

UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO
W WARSZAWIE

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY
SZKOŁA NAUK ŚCISŁYCH

Katarzyna Mitrus

Michał Słotwiński

Wprowadzenie do Przetwarzania Obrazów

Sprawozdanie z laboratorium

Prowadzący:
prof. Wojciech Mokrzycki

Warszawa, 2018

Spis treści

Spis rysunków	4
Rozdział 1. Wstęp	6
1.1 Specyfikacja wykorzystanego fortformatu obrazu	6
1.2 Instrukcja obsługi programu	6
Rozdział 2. Operacje ujednoliciania obrazów	7
Rozdział 3. Operacje sumowania arytmetycznego obrazów szarych	8
3.1 Sumowanie (określonej) stałej z obrazem	8
3.2 Sumowanie dwóch obrazów	10
3.3 Mnożenie obrazu przez zadaną liczbę	13
3.4 Mnożenie obrazu przez inny obraz	15
3.5 Mieszanie obrazów z określonym współczynnikiem	18
3.6 Potęgowanie obrazu (z zadaną potęgą)	19
3.7 Dzielenie obrazu przez (zadaną) liczbę	21
3.8 Dzielenie obrazu przez inny obraz	23
3.9 Pierwiastkowanie obrazu	26
3.10 Logarytmowanie obrazu	28
Rozdział 4. Operacje sumowania arytmetycznego obrazów barwowych	31
4.1 Sumowanie (określonej) stałej z obrazem	31
4.2 Sumowanie dwóch obrazów	33
4.3 Mnożenie obrazu przez zadaną liczbę	36
4.4 Mnożenie obrazu przez inny obraz	39
4.5 Mieszanie obrazów z określonym współczynnikiem	42
4.6 Potęgowanie obrazu	44
4.7 Dzielenie obrazu przez (zadaną) liczbę	47
4.8 Dzielenie obrazu przez inny obraz	49
4.9 Pierwiastkowanie obrazu	52
4.10 Logarytmowanie obrazu	54
Rozdział 5. Operacje geometryczne na obrazie	57
Rozdział 6. Operacje na histogramie obrazu szarego	58
Rozdział 7. Operacje na histogramie obrazu barwowego	59
Rozdział 8. Operacje morfologiczne na obrazach binarnych	60
Rozdział 9. Operacje morfologiczne na obrazach szarych	61
Rozdział 10. Filtrowanie liniowe i nielinowe	62

<i>Spis treści</i>	3
Rozdział 11. Podsumowanie	63
Bibliografia	64

Spis rysunków

3.1	(Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	8
3.2	(Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	9
3.3	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstał w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	11
3.4	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstał w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	11
3.5	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji	13
3.6	(Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji	13
3.7	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstał w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	15
3.8	(Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstał w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	16
3.9	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.5$, poniżej obraz wynikowy po normalizacji	18
3.10	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.8$, poniżej obraz wynikowy po normalizacji	18
3.11	(Od lewej) Szary obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=2$, obraz po normalizacji	20
3.12	(Od lewej) Szary obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=3$, obraz po normalizacji	20
3.13	(Od lewej) Szary obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji	22
3.14	(Od lewej) Szary obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji	22
3.15	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku podzielenia obrazów, poniżej obraz wynikowy po normalizacji	24
3.16	(Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku podzielenia obrazów, poniżej obraz wynikowy po normalizacji	24
3.17	(Od lewej) Szary obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym ($\alpha=1/2$), obraz po normalizacji	26
3.18	(Od lewej) Szary obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego ($\alpha=1/3$), obraz po normalizacji	26

3.19 (Od lewej) Szary obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	28
3.20 (Od lewej) Szary obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	28
 4.1 (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	31
4.2 (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	32
4.3 (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstał w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	34
4.4 (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstał w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji	34
4.5 (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 50, obraz po normalizacji	37
4.6 (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 100, obraz po normalizacji	37
4.7 (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstał w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	39
4.8 (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstał w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji	40
4.9 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.5$, poniżej obraz wynikowy po normalizacji	42
4.10 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.8$, poniżej obraz wynikowy po normalizacji	43
4.11 (Od lewej) Barwowy obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=2$, obraz po normalizacji	44
4.12 (Od lewej) Barwowy obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=3$, obraz po normalizacji	45
4.13 (Od lewej) Barwowy obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji	47
4.14 (Od lewej) Barwowy obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji	47
4.15 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku dzielenia obrazów, poniżej obraz wynikowy po normalizacji	49
4.16 (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku dzielenia obrazów, poniżej obraz wynikowy po normalizacji	50
4.17 (Od lewej) Barwowy obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym ($\alpha=1/2$), obraz po normalizacji	52
4.18 (Od lewej) Barwowy obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego ($\alpha=1/3$), obraz po normalizacji	52
4.19 (Od lewej) Barwowy obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	54
4.20 (Od lewej) Barwowy obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	54

Rozdział 1

Wstęp

Laboratoria oh oh... [1]

1.1 Specyfikacja wykorzystanego formatu obrazu

1.2 Instrukcja obsługi programu

Rozdział 2

Operacje ujednolicania obrazów

1. ujednolicenie obrazów szarych geometryczne (liczba wierszy i kolumn piksli)
2. ujednolicenie obrazów szarych rozdzielczościowe (w rastrze)
3. ujednolicenie obrazów RGB geometryczne (liczba wierszy i kolumn piksli)
4. ujednolicenie obrazów RGB rozdzielczościowe (w rastrze)

Rozdział 3

Operacje sumowania arytmetycznego obrazów szarych

Arytmetyczne operacje między pikslami p i q dwóch obrazów są używane w wielu dzia-łach przetwarzania obrazów. Przeprowadzane się je wykonując działania na pojedynczych pikslach i są uwarunkowane wymaganiami zależnymi od typu operacji. Po operacjach arytmetycznych zwykle niezbędna jest normalizacja. W przedstawionych zadaniach do normalizacji wykorzystano wzór:

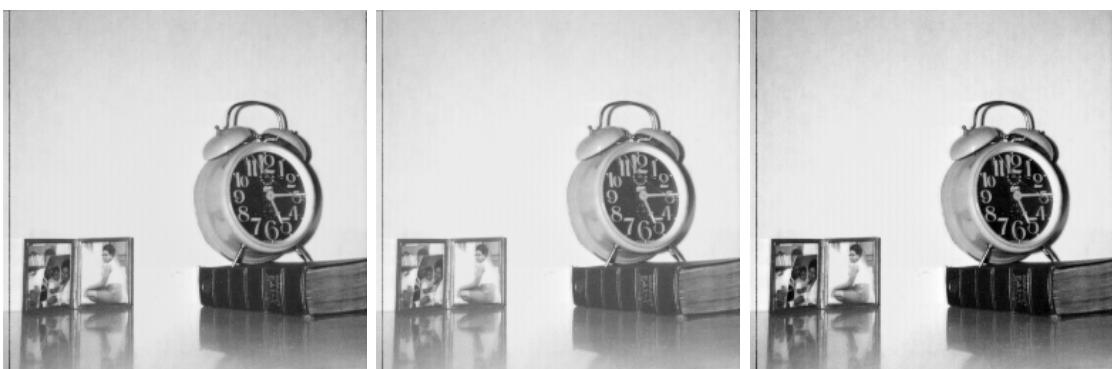
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

3.1 Sumowanie (określonej) stałej z obrazem

Algorytm sumowania obrazu szarego z określona stałą polega na dodaniu do każdej wartości pojedynczego piksla stałej liczby. Po operacji sumowania następuje normalizacja obrazu.

1. Policz sumy wartości kazdego piksla ze stałą (*const*).
2. Jeżeli jedna z tych sum jest większa niż 255 to:
3. Wybierz największą sumę Q_{max} i policz D_{max} ze wzoru: $D_{max}[i, j] = (Q_{max}[i, j] - 255)$
4. Oblicz $X = D_{max}/255$
5. Policz sumy ze wzoru

$$Q[i, j] = P[i, j] - (P[i, j] * X) + const - (const * X)$$



Rysunek 3.1: (Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji



Rysunek 3.2: (Od lewej) Szary obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji

Listing 3.1: Sumowanie obrazu szarego ze stałą

```

image_matrix = self.im1
width = image_matrix.shape[1]      # szereoksc
height = image_matrix.shape[0]     # wysokosc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):
        # Obliczanie sumy
        L = int(image_matrix[x][y]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < L:
            Q_max = L

# Sprawdzenie czy przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255)

```

```

# Obliczenie sumy z uwzględnieniem zakresu
for y in range(height):
    for x in range(width):
        L = (image_matrix[x][y] - (image_matrix[x][y] * X)) + (
            const - (const * X))

    # Zaokrąglenie do najbliższej wartości całkowitej z
    # gory
    # i przypisanie wartości
    result_matrix[x][y] = math.ceil(L)

    # Poszukiwanie minimum i maksimum
    if f_min > L:
        f_min = L
    if f_max < L:
        f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min) /
            (f_max - f_min))

```

3.2 Sumowanie dwóch obrazów



Rysunek 3.3: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 3.4: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji

Listing 3.2: Sumowanie obrazów szarych

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sumy
        L = int(image1_matrix[x][y]) + int(image2_matrix[x][y])

        # Poszukiwanie maksimum
        if Q_max < L:
            Q_max = L

# Sprawdzenie czy przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255)

# Obliczenie sumy z uwzglednieniem zakresu
for y in range(height):
    for x in range(width):
        L = (image1_matrix[x][y] - (image1_matrix[x][y] * X)) +
            (image2_matrix[x][y] - (image2_matrix[x][y] * X))

        # Zaokroglenie do najblzszej wartosci calkowitej z gory
        # i przypisanie wartosci
        result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum

```

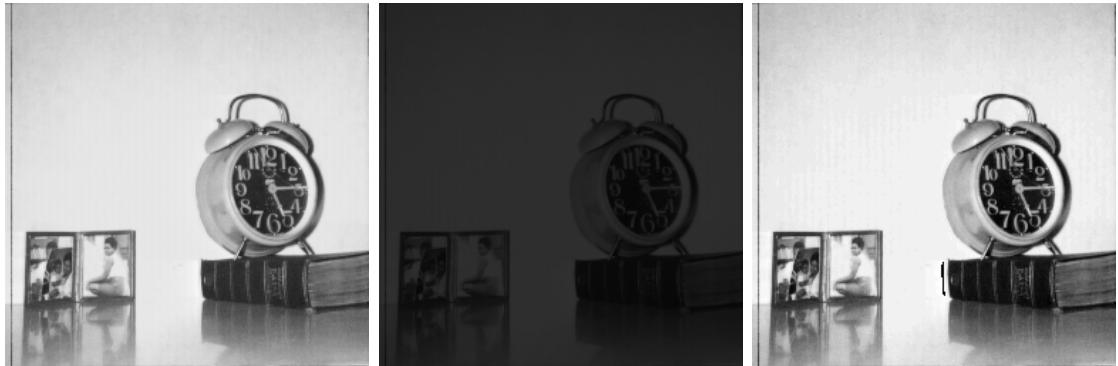
```

if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

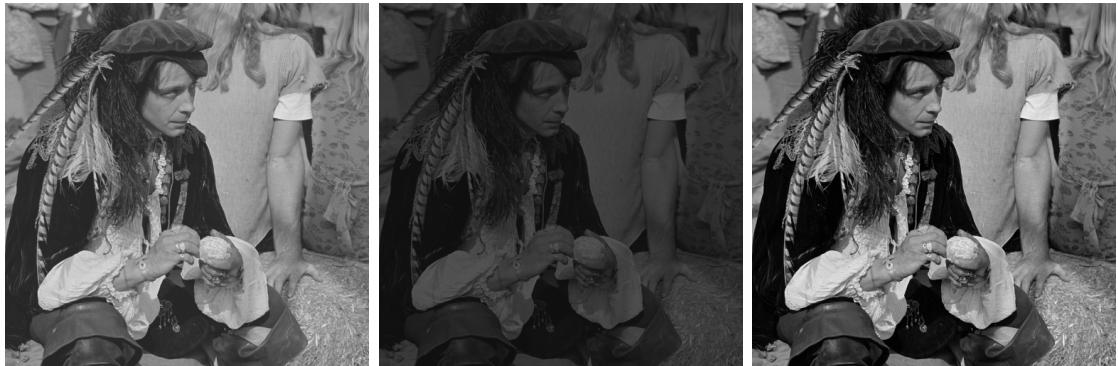
# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min
            ) / (f_max - f_min))

```

3.3 Mnożenie obrazu przez zadaną liczbę



Rysunek 3.5: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji



Rysunek 3.6: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji

Listing 3.3: Mnożenie obrazu szarego przez zadaną liczbę

```

image1_matrix = self.im1
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

# Mnożenie
for y in range(height):
    for x in range(width):

        L = int(image1_matrix[x][y])
        if L == 255:
            L = const
        elif L == 0:
            L = 0
        else:
            L = (int(image1_matrix[x][y]) * const)/255

        # Zaokrąglenie do najbliższej wartości całkowitej z gory
        # i przypisanie wartości
        result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum
if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min)
            / (f_max - f_min))

```

3.4 Mnożenie obrazu przez inny obraz



Rysunek 3.7: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstał w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 3.8: (Od lewej) Pierwsze dwa to szare obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 3.4: Mnożenie obrazu szarego przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        L = int(image1_matrix[x][y])
        if L == 255:
            L = image2_matrix[x][y]
        elif L == 0:
            L = 0

```

```
else:  
    L = (int(image1_matrix[x][y]) * int(image2_matrix[x]  
        ][y]))/255  
  
    # Zaokrąglenie do najbliższej wartości całkowitej z  
    # gory  
    # i przypisanie wartości  
    result_matrix[x][y] = math.ceil(L)  
  
    # Poszukiwanie minimum i maksimum  
    if f_min > L:  
        f_min = L  
    if f_max < L:  
        f_max = L  
  
# Normalizacja  
norm_matrix = np.zeros((width, height), dtype=np.uint8)  
for y in range(height):  
    for x in range(width):  
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min  
            ) / (f_max - f_min))
```



Rysunek 3.9: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.5$, poniżej obraz wynikowy po normalizacji



Rysunek 3.10: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.8$, poniżej obraz wynikowy po normalizacji

Listing 3.5: Mieszanie obrazów szarych z określonym współczynnikiem

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        L = float(image1_matrix[x][y]) * alfa + (1-alfa) *
            float(image2_matrix[x][y])

        # Zaokroglenie do najblizszej wartosci calkowitej z
        gory
# i przypisanie wartosci
        result_matrix[x][y] = math.ceil(L)

        # Poszukiwanie minimum i maksimum
        if f_min > L:
            f_min = L
        if f_max < L:
            f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min) /
            (f_max - f_min))

```

3.6 Potęgowanie obrazu (z zadaną potęgą)



Rysunek 3.11: (Od lewej) Szary obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=2$, obraz po normalizacji



Rysunek 3.12: (Od lewej) Szary obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=3$, obraz po normalizacji

Listing 3.6: Potęgowanie obrazu szarego (z zadaną potęgą)

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereokosc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
f_img_max = 0

for y in range(height):
    for x in range(width):
        
```

```

L = int(image_matrix[x][y])

# Poszukiwanie maksimum
if f_img_max < L:
    f_img_max = L

for y in range(height):
    for x in range(width):

        L = int(image_matrix[x][y])
        if L == 255:
            L = 255
        elif L == 0:
            L = 0
        else:
            L = math.pow(int(image_matrix[x][y]) / f_img_max,
                         alfa) * 255

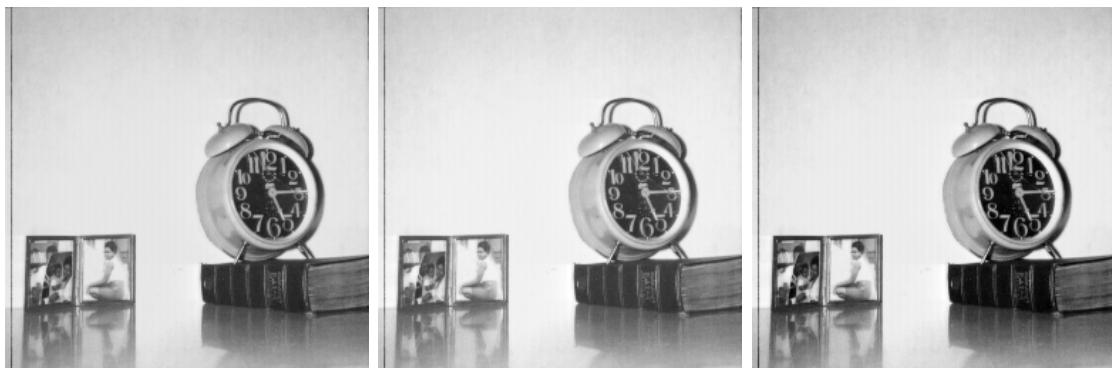
        # Zaokroglenie do najblizszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y] = math.ceil(L)

# Poszukiwanie minimum i maksimum
if f_min > L:
    f_min = L
if f_max < L:
    f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min)
                                    / (f_max - f_min))

```

3.7 Dzielenie obrazu przez (zadaną) liczbę



Rysunek 3.13: (Od lewej) Szary obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji



Rysunek 3.14: (Od lewej) Szary obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji

Listing 3.7: Dzielenie obrazu szarego przez (zadaną) liczbę

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
Q_max = 0

for y in range(height):
    for x in range(width):
        
```

```

L = int(image_matrix[x][y]) + int(const)

# Poszukiwanie maksimum
if Q_max < L:
    Q_max = L

for y in range(height):
    for x in range(width):

        L = int(image_matrix[x][y]) + int(const)
        Q_L = (L * 255) / Q_max

        # Zaokroglenie do najblzszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y] = math.ceil(Q_L)

        # Poszukiwanie minimum i maksimum
        if f_min > Q_L:
            f_min = Q_L
        if f_max < Q_L:
            f_max = Q_L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min)
            / (f_max - f_min))

```

3.8 Dzielenie obrazu przez przez inny obraz



Rysunek 3.15: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku podzielenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 3.16: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku podzielenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 3.8: Dzielenie obrazu szarego przez przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
Q_max = 0

for y in range(height):
    for x in range(width):
        # Obliczanie sumy
        L = int(image1_matrix[x][y]) + int(image2_matrix[x][y])

        # Poszukiwanie maksimum
        if Q_max < L:
            Q_max = L

for y in range(height):
    for x in range(width):

        # Obliczanie sumy
        L = int(image1_matrix[x][y]) + int(image2_matrix[x][y])
        Q_L = (L * 255) / Q_max

        # Zaokroglenie do najblzszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y] = math.ceil(Q_L)

        # Poszukiwanie minimum i maksimum
        if f_min > Q_L:
            f_min = Q_L
        if f_max < Q_L:
            f_max = Q_L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)

```

```

for y in range( height ):
    for x in range( width ):
        norm_matrix[x][y] = 255 * (( result_matrix[x][y] - f_min
        ) / (f_max - f_min))
    
```

3.9 Pierwiastkowanie obrazu



Rysunek 3.17: (Od lewej) Szary obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym ($\alpha=1/2$), obraz po normalizacji



Rysunek 3.18: (Od lewej) Szary obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego ($\alpha=1/3$), obraz po normalizacji

Listing 3.9: Pierwiastkowanie obrazu szarego

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc
    
```

```

result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
f_img_max = 0

alfa = 1/deg # Zamiana stopnia pierwiastka na ulamek

for y in range(height):
    for x in range(width):

        L = int(image_matrix[x][y])

        # Poszukiwanie maksimum
        if f_img_max < L:
            f_img_max = L

        for y in range(height):
            for x in range(width):

                L = int(image_matrix[x][y])
                if L == 255:
                    L = 255
                elif L == 0:
                    L = 0
                else:
                    L = math.pow(int(image_matrix[x][y]) / f_img_max,
                                alfa) * 255

                # Zaokroglenie do najblizszej wartosci calkowitej z
                gory
                # i przypisanie wartosci
                result_matrix[x][y] = math.ceil(L)

        # Poszukiwanie minimum i maksimum
        if f_min > L:
            f_min = L
        if f_max < L:
            f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)

```

```

for y in range( height ):
    for x in range( width ):
        norm_matrix[x][y] = 255 * (( result_matrix[x][y] - f_min )
            ) / (f_max - f_min))
    
```

3.10 Logarytmowanie obrazu



Rysunek 3.19: (Od lewej) Szary obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji



Rysunek 3.20: (Od lewej) Szary obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji

Listing 3.10: Logarytmowanie obrazu szarego

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc
    
```

```
result_matrix = np.zeros((width, height), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
f_img_max = 0

for y in range(height):
    for x in range(width):

        L = int(image_matrix[x][y])

        # Poszukiwanie maksimum
        if f_img_max < L:
            f_img_max = L

for y in range(height):
    for x in range(width):

        L = int(image_matrix[x][y])

        if L == 0:
            L = 0
        else:
            L = (math.log(1 + L) / math.log(1 + f_img_max)) *
                 255

        # Zaokrąglenie do najbliższej wartości całkowitej z
        # gory
        # i przypisanie wartości
        result_matrix[x][y] = math.ceil(L)

        # Poszukiwanie minimum i maksimum
        if f_min > L:
            f_min = L
        if f_max < L:
            f_max = L

# Normalizacja
norm_matrix = np.zeros((width, height), dtype=np.uint8)
for y in range(height):
    for x in range(width):
```

```
norm_matrix[x][y] = 255 * ((result_matrix[x][y] - f_min  
) / (f_max - f_min))
```

Rozdział 4

Operacje sumowania arytmetycznego obrazów barwowych

Arytmetyczne operacje na obrazach barwowych przeprowadza się wykonując działania na pojedynczych pikslach i są uwarunkowane wymaganiami zależnymi od typu operacji. W przedstawionych zadaniach poruszamy się w przestrzeni barw RGB, a do normalizacji wykorzystano wzór:

$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

4.1 Sumowanie (określonej) stałej z obrazem

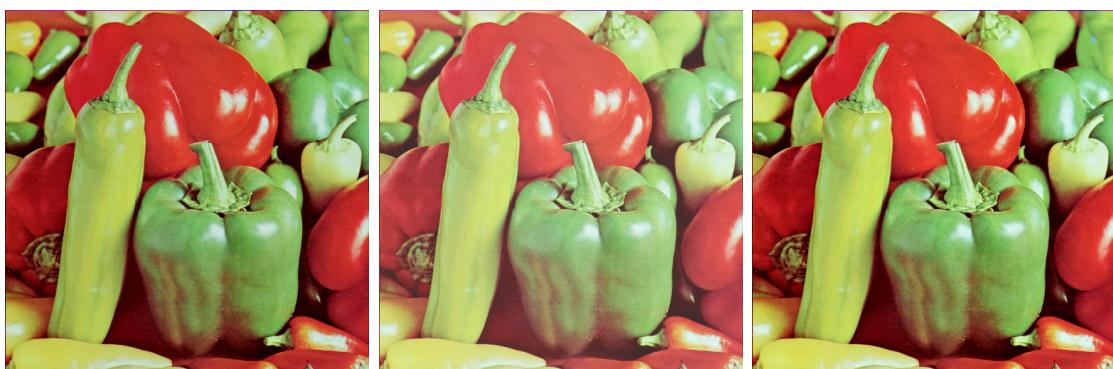
Algorytm sumowania obrazu barwowego z określona stałą polega na dodaniu do każdej składowej barwowej pojedynczego piksla stałej liczby. Po operacji sumowania następuje normalizacja obrazu.

1. Policz sumy wartości kazdego piksla ze stałą (*const*).
2. Jeżeli jedna z tych sum jest większa niż 255 to:
 3. Wybierz największą sumę Q_{max} i policz D_{max} ze wzoru: $D_{max}[i, j] = (Q_{max}[i, j] - 255)$
 4. Oblicz $X = D_{max}/255$
 5. Policz sumy ze wzoru

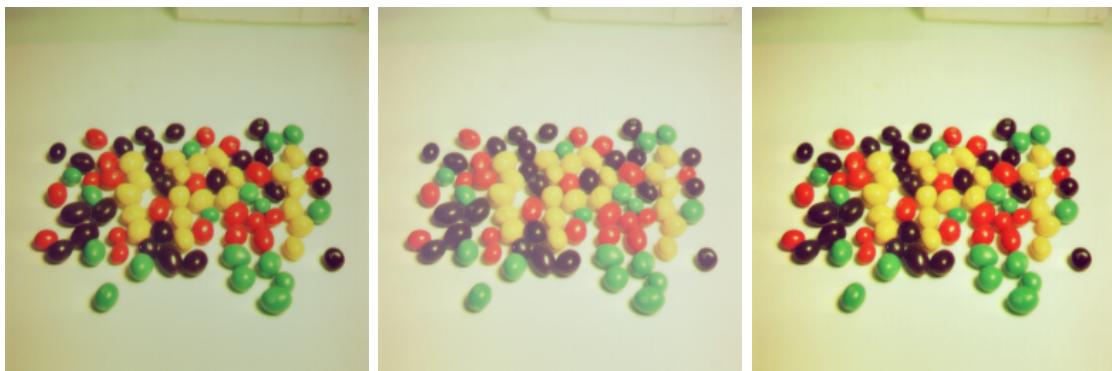
$$Q_R[i, j] = P_R[i, j] - (P_{1R} * X) + const - (const * X) - 1$$

$$Q_G[i, j] = P_G[i, j] - (P_G * X) + const - (const * X) - 1$$

$$Q_B[i, j] = P_B[i, j] - (P_B * X) + const - (const * X) - 1$$



Rysunek 4.1: (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 50, obraz po normalizacji



Rysunek 4.2: (Od lewej) Barwowy obraz wejściowy, obraz po sumowaniu ze stałą = 100, obraz po normalizacji

Listing 4.1: Sumowanie obrazu barwowego ze stałą

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R = int(image_matrix[x][y][0]) + int(const)
        G = int(image_matrix[x][y][1]) + int(const)
        B = int(image_matrix[x][y][2]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < max([R, G, B]):
            Q_max = max([R, G, B])

# Sprawdzenie czy maksimum przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255

```

```

X = (D_max/255) # Obliczenie proporcji

# Obliczenie sum z uwzglednieniem zakresu
for y in range(height):
    for x in range(width):
        R = (image_matrix[x][y][0] - (image_matrix[x][y][0] * X
            )) + (const - (const * X))
        G = (image_matrix[x][y][1] - (image_matrix[x][y][1] * X
            )) + (const - (const * X))
        B = (image_matrix[x][y][2] - (image_matrix[x][y][2] * X
            )) + (const - (const * X))

        # Zaokroglenie do najblizszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y][0] = math.ceil(R)
        result_matrix[x][y][1] = math.ceil(G)
        result_matrix[x][y][2] = math.ceil(B)

        # Poszukiwanie minimum i maksimum
        if f_min > min([R, G, B]):
            f_min = min([R, G, B])
        if f_max < max([R, G, B]):
            f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
            f_min) / (f_max - f_min))

```

4.2 Sumowanie dwóch obrazów



Rysunek 4.3: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 4.4: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku sumowania obrazów, poniżej obraz wynikowy po normalizacji

Listing 4.2: Sumowanie obrazów barwowych

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
Q_max = 0
D_max = 0
X = 0
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R = int(image1_matrix[x][y][0]) + int(image2_matrix[x][y][0])
        G = int(image1_matrix[x][y][1]) + int(image2_matrix[x][y][1])
        B = int(image1_matrix[x][y][2]) + int(image2_matrix[x][y][2])

        # Poszukiwanie maksimum
        if Q_max < max([R, G, B]):
            Q_max = max([R, G, B])

# Sprawdzenie czy maximum przekracza zakres
if Q_max > 255:
    D_max = Q_max - 255
    X = (D_max/255) # Obliczenie proporcji

# Obliczenie sum z uwzglednieniem zakresu
for y in range(height):
    for x in range(width):
        R = (image1_matrix[x][y][0] - (image1_matrix[x][y][0] * X)) + (image2_matrix[x][y][0] - (image2_matrix[x][y][0] * X))

```

```

G = (image1_matrix[x][y][1] - (image1_matrix[x][y][1] *
    X)) + (image2_matrix[x][y][1] - (image2_matrix[x][
    y][1] * X))
B = (image1_matrix[x][y][2] - (image1_matrix[x][y][2] *
    X)) + (image2_matrix[x][y][2] - (image2_matrix[x][
    y][2] * X))

# Zaokroglenie do najblizszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
            f_min) / (f_max - f_min))

```

4.3 Mnożenie obrazu przez zadaną liczbę



Rysunek 4.5: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 50, obraz po normalizacji



Rysunek 4.6: (Od lewej) Szary obraz wejściowy, obraz po przemnożeniu przez liczbę 100, obraz po normalizacji

Listing 4.3: Mnożenie obrazu barwowego przez zadaną liczbę

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        R = int(image_matrix[x][y][0])

```

```

G = int(image_matrix[x][y][1])
B = int(image_matrix[x][y][2])

if R == 255:
    R = const
elif R == 0:
    R = 0
else:
    R = (int(image_matrix[x][y][0]) * int(const))/255

if G == 255:
    G = const
elif G == 0:
    G = 0
else:
    G = (int(image_matrix[x][y][1]) * int(const))/255

if B == 255:
    B = const
elif B == 0:
    B = 0
else:
    B = (int(image_matrix[x][y][2]) * int(const))/255

# Zaokroglenie do najblzszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))

```

```
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -  
    f_min) / (f_max - f_min))
```

4.4 Mnożenie obrazu przez inny obraz



Rysunek 4.7: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstający w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 4.8: (Od lewej) Pierwsze dwa to barwowe obrazy wejściowe, następnie obraz powstały w wyniku przemnożenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 4.4: Mnożenie obrazu barwowego przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        R = int(image1_matrix[x][y][0])
        G = int(image1_matrix[x][y][1])
        B = int(image1_matrix[x][y][2])

        if R == 255:

```

```

R = image2_matrix[x][y][0]
elif R == 0:
    R = 0
else:
    R = (int(image1_matrix[x][y][0]) * int(
        image2_matrix[x][y][0]))/255

if G == 255:
    G = image2_matrix[x][y][1]
elif G == 0:
    G = 0
else:
    G = (int(image1_matrix[x][y][1]) * int(
        image2_matrix[x][y][1]))/255

if B == 255:
    B = image2_matrix[x][y][2]
elif B == 0:
    B = 0
else:
    B = (int(image1_matrix[x][y][2]) * int(
        image2_matrix[x][y][2]))/255

# Zaokrąglenie do najbliższej wartości całkowitej z
# gory
# i przypisanie wartości
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))

```

```
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -  
    f_min) / (f_max - f_min))
```

4.5 Mieszanie obrazów z określonym współczynnikiem



Rysunek 4.9: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstał w wyniku mieszania obrazów ze współczynnikami $\alpha=0.5$, poniżej obraz wynikowy po normalizacji



Rysunek 4.10: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku mieszania obrazów ze współczynnikiem $\alpha=0.8$, poniżej obraz wynikowy po normalizacji

Listing 4.5: Mieszanie obrazów barwowych z określonym współczynnikiem

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0

for y in range(height):
    for x in range(width):

        R = float(image1_matrix[x][y][0]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][0])
        G = float(image1_matrix[x][y][1]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][1])
        B = float(image1_matrix[x][y][2]) * alfa + (1-alfa) *
            float(image2_matrix[x][y][2])

        result_matrix[x][y] = [R, G, B]
    
```

```

B = float(image1_matrix[x][y][2]) * alfa + (1-alfa) *
    float(image2_matrix[x][y][2])

# Zaokroglenie do najblizszej wartosci calkowitej z gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
                                         f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
                                         f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
                                         f_min) / (f_max - f_min))

```

4.6 Potęgowanie obrazu



Rysunek 4.11: (Od lewej) Barwowy obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=2$, obraz po normalizacji



Rysunek 4.12: (Od lewej) Barwowy obraz wejściowy, obraz po podniesieniu do potęgi $\alpha=3$, obraz po normalizacji

Listing 4.6: Potęgowanie obrazu barwowego

```

image1_matrix = self.im1
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
f_img_max = 0

for y in range(height):
    for x in range(width):

        R = int(image1_matrix[x][y][0])
        G = int(image1_matrix[x][y][1])
        B = int(image1_matrix[x][y][2])

        if f_img_max < max([R, G, B]):
            f_img_max = max([R, G, B])

for y in range(height):
    for x in range(width):

        R = int(image1_matrix[x][y][0])
        G = int(image1_matrix[x][y][1])
        B = int(image1_matrix[x][y][2])
    
```

```

if R == 0:
    R = 0
else:
    R = 255 * (math.pow(int(image1_matrix[x][y][0]) /
        f_img_max, alfa))

if G == 0:
    G = 0
else:
    G = 255 * (math.pow(int(image1_matrix[x][y][1]) /
        f_img_max, alfa))

if B == 0:
    B = 0
else:
    B = 255 * (math.pow(int(image1_matrix[x][y][2]) /
        f_img_max, alfa))

# Zaokroglenie do najblzszej wartosci calkowitej z gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
            f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
            f_min) / (f_max - f_min))

```

4.7 Dzielenie obrazu przez (zadaną) liczbę



Rysunek 4.13: (Od lewej) Barwowy obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji



Rysunek 4.14: (Od lewej) Barwowy obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji

Listing 4.7: Dzielenie obrazu barwowego przez (zadaną) liczbę

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
Q_max = 0

```

```

for y in range( height ):
    for x in range( width ):

        # Obliczanie sum
        R_S = int(image_matrix[x][y][0]) + int(const)
        G_S = int(image_matrix[x][y][1]) + int(const)
        B_S = int(image_matrix[x][y][2]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < max([R_S, G_S, B_S]):
            Q_max = max([R_S, G_S, B_S])

for y in range( height ):
    for x in range( width ):

        # Obliczanie sum
        R_S = int(image_matrix[x][y][0]) + int(const)
        G_S = int(image_matrix[x][y][1]) + int(const)
        B_S = int(image_matrix[x][y][2]) + int(const)

        Q_R = (R_S * 255)/Q_max
        Q_G = (G_S * 255)/Q_max
        Q_B = (B_S * 255)/Q_max

        # Zaokroglenie do najblizszej wartosci calkowitej z gory
        # i przypisanie wartosci
        result_matrix[x][y][0] = math.ceil(Q_R)
        result_matrix[x][y][1] = math.ceil(Q_G)
        result_matrix[x][y][2] = math.ceil(Q_B)

        # Poszukiwanie minimum i maksimum
        if f_min > min([Q_R, Q_G, Q_B]):
            f_min = min([Q_R, Q_G, Q_B])
        if f_max < max([Q_R, Q_G, Q_B]):
            f_max = max([Q_R, Q_G, Q_B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range( height ):
    for x in range( width ):

```

```
norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -  
    f_min) / (f_max - f_min))  
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -  
    f_min) / (f_max - f_min))
```

4.8 Dzielenie obrazu przez inny obraz



Rysunek 4.15: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku dzielenia obrazów, poniżej obraz wynikowy po normalizacji



Rysunek 4.16: (Od lewej) Dwa obrazy wejściowe, następnie obraz powstały w wyniku dzielenia obrazów, poniżej obraz wynikowy po normalizacji

Listing 4.8: Dzielenie obrazu barwowego przez inny obraz

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
Q_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R_S = int(image1_matrix[x][y][0]) + int(image2_matrix[x][y][0])

```

```

G_S = int(image1_matrix[x][y][1]) + int(image2_matrix[x]
    ][y][1])
B_S = int(image1_matrix[x][y][2]) + int(image2_matrix[x]
    ][y][2])

# Poszukiwanie maksimum
if Q_max < max([R_S, G_S, B_S]):
    Q_max = max([R_S, G_S, B_S])

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R_S = int(image1_matrix[x][y][0]) + int(image2_matrix[x]
            ][y][0])
        G_S = int(image1_matrix[x][y][1]) + int(image2_matrix[x]
            ][y][1])
        B_S = int(image1_matrix[x][y][2]) + int(image2_matrix[x]
            ][y][2])

        Q_R = (R_S * 255)/Q_max
        Q_G = (G_S * 255)/Q_max
        Q_B = (B_S * 255)/Q_max

        # Zaokroglenie do najblizszej wartosci calkowitej z
        # gory
        # i przypisanie wartosci
        result_matrix[x][y][0] = math.ceil(Q_R)
        result_matrix[x][y][1] = math.ceil(Q_G)
        result_matrix[x][y][2] = math.ceil(Q_B)

        # Poszukiwanie minimum i maksimum
        if f_min > min([Q_R, Q_G, Q_B]):
            f_min = min([Q_R, Q_G, Q_B])
        if f_max < max([Q_R, Q_G, Q_B]):
            f_max = max([Q_R, Q_G, Q_B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
            f_min) / (f_max - f_min))

```

```

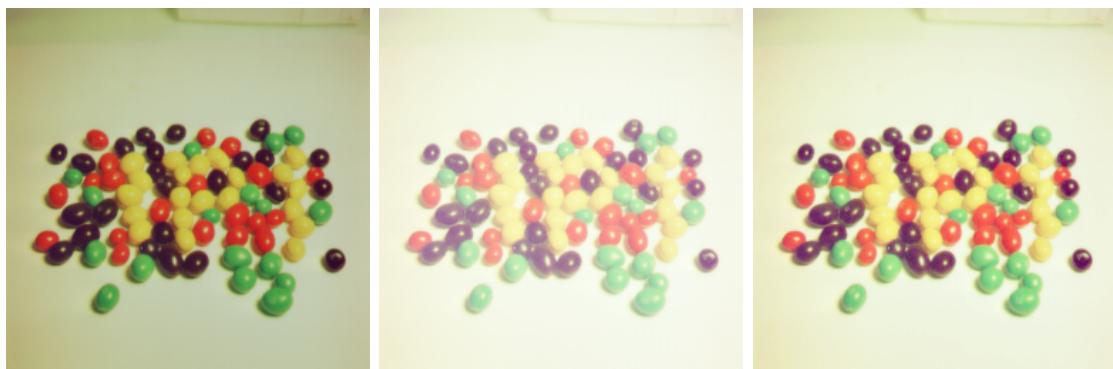
norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
    f_min) / (f_max - f_min))
norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
    f_min) / (f_max - f_min))

```

4.9 Pierwiastkowanie obrazu



Rysunek 4.17: (Od lewej) Barwowy obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym ($\alpha=1/2$), obraz po normalizacji



Rysunek 4.18: (Od lewej) Barwowy obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego ($\alpha=1/3$), obraz po normalizacji

Listing 4.9: Pierwiastkowanie obrazu barwowego

```

image_matrix = self.im1
height = image_matrix.shape[0]      # wysokosc
width = image_matrix.shape[1]       # szereoksc

```

```

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
Q_max = 0

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R_S = int(image_matrix[x][y][0]) + int(const)
        G_S = int(image_matrix[x][y][1]) + int(const)
        B_S = int(image_matrix[x][y][2]) + int(const)

        # Poszukiwanie maksimum
        if Q_max < max([R_S, G_S, B_S]):
            Q_max = max([R_S, G_S, B_S])

for y in range(height):
    for x in range(width):

        # Obliczanie sum
        R_S = int(image_matrix[x][y][0]) + int(const)
        G_S = int(image_matrix[x][y][1]) + int(const)
        B_S = int(image_matrix[x][y][2]) + int(const)

        Q_R = (R_S * 255) / Q_max
        Q_G = (G_S * 255) / Q_max
        Q_B = (B_S * 255) / Q_max

        # Zaokroglenie do najblizszej wartosci calkowitej z gory
        # i przypisanie wartosci
        result_matrix[x][y][0] = math.ceil(Q_R)
        result_matrix[x][y][1] = math.ceil(Q_G)
        result_matrix[x][y][2] = math.ceil(Q_B)

        # Poszukiwanie minimum i maksimum
        if f_min > min([Q_R, Q_G, Q_B]):
            f_min = min([Q_R, Q_G, Q_B])
        if f_max < max([Q_R, Q_G, Q_B]):
            f_max = max([Q_R, Q_G, Q_B])

```

```
# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
                                         f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
                                         f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
                                         f_min) / (f_max - f_min))
```

4.10 Logarytmowanie obrazu



Rysunek 4.19: (Od lewej) Barwowy obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji



Rysunek 4.20: (Od lewej) Barwowy obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji

Listing 4.10: Logarytmowanie obrazu barwowego

```

image1_matrix = self.im1
image2_matrix = self.im2
height = image1_matrix.shape[0]      # wysokosc
width = image1_matrix.shape[1]       # szereoksc

result_matrix = np.zeros((width, height, 3), dtype=np.uint8)

# Inicjalizacja zmiennych
f_min = 255
f_max = 0
f_img_max = 0

for y in range(height):
    for x in range(width):

        R = int(image1_matrix[x][y][0])
        G = int(image1_matrix[x][y][1])
        B = int(image1_matrix[x][y][2])

        # Poszukiwanie maksimum
        if f_img_max < max([R, G, B]):
            f_img_max = max([R, G, B])

for y in range(height):
    for x in range(width):

        R = int(image1_matrix[x][y][0])
        G = int(image1_matrix[x][y][1])
        B = int(image1_matrix[x][y][2])

        if R == 0:
            R = 0
        else:
            R = math.log(1 + int(image1_matrix[x][y][0])) /
                math.log(1 + int(f_img_max)) * 255

        if G == 0:
            G = 0
        else:

```

```

G = math.log(1 + int(image1_matrix[x][y][1])) /
math.log(1 + int(f_img_max)) * 255

if B == 0:
    B = 0
else:
    B = math.log(1 + int(image1_matrix[x][y][2])) /
math.log(1 + int(f_img_max)) * 255

# Zaokroglenie do najblizszej wartosci calkowitej z
# gory
# i przypisanie wartosci
result_matrix[x][y][0] = math.ceil(R)
result_matrix[x][y][1] = math.ceil(G)
result_matrix[x][y][2] = math.ceil(B)

# Poszukiwanie minimum i maksimum
if f_min > min([R, G, B]):
    f_min = min([R, G, B])
if f_max < max([R, G, B]):
    f_max = max([R, G, B])

# Normalizacja
norm_matrix = np.zeros((width, height, 3), dtype=np.uint8)
for y in range(height):
    for x in range(width):
        norm_matrix[x][y][0] = 255 * ((result_matrix[x][y][0] -
f_min) / (f_max - f_min))
        norm_matrix[x][y][1] = 255 * ((result_matrix[x][y][1] -
f_min) / (f_max - f_min))
        norm_matrix[x][y][2] = 255 * ((result_matrix[x][y][2] -
f_min) / (f_max - f_min))

```

Rozdział 5

Operacje geometryczne na obrazie

1. przemieszczenie obrazu o zadany wektor
2. jednorodne i niejednorodne skalowanie obrazu
3. obracanie obrazu o dowolny kąt
4. symetrie względem osi układu i zadanej prostej
5. wycinanie fragmentów obrazu
6. kopiowanie fragmentów obrazów

Rozdział 6

Operacje na histogramie obrazu szarego

1. obliczanie histogramu
2. przemieszczanie histogramu
3. rozciąganie histogramu
4. progowanie lokalne
5. progowanie globalne

Rozdział 7

Operacje na histogramie obrazu barwowego

1. obliczanie histogramu
2. przemieszczanie histogramu
3. rozciąganie histogramu
4. progowanie 1-progowe
5. progowanie wieloprogowe
6. progowanie lokalne
7. progowanie globalne

Rozdział 8

Operacje morfologiczne na obrazach binarnych

1. okrawanie(erozja)
2. nakładanie (dylatacja)
3. otwarcie
4. zamknięcie

Rozdział 9

Operacje morfologiczne na obrazach szarych

1. okrawanie(erozja)
2. nakładanie (dylatacja)
3. otwarcie
4. zamknięcie

Rozdział 10

Filtrowanie liniowe i nieliniowe

1. dolnoprzepustowe (dwa do wyboru)
2. górnoprzepustowe (Roberts, Prewitta, Sobella,)
3. gradientowe (kompasowe, płaskorzeźbowe kierunkowe, gradientu wektorowego VGO, gradientu wektora kierunkowego VDG).
4. medianowe
5. ekstremalne

Rozdział 11

Podsumowanie

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Bibliografia

- [1] Wojciech S. Mokrzycki. *Wprowadzenie do przetwarzania informacji wizualnej Tom II*. Akademicka Oficyna Wydawnicza EXIT, 2012.