

# Laboratorium 2

Autorzy: Krzysztof Zalewa 273032, Michał Pakuła 272828

Data: 7 Kwietnia 2025

## Ćwiczenie 5

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.

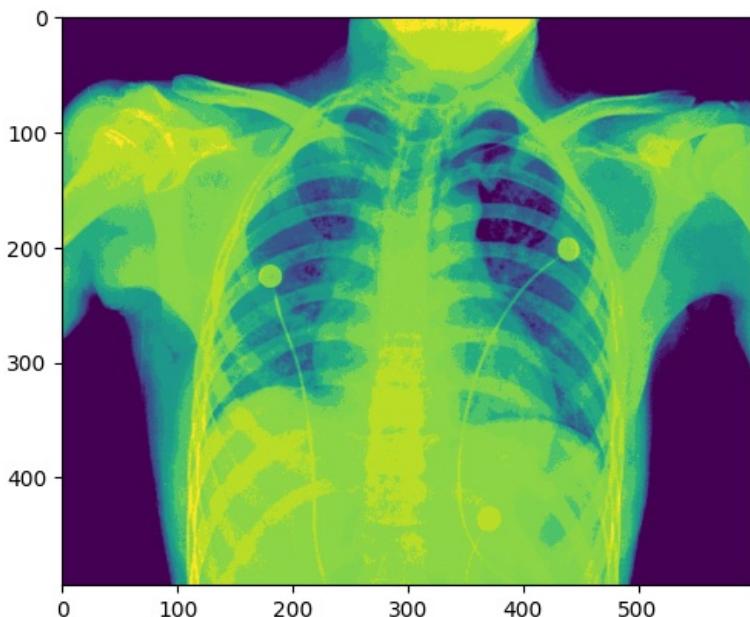
```
In [12]: import matplotlib.pyplot as plt
import tifffile as tiff
import numpy as np
import os
```

### Zadanie 1

```
In [14]: # Załadowanie pliku .tiff
if os.name == 'nt':
    file_name = "./src/" + input("Podaj nazwe pliku z danymi: ")
elif os.name == 'posix':
    file_name = "./src//"+input("Podaj nazwe pliku z danymi: ")
else:
    print("Nieznaný system")
img = tiff.imread(file_name)
```

```
In [ ]: #Wyświetlenie załadowanego obrazu
plt.figure()
plt.imshow(img)

plt.show()
```



Obraz został wczytany z pliku o podanej nazwie z rozszerzeniem i wyświetlony za pomocą biblioteki matplotlib.

Skrypt działa zarówno w systemie Linux, jak i Windows - ścieżka do pliku jest automatycznie dostosowywana w zależności od wykrytego systemu operacyjnego.

### Zadanie 2

Stworzenie histogramu obrazu

Histogram jest tworzony na podstawie jednej linii (poziomej lub pionowej) która jest wybrana przez użytkownika

```
In [32]: if len(img.shape)==3:
    img = np.mean(img, axis=2).astype(np.uint8)
mode = input("Podaj pozioma/pionowa: ")
```

```

if( mode == "pozioma"):
    line_num = int(input("Podaj wysokość: "))
    gray_val = img[line_num,:]

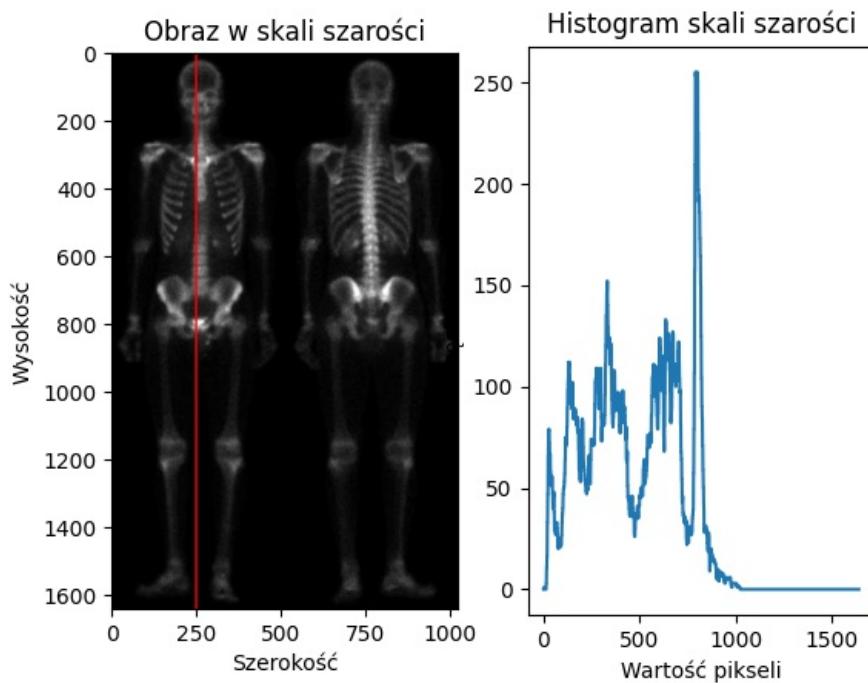
elif(mode == "pionowa"):
    line_num = int(input("Podaj szerokość: "))
    gray_val = img[:,line_num]

else:
    print("Eee")

if(mode == "pozioma"):
    x = np.arange(img.shape[1])
    y = line_num * np.ones(img.shape[1])
elif mode == "pionowa":
    x = line_num * np.ones(img.shape[0])
    y = np.arange(img.shape[0])

plt.subplot(1, 2, 1)
plt.plot(x, y, color='red', linewidth=1)
plt.imshow(img, cmap='gray')
plt.title('Obraz w skali szarości')
plt.xlabel('Szerokość')
plt.ylabel('Wysokość')
plt.subplot(1, 2, 2)
plt.plot(gray_val)
plt.title('Histogram skali szarości')
plt.xlabel('Wartość pikseli')
plt.ylabel('Częstość')
plt.show()

```



Na histogramie widoczne są poziomy szarości pikseli znajdujących się na wybranej linii obrazu

Oś X przedstawia pozycję piksela na tej linii, natomiast oś Y odpowiada wartości poziomu szarości (od 0 do 255).

Analizując wykres, można zauważać rozkład jasności na tej konkretnej linii obrazu, co pozwala ocenić kontrast, obecność struktur lub jednolitość tła w tym fragmencie zdjęcia.

### Zadanie 3

Stworzenie wycinka obrazu

Użytkownik podaje współrzędne a następnie wyznacza szerokość oraz wysokość wycinka od danego punktu

```

In [37]: try:
    x, y, w, h = map(int, input("Podaj współrzędne X, Y, szerokość i wysokość prostokąta (oddzielone spacjami):"))

    if w <= 0 or h <= 0:
        print("Szerokość i wysokość muszą być dodatnie.")

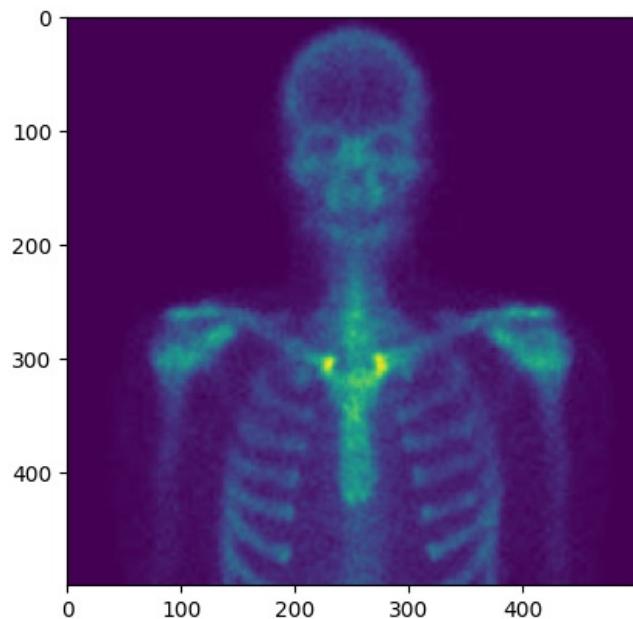
except ValueError:
    print("To nie są liczby całkowite. Proszę spróbować ponownie.")

region_of_intrest = img[y:y+h , x:x+w]

plt.figure()
plt.imshow(region_of_intrest)

```

```
plt.show()
```



Podane współrzędne są skrajną współrzędną w lewym górnym rogu, a następnie wycinek rozszerzany jest w stronę prawą oraz w dół dając odcinek wynikowy.

## Ćwiczenie 6

Zaobserwuj działanie następujących przekształceń punktowych:

1. Mnożenie obrazu przez stałą
2. Transformacja logarytmiczna
3. Zmiana dynamiki skali szarości (kontrastu)
4. Korekcja gamma

```
In [1]: import matplotlib.pyplot as plt
import tifffile as tiff
import numpy as np
import os
```

Załadowanie wyznaczonych plików:

```
In [17]: # Załadowanie pliku .tif
img_a = tiff.imread("src/pollen-dark.tif")
img_b = tiff.imread("src/spectrum.tif")
img_c = tiff.imread("src/einstein-low-contrast.tif")
img_d = tiff.imread("src/aerial_view.tif")
```

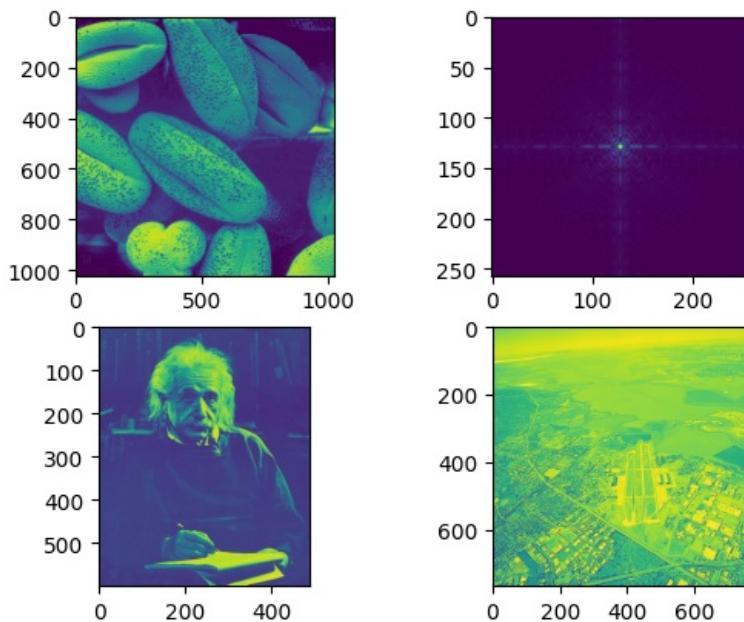
Obrazy bazowe:

```
In [19]: # Wyświetlenie załadowanego obrazu
plt.figure()
plt.subplot(2,2,1)
plt.imshow(img_a)

plt.subplot(2,2,2)
plt.imshow(img_b)

plt.subplot(2,2,3)
plt.imshow(img_c)

plt.subplot(2,2,4)
plt.imshow(img_d)
plt.show()
```



## Funkcje przekształcające

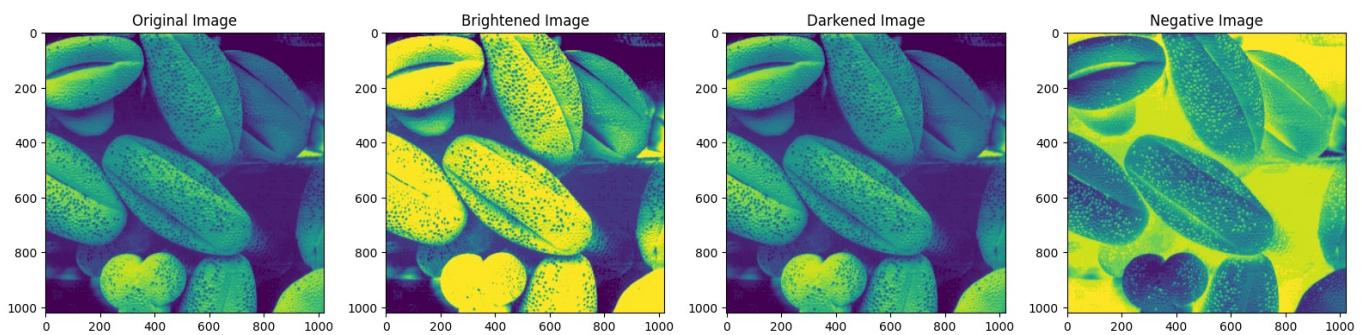
Przyjmujemy że 'r', to nasz przetwarzany obraz  
c jest stałą podaną przez użytkownika

1.  $T(r) = c * r$
2.  $T(r) = c * \log(1+r)$
3.  $T(r) = 1 / (1 + (m/r)^e)$   
*m oraz e są ustalonymi wartościami całkowitymi*
4.  $T(r) = c * (r^\gamma)$ ; gdzie  $c > 0$  oraz  $\gamma > 0$

In [254]

```
def TransformByConst(img, c=1.0, inv_factor=0.0):
    """
    Transform the image using a constant c.
    """
    max_val = np.iinfo(img.dtype).max
    transformed_img = c * img.astype(np.float32)
    inverted = max_val - transformed_img
    result = (1 - inv_factor) * transformed_img + inv_factor * inverted
    return np.clip(result, 0, max_val).astype(img.dtype)

#pollen-dark.tif
processing = img_a
plt.figure(figsize=(16, 4))
plt.subplot(1,4,1)
plt.title("Original Image")
plt.imshow(processing)
# rozjaśnienie
plt.subplot(1,4,2)
plt.title("Brightened Image")
img_an = TransformByConst(processing, c = 5.0)
plt.imshow(img_an)
# sciemienienie
plt.subplot(1,4,3)
plt.title("Darkened Image")
img_ab = TransformByConst(processing, c = 0.6)
plt.imshow(img_ab)
# negatyw
plt.subplot(1,4,4)
plt.title("Negative Image")
img_ac = TransformByConst(processing, c = 1.0, inv_factor = 0.6)
plt.imshow(img_ac)
plt.tight_layout()
plt.show()
```



## 1. Przekształcenie przez stałą

Przekształcenie przy pomocy mnożenia przez stałą pozwala na zwiększenie (wartości > 1) lub zmniejszenie (wartości w zakresie (0,1)) jasności obrazu.

Mnożenie przez liczby ujemne daje obraz negatywowy gdzie szarości odwrócone są liniowo.

In [255]

```
def LogarithmicTransform(img, c):
    """
    Apply logarithmic transformation to the image.
    """
    img_float = img.astype(np.float32) + 1e-9 # Avoid log(0)
    return c * np.log(img_float) # Apply log only to positive values

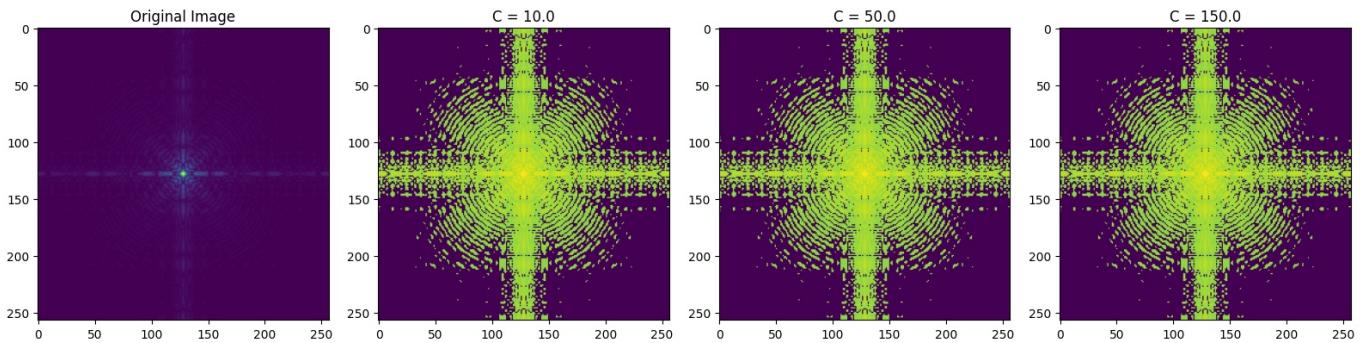
#spectrum.tif
processing = img_b
plt.figure(figsize=(16, 4))
plt.subplot(1,4,1)
plt.title("Original Image")
plt.imshow(processing)

plt.subplot(1,4,2)
img_ba = LogarithmicTransform(processing, c = 10.0)
plt.title("C = 10.0")
plt.imshow(img_ba)

plt.subplot(1,4,3)
img_bb = LogarithmicTransform(processing, c = 50.0)
plt.title("C = 50.0")
plt.imshow(img_bb)

plt.subplot(1,4,4)
img_bc = LogarithmicTransform(processing, c = 150.0)
plt.title("C = 150.0")
plt.imshow(img_bc)
```

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



## 2. Przekształcenie logarytmiczne

Przekształcenie logarytmiczne ma za zadanie wzmacnić ciemne obszary poprzez wzór -  $T(r) = c \cdot \log(1 + r)$ , gdzie  $c = \text{const}$   
Przez wzgląd na działanie logarytmiczne  $c > 0$

W wyniku badań, wartości  $c=[10, 50, 150]$  nie przyniosły widocznych zmian

```
In [256]: def GammaCorrection(img, m=1, e=1):
    """
    Apply contrast transformation to the image. Using formula:
    T(r) = 1 / (1 + (m/img)^e)

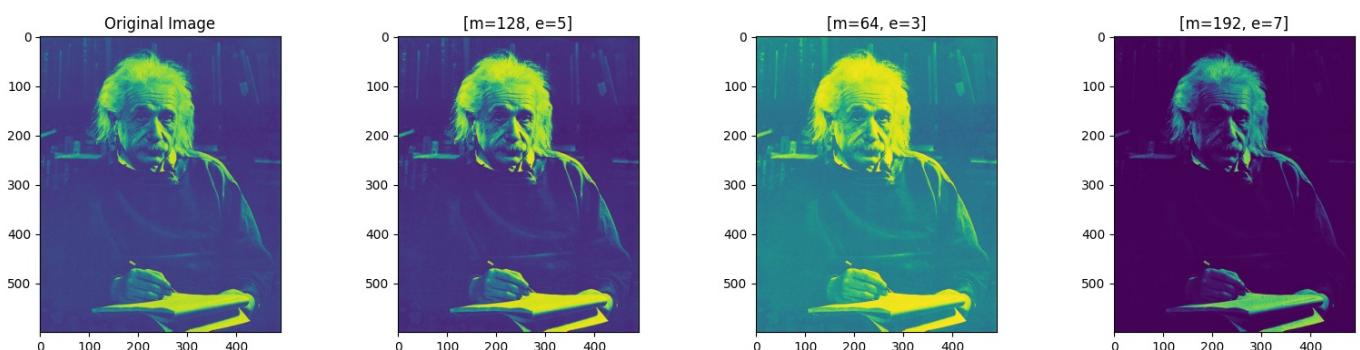
    :param img: Input image
    :param m: Contrast factor
    :param e: Exponent factor (default is 1)
    :return: Transformed image
    """
    img_float = img.astype(np.float32) + 1e-9 # Avoid division by zero
    transformed_img = 1 / (1 + (m / img_float) ** e)
    return np.clip(transformed_img * 255, 0, 255).astype(np.uint8)

# einstein-low-contrast.tif
processing = img_c
plt.figure(figsize=(16, 4))
plt.subplot(1, 4, 1)
plt.title("Original Image")
plt.imshow(processing)

plt.subplot(1, 4, 2)
img_ca = GammaCorrection(processing, m = 128, e = 5)
plt.title("[m=128, e=5]")
plt.imshow(img_ca)

plt.subplot(1, 4, 3)
img_cb = GammaCorrection(processing, m = 64, e = 3)
plt.title("[m=64, e=3]")
plt.imshow(img_cb)

plt.subplot(1, 4, 4)
img_cc = GammaCorrection(processing, m = 192, e = 7)
plt.title("[m=192, e=7]")
plt.imshow(img_cc)
plt.tight_layout()
plt.show()
```



## 3. Przekształcenie Sigmoid (odwrotność funkcji potęgowej)

Przekształcenie to przy pomocy parametrów  $m$  oraz  $e$  zmieniają kontrast na podstawie wybranego poziomu szarości  
 $m$  odpowiada za poziom szarości wokół którego następuje szybka zmiana kontrastu, natomiast  $e$  definiuje stromość czyli kontrast

W celu przeprowadzenia badań wybrano wartości

- $m=128, e=5.0$   
Obraz z podwyższonym kontrastem wokół średniej szarości. Ciemne stają się ciemniejsze, jasne jaśniejsze.
- $m=64, e=3.0$   
Próg przesunięty w stronę ciemnych tonów.
- $m=192, e=7.0$   
Próg przesunięty w stronę jasnych tonów, bardzo ostry kontrast.

```
In [257]: def GammaCorrection(img, c=1, gamma=1):
    """
    Apply gamma correction to the image. Using formula:
    s = c * (r ** gamma)
    where s is the output pixel value, r is the input pixel value, c is a constant, and gamma is the exponent.

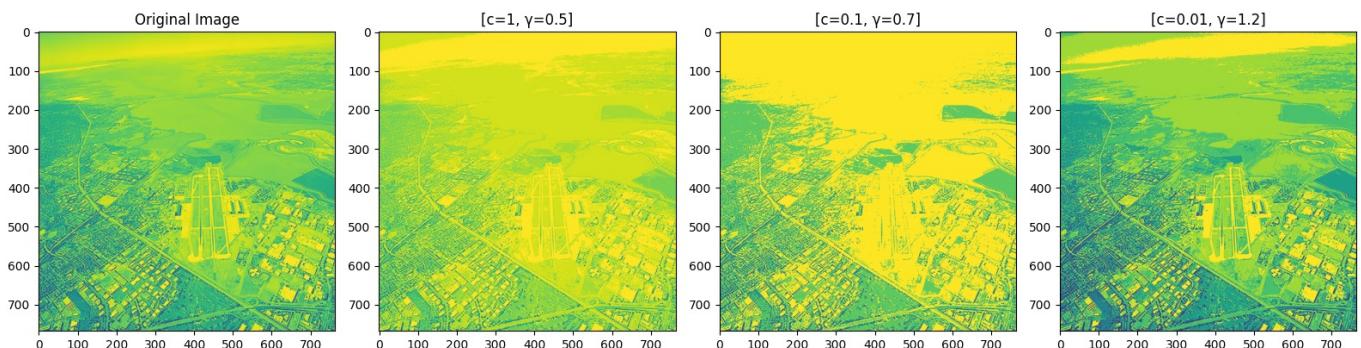
    :param img: Input image
    :param c: Constant factor (default is 1)
    :param gamma: Exponent factor (default is 1)
    :return: Transformed image
    """
    img_float = img.astype(np.float32) + 1e-9 # Avoid division by zero
    transformed_img = c * (img_float ** gamma)
    return np.clip(transformed_img, 0, 255).astype(np.uint8)

#aerial_view.tif
processing = img_d
plt.figure(figsize=(16, 4))
plt.subplot(1, 4, 1)
plt.title("Original Image")
plt.imshow(processing)

plt.subplot(1, 4, 2)
img_da = GammaCorrection(processing, c=1, gamma=0.5)
plt.title("[c=1, gamma=0.5]")
plt.imshow(img_da)

plt.subplot(1, 4, 3)
img_db = GammaCorrection(processing, c=0.1, gamma=0.7)
plt.title("[c=0.1, gamma=0.7]")
plt.imshow(img_db)

plt.subplot(1, 4, 4)
img_dc = GammaCorrection(processing, c=0.01, gamma=1.2)
plt.title("[c=0.01, gamma=1.2]")
plt.imshow(img_dc)
plt.tight_layout()
plt.show()
```



#### 4. Konwersja gamma

Przekształcenie to przy pomocy parametrów  $c$  oraz  $\gamma$  zmieniają kontrast lub jasność obrazu

W celu przeprowadzenia badań wybrano wartości

- $c=1, \gamma=0.5$   
Obraz został wyraźnie rozjaśniony. Wartość  $\gamma < 0$  powoduje, że ciemne piksele stają się jaśniejsze. Cienie są podbite.
- $c=0.1, \gamma=0.7$   
Z powodu niskiego współczynnika  $c$  obraz jest przyciemniony pomimo wartości  $\gamma < 0$ , dzięki temu cienie są tylko nieznacznie podbite.
- $c=0.01, \gamma=1.2$   
Silnie przyciemniony obraz z przyciemnieniem jasnych obszarów

## Ćwiczenie 7

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.

```
In [1]: import matplotlib.pyplot as plt
import tifffile as tiff
from skimage import exposure
import numpy as np
```

```
In [9]: # Załadowanie pliku .tif
img_a = tiff.imread("src/chester-xray.tif")
img_b = tiff.imread("src/pollen-dark.tif")
img_c = tiff.imread("src/pollen-ligt.tif")
img_d = tiff.imread("src/pollen-lowcontrast.tif")
img_e = tiff.imread("src/pout.tif")
img_f = tiff.imread("src/spectrum.tif")
```

```
In [13]: def histOfImg(img, title, index):
    if len(img.shape) == 3:
        vals = np.mean(img, axis=2).astype(np.uint8)
    else:
        vals = img

    counts, bins = np.histogram(vals, range(257))

    plt.subplot(2, 2, index)
    plt.imshow(img)
    plt.title(title)

    plt.subplot(2, 2, index + 1)
    plt.bar(bins[:-1] - 0.5, counts, width=1, edgecolor='none')
    plt.xlim([-0.5, 255.5])
    plt.title("Histogram ")
```

```
In [17]: def hist(img):
    plt.figure(figsize=(10, 8))
    histOfImg(img, "Obraz oryginalny", 1)
    equ = exposure.equalize_hist(img)
    equ_uint8 = (equ * 255).astype(np.uint8)
    histOfImg(equ_uint8, "Obraz wyrównany", 3)
    plt.tight_layout()
    plt.show()
```

```
In [18]: print("Image: chest-xray.tif")
hist(img_a)

print("Image: pollen-dark.tif")
hist(img_b)

print("Image: pollen-ligt.tif")
hist(img_c)

print("Image: pollen-lowcontrast.tif")
hist(img_d)

print("Image: chest-pout.tif")
hist(img_e)

print("Image: chest-spectrum.tif")
hist(img_f)
```

Image: chest-xray.tif

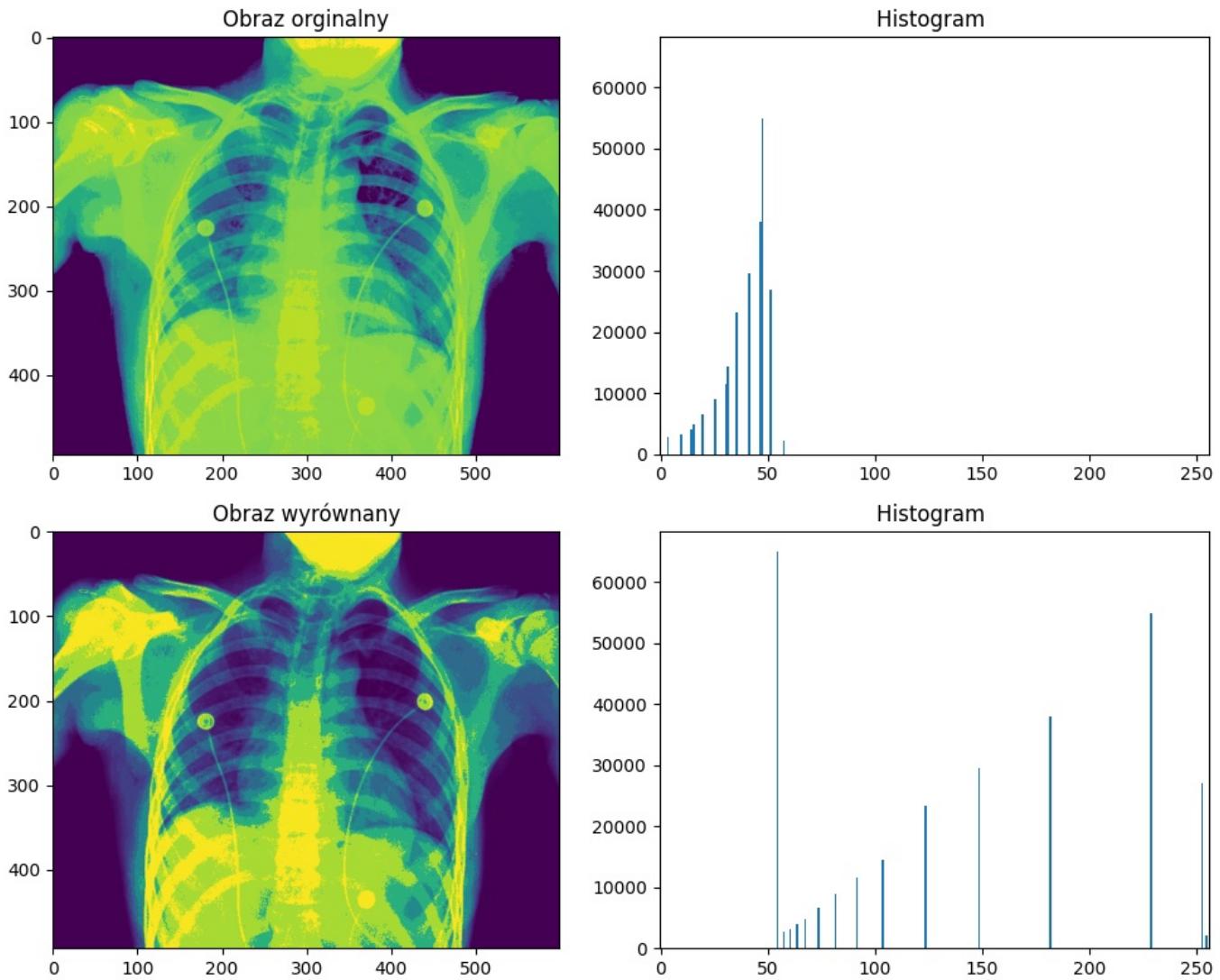


Image: pollen-dark.tif

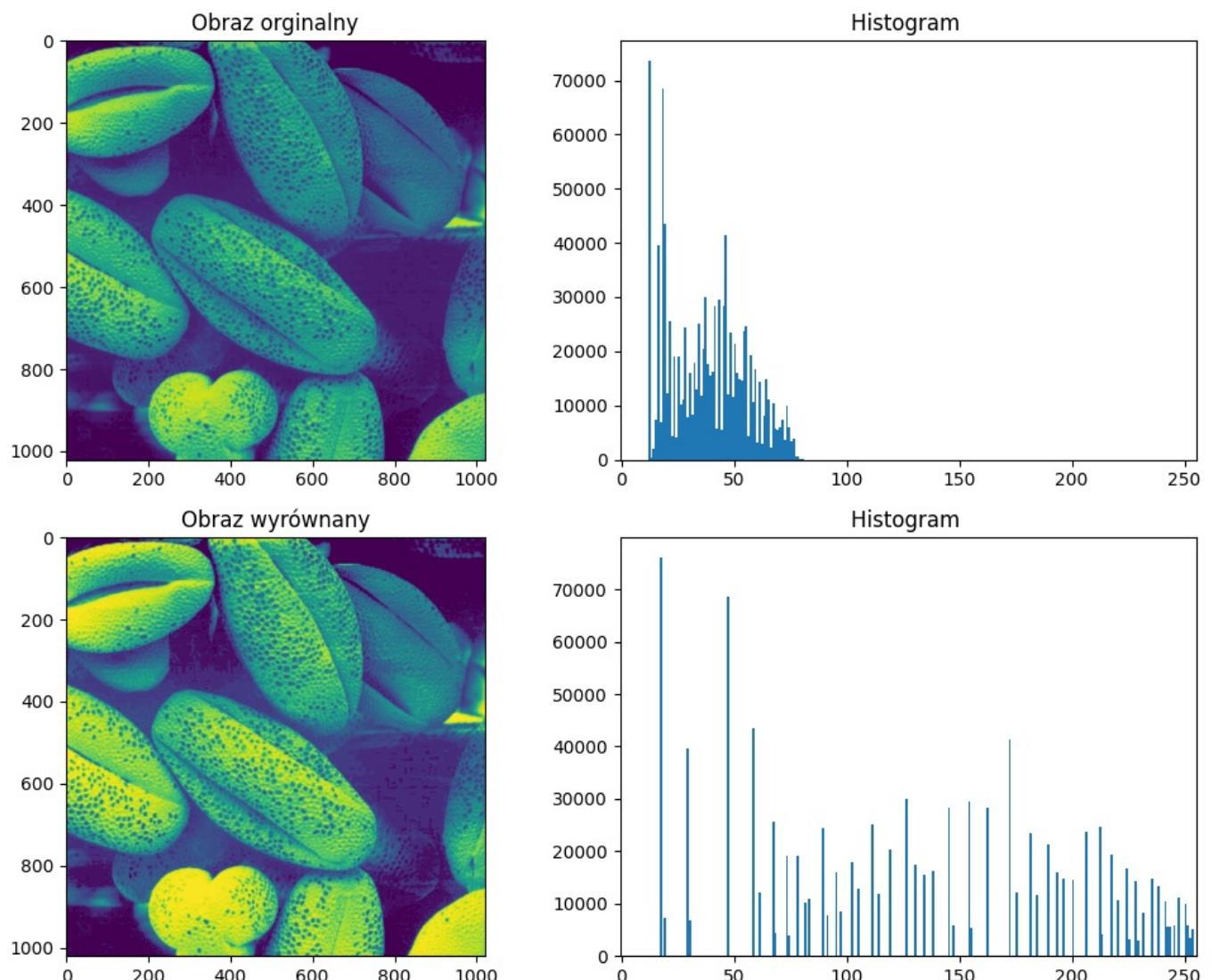


Image: pollen-ligt.tif

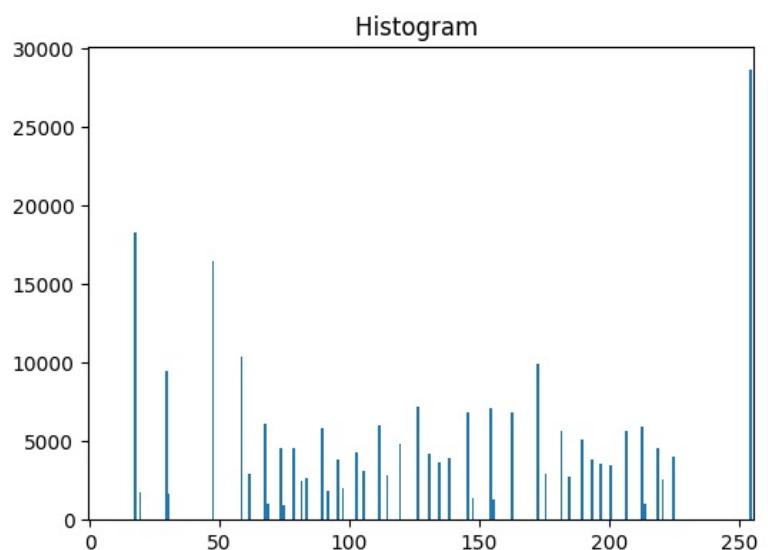
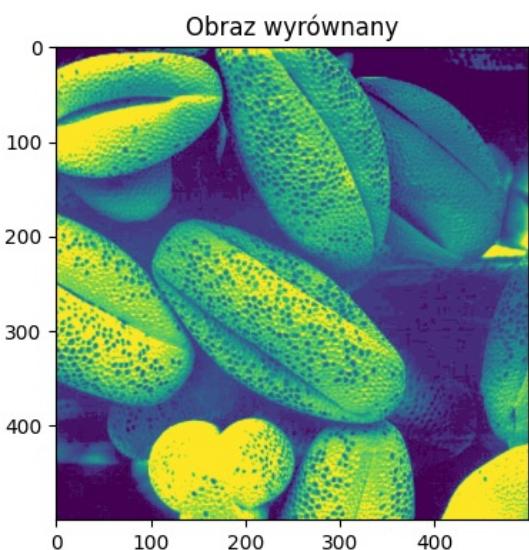
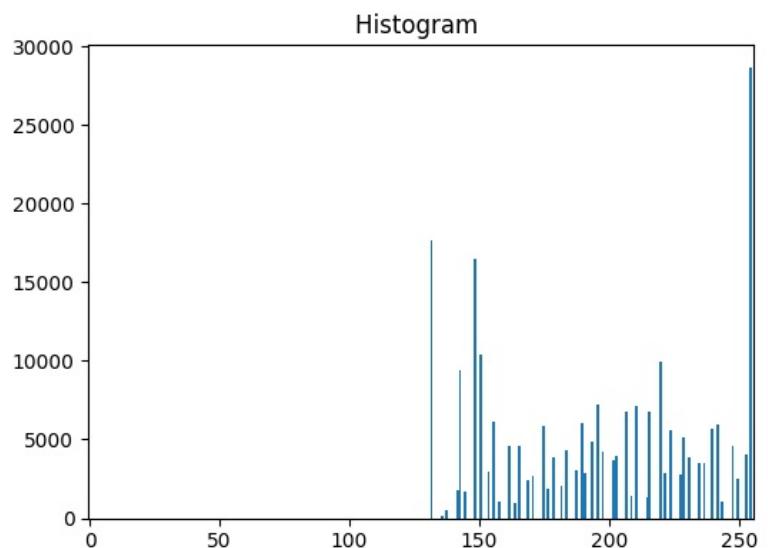
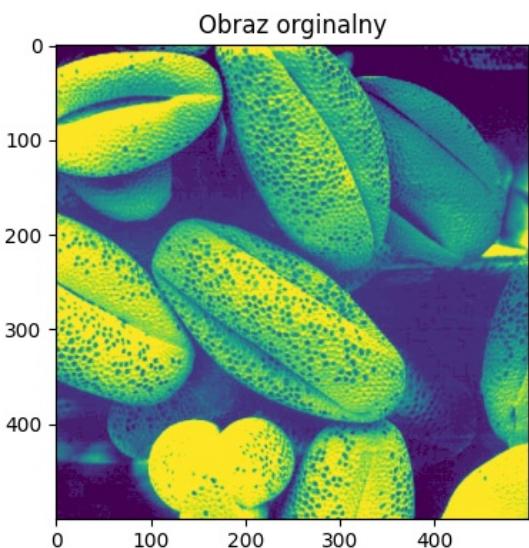


Image: pollen-lowcontrast.tif

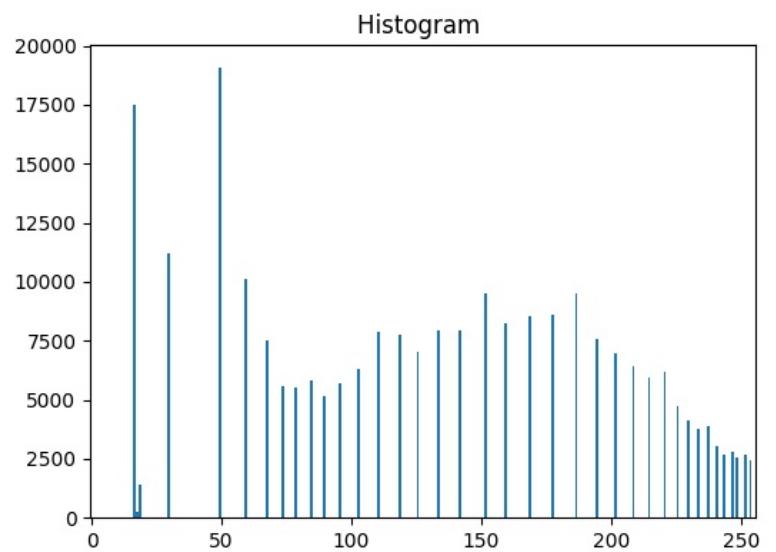
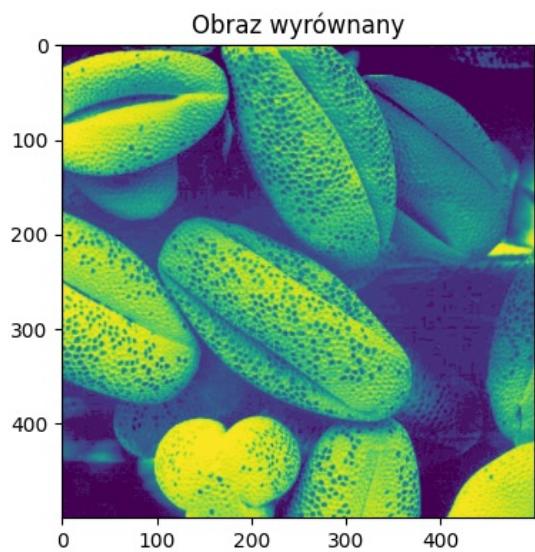
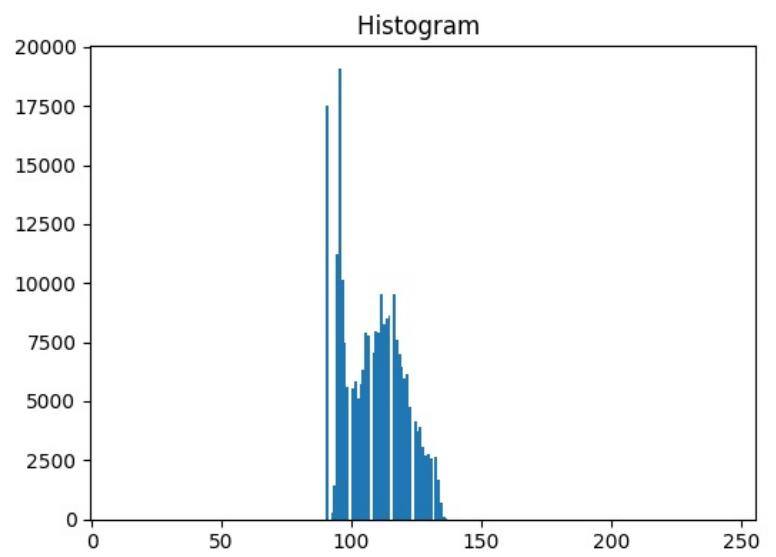
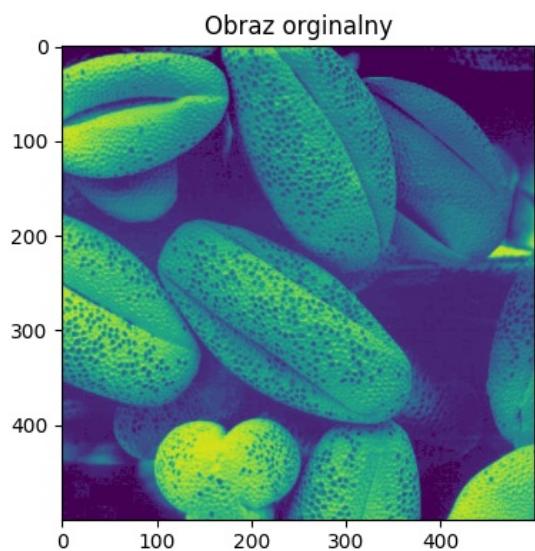


Image: chest-pout.tif

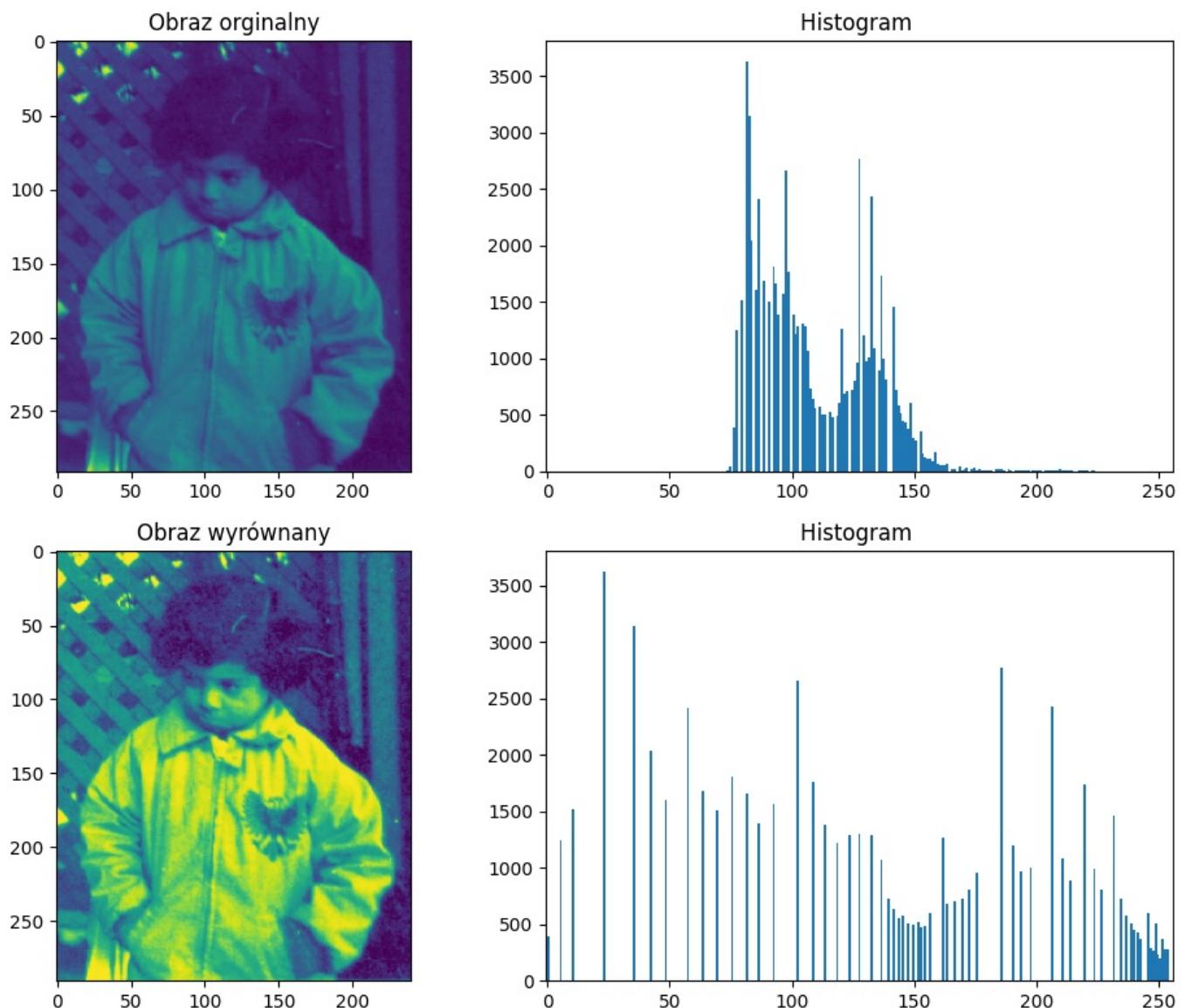
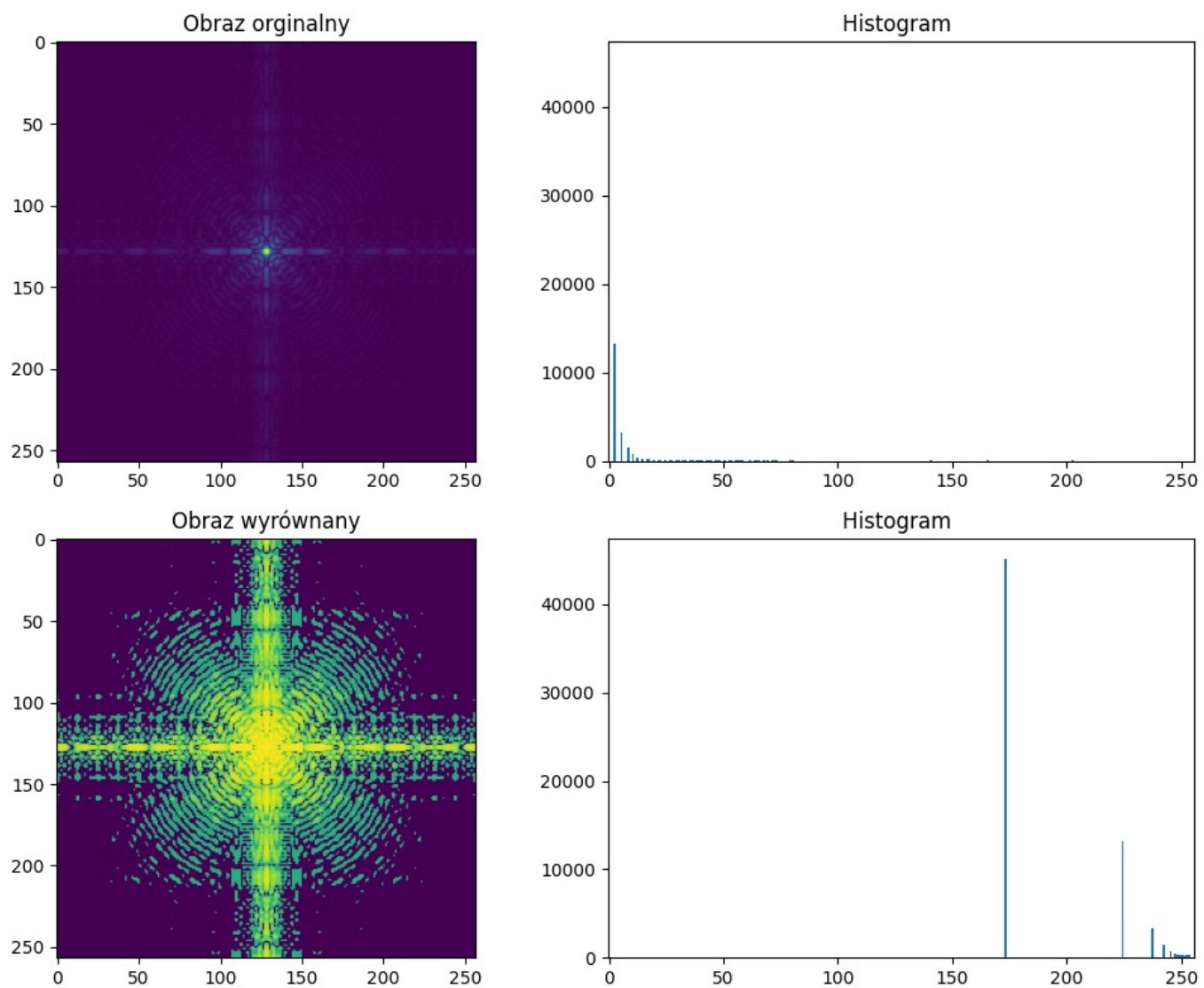


Image: chest-spectrum.tif



Na podstawie przeprowadzonych badań zauważono, że wyrównanie obrazu powiększa zakres wartości histogramu. Dzięki temu linie są bardziej widoczne, kształty mają intensywniejsze kontury oraz obrazy są jaśniejsze.  
Dzięki takiemu wyrównaniu można zauważać więcej szczegółów które na obrazie są bardzo ciemne (np. spectrum.tif)

```
In [14]: import matplotlib.pyplot as plt
import tifffile as tiff
from skimage import exposure
import skimage.morphology as morph
from skimage.filters import rank
```

```
In [3]: # Załadowanie pliku .tiff
img = tiff.imread("src/hidden-symbols.tif")
```

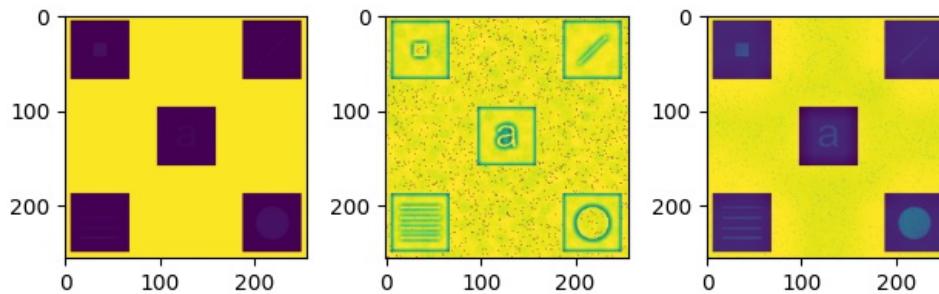
```
In [4]: # Wyświetlenie załadowanego obrazu
img_local = rank.equalize(img, morph.disk(5))
img_stat = exposure.equalize_adapthist(img, clip_limit=0.15)

plt.figure()
plt.subplot(1,3,1)
plt.imshow(img)

plt.subplot(1,3,2)
plt.imshow(img_local)

plt.subplot(1,3,3)
plt.imshow(img_stat)

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



## Ćwiczenie 9

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

1. liniowego filtra uśredniającego z kwadratową maską, rozpoczynając od maski rozmiaru  $3 \times 3$ .
2. nieliniowego filtra medianowego
3. filtrów minimum i maksimum.

```
In [61]: import matplotlib.pyplot as plt
import tifffile as tiff
import skimage.morphology as morph
from skimage.filters import rank
```

```
In [62]: # Załadowanie pliku .tif
img_a = tiff.imread("src/cboard_pepper_only.tif")
img_b = tiff.imread("src/cboard_salt_only.tif")
img_c = tiff.imread("src/cboard_salt_pepper.tif")
```

### Zadanie 1

Filtr uśredniający z kwadratową maską  $3 \times 3$

```
In [ ]: def meanFilter(img):
    plt.subplot(2,3,2)
    mean_img = rank.mean(img,morph.footprint_rectangle(square_mask))
    plt.imshow(mean_img)
    plt.title("Filtr uśredniający")
```

### Zadanie 2

Nieliniowy filtr medianowy

```
In [64]: def mediFilter(img):
    plt.subplot(2,3,3)
    medi_img = rank.median(img,morph.footprint_rectangle(square_mask))
    plt.imshow(medi_img)
    plt.title("Filtr medianowy")
```

### Zadanie 3

Filtры minimum i maximum

```
In [65]: def minMaxFilter(img):
    plt.subplot(2,3,4)
    min_img = rank.minimum(img,morph.footprint_rectangle(square_mask))
    plt.imshow(min_img)
    plt.title("Filtr min")

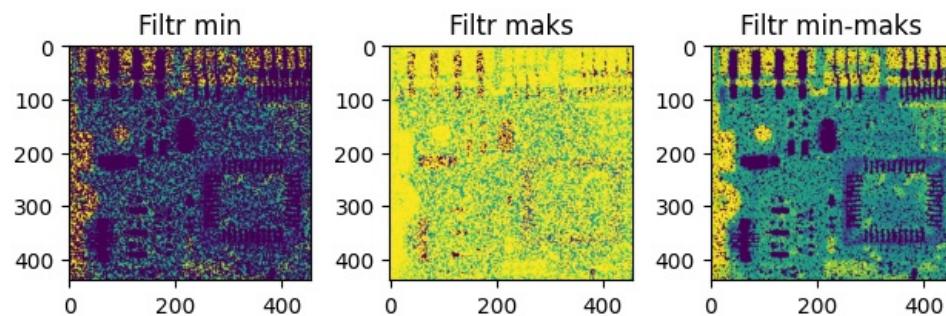
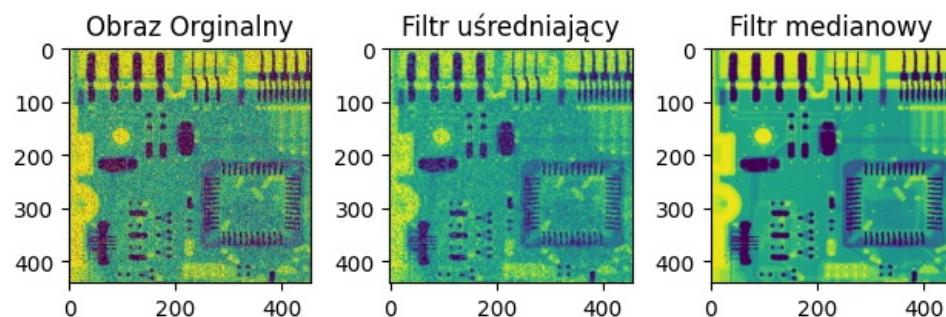
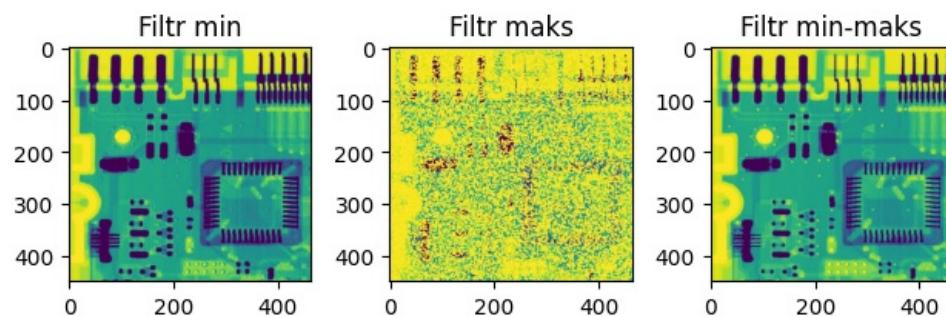
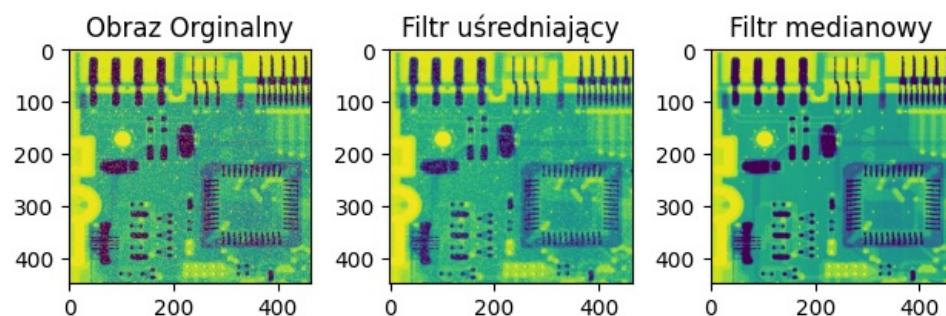
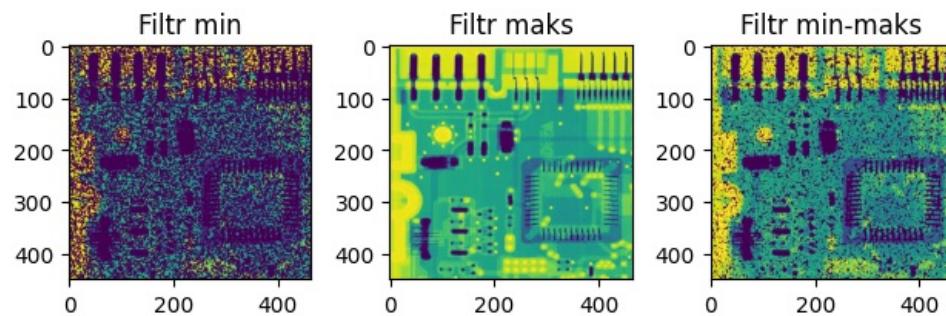
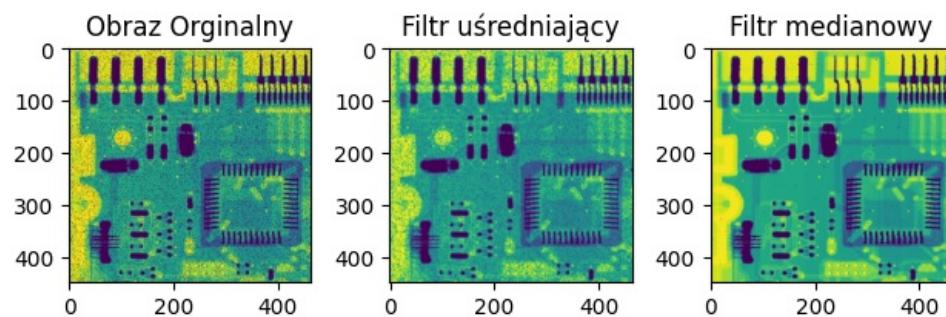
    plt.subplot(2,3,5)
    max_img = rank.maximum(img,morph.footprint_rectangle(square_mask))
    plt.imshow(max_img)
    plt.title("Filtr maks")

    plt.subplot(2,3,6)
    minMax_img = rank.maximum(min_img,morph.footprint_rectangle(square_mask))
    plt.imshow(minMax_img)
    plt.title("Filtr min-maks")
```

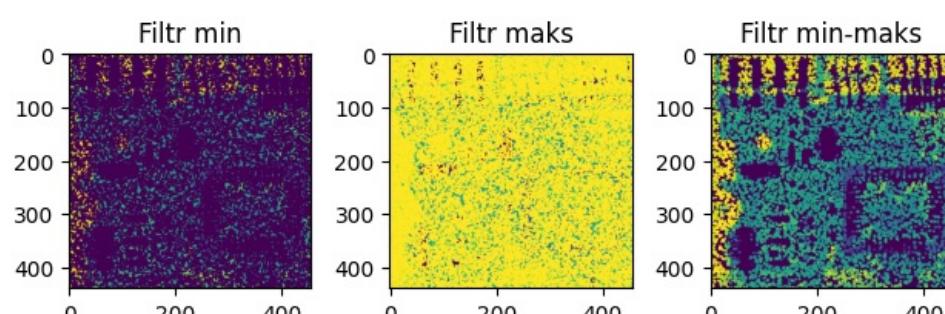
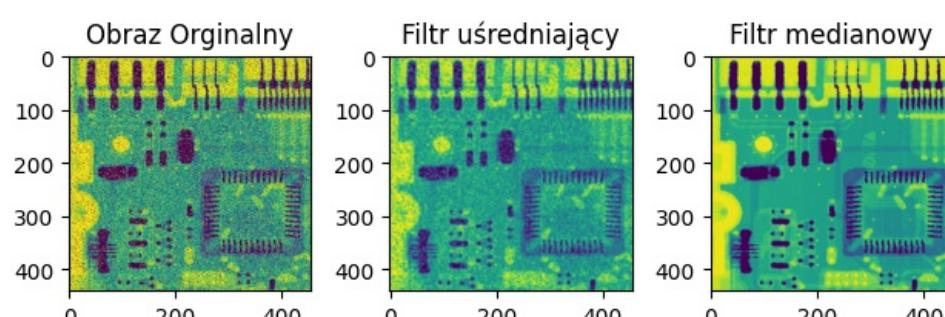
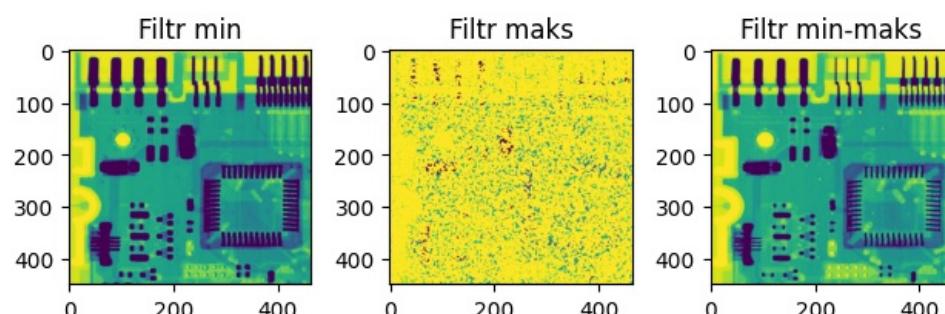
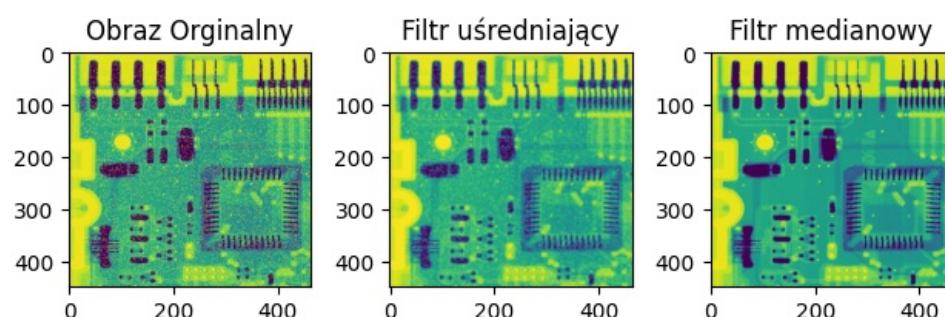
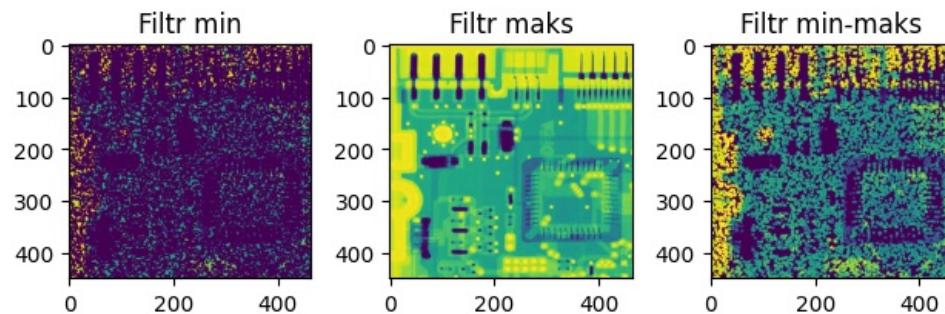
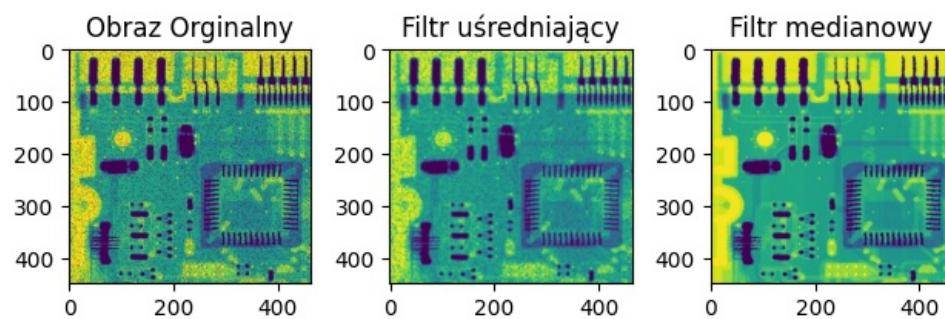
```
In [66]: def display(img):
    plt.subplot(2,3,1)
    plt.imshow(img)
    plt.title("Obraz Orginalny")
    meanFilter(img)
    mediFilter(img)
    minMaxFilter(img)
    plt.tight_layout()
    plt.show()

square_mask = (3, 3)
for i in range(3):
    print("Obrazy wynikowe dla maski {}x{}:".format(square_mask[0], square_mask[1]))
    display(img_a) #pepper
    display(img_b) #salt
    display(img_c) #salt and pepper
    square_mask = (square_mask[0] + 1, square_mask[1] + 1)
```

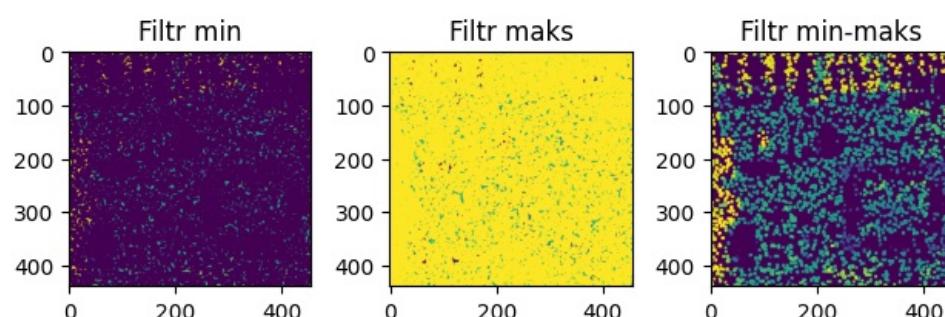
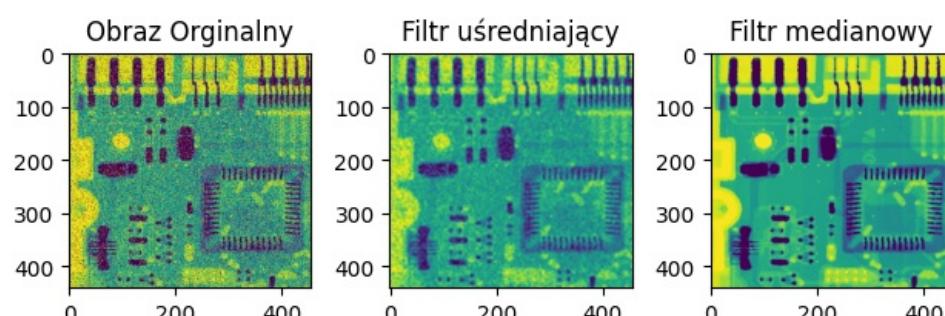
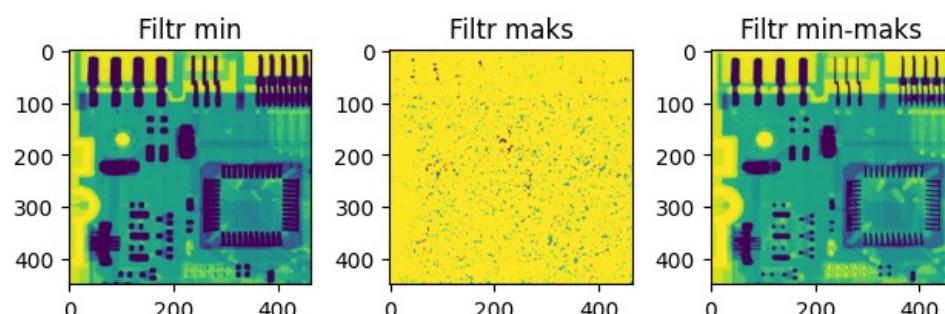
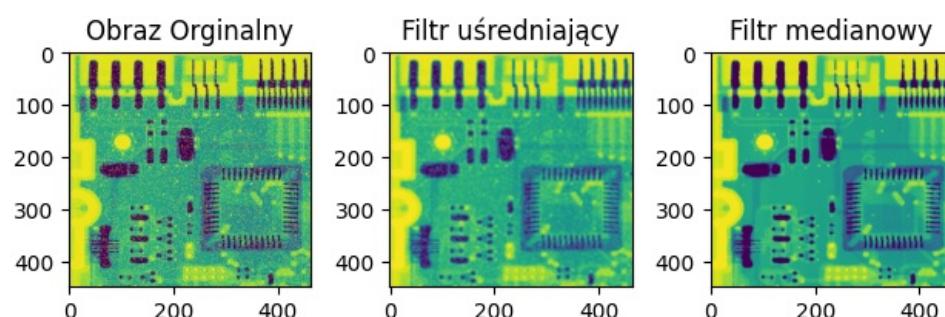
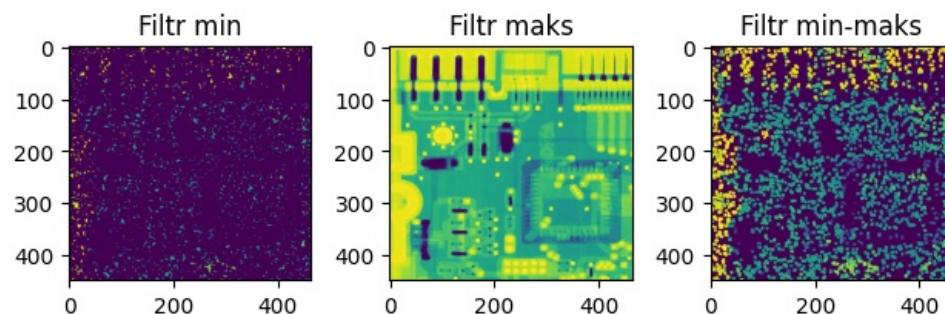
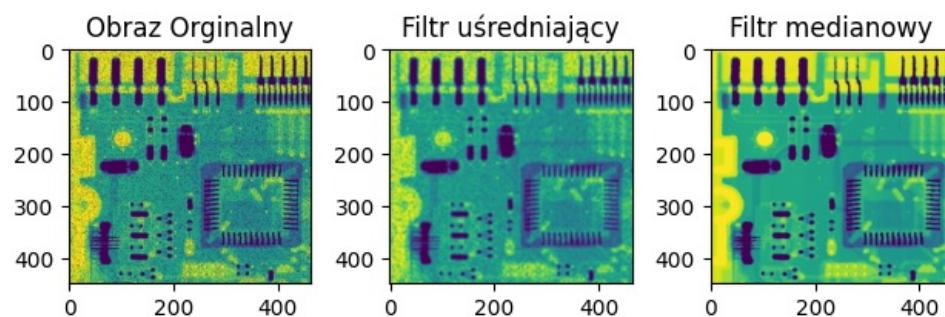
Obrazy wynikowe dla maski 3x3:



Obrazy wynikowe dla maski 4x4:



Obrazy wynikowe dla maski 5x5:



W badaniach porównano skuteczność filtrów uśredniającego, medianowego, minimalnego, maksymalnego oraz min-maks na obrazach z szumem typu "pieprz", "sól" oraz "sól i pieprz" dla masek o rozmiarach od  $3 \times 3$  do  $5 \times 5$ .

- Zastosowane filtry i efekty

- Filtr uśredniający (mean): Skutecznie wygładza szum, jednak powoduje rozmycie krawędzi i utratę szczegółów. Im większa maska, tym silniejsze rozmycie obrazu. Najlepiej sprawdza się przy słabym szumie, ale nie radzi sobie z pojedynczymi impulsami szumu typu "sól" lub "pieprz".
  - Filtr medianowy (median): Najlepiej usuwa szum impulsowy ("sól", "pieprz", "sól i pieprz"), zachowując przy tym ostrość krawędzi. Nawet przy większych maskach skutecznie eliminuje zakłócenia bez nadmiernego rozmycia obrazu. Jest rekomendowany do usuwania szumu impulsowego.
  - Filtr minimalny (min): Usuwa głównie szum typu "sól" (białe piksele), ale może prowadzić do przyciemnienia obrazu i utraty detali. Przy większych maskach efekt ciemnienia jest silniejszy.
  - Filtr maksymalny (max): Usuwa głównie szum typu "pieprz" (czarne piksele), ale może prowadzić do rozjaśnienia obrazu. Zwiększenie maski potęguje ten efekt.
  - Filtr min-maks: Kombinacja filtrów min i max pozwala na usunięcie obu typów szumu ("sól i pieprz"), jednak kosztem utraty kontrastu i szczegółów, zwłaszcza przy większych maskach.
- Wnioski z badań dla masek  $3 \times 3$ ,  $4 \times 4$ ,  $5 \times 5$
- Maska  $3 \times 3$ : Najlepszy kompromis między usuwaniem szumu a zachowaniem szczegółów. Filtr medianowy skutecznie usuwa szum impulsowy, a efekt rozmycia jest minimalny.
  - Maska  $4 \times 4$  i  $5 \times 5$ : Skuteczność usuwania szumu rośnie, ale jednocześnie rośnie rozmycie i utrata detali. Filtr medianowy nadal radzi sobie najlepiej, natomiast filtry uśredniający, min i max powodują coraz większą degradację jakości obrazu.

## Zadanie 10

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

```
In [2]: import matplotlib.pyplot as plt
import tifffile as tiff
import skimage.filters as flt
import skimage.morphology as morph
from skimage.filters import rank
```

```
In [ ]: # Załadowanie pliku .tif
img_a = tiff.imread("src/characters_test_pattern.tif")
img_b = tiff.imread("src/zoneplate.tif")
```

```
In [22]: plt.figure(figsize=(25,7))
plt.subplot(2,6,1)
plt.imshow(img_a)
processing = img_a
plt.title("Obraz oryginalny")

plt.subplot(2,6,2)
gaus_b = flt.gaussian(processing,sigma=1)
plt.imshow(gaus_b)
plt.title("Filtr gausa sigma=1")

plt.subplot(2,6,3)
gaus_b = flt.gaussian(processing,sigma=5)
plt.imshow(gaus_b)
plt.title("Filtr gausa sigma=5")

plt.subplot(2,6,4)
medi_img = rank.median(processing,morph.footprint_rectangle((3,3)))
plt.imshow(medi_img)
plt.title("Filtr medianowy 3x3")

plt.subplot(2,6,5)
medi_img = rank.median(processing,morph.footprint_rectangle((16,16)))
plt.imshow(medi_img)
plt.title("Filtr medianowy 16x16")

plt.subplot(2,6,6)
medi_img = rank.median(processing,morph.footprint_rectangle((90,90)))
plt.imshow(medi_img)
plt.title("Filtr medianowy 90x90")

plt.subplot(2,6,7)
plt.imshow(img_b)
processing = img_b

plt.subplot(2,6,8)
gaus_b = flt.gaussian(processing,sigma=1)
plt.imshow(gaus_b)

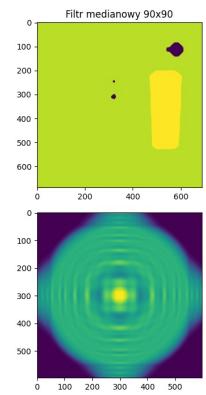
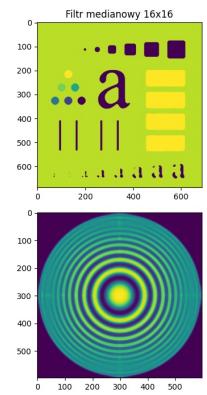
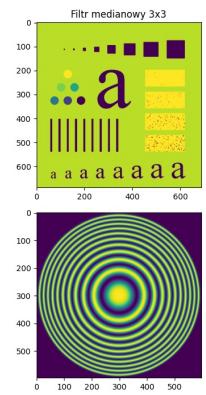
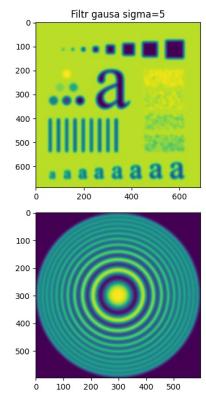
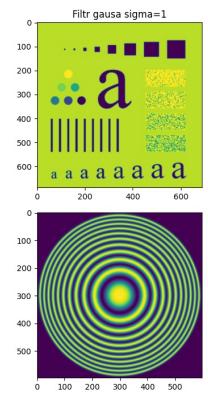
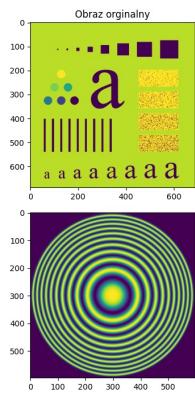
plt.subplot(2,6,9)
gaus_b = flt.gaussian(processing,sigma=5)
plt.imshow(gaus_b)

plt.subplot(2,6,10)
medi_img = rank.median(processing,morph.footprint_rectangle((3,3)))
plt.imshow(medi_img)

plt.subplot(2,6,11)
medi_img = rank.median(processing,morph.footprint_rectangle((16,16)))
plt.imshow(medi_img)

plt.subplot(2,6,12)
medi_img = rank.median(processing,morph.footprint_rectangle((90,90)))
plt.imshow(medi_img)

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



## Ćwiczenie 11

Wykrywanie krawędzi obiektów i poprawa ostrości.

1. Użyj filtra z maską Sobela do wykrywania krawędzi poziomych, pionowych i ukośnych.
2. Zaobserwuj działanie Laplasjanu do wyostrzania szczegółów.
3. Zbadaj działanie filtrów typu „unsharp masking” i „high boost”.

```
In [25]: import matplotlib.pyplot as plt
import tifffile as tiff
import cv2 as cv
import numpy as np
from skimage.filters import sobel, laplace, unsharp_mask, gaussian
```

```
In [26]: # Załadowanie pliku .tiff
img_a = tiff.imread("src/circuitmask.tif")
img_b = cv.imread("src/testpat1.png")
img_c = tiff.imread("src/blurry-moon.tif")
img_d = tiff.imread("src/text-dipxe-blurred.tif")
```

### Zadanie 1

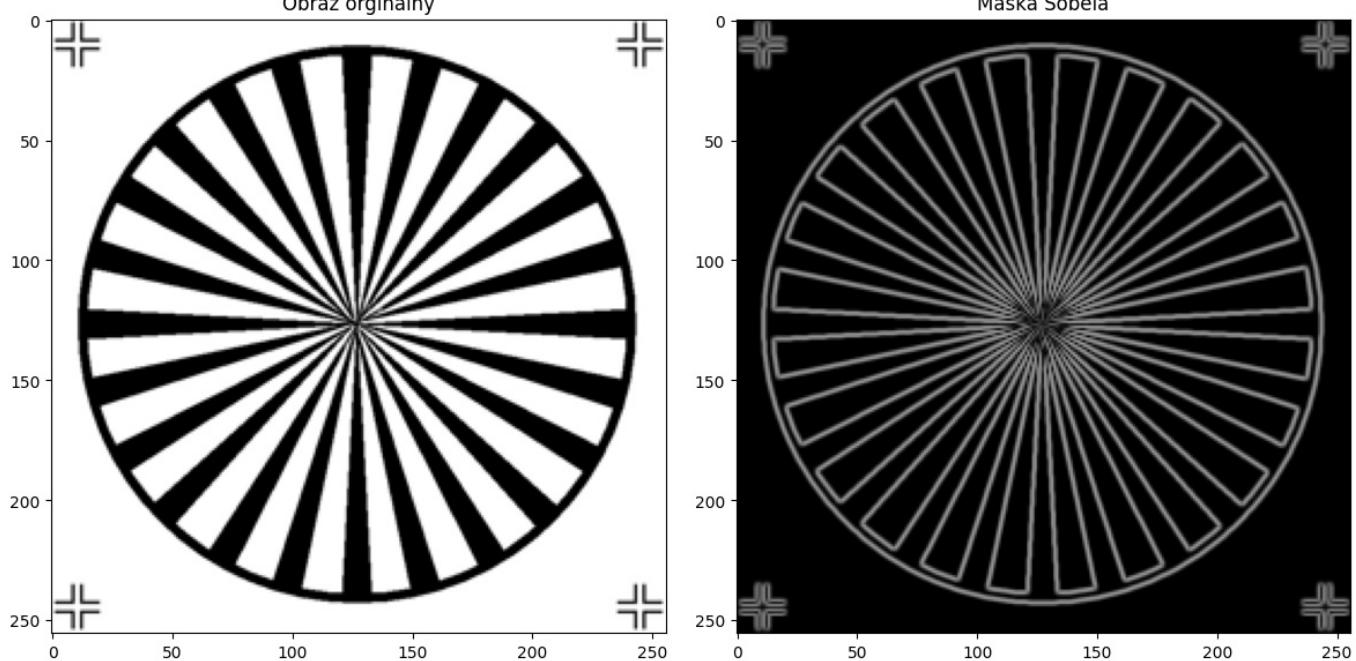
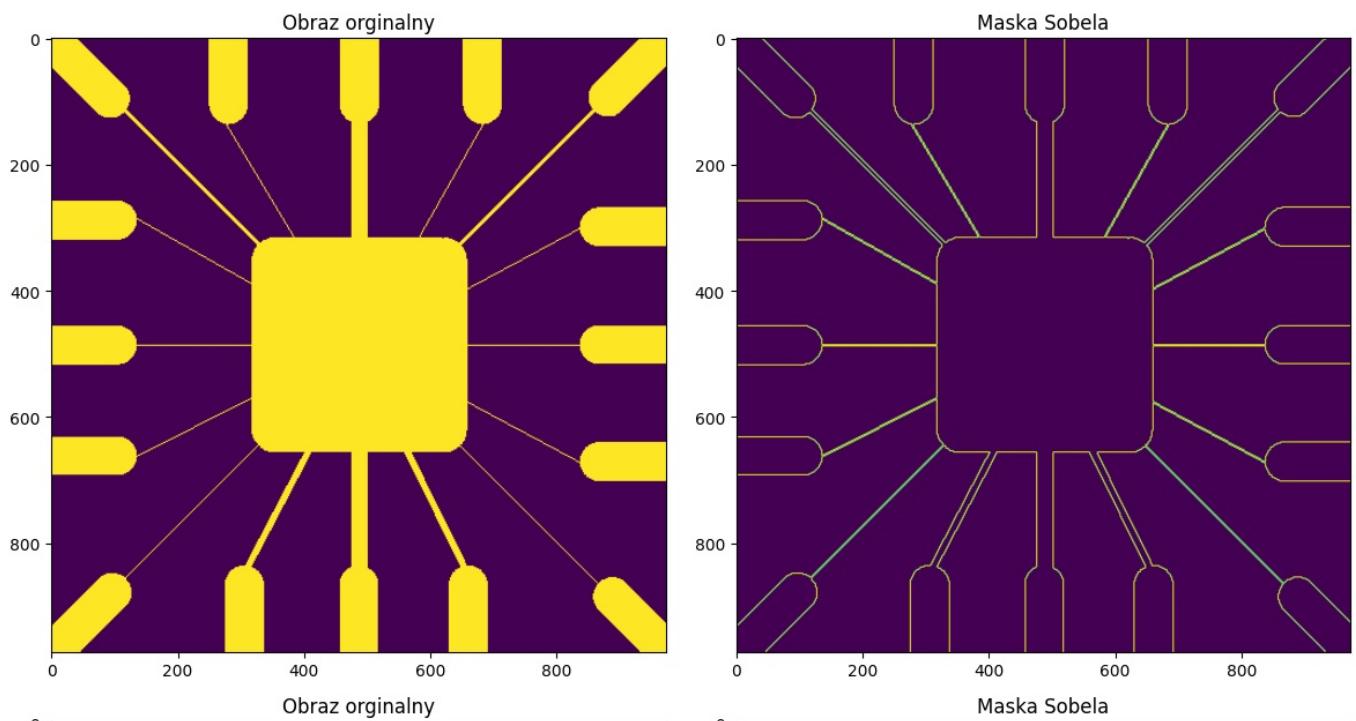
Użycie maski Sobela w celu wykrycia krawędzi poziomych, pionowych i ukośnych.

```
In [27]: plt.figure(figsize=(12, 12))
plt.subplot(2,2,1)
plt.imshow(img_a)
plt.title("Obraz oryginalny")

plt.subplot(2,2,2)
sob_img_a = sobel(img_a)
plt.imshow(sob_img_a)
plt.title("Maska Sobela")

plt.subplot(2,2,3)
plt.imshow(img_b)
plt.title("Obraz oryginalny")

plt.subplot(2,2,4)
sob_img_b = sobel(img_b)
plt.imshow(sob_img_b)
plt.title("Maska Sobela")
plt.tight_layout()
plt.show()
```



Wykorzystanie maski Sobela pozwala na wyznaczenie zarysów kształtów pozбавiając ich wypełnienia. W ten sposób można uzyskać wyraźne kontury kształtów.

## Zadanie 2

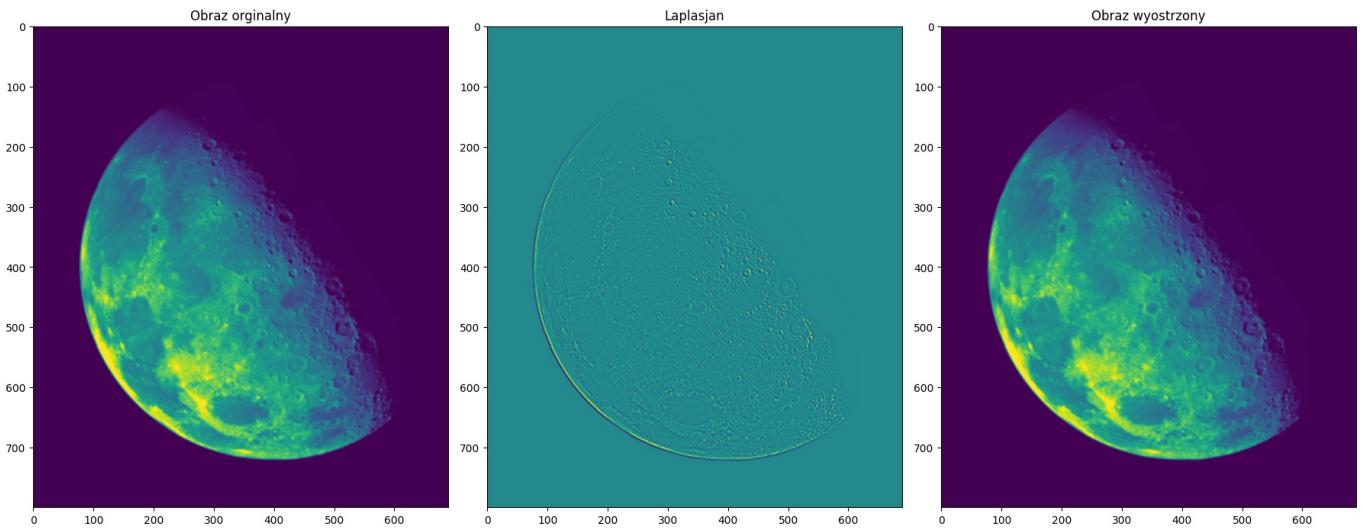
Wykorzystanie Laplasjanu w celu wyostrzenia szczegółów.

```
In [28]: plt.figure(figsize=(18, 10))
plt.subplot(1,3,1)
plt.imshow(img_c)
plt.title("Obraz orginalny")

plt.subplot(1,3,2)
lap_img_c = laplace(img_c)
plt.imshow(lap_img_c)
plt.title("Laplasjan")

plt.subplot(1,3,3)
fin_img_c = img_c - lap_img_c
plt.imshow(fin_img_c)
plt.title("Obraz wyostrzony")

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



Wykorzystanie laplasjanu należy wykonać w dwóch krokach:

1. Wyznaczyć Laplasjan, który jest Wykryciem krawędzi na obrazie
2. Odjąć Laplasjan od oryginalnego obrazu

Laplasjan służy głównie do:

- wykrywania krawędzi,
- podkreślania szczegółów,
- wyostrzania obrazów.

## Zadanie 11

Wykorzystanie filtrów "unsharp masking" oraz "high boost"

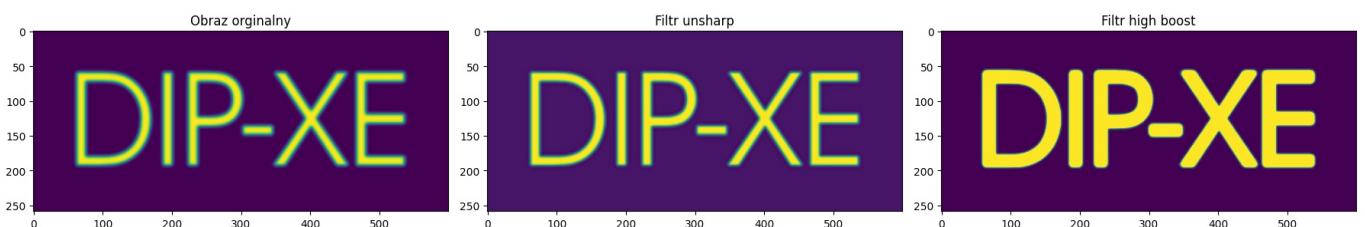
```
In [40]: def highBoostFilter(img, k=1.5, sigma=3):
    blur = gaussian(img, sigma=sigma)
    hb = k * img - blur
    hb = np.clip(hb, 0, 255).astype(np.uint8)
    return hb

plt.figure(figsize=(18, 10))
plt.subplot(1,3,1)
plt.imshow(img_d)
plt.title("Obraz orginalny")

plt.subplot(1,3,2)
us_img_d = unsharp_mask(img_d, radius=1.0, amount=3.5)
plt.imshow(us_img_d)
plt.title("Filtr unsharp")

plt.subplot(1,3,3)
hb_img_d = highBoostFilter(img_d, k=3.5, sigma=6)
plt.imshow(hb_img_d)
plt.title("Filtr high boost")

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



W tym eksperymencie porównano efekty działania filtru unsharp mask oraz filtru high-boost na tym samym obrazie.

- Filtr unsharp mask
  - Zastosowano filtr unsharp mask z parametrami radius=1.0 oraz amount=3.5.  
Efekt: Obraz jest silnie wyostrzony – krawędzie i detale są mocno podkreślone, a kontrast lokalny znacznie wzrasta. Tak wysoka wartość amount powoduje, że efekt wyostrzenia jest bardzo wyraźny, a drobne szczegółы oraz szумy mogą być również wzmacnione. Obraz wydaje się się nienaturalnie ostry, zwłaszcza w miejscach o dużych zmianach jasności.
- Filtr high-boost

- Zastosowano filtr high-boost z parametrami k=3.5 i sigma=6.  
Efekt: Obraz jest ekstremalnie wyostrzony – efekt jest jeszcze silniejszy niż w przypadku filtra unsharp mask. Współczynnik k=3.5 powoduje, że oryginalny obraz jest mocno wzmacniany, a rozmycie z dużym sigma=6 sprawia, że podbijane są również większe struktury. Skutkiem tego krawędzie i detale są bardzo mocno zaakcentowane, ale jednocześnie mogą pojawić się silne artefakty, a szумy i drobne zakłócenia są znacznie wzmacnione.

## Ćwiczenie 12

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”

```
In [15]: import matplotlib.pyplot as plt
import tifffile as tiff
from numpy import emath, abs, sqrt
import skimage.morphology as morph
from skimage.filters import rank,sobel,laplace,unsharp_mask,gaussian
```

```
In [16]: # Załadowanie pliku .tiff
img = tiff.imread("src/bonescan.tif")
```

1. Skan PET ciała człowieka
  - wysoki poziom szumów
  - dominacja ciemnych i jasnych poziomów szarości
2. Laplasjan obrazu 1. z maską  $3 \times 3$ 
  - obraz przeskalowano do zakresu  $[0, 255]$
3. Suma obrazów 1. i 2.
  - uwydawnienie drobnych szczegółów
  - wciąż zauważamy spory poziom szumów
4. Gradient Sobela obrazu 1.
$$M(x, y) \approx |gx| + |gy|$$
  - uwydawnienie brzegów

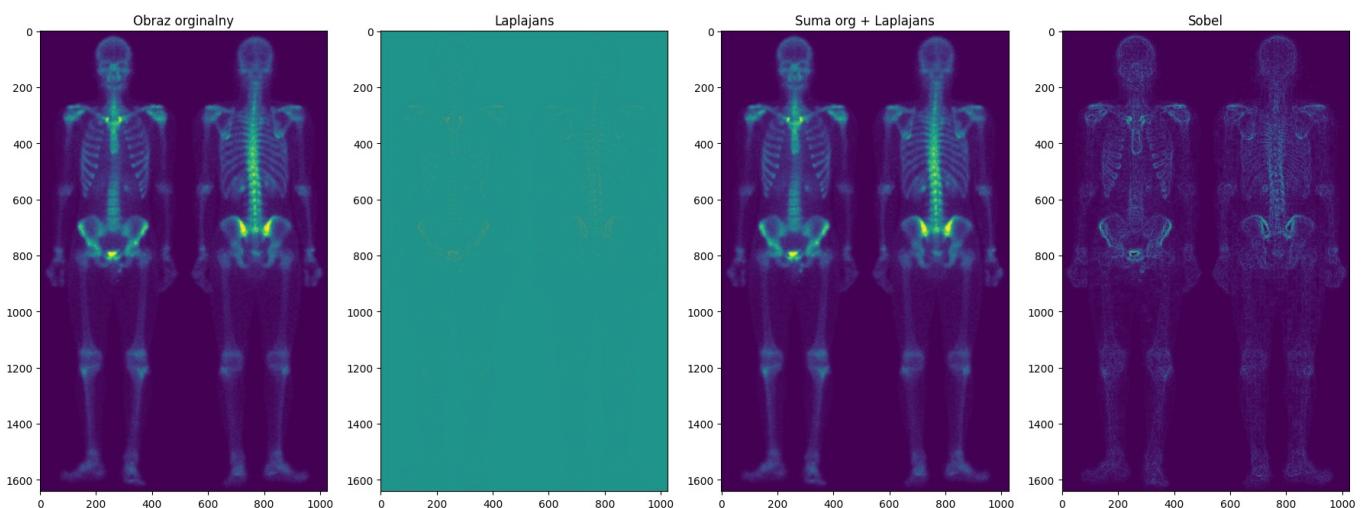
```
In [17]: plt.figure(figsize=(18, 10))
plt.subplot(1,4,1)
plt.imshow(img)
plt.title("Obraz orginalny")

plt.subplot(1,4,2)
lap_img = laplace(img)
plt.imshow(lap_img)
plt.title("Laplajans")

plt.subplot(1,4,3)
sum_img = img + lap_img
plt.imshow(sum_img)
plt.title("Suma org + Laplajans")

plt.subplot(1,4,4)
sob_img = sobel(img)
plt.imshow(sob_img)
plt.title("Sobel")

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



Dzięki zastosowaniu powyższych kroków uzyskano odszumiony obraz z widocznymi krawędziami kości z tomografii komputerowej. Tak widoczne elementy pozwalają na bezproblemowe wyznaczenie granic układu kostnego.

5. Filtracja uśredniająca z maską  $5 \times 5$  obrazu 4.

- redukcja szumu uwydatnionego przez laplasjan
6. iloczyn obrazu 5. i laplasjanu 2.
  7. Suma 1. i 6.
  8. Transformacja potęgowa 7.,  $c = 1, \gamma = 0,5$
- $s = c \cdot r^{\gamma}$
- zwiększenie kontrastu

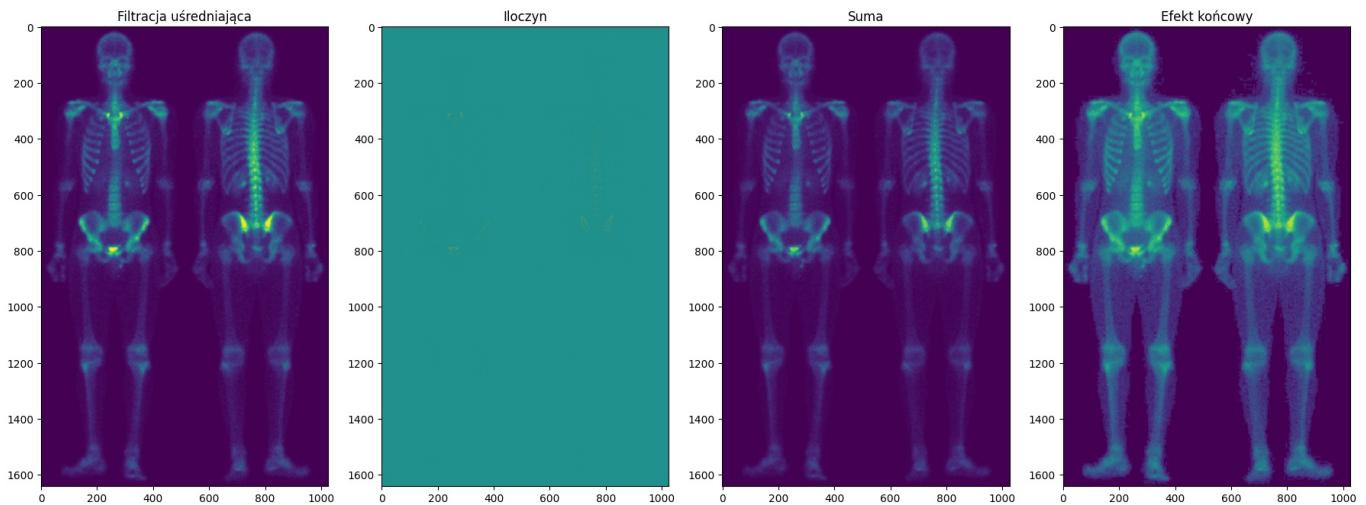
```
In [18]: plt.figure(figsize=(18, 10))
plt.subplot(1,4,1)
mean_img = rank.mean(img,morph.footprint_rectangle((5,5)))
plt.imshow(img)
plt.title("Filtracja uśredniająca")

plt.subplot(1,4,2)
ilo_img = mean_img * lap_img
plt.imshow(ilo_img)
plt.title("Iloczyn")

plt.subplot(1,4,3)
sum2_img = img + ilo_img
plt.imshow(sum2_img)
plt.title("Suma")

plt.subplot(1,4,4)
fin_img = sqrt(abs(sum2_img))
plt.imshow(fin_img)
plt.title("Efekt końcowy")

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



Dzięki zastosowaniu powyższych kroków uzyskano odszumione zdjęcie, gdzie można wyraźnie zauważać obrys ciała, a nie samych kości jak w zdjęciu oryginalnym. Tak przetworzone zdjęcie pozwala na jasne określenie położenia kości względem powierzchni ciała.