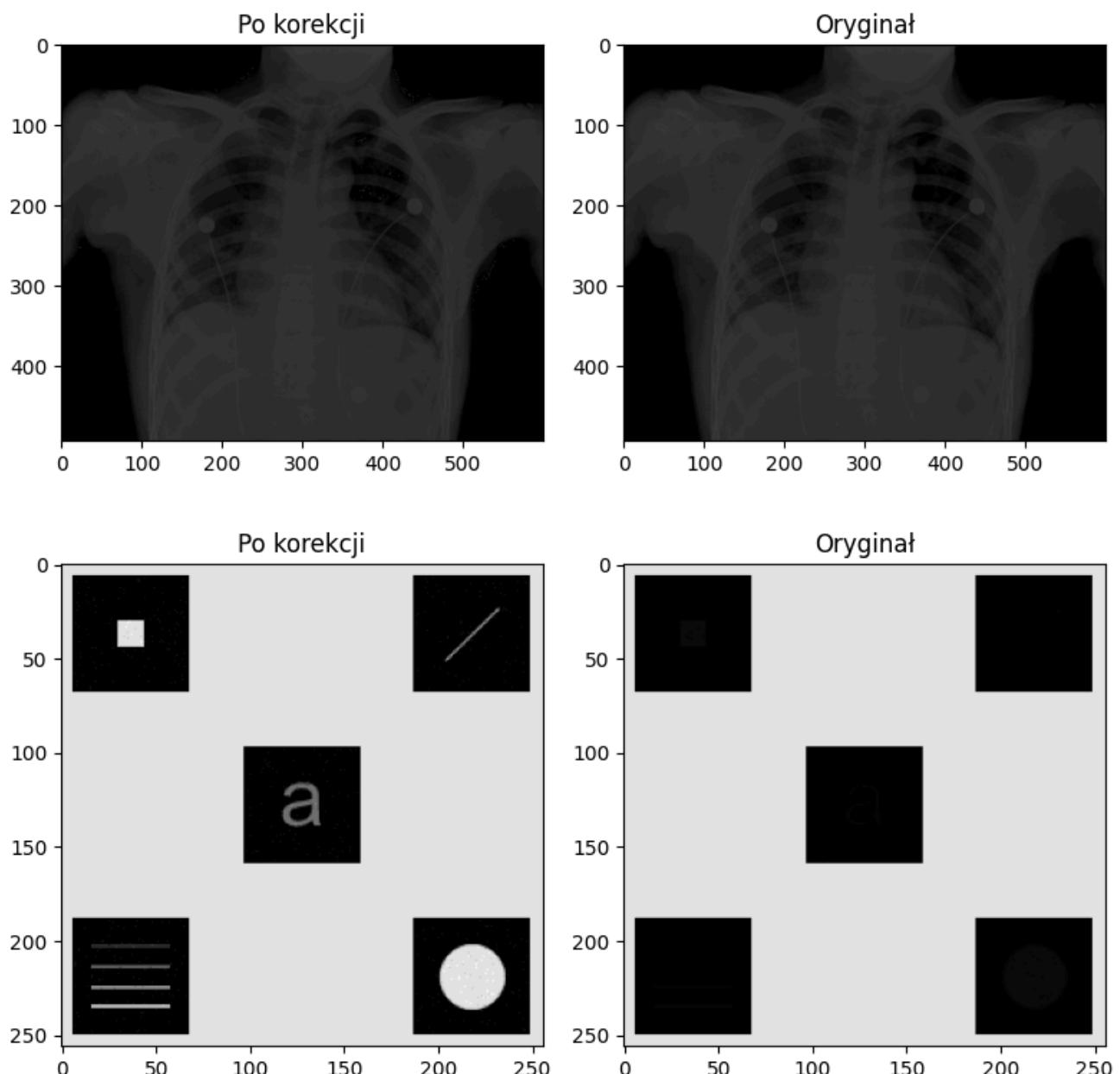
```

ax2.imshow(Image.fromarray(img_chest_xray),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Oryginał')

ax3.imshow(Image.fromarray(eq_stat_symbols),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Po korekcji')
ax4.imshow(Image.fromarray(img_hidden_symbols),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Oryginał')
plt.tight_layout()
plt.show()

```

Figure



Zarówno lokalne wyrównywanie histogramu, jak i poprawa jakości oparta na lokalnych statystykach spełniają swoje zadanie (uwydatnienie ukrytych symboli), natomiast metoda oparta na statystykach nie powoduje tak inwazyjnych zmian w obrazie.

# Zadanie nr 5: Filtracja dolnoprzepustowa

Zbadaj skuteczność redukcji szumu typu „sól i pieprz” za pomocą

- a) liniowego filtra uśredniającego z kwadratową maską, rozpoczynając od maski rozmiaru  $3 \times 3$ .
- b) nieliniowego filtra medianowego
- c) filtrów minimum i maksimum.

```
In [ ]: # funkcja służąca do filtracji usredniającej
def apply_mean_filter(image, filter_size=3):
    kernel = np.ones((filter_size, filter_size), dtype=float) / \
        (filter_size * filter_size)
    filtered_image = convolve(image, kernel, mode='reflect')
    return filtered_image
```

```
In [ ]: # a) Liniowa filtracja uśredniająca z kwadratową maską
img_pepper_only = get_image('resources/cboard_pepper_only.tif')
img_salt_only = get_image('resources/cboard_salt_only.tif')

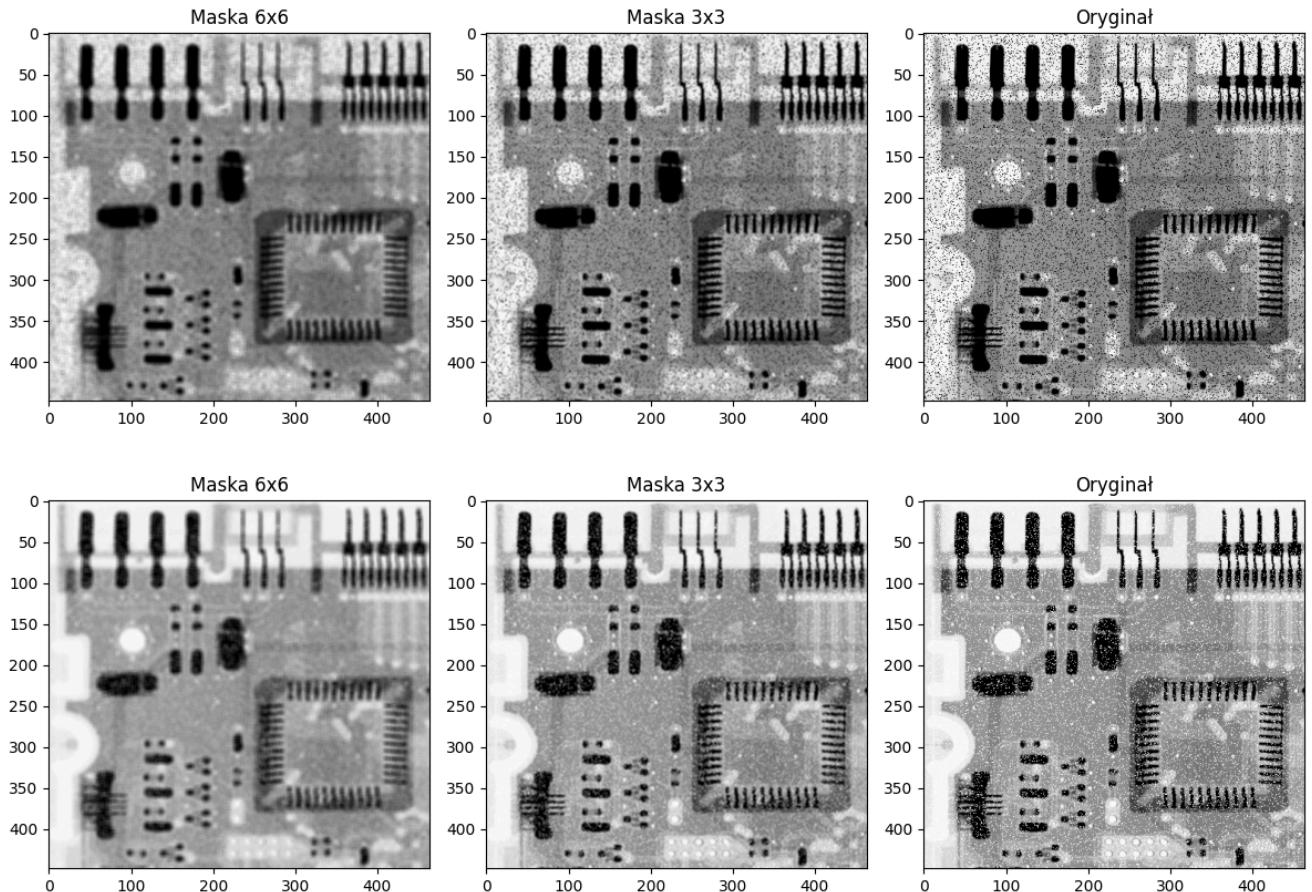
filtered_img_pepper3x3 = apply_mean_filter(img_pepper_only)
filtered_img_salt3x3 = apply_mean_filter(img_salt_only)
filtered_img_pepper6x6 = apply_mean_filter(img_pepper_only, 6)
filtered_img_salt6x6 = apply_mean_filter(img_salt_only, 6)

fig, axes = plt.subplots(2, 3, figsize=(12, 9))
(ax1, ax2, ax3), (ax4, ax5, ax6) = axes

ax1.imshow(Image.fromarray(filtered_img_pepper6x6),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Maska 6x6')
ax2.imshow(Image.fromarray(filtered_img_pepper3x3),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Maska 3x3')
ax3.imshow(Image.fromarray(img_pepper_only),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')

ax4.imshow(Image.fromarray(filtered_img_salt6x6),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Maska 6x6')
ax5.imshow(Image.fromarray(filtered_img_salt3x3),
           cmap='gray', vmin=0, vmax=255)
ax5.set_title('Maska 3x3')
ax6.imshow(Image.fromarray(img_salt_only),
           cmap='gray', vmin=0, vmax=255)
ax6.set_title('Oryginał')
fig.suptitle('Filtrowanie uśredniające z kwadratową maską', fontsize=16)
plt.tight_layout()
plt.show()
```

Figure  
Filtrowanie uśredniające z kwadratową maską



Filtrowanie uśredniające spisuje się zadowalająco dla małego rozmiaru maski, np. 3x3, aczkolwiek efekty nie są tak dobre jak w przypadku następnych metod filtrowania. Dla większego rozmiaru maski powoduje efekt rozmycia szczegółów obrazu.

```
In [ ]: # funkcja służąca do filtracji medianowej
def apply_median_filter(image, filter_size=3):
    filtered_image = median_filter(image, size=filter_size)
    return filtered_image
```

```
In [ ]: # b) filtracja medianowa

global img_pepper_only
global img_salt_only

filtered_img_pepper3x3 = apply_median_filter(img_pepper_only)
filtered_img_salt3x3 = apply_median_filter(img_salt_only)
filtered_img_pepper6x6 = apply_median_filter(img_pepper_only, 6)
filtered_img_salt6x6 = apply_median_filter(img_salt_only, 6)

fig, axes = plt.subplots(2, 3, figsize=(12, 9))
(ax1, ax2, ax3), (ax4, ax5, ax6) = axes

ax1.imshow(Image.fromarray(filtered_img_pepper6x6),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Maska 6x6')
ax2.imshow(Image.fromarray(filtered_img_pepper3x3),
```

```

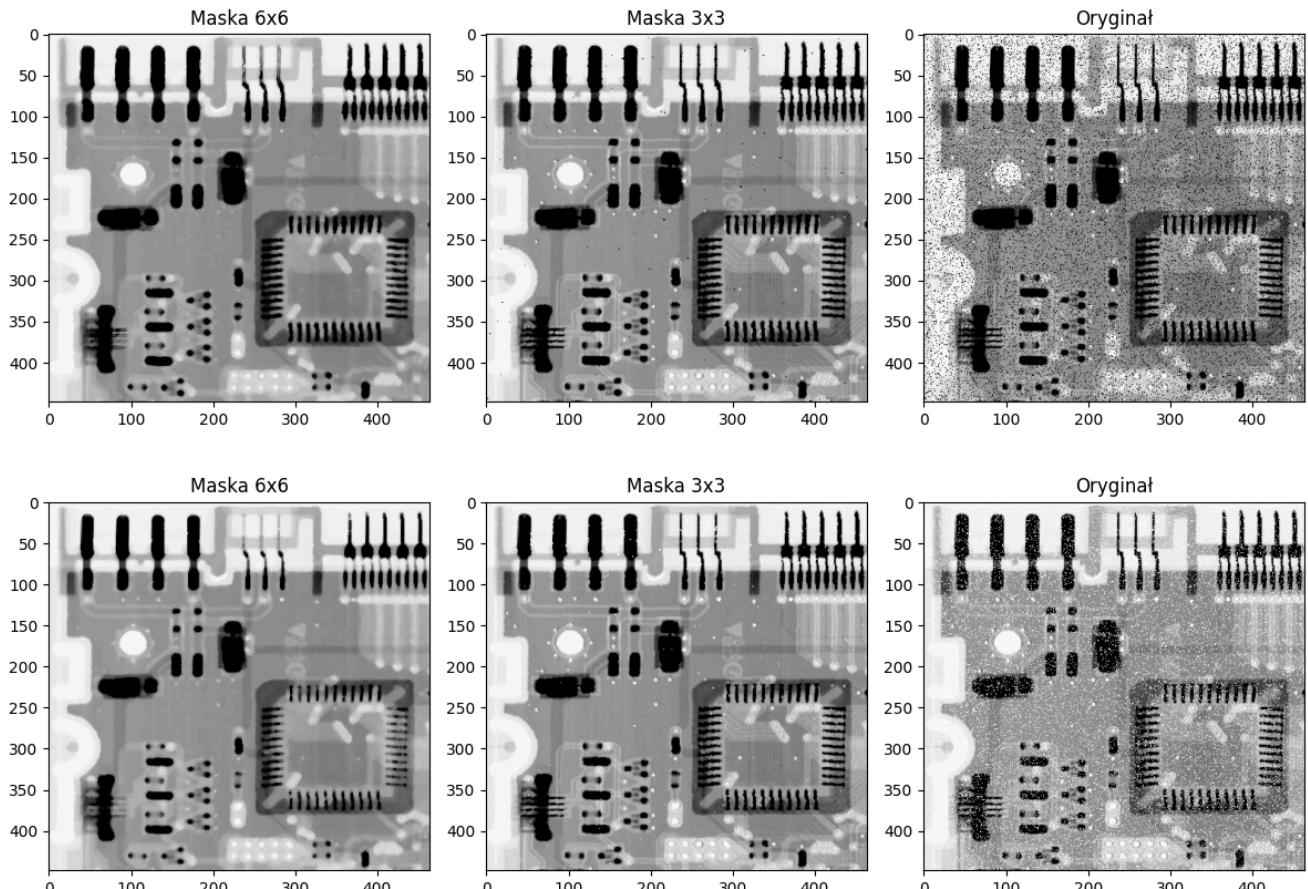
        cmap='gray', vmin=0, vmax=255)
ax2.set_title('Maska 3x3')
ax3.imshow(Image.fromarray(img_pepper_only),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')

ax4.imshow(Image.fromarray(filtered_img_salt6x6),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Maska 6x6')
ax5.imshow(Image.fromarray(filtered_img_salt3x3),
           cmap='gray', vmin=0, vmax=255)
ax5.set_title('Maska 3x3')
ax6.imshow(Image.fromarray(img_salt_only),
           cmap='gray', vmin=0, vmax=255)
ax6.set_title('Oryginał')
fig.suptitle('Filtracja medianowa', fontsize=16)
plt.tight_layout()
plt.show()

```

Figure

## Filtracja medianowa



Filtracja medianowa jest metodą uniwersalną, spisuje się dobrze zarówno dla zakłóceń typu sól, jak i pieprz.

```
In [ ]: # funkcja służąca do filtracji minimalnej
def apply_minimum_filter(image, filter_size=3):
    filtered_image = minimum_filter(image, size=filter_size)
    return filtered_image
```

```
# funkcja służąca do filtracji maksymalnej
def apply_maximum_filter(image, filter_size=3):
    filtered_image = maximum_filter(image, size=filter_size)
    return filtered_image
```

In [ ]: # c) filtracja minimalna i maksymalna

```
global img_pepper_only
global img_salt_only

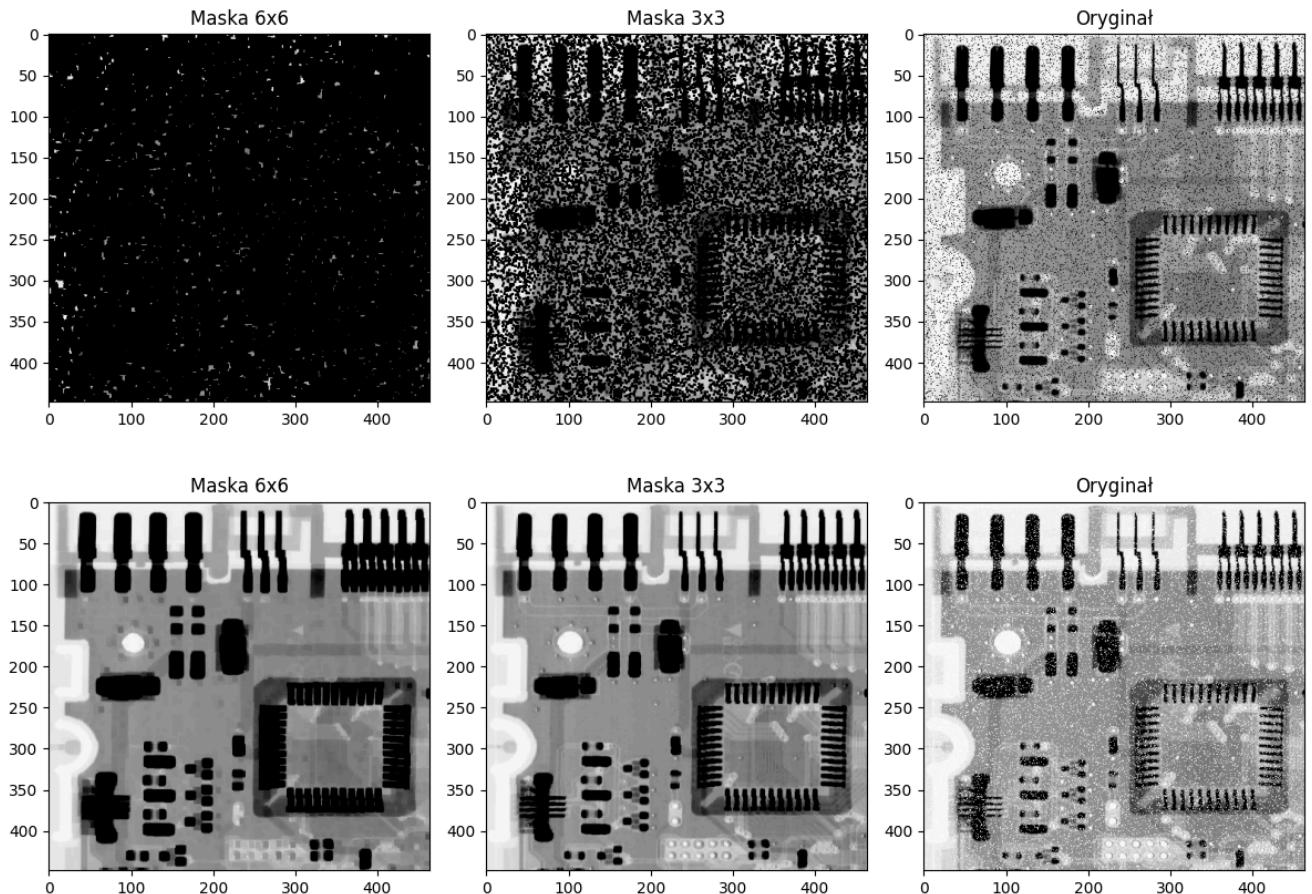
filtered_min_pepper3x3 = apply_minimum_filter(img_pepper_only)
filtered_min_salt3x3 = apply_minimum_filter(img_salt_only)
filtered_min_pepper6x6 = apply_minimum_filter(img_pepper_only, 6)
filtered_min_salt6x6 = apply_minimum_filter(img_salt_only, 6)

fig, axes = plt.subplots(2, 3, figsize=(12, 9))
(ax1, ax2, ax3), (ax4, ax5, ax6) = axes

ax1.imshow(Image.fromarray(filtered_min_pepper6x6),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Maska 6x6')
ax2.imshow(Image.fromarray(filtered_min_pepper3x3),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Maska 3x3')
ax3.imshow(Image.fromarray(img_pepper_only),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')

ax4.imshow(Image.fromarray(filtered_min_salt6x6),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Maska 6x6')
ax5.imshow(Image.fromarray(filtered_min_salt3x3),
           cmap='gray', vmin=0, vmax=255)
ax5.set_title('Maska 3x3')
ax6.imshow(Image.fromarray(img_salt_only),
           cmap='gray', vmin=0, vmax=255)
ax6.set_title('Oryginał')
fig.suptitle('Filtracja minimalna', fontsize=16)
plt.tight_layout()
plt.show()
```

Figure  
Filtracja minimalna



Filtracja minimalna spisuje się bardzo dobrze dla szumu typu sól, natomiast dla szumu typu pieprz prowadzi do zniczenia obrazu.

```
In [ ]: global img_pepper_only
global img_salt_only

filtered_max_pepper3x3 = apply_maximum_filter(img_pepper_only)
filtered_max_salt3x3 = apply_maximum_filter(img_salt_only)
filtered_max_pepper6x6 = apply_maximum_filter(img_pepper_only, 6)
filtered_max_salt6x6 = apply_maximum_filter(img_salt_only, 6)

fig, axes = plt.subplots(2, 3, figsize=(12, 9))
(ax1, ax2, ax3), (ax4, ax5, ax6) = axes

ax1.imshow(Image.fromarray(filtered_max_pepper6x6),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Maska 6x6')
ax2.imshow(Image.fromarray(filtered_max_pepper3x3),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Maska 3x3')
ax3.imshow(Image.fromarray(img_pepper_only),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')

ax4.imshow(Image.fromarray(filtered_max_salt6x6),
           cmap='gray', vmin=0, vmax=255)
```

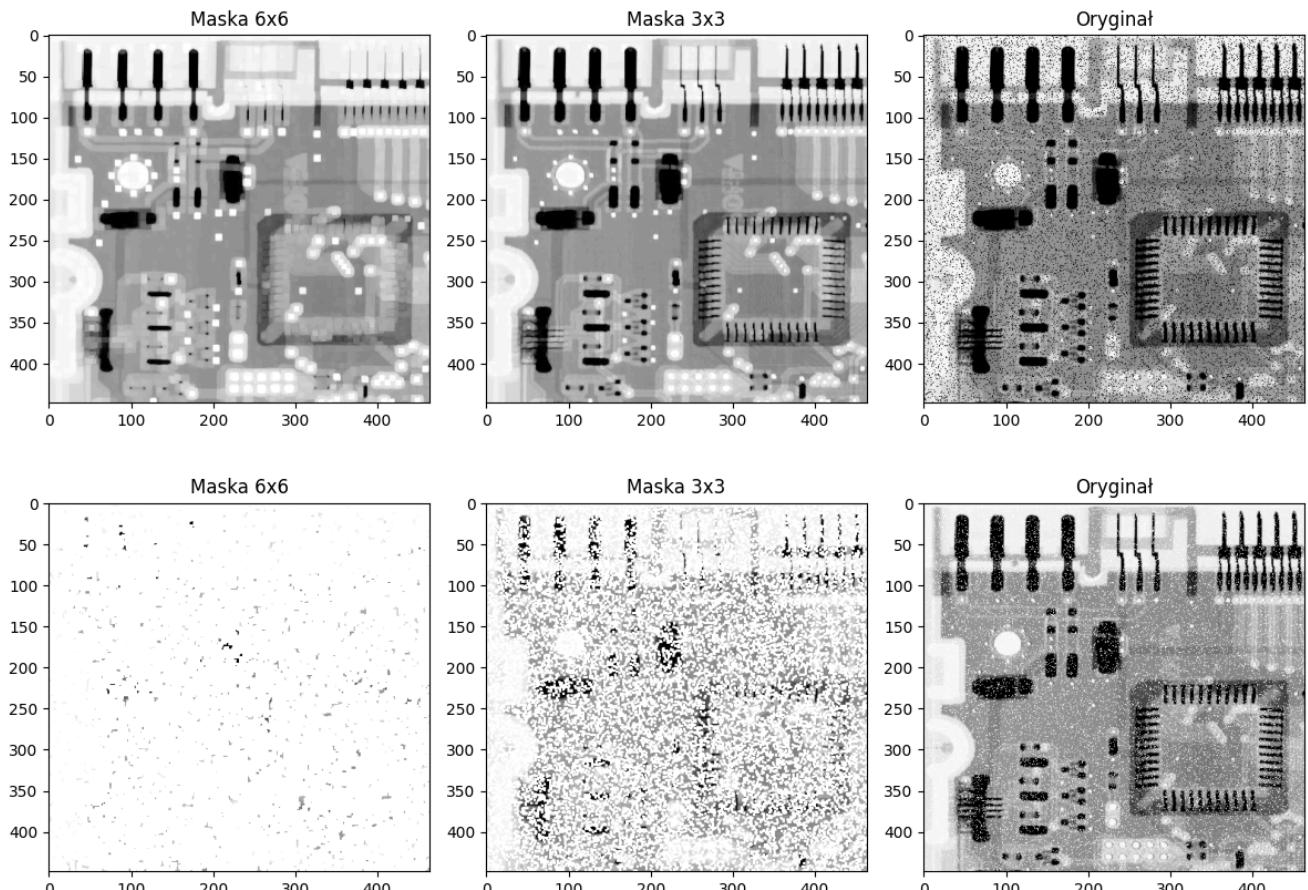
```

        cmap='gray', vmin=0, vmax=255)
ax4.set_title('Maska 6x6')
ax5.imshow(Image.fromarray(filtered_max_salt3x3),
            cmap='gray', vmin=0, vmax=255)
ax5.set_title('Maska 3x3')
ax6.imshow(Image.fromarray(img_salt_only),
            cmap='gray', vmin=0, vmax=255)
ax6.set_title('Oryginał')
fig.suptitle('Filtracja maksymalna', fontsize=16)
plt.tight_layout()
plt.show()

```

Figure

Filtracja maksymalna



Filtracja maksymalna spisuje się bardzo dobrze dla szumu typu pieprz, natomiast dla szumu typu sól prowadzi do zniczenia obrazu.

## Zadanie nr 6

Zbadaj działanie dolnoprzepustowych filtrów uśredniającego i gaussowskiego dla danych obrazów. Zaobserwuj wpływ rozmiaru masek na wynik filtracji.

```
In [ ]: def apply_low_pass_filter(image, filter_size=3):
    filtered_image = uniform_filter(image, size=filter_size)
    return filtered_image
```

```
def apply_gaussian_filter(image, sigma=1):
    filtered_image = gaussian_filter(image, sigma=sigma)
    return filtered_image
```

```
In [ ]: img_zone = get_image('resources/zoneplate.tif')

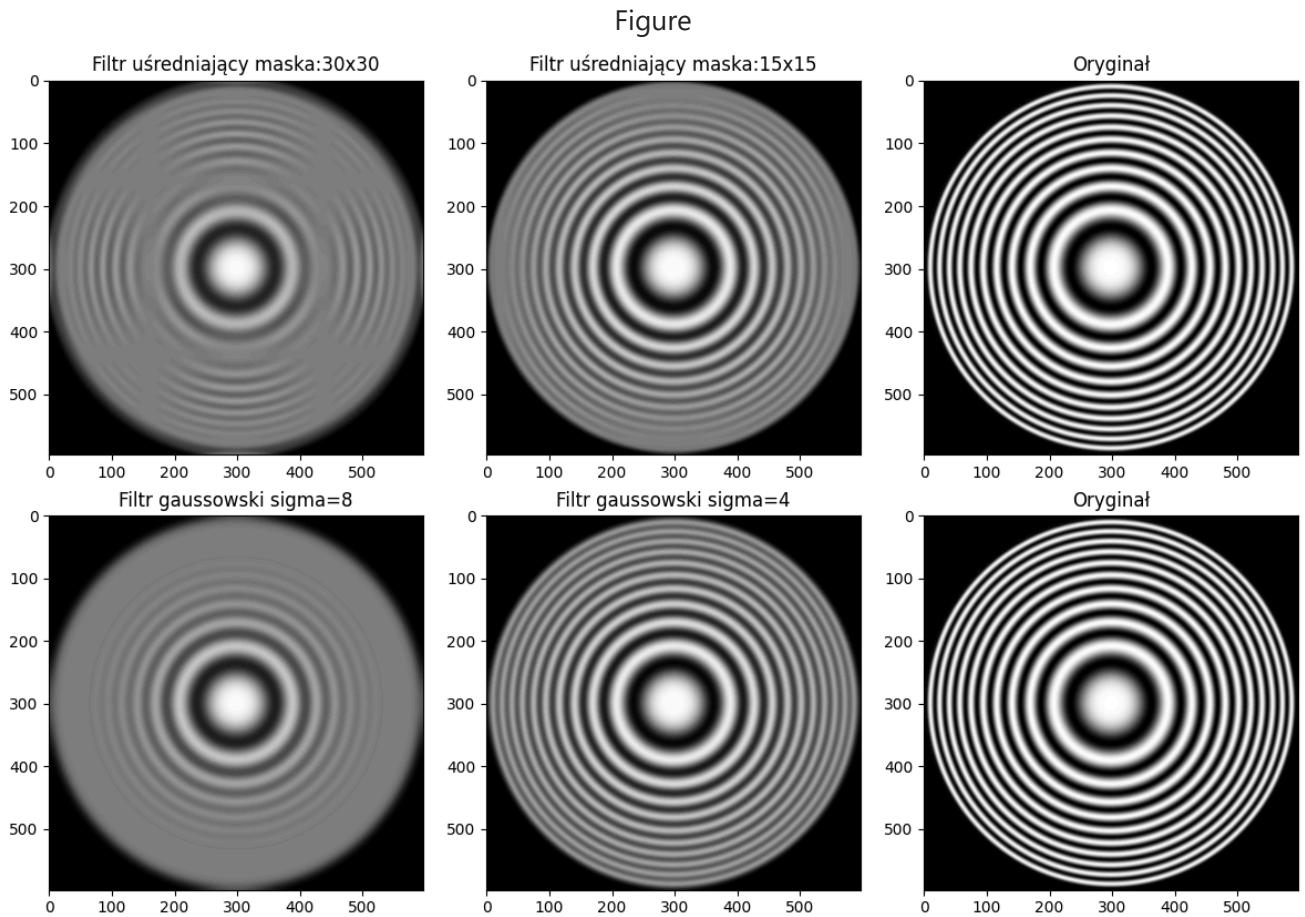
filtered_lowpass_15x15 = apply_low_pass_filter(img_zone, 15)
filtered_lowpass_30x30 = apply_low_pass_filter(img_zone, 30)

filtered_gaussian_s4 = apply_gaussian_filter(img_zone, 4)
filtered_gaussian_s8 = apply_gaussian_filter(img_zone, 8)

fig, axes = plt.subplots(2, 3, figsize=(12, 8))
(ax1, ax2, ax3), (ax4, ax5, ax6) = axes

ax1.imshow(Image.fromarray(filtered_lowpass_30x30),
           cmap='gray', vmin=0, vmax=255)
ax1.set_title('Filtr uśredniający maska:30x30')
ax2.imshow(Image.fromarray(filtered_lowpass_15x15),
           cmap='gray', vmin=0, vmax=255)
ax2.set_title('Filtr uśredniający maska:15x15')
ax3.imshow(Image.fromarray(img_zone),
           cmap='gray', vmin=0, vmax=255)
ax3.set_title('Oryginał')

ax4.imshow(Image.fromarray(filtered_gaussian_s8),
           cmap='gray', vmin=0, vmax=255)
ax4.set_title('Filtr gaussowski sigma=8')
ax5.imshow(Image.fromarray(filtered_gaussian_s4),
           cmap='gray', vmin=0, vmax=255)
ax5.set_title('Filtr gaussowski sigma=4')
ax6.imshow(Image.fromarray(img_zone),
           cmap='gray', vmin=0, vmax=255)
ax6.set_title('Oryginał')
plt.tight_layout()
plt.show()
```



Można zauważyć, że po użyciu filtra uśredniającego dla dużej maski (30x30), pojawia się artefakt w postaci kwadratu na środku obrazu. W przypadku filtra gaussowskiego nie występują tego typu efekty.

## Zadanie nr 7

Wykrywanie krawędzi obiektów i poprawa ostrości.

a) Użyj filtra z maską Sobela do wykrywania krawędzi poziomych, pionowych i ukośnych.

Dane: circuitmask.tif, testpat1.png

b) Zaobserwuj działanie Laplasjanu do wyostrzania szczegółów.

Dane: blurry-moon.tif

c) Zbadaj działanie filtrów typu „unsharp masking” i „high boost”.

Dane: text-dipxe-blurred.tif

```
In [ ]: # a) Filtr Sobela
img_circuitmask = cv.imread('resources/circuitmask.tif', cv.IMREAD_COLOR)

img_circuitmask = cv.GaussianBlur(img_circuitmask, (3, 3), 0)
gray = cv.cvtColor(img_circuitmask, cv.COLOR_BGR2GRAY)

grad_x = cv.Sobel(gray, cv.CV_16S, 1, 0, ksize=3, scale=1,
                  delta=0, borderType=cv.BORDER_DEFAULT)
```

```
grad_y = cv.Sobel(gray, cv.CV_16S, 0, 1, ksize=3, scale=1,
                  delta=0, borderType=cv.BORDER_DEFAULT)

grad_x = cv.convertScaleAbs(grad_x)
grad_y = cv.convertScaleAbs(grad_y)

grad = cv.addWeighted(grad_x, 0.5, grad_y, 0.5, 0)

fig, axes = plt.subplots(2, 2, figsize=(8, 8))
(ax1, ax2), (ax3, ax4) = axes

ax1.imshow(Image.fromarray(img_circuitmask))
ax1.set_title('Oryginał')

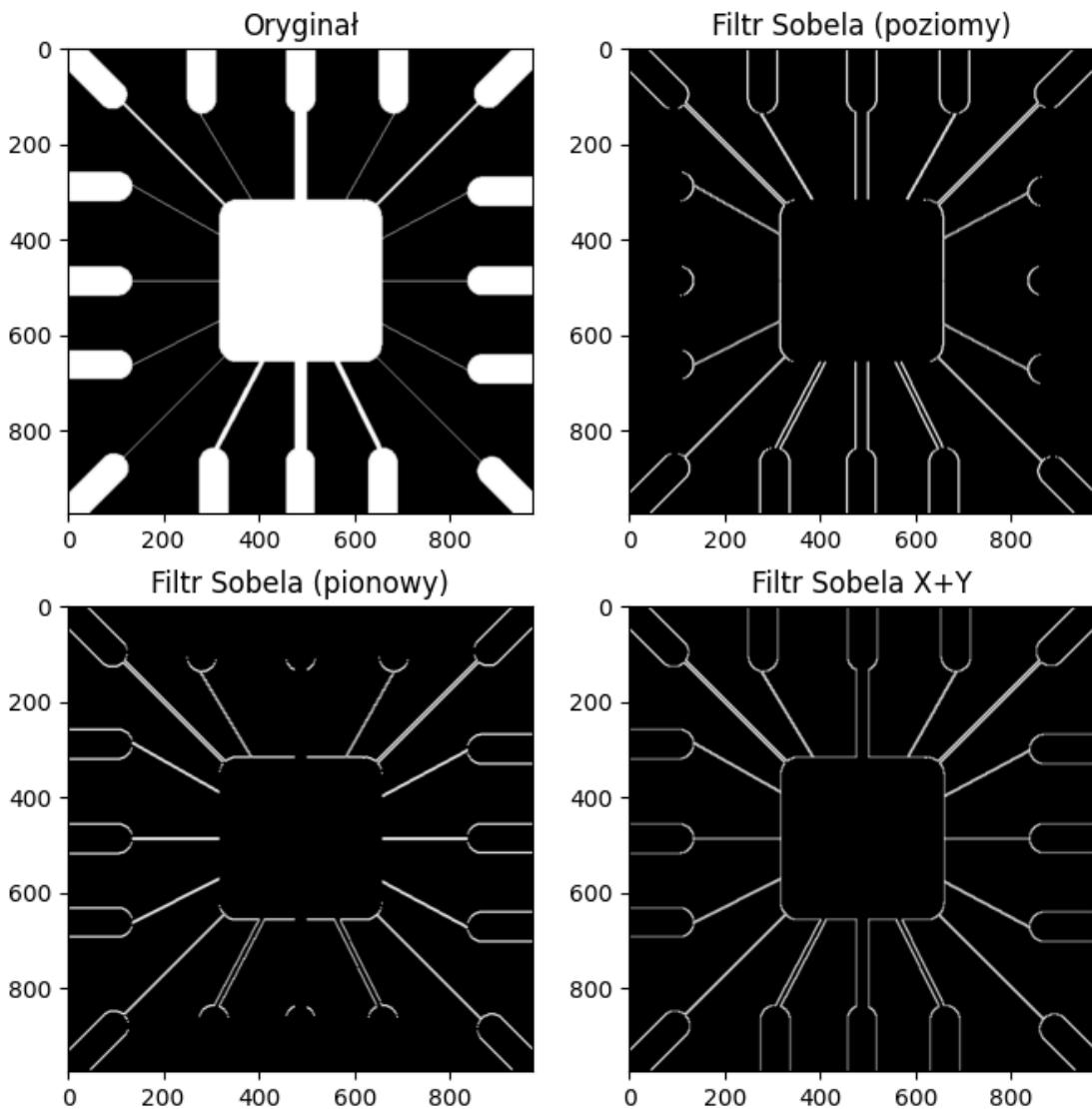
ax2.imshow(grad_x, cmap='gray', vmin=0, vmax=255)
ax2.set_title('Filtr Sobela (poziomy)')

ax3.imshow(grad_y, cmap='gray', vmin=0, vmax=255)
ax3.set_title('Filtr Sobela (pionowy)')

ax4.imshow(grad, cmap='gray', vmin=0, vmax=255)
ax4.set_title('Filtr Sobela X+Y')
```

Out[ ]: Text(0.5, 1.0, 'Filtr Sobela X+Y')

Figure



```
In [ ]: # b) Filtr Laplace'a
img_moon = cv.imread('resources/blurry-moon.tif', cv.IMREAD_COLOR)

moon = cv.GaussianBlur(img_moon, (3, 3), 0)
moon_gray = cv.cvtColor(moon, cv.COLOR_BGR2GRAY)

laplasjan = cv.Laplacian(moon_gray, cv.CV_16S, ksize=3)
laplasjan = cv.convertScaleAbs(laplasjan)

laplasjan_scaled = cv.Laplacian(moon_gray, cv.CV_16S, ksize=3,
                                 delta=10, scale=2, borderType=cv.BORDER_DEFAULT, dst=None)
laplasjan_scaled = cv.convertScaleAbs(laplasjan_scaled)

laplacian = cv.Laplacian(moon_gray, cv.CV_64F)
laplacianabs = cv.convertScaleAbs(laplacian)
sharpened = cv.add(moon_gray, laplacianabs)
```

```

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

plt.subplot(1, 3, 1)
plt.title('Obraz oryginalny')
plt.imshow(img_moon, cmap='gray', vmin=0, vmax=255)

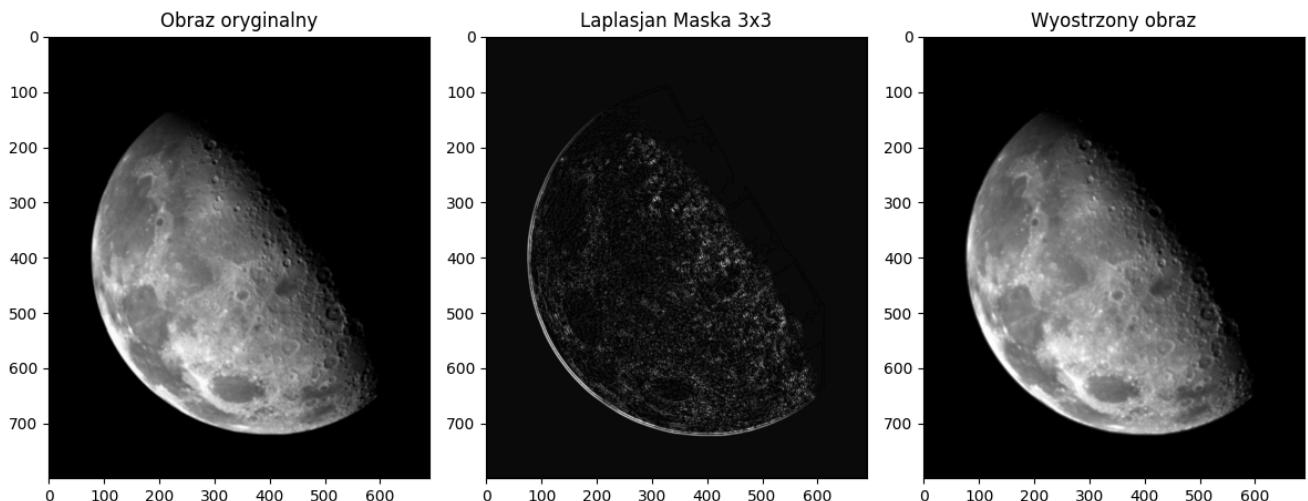
plt.subplot(1, 3, 2)
plt.title('Laplasjan Maska 3x3')
plt.imshow(laplasjan_scaled, cmap='gray', vmin=0, vmax=255)

plt.subplot(1, 3, 3)
plt.title('Wyostrzony obraz')
plt.imshow(sharpened, cmap='gray', vmin=0, vmax=255)

plt.tight_layout()
plt.show()

```

Figure



In [ ]: # c) Filtry typu „unsharp masking” i „high boost”.

```

img_dipxe = cv.imread(
    'resources/text-dipxe-blurred.tif', cv.IMREAD_COLOR)

gaussian_blur_dipxe = cv.GaussianBlur(img_dipxe, (31, 31), 5)

gmask = img_dipxe - gaussian_blur_dipxe

k = 1
unsharp_masking = img_dipxe + k*gmask

k = 5
high_boost = img_dipxe + k*gmask

plt.figure(figsize=(12, 8))

plt.subplot(3, 2, 1)
plt.title('Obraz oryginalny')
plt.imshow(img_dipxe, cmap='gray', vmin=0, vmax=255)

```

```

plt.subplot(3, 2, 2)
plt.title('Rozmyty filtrem Gaussa')
plt.imshow(gaussian_blur_dipxe, cmap='gray', vmin=0, vmax=255)

plt.subplot(3, 2, 3)
plt.title('Nieostra maska')
plt.imshow(gmask, cmap='gray', vmin=0, vmax=255)

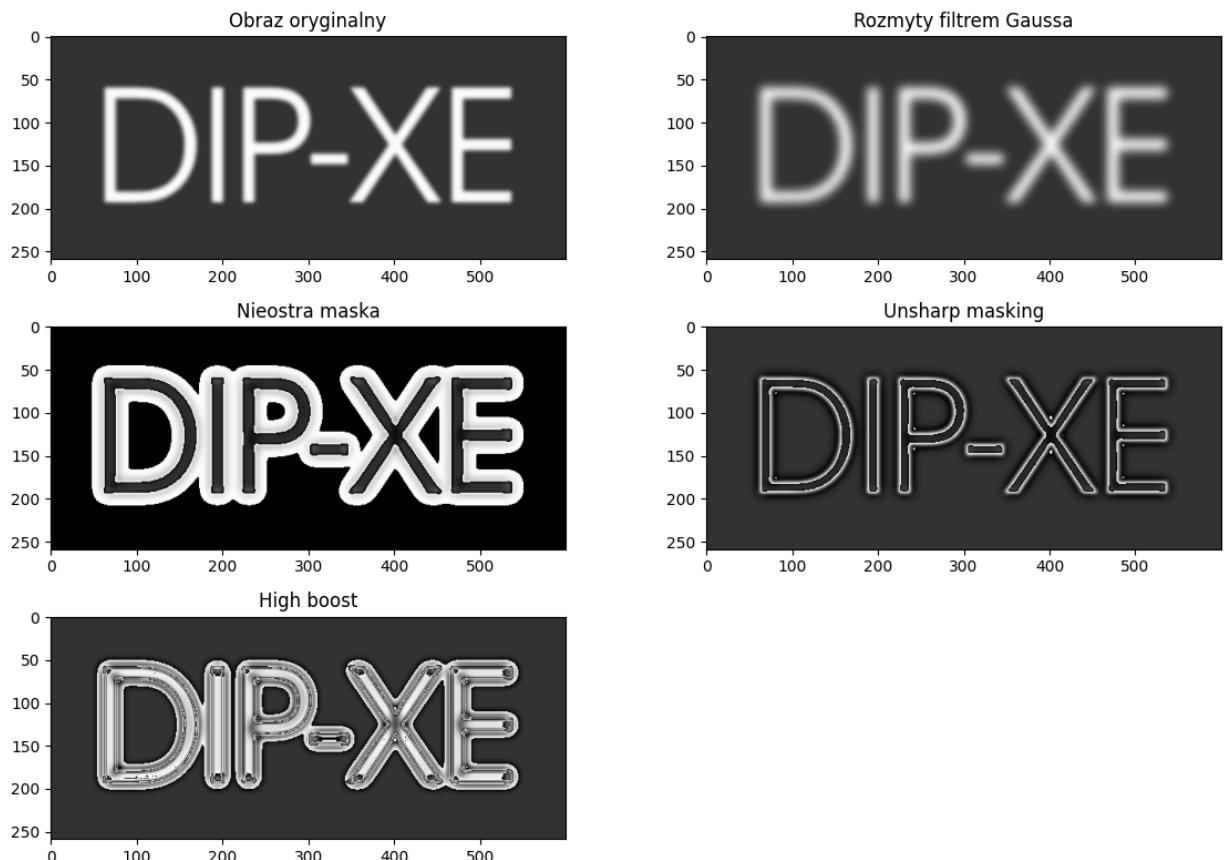
plt.subplot(3, 2, 4)
plt.title('Unsharp masking')
plt.imshow(unsharp_masking, cmap='gray', vmin=0, vmax=255)

plt.subplot(3, 2, 5)
plt.title('High boost')
plt.imshow(high_boost, cmap='gray', vmin=0, vmax=255)

plt.tight_layout()
plt.show()

```

Figure



## Zadanie nr 8

Naszym celem jest poprawa jakości obrazu za pomocą kolejnego stosowania różnych przekształceń i filtrów. Zastosuj złożone, wieloetapowe podejście do poprawy jakości przedstawione na wykładzie pt. „Filtracja w dziedzinie przestrzennej”.

Dane: bonescan.tif

```
In [ ]: def normalize_image(image):
    norm_image = cv.normalize(image, None, 0, 255, cv.NORM_MINMAX)
    return cv.convertScaleAbs(norm_image)

# Poniżej przeprowadzono wieloetapowe podejście do poprawy jakości obrazu
# Powtórzone wszystkie kroki przedstawione na wykładzie nr 4

# a) Obraz oryginalny
img_bonescan = get_image('resources/bonescan.tif')

# b) Laplasjan obrazu (a) z maską  $3 \times 3$ 
laplasjan = cv.Laplacian(img_bonescan, cv.CV_16S, ksize=3,
                         scale=8, delta=0, borderType=cv.BORDER_DEFAULT)
laplasjan = cv.convertScaleAbs(laplasjan, alpha=0.2)

# c) Suma obrazów (a) i (b)
sum = cv.add(img_bonescan, laplasjan)

# d) Gradient Sobela obrazu (a) uwydatnienie brzegów
sobel_x = cv.Sobel(img_bonescan, cv.CV_16S, 1, 0, ksize=3)
sobel_y = cv.Sobel(img_bonescan, cv.CV_16S, 0, 1, ksize=3)
sobel_x = cv.convertScaleAbs(sobel_x, alpha=4)
sobel_y = cv.convertScaleAbs(sobel_y, alpha=4)
sobel = cv.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)

# e) Filtracja uśredniająca
# z maską  $5 \times 5$  obrazu (d)
# • redukcja szumu uwydatnionego
# przez laplasjan
mean_filtered = apply_mean_filter(sobel.copy(), 5)

# f) Iloczyn obrazu (e) i Laplasjanu (b)
mlt_bonescan = cv.bitwise_and(mean_filtered, laplasjan)
mlt_bonescan = normalize_image(mlt_bonescan)

# (g) Suma (a) i (f)
sum_bonescan = cv.add(img_bonescan, mlt_bonescan)

# (h) Transformacja potęgowa (g),  $c = 1$ ,  $\gamma = 0,5$ 
def contrast_stretching_transform_bs(x): return 255*1*np.power(x/255, 0.5)

img_contrast = image_transform_optimized(
    sum_bonescan, contrast_stretching_transform_bs)

plt.figure(figsize=(8, 48))

plt.subplot(8, 1, 1)
plt.title('Obraz oryginalny')
plt.imshow(img_bonescan, cmap='gray', vmin=0, vmax=255)

plt.subplot(8, 1, 2)
plt.title('Laplasjan maska 3x3')
plt.imshow(laplasjan, cmap='gray', vmin=0, vmax=255)

plt.subplot(8, 1, 3)
```

```
plt.title('Suma obrazów (a) i (b)')
plt.imshow(sum, cmap='gray', vmin=0, vmax=255)

plt.subplot(8, 1, 4)
plt.title('Gradient Sobela obrazu (a) uwydatnienie brzegów')
plt.imshow(sobel, cmap='gray', vmin=0, vmax=255)

plt.subplot(8, 1, 5)
plt.title('Filtr uśredniający z maską 5x5')
plt.imshow(mean_filtered, cmap='gray', vmin=0, vmax=255)

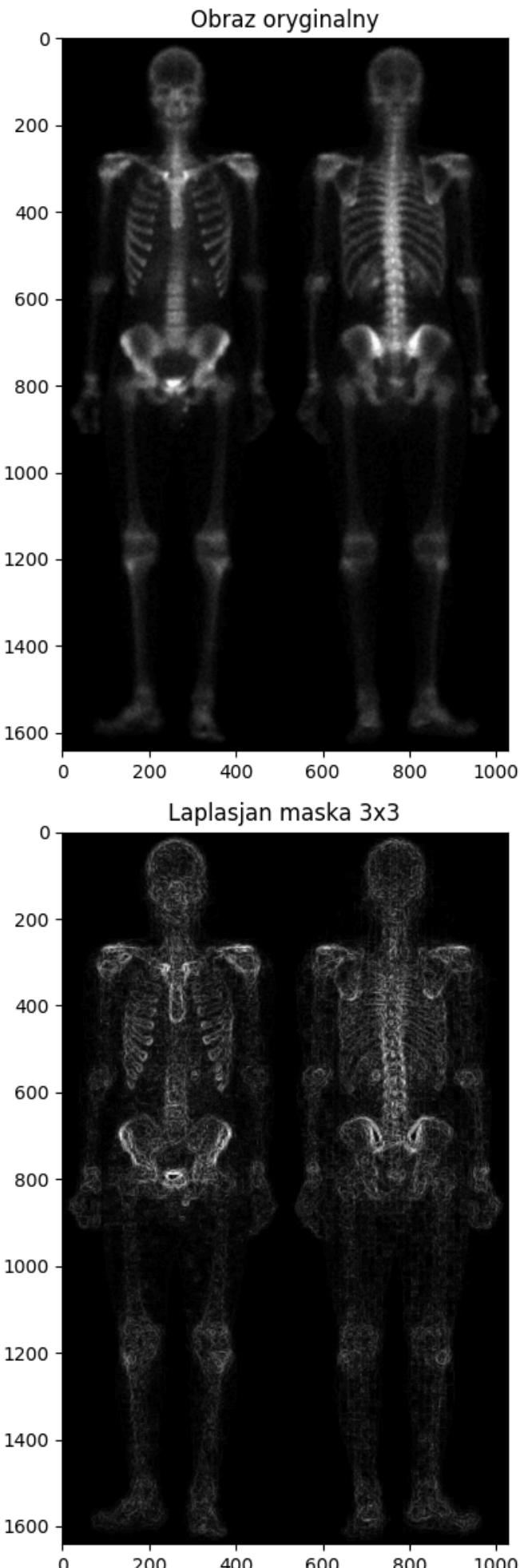
plt.subplot(8, 1, 6)
plt.title('Iloczyn obrazu (e) i laplasjanu (b)')
plt.imshow(mlt_bonescan, cmap='gray', vmin=0, vmax=255)

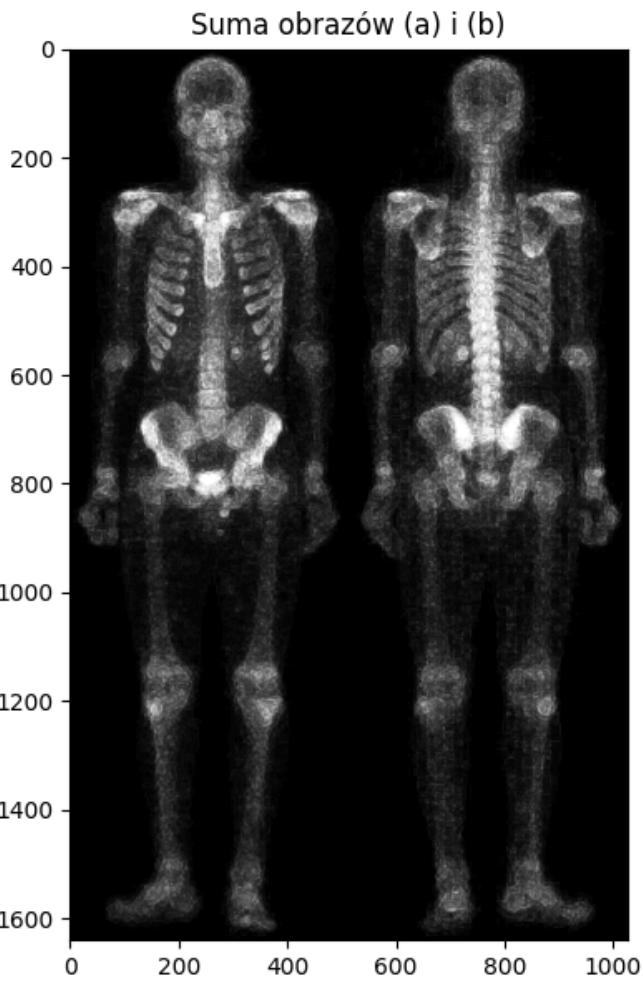
plt.subplot(8, 1, 7)
plt.title('Suma (a) i (f)')
plt.imshow(sum_bonescan, cmap='gray', vmin=0, vmax=255)

plt.subplot(8, 1, 8)
plt.title('Transformacja potęgowa (g)')
plt.imshow(img_contrast, cmap='gray', vmin=0, vmax=255)

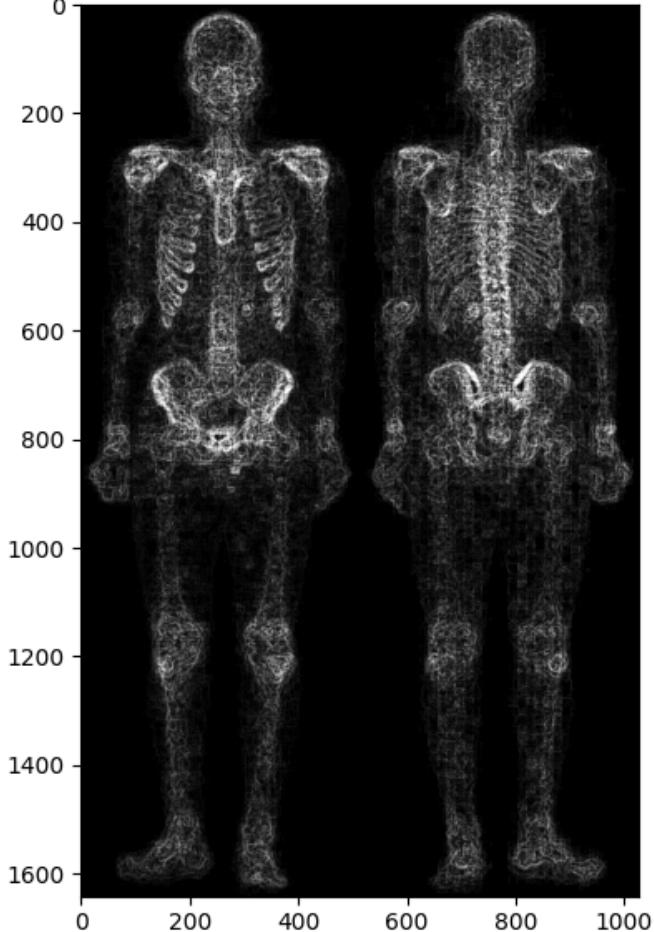
plt.tight_layout()
plt.show()
```

Figure



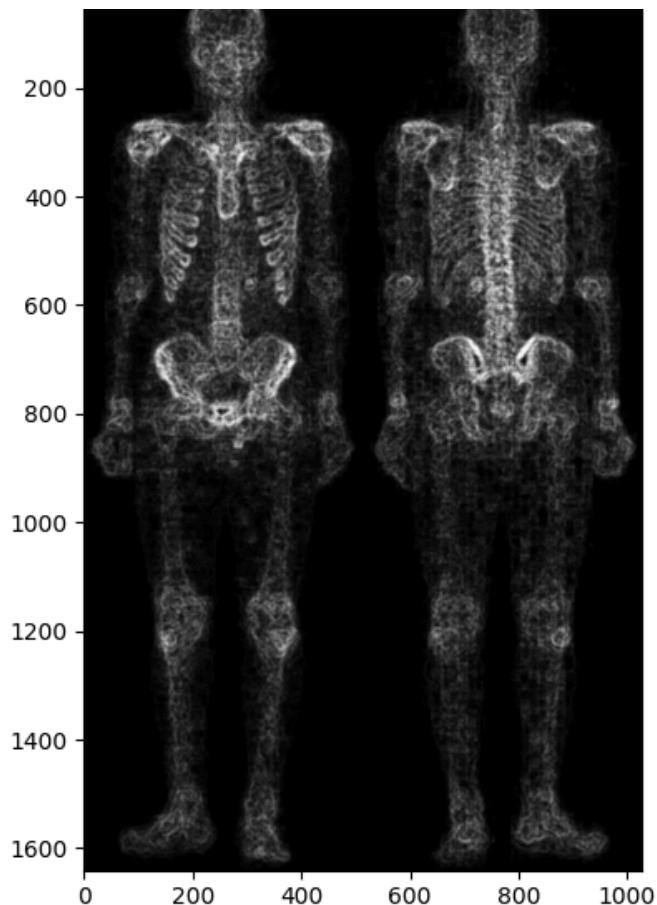


Gradient Sobela obrazu (a) uwydatnienie brzegów

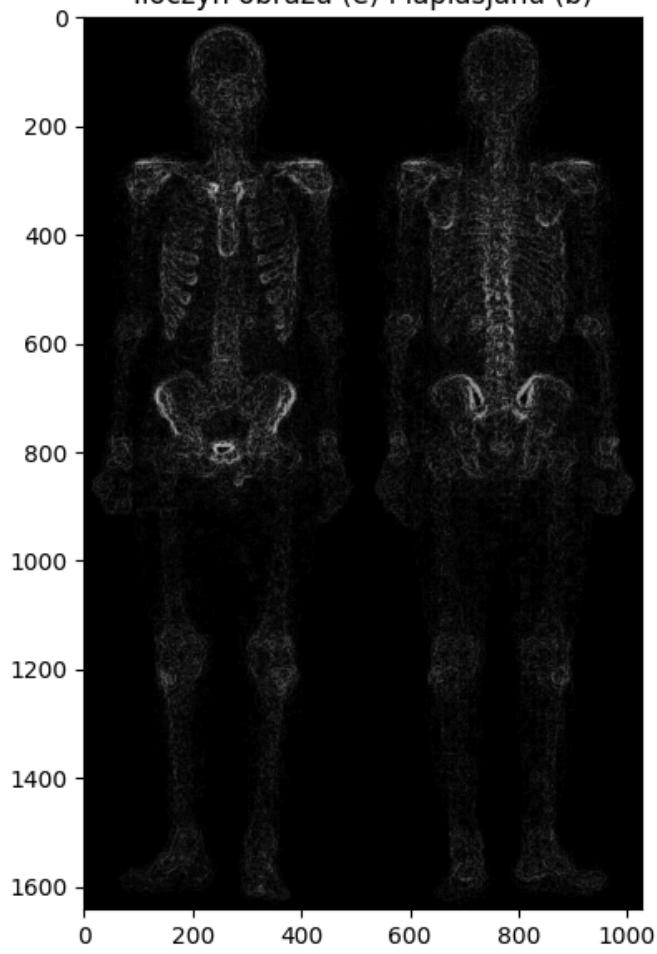


Filtr uśredniający z maską 5x5

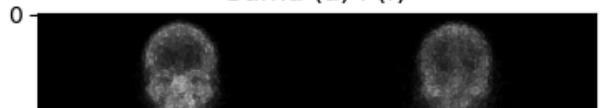


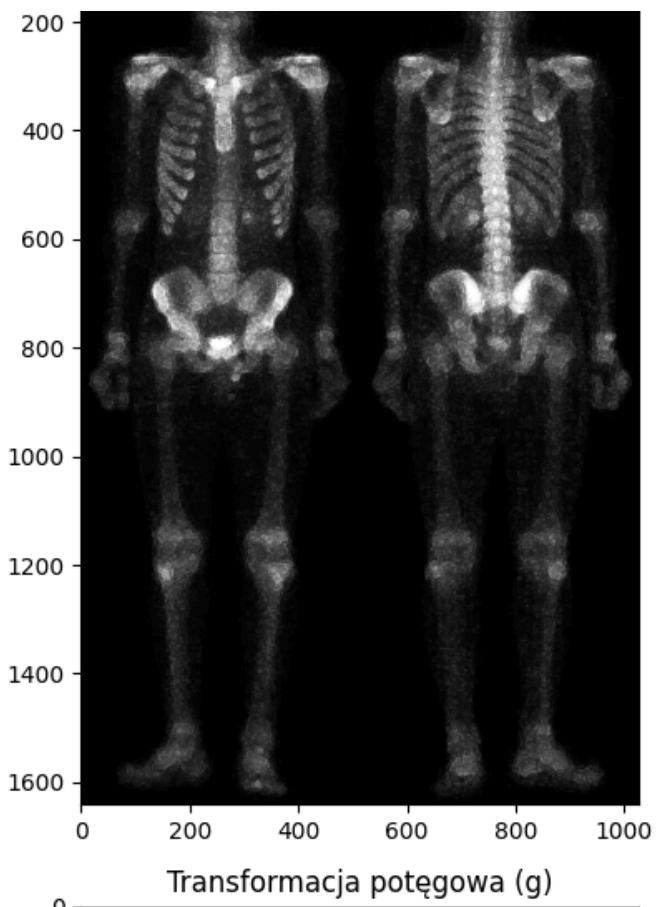


Iloczyn obrazu (e) i laplasjanu (b)

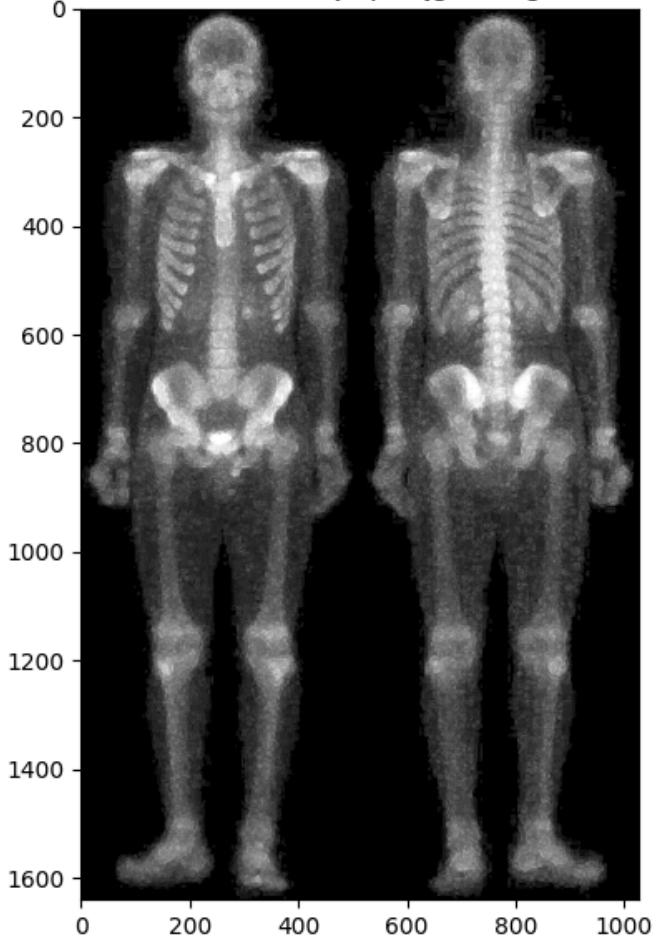


Suma (a) i (f)





Transformacja potęgowa (g)



Po dostosowaniu parametrów filtrów obraz został poprawiony i uwydatnione zostały jego brzegi.

Wykorzystując metody przetwarzania obrazów z biblioteki OpenCV, manipulowano głównie parametrem alpha, który skaluje wartości gradientu Sobela przed ich konwersją. Jest to użyteczne, gdy wartości

gradientu są zbyt małe i chcemy je wzmacnić (uwydatnić brzegi). Skalowanie powoduje, że różnice między wartościami pikseli są bardziej wyraźne, co może pomóc w lepszym wykrywaniu krawędzi. W przypadku laplasjanu,  $\alpha=0.2$  zmniejsza wartości pikseli do 20% ich pierwotnej wartości.