

UNIWERSYTET KARDYNAŁA STEFANA WYSZYŃSKIEGO
W WARSZAWIE

WYDZIAŁ MATEMATYCZNO-PRZYRODNICZY
SZKOŁA NAUK ŚCISŁYCH

Jakub Kowalczyk

Szymon Kozakiewicz

**Wprowadzenie do Przetwarzania Obrazów
Sprawozdanie z projektu**

Prowadzący:
prof. Wojciech Mokrzycki

WARSZAWA 2019

Spis treści

1. Wstęp	3
1.1. Wykorzystny fortmat pliku obrazu	3
1.2. Instrukcja obsługi programu	3
2. Operacje ujednolicania obrazów	4
2.1. Ujednolicenie obrazów szarych geometryczne	4
2.2. Ujednolicenie obrazów szarych rozdzielczościowe	9
2.3. Ujednolicenie obrazów RGB geometryczne	15
2.4. Ujednolicenie obrazów RGB rozdzielczościowe	20
3. Operacje sumowania arytmetycznego obrazów szarych	24
3.1. Sumowanie (okresłonej) stałej z obrazem	24
3.2. Sumowanie dwóch obrazów	28
3.3. Mnożenie obrazu przez zadaną liczbę	33
3.4. Mnożenie obrazu przez inny obraz	37
3.5. Mieszanie obrazów z określonym współczynnikiem	42
3.6. Potęgowanie obrazu (z zadaną potęgą)	46
3.7. Dzielenie obrazu przez (zadaną) liczbę	49
3.8. Dzielenie obrazu przez inny obraz	52
3.9. Pierwiastkowanie obrazu	57
3.10. Logarytmowanie obrazu	61
4. Operacje sumowania arytmetycznego obrazów barwowych	64
4.1. Sumowanie (okresłonej) stałej z obrazem	64
4.2. Sumowanie dwóch obrazów	68
4.3. Mnożenie obrazu przez zadaną liczbę	74
4.4. Mnożenie obrazu przez inny obraz	78
4.5. Mieszanie obrazów z określonym współczynnikiem	83
4.6. Potęgowanie obrazu (z zadaną potęgą)	87
4.7. Dzielenie obrazu przez (zadaną) liczbę	91
4.8. Dzielenie obrazu przez inny obraz	95
4.9. Pierwiastkowanie obrazu	100
4.10. Logarytmowanie obrazu	104
Spis rysunków	109
Bibliografia	110

Rozdział 1

Wstęp

1.1. Wykorzystyty fortmat pliku obrazu

Do wykonania wymienionych w projekcie zadań został użyty format TIFF.

1.2. Instrukcja obsługi programu

Program został napisany w języku Python3. Do poprawnego działania programu potrzebny będzie za-instalowany język Python w wersji 3.6 i wyżej. Aby rozpocząć otrzymywanie wyników przetwarzania poszczególnych obrazów należy od komentować poszczególne linie kodu w pliku Main.py, w każdej selekcji rozdziałów jest pokazany jaki kod powinien być od komentowany(jego nagłówek nazywa się "Kod do wykonania danego problemu") , a potem należy uruchomić plik Main.py

Przykład uruchomienia programu w Windows: Uruchamiamy interpreter poleceń zwany cmd i wpisujemy komendę: python Main.py

Rozdział 2

Operacje ujednolicania obrazów

2.1. Ujednolicenie obrazów szarych geometryczne

Opis ćwiczenia

Algorytm geometrycznego ujednolicenia obrazów polega na doprowadzeniu z mniejszymi rozmiarami obrazu do większego rozmiarami obrazu, czyli doprowadzenie obrazów do takiej samej liczby wierszy i kolumn pikseli w każdym obrazie. Ten sposób ujednolicania nie powoduje spadku jakości obrazów.

Opis realizowanych operacji

1. Wybierz największą szerokość i największą wysokość z dwóch obrazów.
2. Dodaj i wypełni różnicę pikselami o wartości 1 w obrazie, który ma mniejszą szerokość albo wysokość.

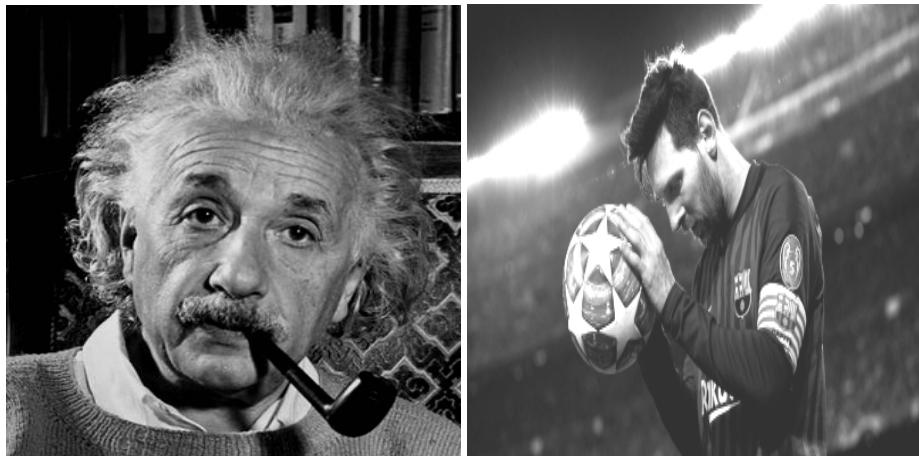
Kod do wykonania danego problemu

```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_GrayScale_Geometric(imageOne, imageTwo)
writeTiff("Geometric_GRAY_1", imageOne)
writeTiff("Geometric_GRAY_2", imageTwo)

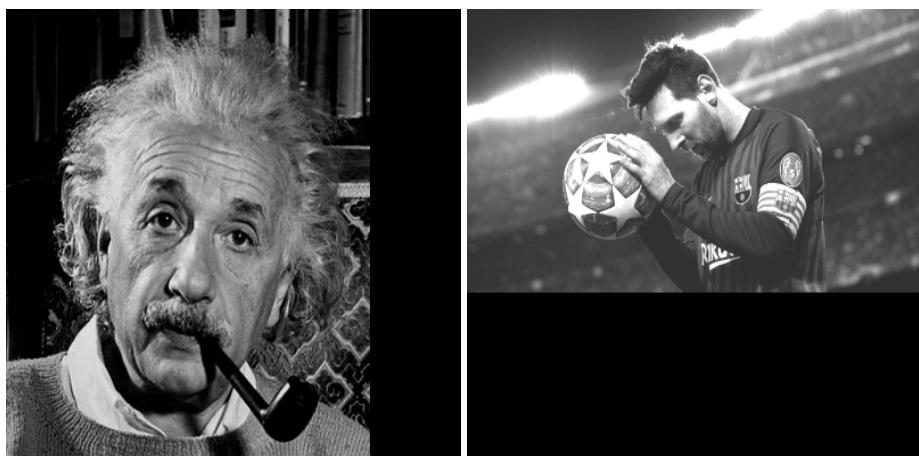
nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_GrayScale_Geometric(imageThree, imageFour)
```

```
writeTiff("Geometric_GRAY_3", imageThree)
writeTiff("Geometric_GRAY_4", imageFour)
```

Przeprowadzone testy



Rysunek 2.1: Obrazy wejściowe (od lewej): obraz 1 (320x361), obraz 2 (400x225)



Rysunek 2.2: Obrazy wyjściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)



Rysunek 2.3: Obrazy wejściowe (od lewej): obraz 1 (605x512), obraz 2 (332x250)



Rysunek 2.4: Obrazy wyjściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)

Kod funkcji

Listing 2.1: Geometryczne ujednolicianie obrazów szarych

```
def unification_Grayscale_Geometric(image1, image2):

    if (image1.imageColor == 0 or image1.imageColor == 1) and
       (image2.imageColor == 0 or image2.imageColor == 1):

        # Dodanie i wypelnienie roznice pikselami o wartosci 1
        # w obrazie 2, ktory ma mniejsza wysokosc.
        if image1.imageLength > image2.imageLength:
            print("Dlugosc obrazu 1 jest wieksza.")

            for i in range(image2.imageLength, image1.imageLength):
                image2.imageData.append([[1]])
                for j in range(image2.imageWidth - 1):
                    image2.imageData[i].append([1])

        image2.imageLength = image1.imageLength

    else:
        # Dodanie i wypelnienie roznice pikselami o wartosci 1
        # w obrazie 1, ktory ma mniejsza wysokosc.
        if image1.imageLength < image2.imageLength:
            print("Dlugosc obrazu 2 jest wieksza.")

            for i in range(image1.imageLength, image2.imageLength):
                image1.imageData.append([[1]])
                for j in range(image1.imageWidth - 1):
                    image1.imageData[i].append([1])

        image1.imageLength = image2.imageLength

    else:
        print("Dlugosc obrazu 1 i 2 jest rowna.")

    # Dodanie i wypelnienie roznice pikselami o wartosci 1
    # w obrazie 2, ktory ma mniejsza szerokosc.
    if image1.imageWidth > image2.imageWidth:
        print("Szerokosc obrazu 1 jest wieksza.")

        for i in range(image2.imageLength):
            for j in range(image2.imageWidth, image1.imageWidth):
                image2.imageData[i].append([1])

    image2.imageWidth = image1.imageWidth
```

```

else:
    # Dodanie i wypelnienie roznice pikselami o wartosci 1
    # w obrazie 1, ktory ma mniejsza szerokosc.
    if image1.imageWidth < image2.imageWidth:
        print("Szerokosc obrazu 2 jest wieksza.")

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth, image2.imageWidth):
            image1.imageData[i].append([1])

    image1.imageWidth = image2.imageWidth

else:
    print("Szerokosc obrazu 1 i 2 jest rowna.")

else:
    raise Exception("Ta funkcja sluzy do ujednolicenia geometrycznie
    obrazow SZARYCH, a ktorys obraz jest RGB.")

```

2.2. Ujednolicenie obrazów szarych rozdzielczościowe

Opis ćwiczenia

Algorytm rozdzielczościowego ujednolicenia obrazów następuje po ujednoliceniu geometrycznym i polega na doprowadzenie obydwóch obrazów do takie samej interpretacji fotometrycznej, głębi kolorów.

Opis realizowanych operacji

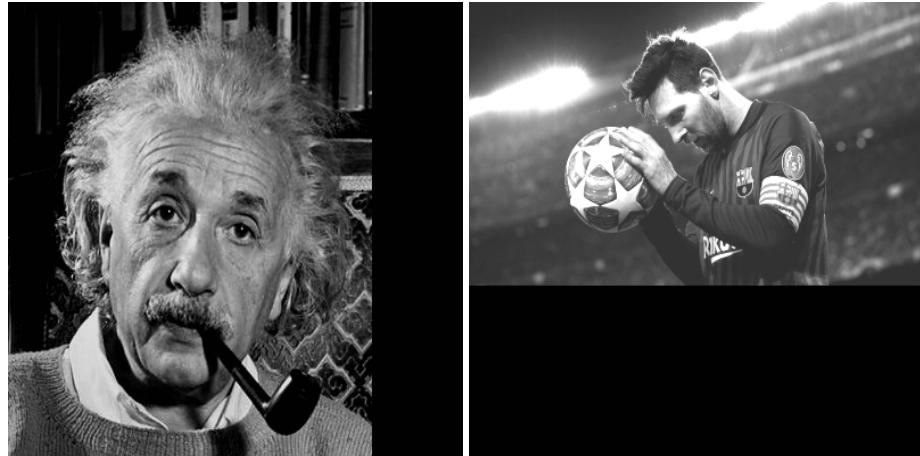
1. Doprowadź obydwa obrazy do takiej samej interpolacji fonetycznej.
2. Doprowadź obydwa obrazy do takiej samej głębi kolorów.

Kod do wykonania danego problemu

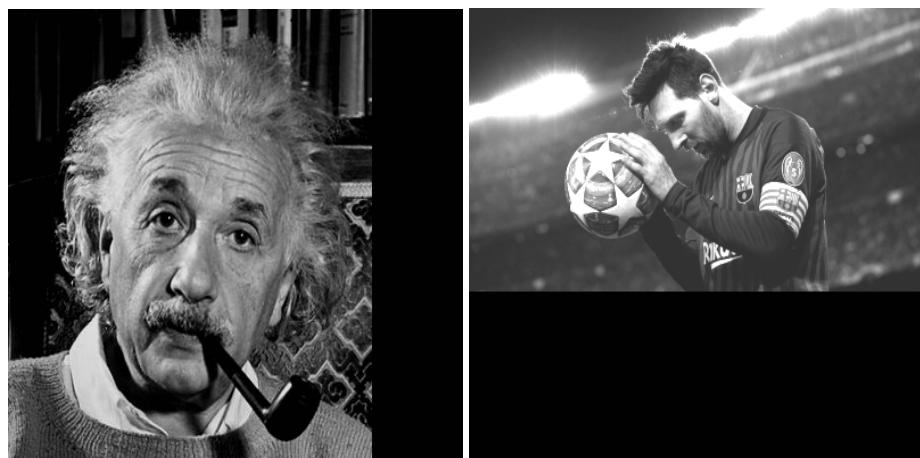
```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_Grayscale_Resolution(imageOne, imageTwo)
writeTiff("Resolution_GRAY_3", imageOne)
writeTiff("Resolution_GRAY_4", imageTwo)

nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_Grayscale_Resolution(imageThree, imageFour)
writeTiff("Resolution_GRAY_3", imageThree)
writeTiff("Resolution_GRAY_4", imageFour)
```

Przeprowadzone testy



Rysunek 2.5: Obrazy wejściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)



Rysunek 2.6: Obrazy wyjściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)



Rysunek 2.7: Obrazy wejściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)



Rysunek 2.8: Obrazy wyjściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)

Kod funkcji

Listing 2.2: Rozdzielczościowe ujednolicianie obrazów szarych

```
def unification_Grayscale_Resolution(image1, image2):

    if (image1.imageColor == 0 or image1.imageColor == 1) and
       (image2.imageColor == 0 or image2.imageColor == 1):

        # Doprzeprowadzenie obydwoch obrazow do takiej samej glebi kolorow
        if image1.imageBitsColor[0] != image2.imageBitsColor[0]:

            if image1.imageBitsColor[0] == 4:
                print("Zmiana rozdzielczosci obrazu 1 z 4 na 8 bitow .")
                for i in range(image1.imageLength):
                    for j in range(image1.imageWidth):
                        image1.imageData[i][j][0] =
                            image1.imageData[i][j][0] * 16
                image1.imageBitsColor[0] = 8

            for i in range(image1.originalImageLength):
                for j in range(image1.originalImageWidth):
                    image1.originalImageData[i][j][0] =
                        image1.originalImageData[i][j][0] * 16

        else:
            print("Zmiana rozdzielczosci obrazu 2 z 4 na 8 bitow .")
            if image2.imageBitsColor[0] == 4:
                for i in range(image2.imageLength):
                    for j in range(image2.imageWidth):
                        image2.imageData[i][j][0] =
                            image2.imageData[i][j][0] * 16
                image2.imageBitsColor[0] = 8

            for i in range(image2.originalImageLength):
                for j in range(image2.originalImageWidth):
                    image2.originalImageData[i][j][0] =
                        image2.originalImageData[i][j][0] * 16

        else:
            raise Exception("programu 3")

    else:
        print("Rozdzielczosc obrazow 1 i 2 SZARYCH jest taka sama .")

    # Doprzeprowadzenie obydwoch obrazow do takiej samej
    # interpretacji fotometrycznej
    if image1.imageColor != image2.imageColor:
```

```

if image1.imageColor == 0:
    if image1.imageBitsColor[0] == 8:
        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                image1.imageData[i][j][0] =
                    255 - image1.imageData[i][j][0]

        for i in range(image1.originalImageLength):
            for j in range(image1.originalImageWidth):
                image1.originalImageData[i][j][0] =
                    255 - image1.originalImageData[i][j][0]

    else:
        if image1.imageBitsColor[0] == 4:
            for i in range(image1.imageLength):
                for j in range(image1.imageWidth):
                    image1.imageData[i][j][0] =
                        15 - image1.imageData[i][j][0]

            for i in range(image1.originalImageLength):
                for j in range(image1.originalImageWidth):
                    image1.originalImageData[i][j][0] =
                        15 - image1.originalImageData[i][j][0]

    else:
        raise Exception("programu_4")
image1.imageColor = 1

else:
    if image2.imageColor == 0:
        if image2.imageBitsColor[0] == 8:
            for i in range(image2.imageLength):
                for j in range(image2.imageWidth):
                    image2.imageData[i][j][0] =
                        255 - image2.imageData[i][j][0]

            for i in range(image2.originalImageLength):
                for j in range(image2.originalImageWidth):
                    image2.originalImageData[i][j][0] =
                        255 - image2.originalImageData[i][j][0]

    else:
        if image2.imageBitsColor[0] == 4:
            for i in range(image2.imageLength):
                for j in range(image2.imageWidth):
                    image2.imageData[i][j][0] =
                        15 - image2.imageData[i][j][0]

```


2.3. Ujednolicenie obrazów RGB geometryczne

Opis ćwiczenia

Algorytm geometrycznego ujednolicenia obrazów polega na doprowadzeniu z mniejszymi rozmiarami obrazu do większego rozmiarami obrazu, czyli doprowadzenie obrazów do takiej samej liczby wierszy i kolumn piksli w każdym obrazie. Ten sposób ujednolicania nie powoduje spadku jakości obrazów.

Opis realizowanych operacji

1. Wybierz największą szerokość i największą wysokość z dwóch obrazów.
2. Dodaj i wypełni różnicę pikselami o wartości 1 dla każdego z kanałów w obrazie, który ma mniejszą szerokość albo wysokość.

Kod do wykonania danego problemu

```
nameFileONE = 'img/RGBMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Geometric(imageOne, imageTwo)
writeTiff("Geometric_RGB_1", imageOne)
writeTiff("Geometric_RGB_2", imageTwo)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Geometric(imageThree, imageFour)
writeTiff("Geometric_RGB_3", imageThree)
writeTiff("Geometric_RGB_4", imageFour)
```

Przeprowadzone testy



Rysunek 2.9: Obrazy wejściowe (od lewej): obraz 1 (475x355), obraz 2 (425x275)



Rysunek 2.10: Obrazy wyjściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)



Rysunek 2.11: Obrazy wejściowe (od lewej): obraz 1 (500x400), obraz 2 (450x450)



Rysunek 2.12: Obrazy wyjściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)

Kod funkcji

Listing 2.3: Geometryczne ujednolicianie obrazów RGB

```
def unification_RGB_Geometric(image1, image2):

    if image1.imageColor == 2 and image2.imageColor == 2:

        # Dodanie i wypelnienie roznice pikselami o wartosci 1 w obrazie 2,
        # ktory ma mniejsza wysokosc.
        if image1.imageLength > image2.imageLength:
            print("Dlugosc obrazu 1 jest wieksza.")

            for i in range(image2.imageLength, image1.imageLength):
                image2.imageData.append([[1, 1, 1]])
                for j in range(image2.imageWidth - 1):
                    image2.imageData[i].append([1, 1, 1])
            image2.imageLength = image1.imageLength

        else:
            # Dodanie i wypelnienie roznice pikselami o wartosci 1 w obrazie 1,
            # ktory ma mniejsza wysokosc.
            if image1.imageLength < image2.imageLength:
                print("Dlugosc obrazu 2 jest wieksza.")

                for i in range(image1.imageLength, image2.imageLength):
                    image1.imageData.append([[1, 1, 1]])
                    for j in range(image1.imageWidth - 1):
                        image1.imageData[i].append([1, 1, 1])
                image1.imageLength = image2.imageLength

            else:
                print("Dlugosc obrazu 1 i 2 jest rowna.")

        # Dodanie i wypelnienie roznice pikselami o wartosci 1 w obrazie 2,
        # ktory ma mniejsza szerokosc.
        if image1.imageWidth > image2.imageWidth:
            print("Szerokosc obrazu 1 jest wieksza.")

            for i in range(image2.imageLength):
                for j in range(image2.imageWidth, image1.imageWidth):
                    image2.imageData[i].append([1, 1, 1])
            image2.imageWidth = image1.imageWidth

        else:
            # Dodanie i wypelnienie roznice pikselami o wartosci 1 w obrazie 1,
            # ktory ma mniejsza szerokosc.
```

```
if image1.imageWidth < image2.imageWidth:  
    print("Szerokosc obrazu 2 jest wieksza .")  
  
    for i in range(image1.imageLength):  
        for j in range(image1.imageWidth, image2.imageWidth):  
            image1.imageData[i].append([1, 1, 1])  
    image1.imageWidth = image2.imageWidth  
  
else:  
    print("Szerokosc obrazu 1 i 2 jest rowna .")  
  
else:  
    raise Exception("Ta funkcja sluzy do ujednolicenia geometrycznie  
    obrazow RGB, a ktorys obraz jest SZARY. ")
```

2.4. Ujednolicenie obrazów RGB rozdzielczościowe

Opis ćwiczenia

Algorytm rozdzielczościowego ujednolicenia obrazów następuje po ujednoliceniu geometrycznym i polega na doprowadzenie obydwóch obrazów do takie samej interpretacji fotometrycznej, głębi kolorów.

Opis realizowanych operacji

1. Doprowadź obydwa obrazy do takiej samej interpolacji fonetycznej.
2. Doprowadź obydwa obrazy do takiej samej głębi kolorów.

Kod do wykonania danego problemu

```
nameFileONE = 'img/RGBMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Resolution(imageOne, imageTwo)
writeTiff("Resolution_RGB_1", imageOne)
writeTiff("Resolution_RGB_2", imageTwo)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Resolution(imageThree, imageFour)
writeTiff("Resolution_RGB_3", imageThree)
writeTiff("Resolution_RGB_4", imageFour)
```

Przeprowadzone testy



Rysunek 2.13: Obrazy wejściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)



Rysunek 2.14: Obrazy wyjściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)



Rysunek 2.15: Obrazy wejściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)



Rysunek 2.16: Obrazy wyjściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)

Kod funkcji

Listing 2.4: Rozdzielczościowe ujednolicianie obrazów RGB

```
def unification_RGB_Resolution(image1, image2):

    if image1.imageColor == 2 and image2.imageColor == 2:

        # Doprowadzenie obydwoch obrazow do takiej samej glebi kolorow
        if image1.imageBitsColor[0] != image2.imageBitsColor[0]:

            if image1.imageBitsColor[0] == 4:
                for i in range(image1.imageLength):
                    for j in range(image1.imageWidth):
                        for k in range(3):
                            image1.imageData[i][j][k] =
                                image1.imageData[i][j][k] * 16
                for l in range(3):
                    image1.imageBitsColor[1] = 8

            for i in range(image1.originalImageLength):
                for j in range(image1.originalImageWidth):
                    for k in range(3):
                        image1.originalImageData[i][j][k] =
                            image1.originalImageData[i][j][k] * 16

        else:
            if image2.imageBitsColor[0] == 4:
                for i in range(image2.imageLength):
                    for j in range(image2.imageWidth):
                        for k in range(3):
                            image2.imageData[i][j][k] =
                                image2.imageData[i][j][k] * 16
                for l in range(3):
                    image2.imageBitsColor[1] = 8

                for i in range(image2.originalImageLength):
                    for j in range(image2.originalImageWidth):
                        for k in range(3):
                            image2.originalImageData[i][j][k] =
                                image2.originalImageData[i][j][k] * 16

            else:
                raise Exception("programu<u>3")

    else:
        print("Rozdzielczosc<u>obrazow<u>1<u>i<u>2<u>RGB<u>jest<u>taka<u>sama.")

    else:
        raise Exception("Ta<u>funkcja<u>sluzy<u>do<u>ujednolicenia<u>rozdzielczościowo<u>obrazow<u>RGB,<u>a<u>kto<u>y<u>obraz<u>jest<u>SZARY.")
```

Rozdział 3

Operacje sumowania arytmetycznego obrazów szarych

3.1. Sumowanie (okresłonej) stałej z obrazem

Opis ćwiczenia

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

Opis realizowanych operacji

1. Policz sumy wartości każdego piksla ze stałą.
2. Jeśli przynajmniej jedna z tych sum jest większa od głębi kolorów obrazu to:
3. Wybierz największą sumę Q_{max} i D_{max} policz ze wzoru
$$D_{max}[i, j] = (Q_{max}[i, j] - \text{głęb. koloru obrazu})$$
4. Policz proporcję $X = D_{max}[i, j]/\text{głęb. koloru obrazu}$
i zaokrąglając wynik do najbliższej z góry liczby całkowitej.
5. Jeśli nie ma sumy większej od głębi kolorów obrazu to $X=0$.
6. Policz sumy:
$$Q[i, j] = P[i, j] - (P[i, j] * X) + const - (const * X)$$
7. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'  
readFileTwo = ReadTiff(nameFileTWO)  
imageTwo = Image(readFileTwo)  
sum_const_grayscale(imageTwo, 50)  
  
nameFileTHREE = 'img/gLU.tif'  
readFileThree = ReadTiff(nameFileTHREE)  
imageThree = Image(readFileThree)  
sum_const_grayscale(imageThree, 100)
```

Przeprowadzone testy



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



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

Kod funkcji

Listing 3.1: Sumowanie (okresłonej) stałej z obrazem szarym

```
def sum_const_grayscale(image1, const=0):
    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
        if not (0 <= const <= 15):
            raise Exception("program do obrazow SZARYCH 4 bitowych
mozne dodac liczbe z zakresu 0-15, a podana liczba "
"to %d." % const)
    else:
        if image1.imageBitsColor[0] == 8:
            maxBitsColor = 255
            if not (0 <= const <= 255):
                raise Exception("program do obrazow SZARYCH 8 bitowych
mozne dodac liczbe z zakresu 0-255, a podana "
"podana to %d." % const)
        else:
            raise Exception("program dodaje jedynie obrazy SZARE 4, 8 bitowe
ze stala")
    Qmax = 0
    Dmax = 0
    X = 0
    fmax = 0
    fmin = 256

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):

            # obliczanie sumy obrazu z stala
            temp = image1.imageData[i][j][0] + const

            # poszukiwanie maksimum w obrazie
            if temp > Qmax:
                Qmax = temp

    # sprawdzenie, czy maksimum obrazu przekracza zakres
    if Qmax > maxBitsColor:
        Dmax = Qmax - maxBitsColor
        X = round(Dmax/maxBitsColor, 2)

    if X == 1.0:
        X = 0.99
```

```

# dodawanie obrazu ze stała z uzupełnieniem zakresu
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        tempSum = ceil((image1.imageData[i][j][0] -
                        (image1.imageData[i][j][0] * X)) + (const - (const * X)))
        image1.imageData[i][j][0] = tempSum

    # poszukiwanie maksimum
    if tempSum > fmax:
        fmax = tempSum

    # poszukiwanie minimum
    if tempSum < fmin:
        fmin = tempSum

writeTiff('add_const', image1)

# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
                                         ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
writeTiff('normalization_add_const', image1)

```

3.2. Sumowanie dwóch obrazów

Opis ćwiczenia

Algebraiczne sumowanie obrazów f i f' jest określone jedynie dla obrazów o tych samych wymiarach $M \times N$ i strukturze ich macierzy. Algorytm sumowania obrazu z obrazem polega na dodaniu do wartości piksla z pierwszego obrazu, wartości odpowiadającego piksla z drugiego obrazu. Po operacji sumowania następuje normalizacja obrazu. Dodawanie obrazów jest użyteczne w uśrednianiu obrazów, wykonywanym w celu zredukowania na nich szumu.

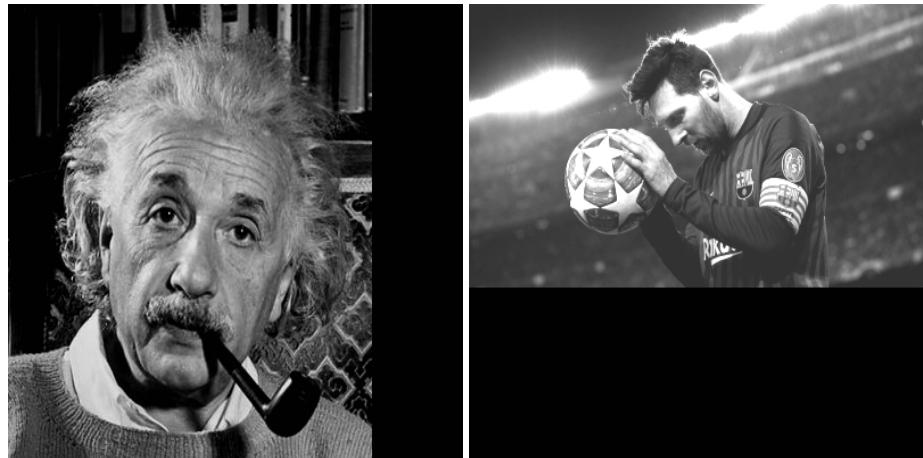
Opis realizowanych operacji

1. Policz sumy dla wszystkich piksli, odpowiadających składowych barw.
2. Jeśli przynajmniej jedna z tych sum jest większa od głębi kolorów obrazu to:
3. Wybierz największą sumę Q_{max} i D_{max} policz ze wzoru
$$D_{max}[i, j] = (Q_{max}[i, j] - \text{głęb. koloru obrazu})$$
4. Policz proporcję $X = D_{max}[i, j]/\text{głęb. koloru obrazu}$
i zaokrąglając wynik do najbliższej z góry liczby całkowitej.
5. Jeśli nie ma sumy większej od głębi kolorów obrazu to $X=0$.
6. Policz sumy:
$$Q[i, j] = P1[i, j] - (P1[i, j] * X) + P2[i, j] - (P2[i, j] * X)$$
7. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

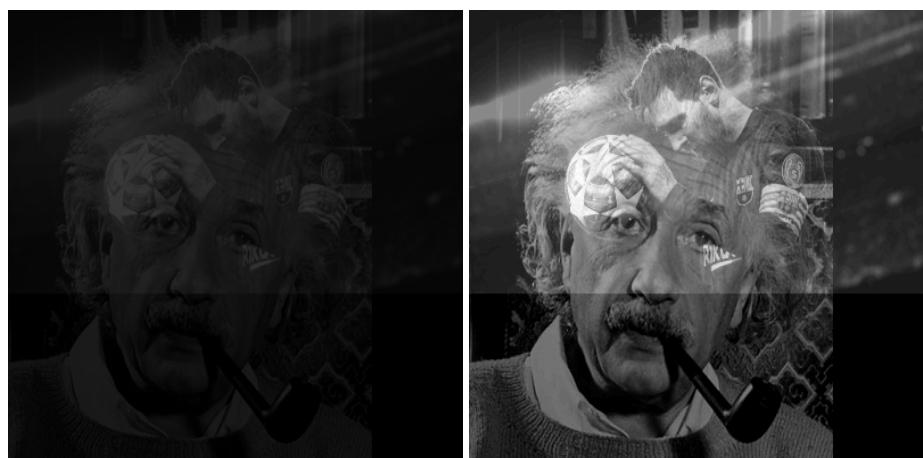
Kod do wykonania danego problemu

```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_GrayScale_Geometric(imageOne, imageTwo)
unification_GrayScale_Resolution(imageOne, imageTwo)
sum_two_images_grayscale(imageOne, imageTwo)
nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_GrayScale_Geometric(imageThree, imageFour)
unification_GrayScale_Resolution(imageThree, imageFour)
sum_two_images_grayscale(imageThree, imageFour)
```

Przeprowadzone testy



Rysunek 3.3: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.4: Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji



Rysunek 3.5: Obrazy wejściowe (od lewej):pierwszy obraz wejściowy,drugi obraz wejściowy



Rysunek 3.6: Obrazy wyjściowe (od lewej): obraz powstający w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 3.2: Sumowanie dwóch obrazów szarych

```
def sum_two_images_grayscale(image1, image2):

    global maxBitsColor
    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        else:
            if image1.imageBitsColor[0] == 8:
                maxBitsColor = 255

        else:
            raise Exception("program dodaje jedynie obrazy SZARE 4, 8 bitowe
                           oraz obrazy musza miec takie same rozmiary .")

    Qmax = 0
    Dmax = 0
    X = 0
    fmax = 0
    fmin = 256

    # obliczanie sumy dwoch obrazow
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            temp = image1.imageData[i][j][0] + image2.imageData[i][j][0]

            # poszukiwanie maksimum w sumie obrazow
            if temp > Qmax:
                Qmax = temp

    # sprawdzenie , czy maksimum sumowanego obrazu przekracza zakres
    if Qmax > maxBitsColor:
        Dmax = Qmax - maxBitsColor
        X = round(Dmax/maxBitsColor, 2)

    if X == 1.0:
        X = 0.99

    # dodawanie dwoch obrazu z uzwgljeniem zakresu
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
```

```

tempSum = round((image1.imageData[i][j][0] -
(image1.imageData[i][j][0] * X)) +
(image2.imageData[i][j][0] - (image2.imageData[i][j][0] * X)))
image1.imageData[i][j][0] = tempSum

# poszukiwanie maksimum
if tempSum > fmax:
    fmax = tempSum

# poszukiwanie minimum
if tempSum < fmin:
    fmin = tempSum

writeTiff('add_two_image', image1)

# normalizacja wynikowego obrazu
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
        ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

writeTiff('normalization_add_two_image', image1)

```

3.3. Mnożenie obrazu przez zadaną liczbę

Opis ćwiczenia

Mnożenie obrazu f przez skalar α wykonuję się mnożąc każdy element obrazu $f_{i,j}$ przez skalar $f_{i,j} * \alpha$.

Opis realizowanych operacji

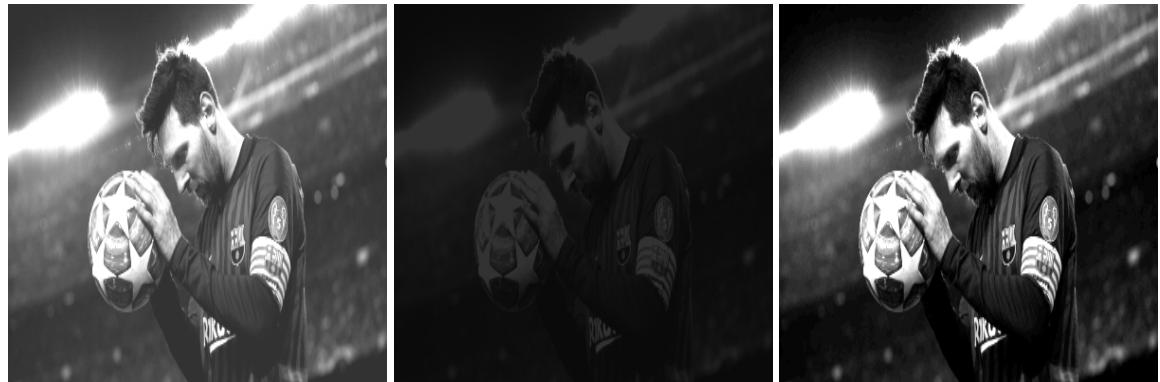
1. Dla wszystkich pikseli w obrazie wykonaj:
2. Jeśli składowa barwy piksla ma maksymalną głębię koloru to składowa wynikowa otrzymuje wartość odpowiadającą wartości stałej
3. W przeciwnym wypadku, jeśli składowa barwy pikslu ma wartość 0 to składowa wynikowa otrzymuje wartość 0
4. W przeciwnym wypadku mnóż odpowiednie sobie składowe, a wynik dziel przez maksymalną głębię koloru, zaokrąglając do najbliższej większej liczby całkowitej
5. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
multiplication_const_grayscale(imageTwo, 50)

nameFileTHREE = 'img/gLU.tif'
readFileThree = ReadTiff(nameFileTHREE)
multiplication_const_grayscale(imageThree, 100)
```

Przeprowadzone testy



Rysunek 3.7: (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji



Rysunek 3.8: (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji

Kod funkcji

Listing 3.3: Mnożenie obrazu przez zadaną liczbę

```
def multiplication_const_grayscale(image1, const=1):

    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == 4:

        maxBitsColor = 15
        if not (0 < const <= 15):
            raise Exception("program do obrazow SZARYCH 4 bitowych moze pomnozyc liczbe z zakresu 0-15, a podana liczba"
                             " to %d." % const)
    else:
        if image1.imageBitsColor[0] == 8:

            maxBitsColor = 255
            if not (0 < const <= 255):
                raise Exception("program do obrazow SZARYCH 8 bitowych moze pomnozyc liczbe z zakresu 0-255, a podana"
                                 " podana to %d." % const)
        else:
            raise Exception("program mnozy jedynie obrazy SZARE 4, 8 bitowe
                           ze stala")

    # mnozenie
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempMult = image1.imageData[i][j][0]

            if tempMult == maxBitsColor:
                tempMult = const

            elif tempMult == 0:
                tempMult = 0

            else:
                tempMult = ceil((image1.imageData[i][j][0] * const) /
                               maxBitsColor)

            image1.imageData[i][j][0] = tempMult

            if tempMult > fmax:
                fmax = tempMult
```

```
if tempMult < fmin:  
    fmin = tempMult  
  
writeTiff('multi_const', image1)  
  
# normalization  
for i in range(image1.imageLength):  
    for j in range(image1.imageWidth):  
        image1.imageData[i][j][0] = round(maxBitsColor *  
            ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))  
  
writeTiff('normalization_multi_const', image1)
```

3.4. Mnożenie obrazu przez inny obraz

Opis ćwiczenia

Algebraiczne mnożenie obrazów $f \times f'$ jest określone jedynie dla obrazów o tych samych wymiarach $M \times N$ i strukturze ich macierzy i wykonuje się mnożąc każdy element obrazu P_1 przez odpowiadający piksel drugiego obrazu P_2 .

Opis realizowanych operacji

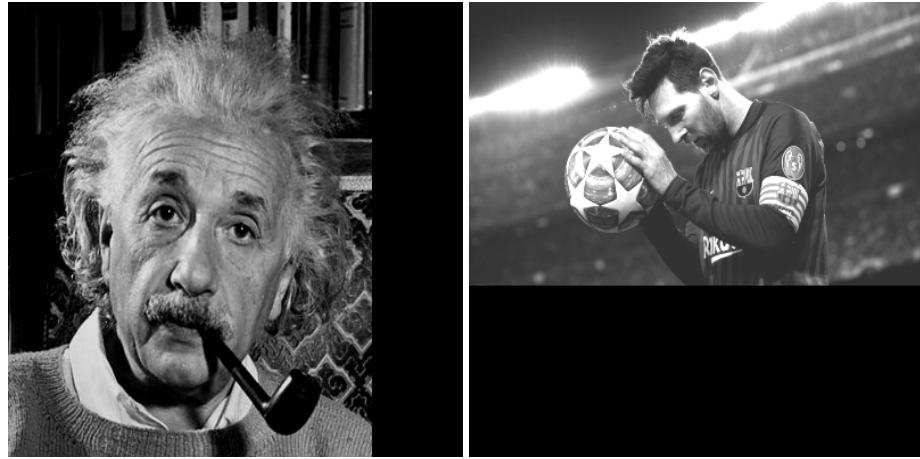
1. Weź dwa identycznych rozmiarów obrazy $P_1 \times P_2$
2. Dla wszystkich pikseli w obrazie wykonaj:
3. Jeśli składowa barwy piksla ma maksymalną głębię koloru to składowa wynikowa otrzymuje wartość odpowiadającą wartości składowej drugiego obrazu
4. W przeciwnym wypadku, jeśli składowa barwy pikslu ma wartość 0 to składowa wynikowa otrzymuje wartość 0
5. W przeciwnym wypadku mnóż odpowiednie sobie składowe, a wynik dziel przez maksymalną głębię koloru, zaokrąglając do najbliższej większej liczby całkowitej
6. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_Grayscale_Geometric(imageOne, imageTwo)
unification_Grayscale_Resolution(imageOne, imageTwo)
multiplication_two_images_grayscale(imageOne, imageTwo)

nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_Grayscale_Geometric(imageThree, imageFour)
unification_Grayscale_Resolution(imageThree, imageFour)
multiplication_two_images_grayscale(imageThree, imageFour)
```

Przeprowadzone testy



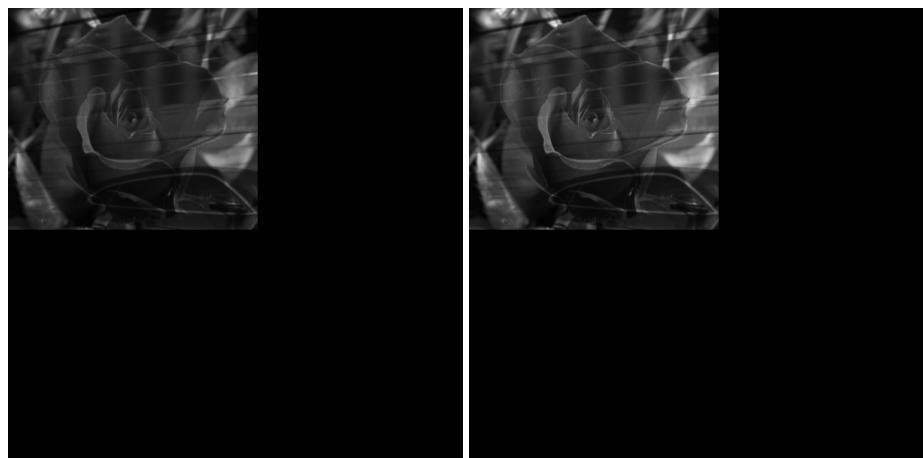
Rysunek 3.9: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.10: Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji



Rysunek 3.11: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.12: Obrazy wyjściowe (od lewej): obraz powstający w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 3.4: Mnożenie dwóch obrazów

```
def multiplication_two_images_grayscale(image1, image2):

    global maxBitsColor
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

        else:
            raise Exception("program umnozy jedynie obrazy SZARE, 8-bitowe
                             oraz obrazy musza miec takie same rozmiary.")

    # mnozenie
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempMult = image1.imageData[i][j][0]

            if tempMult == maxBitsColor:
                tempMult = image2.imageData[i][j][0]

            elif tempMult == 0:
                tempMult = 0

            else:
                tempMult = ceil((image1.imageData[i][j][0] *
                               image2.imageData[i][j][0]) / maxBitsColor)

            image1.imageData[i][j][0] = tempMult

            if tempMult > fmax:
                fmax = tempMult

            if tempMult < fmin:
                fmin = tempMult

    writeTiff('multi_two_images', image1)
```

```
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
            ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

writeTiff('normalization_multi_two_images', image1)
```

3.5. Mieszanie obrazów z określonym współczynnikiem

Opis ćwiczenia

Mieszanie dwóch obrazów polega na sumowaniu ich z wagami α i $(1 - \alpha)$, odpowiednio według wzoru: $f_m = f * \alpha + f' * (1 - \alpha)$, gdzie $\alpha \in [0, 1]$. Płynna zmiana parametru α w przedziale $[0, 1]$ powoduje efekt przechodzenia obrazu f' w obraz f

Opis realizowanych operacji

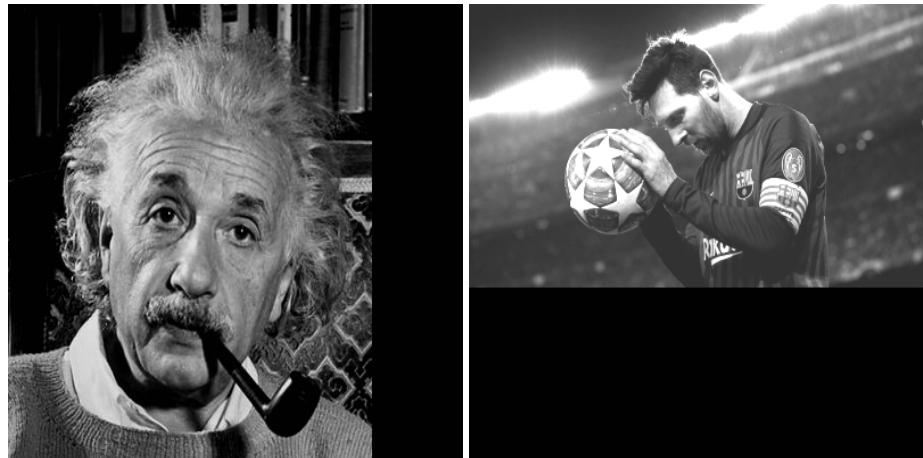
1. Weź dwa identycznych rozmiarów obrazy P_1 i P_2
2. Określ współczynnik mieszania α wyrażony jako liczba rzeczywista z zakresu $<0, 1>$; 0 reprezentuje pełną przezroczystość, 1 - nieprzezroczystości
3. Dla wszystkich pikseli w obrazach wejściowych wykonuj: $Q(i, j) = \alpha * P_1(i, j) + (1 - \alpha) * P_2(i, j)$

Kod do wykonania danego problemu

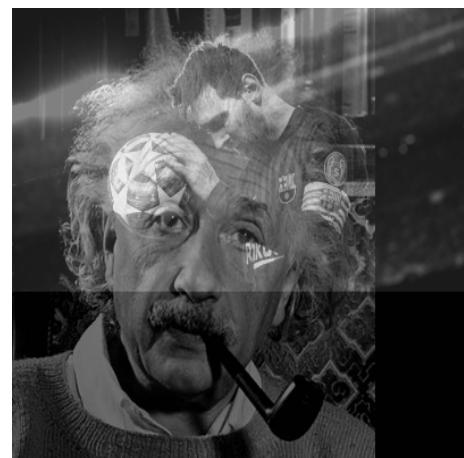
```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_Grayscale_Geometric(imageOne, imageTwo)
unification_Grayscale_Resolution(imageOne, imageTwo)
mixing_images_grayscale(imageOne, imageTwo, 0.5)

nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_Grayscale_Geometric(imageThree, imageFour)
unification_Grayscale_Resolution(imageThree, imageFour)
mixing_images_grayscale(imageThree, imageFour, 0.8)
```

Przeprowadzone testy



Rysunek 3.13: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.14: Obrazy wyjściowe (od lewej): obraz powstały w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.5$



Rysunek 3.15: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.16: Obrazy wyjściowe (od lewej): obraz powstający w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.8$

Kod funkcji

Listing 3.5: Mieszanie obrazów z określonym współczynnikiem

```
def mixing_images_grayscale(image1, image2, scales=0.0):

    global maxBitsColor
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if not (0.0 <= scales <= 1.0):
            raise Exception("program miesza obrazy SZARE z waga
zakresu 0.0 - 1.0, a podana liczba to %f ." %scales)

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

    else:
        raise Exception("program miesza jedynie obrazy 4, 8 bitowe
oraz obrazy musza miec takie same rozmiary .")

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempMix = ceil(scales * image1.imageData[i][j][0] +
                           (1 - scales) * image2.imageData[i][j][0])
            image1.imageData[i][j][0] = tempMix

            if tempMix > fmax:
                fmax = tempMix

            if tempMix < fmin:
                fmin = tempMix

    writeTiff('mix_two_image', image1)
```

3.6. Potęgowanie obrazu (z zadaną potęgą)

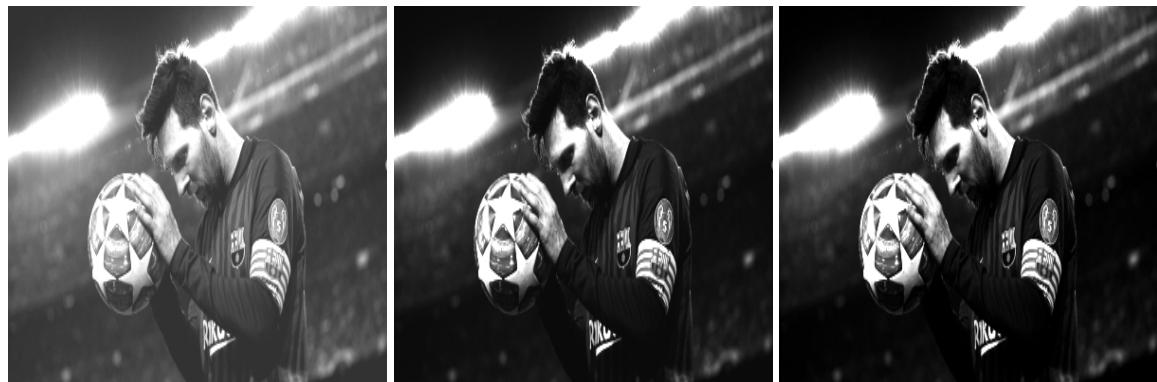
Opis ćwiczenia

Potęgowanie obrazu jest szczególnym przypadkiem operacji mnożenia obrazów.

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'  
readFileTwo = ReadTiff(nameFileTWO)  
imageTwo = Image(readFileTwo)  
  
pow_image_grayscale(imageTwo, 2)  
  
nameFileTHREE = 'img/gLU.tif'  
readFileThree = ReadTiff(nameFileTHREE)  
imageThree = Image(readFileThree)  
  
pow_image_grayscale(imageThree, 3)
```

Przeprowadzone testy



Rysunek 3.17: (Od lewej):obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 2$, obraz po normalizacji



Rysunek 3.18: (Od lewej):obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 3$, obraz po normalizacji

Kod funkcji

Listing 3.6: Potęgowanie obrazu

```

def pow_image_grayscale(image1 , p=1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    if not (0 < p):
        raise Exception("program poteguje obraz SZARY u z zadana potega
uuuuuuuuu z zakresu p > 0, a podana liczba "
                      "to %d ." % p)

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPow = image1.imageData[i][j][0]
            if tempPow > fmaximage:
                fmaximage = tempPow

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPow = image1.imageData[i][j][0]

            if tempPow == maxBitsColor:
                tempPow = maxBitsColor
            elif tempPow == 0:

```

```

tempPow = 0
else :
    tempPow = pow(image1.imageData[i][j][0] / fmaximage , p) *
    maxBitsColor

image1.imageData[i][j][0] = ceil(tempPow)

if tempPow > fmax:
    fmax = tempPow

if tempPow < fmin:
    fmin = tempPow

writeTiff('pow_image' , image1)
# normalization
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
        ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

writeTiff('normalization_pow_image' , image1)

```

3.7. Dzielenie obrazu przez (zadaną) liczbę

Opis realizowanych operacji

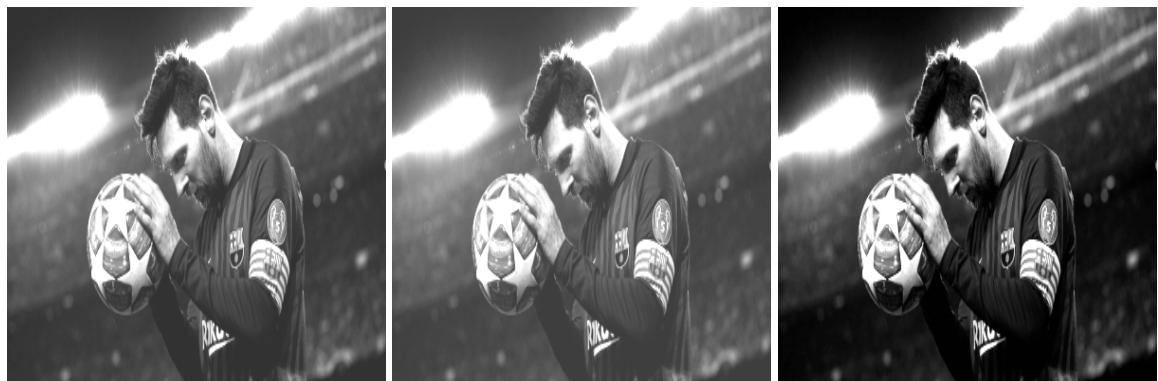
1. Dla wszystkich piksli w tym obrazie wykonaj:
2. Policz sumę piksli ze stałą
3. Wybierz największą sumę Q_{max} i policz równania $Q[i, j] = (S * \text{maksymalna gęścia koloru}) / Q_{max}$. Wynik zaokrąglaj do najbliższej górnej liczby całkowitej
4. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
division_const_grayscale(imageTwo, 15)

nameFileTHREE = 'img/gLU.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
division_const_grayscale(imageThree, 3)
```

Przeprowadzone testy



Rysunek 3.19: (Od lewej):obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji



Rysunek 3.20: (Od lewej):obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji

Kod funkcji

Listing 3.7: Dzielenie obrazu przez liczbę

```
def division_const_grayscale(image1, const=1):

    Qmax = 0
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == 4:

        maxBitsColor = 15
        if not (0 < const <= 15):
            raise Exception("program do obrazow SZARYCH 4 bitowych
moze dzielic liczbe z zakresu 0-15, a podana liczba "
"to %d." % const)
    elif image1.imageBitsColor[0] == 8:

        maxBitsColor = 255
        if not (0 < const <= 255):
            raise Exception("program do obrazow SZARYCH 8 bitowych
moze dzielic liczbe z zakresu 0-255, a podana "
"podana to %d." % const)
    else:
        raise Exception("program dzieli jedynie obrazy SZARE 4, 8 bitowe
ze stala")

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempDiv = image1.imageData[i][j][0] + const

            if Qmax < tempDiv:
                Qmax = tempDiv
```

```

for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        tempDiv = image1.imageData[i][j][0] + const

        resultDiv = (tempDiv * maxBitsColor) / Qmax

        image1.imageData[i][j][0] = ceil(resultDiv)

        if resultDiv > fmax:
            fmax = resultDiv

        if resultDiv < fmin:
            fmin = resultDiv

writeTiff('div_image_const', image1)
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

writeTiff('normalization_div_image_const', image1)

```

3.8. Dzielenie obrazu przez inny obraz

Opis realizowanych operacji

1. Weź dwa identycznych rozmiarów obrazy P_1 i P_2
2. Dla wszystkich pikseli w obrazie wykonaj:
3. Policz sumy pikseli ze stałą
4. Wybierz największą sumę Q_{max} i policz równanie:
$$Q[i, j] = (S - \text{maksymalna gęścia koloru}) / Q_{max}$$

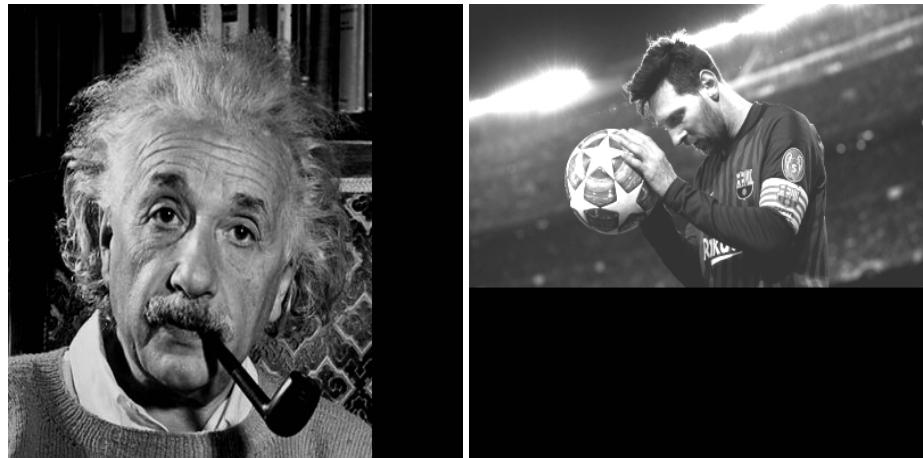
Wynik zaokrągluj do najbliższej górnej liczby całkowitej.
5. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileONE = 'img/gae.tif'
nameFileTWO = 'img/gMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_Grayscale_Geometric(imageOne, imageTwo)
unification_Grayscale_Resolution(imageOne, imageTwo)
division_two_images_grayscale(imageOne, imageTwo)

nameFileTHREE = 'img/gLU.tif'
nameFileFOUR = 'img/groza.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_Grayscale_Geometric(imageThree, imageFour)
unification_Grayscale_Resolution(imageThree, imageFour)
division_two_images_grayscale(imageThree, imageFour)
```

Przeprowadzone testy



Rysunek 3.21: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.22: Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji



Rysunek 3.23: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 3.24: Obrazy wyjściowe (od lewej): obraz powstający w wyniku podzielenia obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 3.8: Dzielenie dwóch obrazów

```
def division_two_images_grayscale(image1, image2):

    global maxBitsColor
    fmax = 0
    fmin = 256
    Qmax = 0

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

        else:
            raise Exception("program dzieli jedynie obrazy SZARE, 8-bitowe
                            oraz obrazy musza miec takie same rozmiary.")

        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                tempDiv = image1.imageData[i][j][0] + image2.imageData[i][j][0]

                if Qmax < tempDiv:
                    Qmax = tempDiv

        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                tempDiv = image1.imageData[i][j][0] + image2.imageData[i][j][0]

                resultDiv = (tempDiv * maxBitsColor) / Qmax

                image1.imageData[i][j][0] = ceil(resultDiv)

                if resultDiv > fmax:
                    fmax = resultDiv

                if resultDiv < fmin:
                    fmin = resultDiv

    writeTiff('div_two_images', image1)
    # normalizacja
```

```
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
            ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

writeTiff('normalization_div_two_images', image1)
```

3.9. Pierwiastkowanie obrazu

Opis ćwiczenia

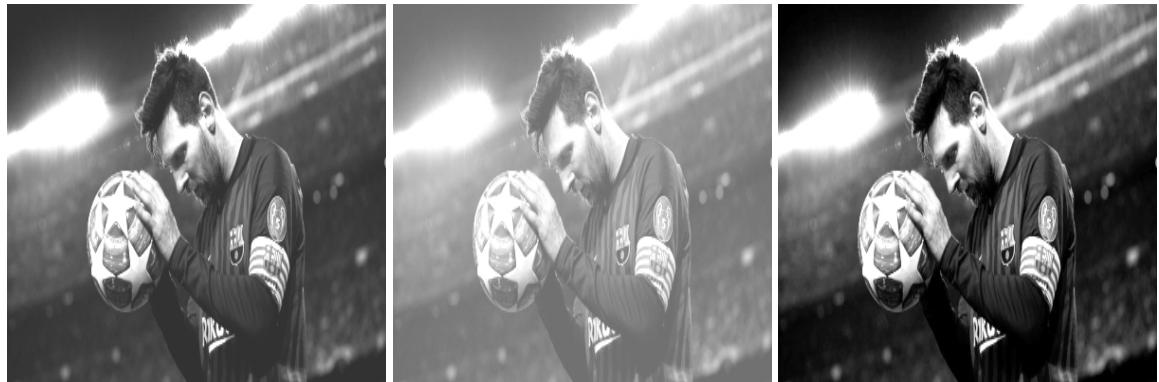
Pierwiastkowanie obrazu jest szczególnym przypadkiem operacji potęgowania obrazów, gdzie wykładnikiem jest ułamek.

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
sqrt_image_grayscale(imageTwo, 2)

nameFileTHREE = 'img/gLU.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
sqrt_image_grayscale(imageThree, 3)
```

Przeprowadzone testy



Rysunek 3.25: (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym, obraz po normalizacji



Rysunek 3.26: (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego, obraz po normalizacji

Kod funkcji

Listing 3.9: Pierwiastkowanie obrazu

```
def sqrt_image_grayscale(image1, deg=1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    p = 1/deg

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPow = image1.imageData[i][j][0]
            if tempPow > fmaximage:
                fmaximage = tempPow

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPow = image1.imageData[i][j][0]

            if tempPow == maxBitsColor:
                tempPow = maxBitsColor
            elif tempPow == 0:
                tempPow = 0
            else:
                tempPow = pow(image1.imageData[i][j][0] / fmaximage, p) * maxBitsColor

            image1.imageData[i][j][0] = ceil(tempPow)

            if tempPow > fmax:
                fmax = tempPow

            if tempPow < fmin:
                fmin = tempPow

    writeTiff('root_image', image1)
    # normalizacja
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
```

```
image1.imageData[i][j][0] = round(maxBitsColor *  
((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))  
  
writeTiff('normalization_root_image', image1)
```

3.10. Logarytmowanie obrazu

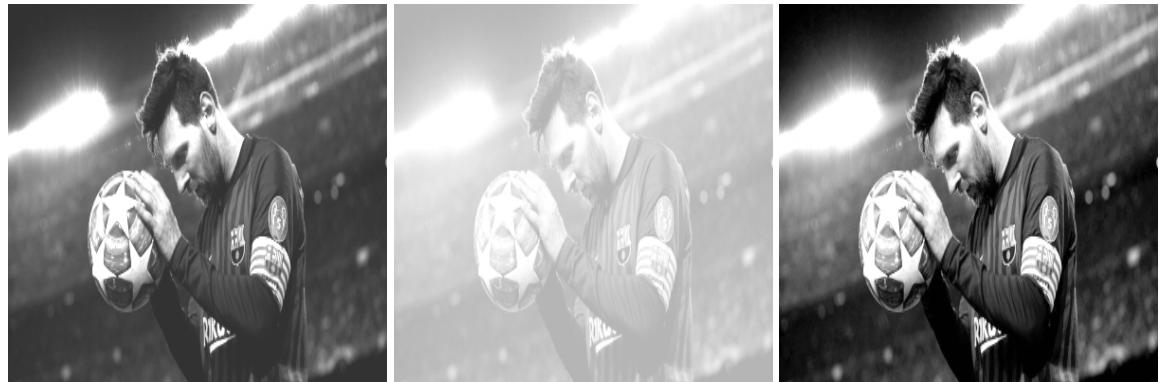
Opis ćwiczenia

Przesunięcie funkcji obrazowej f do góry o 1 przed jej logarytmowaniem wynika z nieokreśloności logarytmu w zerze. Logarytmowanie obrazu powoduje rozjaśnienie i zróżnicowanie najciemniejszych obszarów obrazu.

Kod do wykonania danego problemu

```
nameFileTWO = 'img/gMessi.tif'  
readFileTwo = ReadTiff(nameFileTWO)  
imageTwo = Image(readFileTwo)  
log_image_grayscale(imageTwo)  
  
nameFileTHREE = 'img/gLU.tif'  
readFileThree = ReadTiff(nameFileTHREE)  
imageThree = Image(readFileThree)  
log_image_grayscale(imageThree)
```

Przeprowadzone testy



Rysunek 3.27: (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji



Rysunek 3.28: (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji

Kod funkcji

Listing 3.10: Logarytmowanie obrazu

```
def log_image_grayscale(image1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempLog = image1.imageData[i][j][0]
            if fmaximage < tempLog:
                fmaximage = tempLog

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempLog = image1.imageData[i][j][0]

            if tempLog == 0:
                tempLog = 0
            else:
                tempLog = (log(1 + tempLog) / log(1+fmaximage))*maxBitsColor

            image1.imageData[i][j][0] = ceil(tempLog)

            if tempLog > fmax:
                fmax = tempLog

            if tempLog < fmin:
                fmin = tempLog

    writeTiff('log_image', image1)
    # normalizacja
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            image1.imageData[i][j][0] = round(maxBitsColor *
                ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))

    writeTiff('normalization_log_image', image1)
```

Rozdział 4

Operacje sumowania arytmetycznego obrazów barwowych

4.1. Sumowanie (okresłonej) stałej z obrazem

Opis ćwiczenia

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

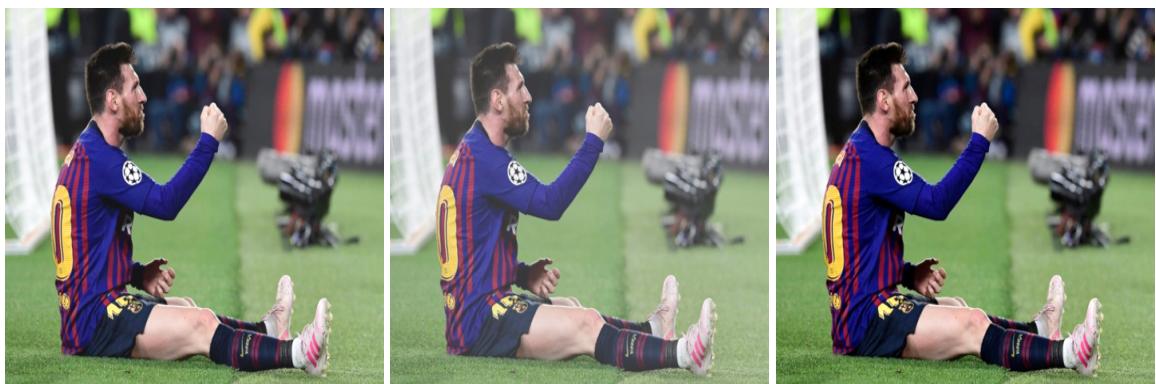
Opis realizowanych operacji

1. Policz sumy wartości każdego piksla ze stałą.
2. Jeśli przynajmniej jedna z tych sum jest większa od głębi kolorów obrazu to:
3. Wybierz największą sumę Q_{max} i D_{max} policz ze wzoru
$$D_{max}[i, j] = (Q_{max}[i, j] - \text{głębia koloru obrazu})$$
4. Policz proporcję $X = D_{max}[i, j]/\text{głębia koloru obrazu}$
i zaokrąglając wynik do najbliższej z góry liczby całkowitej.
5. Jeśli nie ma sumy większej od głębi kolorów obrazu to $X=0$.
6. Policz sumy:
$$Q_R[i, j] = P_R[i, j] - (P_R[i, j] * X) + const - (const * X)$$
$$Q_G[i, j] = P_G[i, j] - (P_G[i, j] * X) + const - (const * X)$$
$$Q_B[i, j] = P_B[i, j] - (P_B[i, j] * X) + const - (const * X)$$
7. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

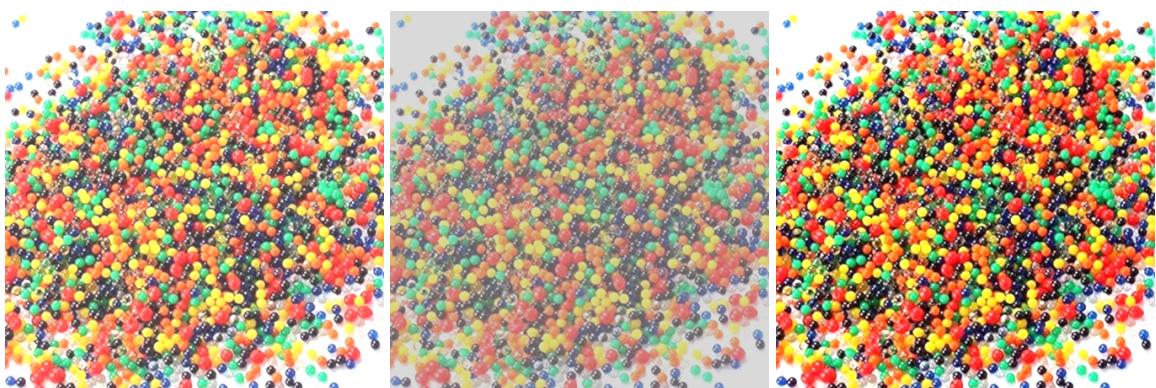
Kod do wykonania danego problemu

```
nameFileONE = 'img/RGBMessi.tif'  
readFileOne = ReadTiff(nameFileONE)  
imageOne = Image(readFileOne)  
sum_const_RGB(imageOne, 50)  
  
nameFileFOUR = 'img/RGBkulki.tif'  
readFileFour = ReadTiff(nameFileFOUR)  
imageFour = Image(readFileFour)  
sum_const_RGB(imageFour, 100)
```

Przeprowadzone testy



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



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

Kod funkcji

Listing 4.1: Sumowanie (okreslonej) stałej z obrazem RGB

```
def sum_const_RGB(image1, const=0):

    if image1.imageBitsColor[0] == 4:

        maxBitsColor = 15
        if not (0 <= const <= 15):
            raise Exception("program do obrazow RGB 4 bitowych
        mozne dodac liczbe z zakresu 0-15, a podana liczba "
                "to %d." % const)
    else:
        if image1.imageBitsColor[0] == 8:

            maxBitsColor = 255
            if not (0 <= const <= 255):
                raise Exception("program do obrazow RGB 8 bitowych
        mozne dodac liczbe z zakresu 0-255, a podana "
                "podana to %d." % const)
        else:
            raise Exception("program dodaje jedynie obrazy RGB 4, 8
        bitowe ze stala")

    Qmax = 0
    Dmax = 0
    X = 0
    fmax = 0
    fmin = 256

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempR = image1.imageData[i][j][0] + const
            tempG = image1.imageData[i][j][1] + const
            tempB = image1.imageData[i][j][2] + const

            if max([tempR, tempG, tempB]) > Qmax:
                Qmax = max([tempR, tempG, tempB])

    if Qmax > maxBitsColor:
        Dmax = Qmax - maxBitsColor
        X = round(Dmax/maxBitsColor, 2)

    if X == 1.0:
        X = 0.99

print(X)
```

```

# dodawanie
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        tempSumR = ceil((image1.imageData[i][j][0] -
                          (image1.imageData[i][j][0] * X)) + (const - (const * X)))
        tempSumG = ceil((image1.imageData[i][j][1] -
                          (image1.imageData[i][j][1] * X)) + (const - (const * X)))
        tempSumB = ceil((image1.imageData[i][j][2] -
                          (image1.imageData[i][j][2] * X)) + (const - (const * X)))

        image1.imageData[i][j][0] = tempSumR
        image1.imageData[i][j][1] = tempSumG
        image1.imageData[i][j][2] = tempSumB

        if max([tempSumR, tempSumG, tempSumB]) > fmax:
            fmax = max([tempSumR, tempSumG, tempSumB])

        if min([tempSumR, tempSumG, tempSumB]) < fmin:
            fmin = min([tempSumR, tempSumG, tempSumB])

writeTiff('add_const_RGB', image1)

# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = ceil(maxBitsColor *
                                         ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][1] = ceil(maxBitsColor *
                                         ((image1.imageData[i][j][1] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][2] = ceil(maxBitsColor *
                                         ((image1.imageData[i][j][2] - fmin) / (fmax - fmin)))
writeTiff('normalization_add_const_RGB', image1)

```

4.2. Sumowanie dwóch obrazów

Opis ćwiczenia

Algebraiczne sumowanie obrazów f i f' jest określone jedynie dla obrazów o tych samych wymiarach $M \times N$ i strukturze ich macierzy. Algorytm sumowania obrazu z obrazem polega na dodaniu do wartości piksla z pierwszego obrazu, wartości odpowiadającego piksla z drugiego obrazu. Po operacji sumowania następuje normalizacja obrazu. Dodawanie obrazów jest użyteczne w uśrednianiu obrazów, wykonywanym w celu zredukowania na nich szumu.

Opis realizowanych operacji

1. Weź dwa identycznych rozmiarów obrazy.
2. Dla wszystkich pikseli w tych obrazach wykonaj:
3. Policz sumy dla wszystkich piksli, odpowiadających składowych barw.
4. Jeśli przynajmniej jedna z tych sum jest większa od maksymalnej głębi kolorów obrazu to:
5. Wybierz największą sumę Q_{max} i D_{max} policz ze wzoru
$$D_{max}[i, j] = (Q_{max}[i, j] - \text{głębia koloru obrazu})$$
6. Policz proporcję $X = D_{max}[i, j]/\text{głębia koloru obrazu}$
i zaokrąglając wynik do najbliższej z góry liczby całkowitej.
7. Jeśli nie ma sumy większej od głębi kolorów obrazu to $X=0$.
8. Policz sumy:
$$Q_R[i, j] = P1_R[i, j] - (P1_R[i, j] * X) + P2_R[i, j] - (P2_R[i, j] * X)$$
$$Q_G[i, j] = P1_G[i, j] - (P1_G[i, j] * X) + P2_G[i, j] - (P2_G[i, j] * X)$$
$$Q_B[i, j] = P1_B[i, j] - (P1_B[i, j] * X) + P2_B[i, j] - (P2_B[i, j] * X)$$
9. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

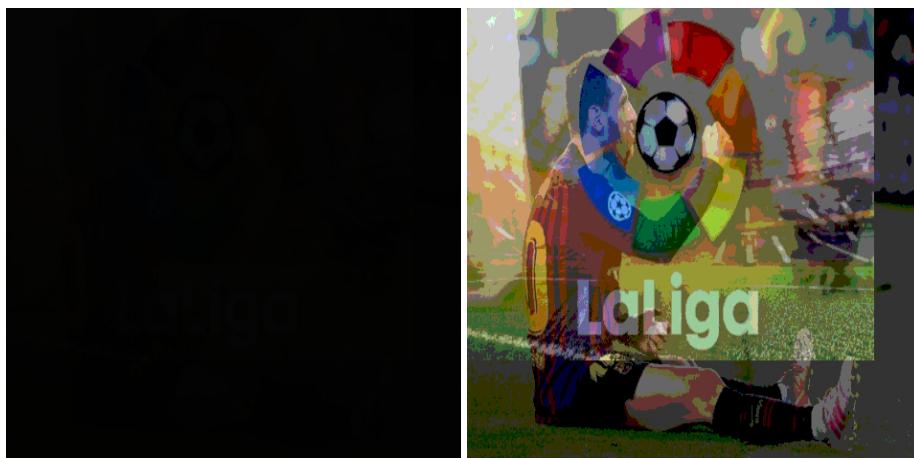
```
nameFileONE = 'img/RGBMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Geometric(imageOne, imageTwo)
unification_RGB_Resolution(imageOne, imageTwo)
sum_two_images_RGB(imageOne, imageTwo)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Geometric(imageThree, imageFour)
unification_RGB_Resolution(imageThree, imageFour)
sum_two_images_RGB(imageThree, imageFour)
```

Przeprowadzone testy



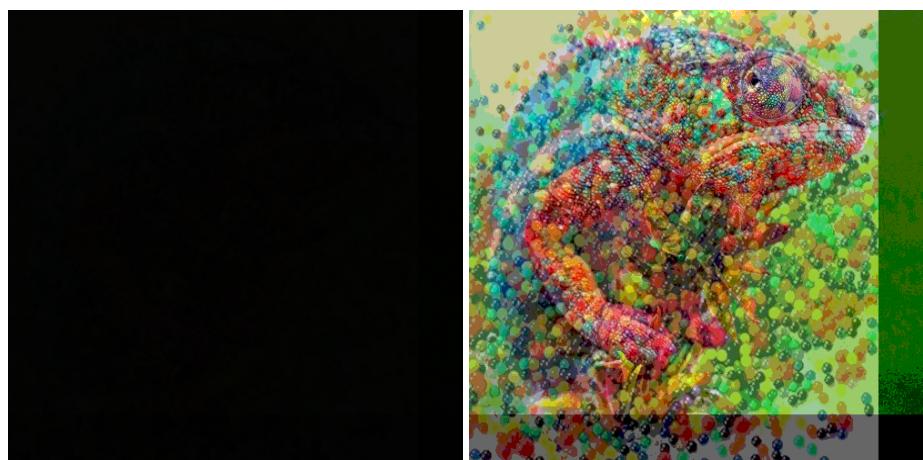
Rysunek 4.3: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.4: Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji



Rysunek 4.5: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.6: Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 4.2: Sumowanie dwóch obrazów barwowych

```
def sum_two_images_RGB(image1, image2):

    global maxBitsColor
    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        else:
            if image1.imageBitsColor[0] == 8:
                maxBitsColor = 255

        else:
            raise Exception("program dodaje jedynie obrazy RGB 4, 8 bitowe
                           oraz obrazy musza miec takie same rozmiary.")

    Qmax = 0
    Dmax = 0
    X = 0
    fmax = 0
    fmin = 256

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempR = image1.imageData[i][j][0] + image2.imageData[i][j][0]
            tempG = image1.imageData[i][j][1] + image2.imageData[i][j][1]
            tempB = image1.imageData[i][j][2] + image2.imageData[i][j][2]

            if max([tempR, tempG, tempB]) > Qmax:
                Qmax = max([tempR, tempG, tempB])

    if Qmax > maxBitsColor:
        Dmax = Qmax - maxBitsColor
        X = round(Dmax/maxBitsColor, 2)

    if X == 1.0:
        X = 0.99

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempSumR = ceil((image1.imageData[i][j][0] -
                            (image1.imageData[i][j][0] * X)) +
```

```

        (image2.imageData[i][j][0] - (image2.imageData[i][j][0] * X)))

    image1.imageData[i][j][0] = tempSumR

    tempSumG = ceil((image1.imageData[i][j][1] -
        (image1.imageData[i][j][1] * X)) +
        (image2.imageData[i][j][1] - (image2.imageData[i][j][1] * X)))
    image1.imageData[i][j][1] = tempSumG

    tempSumB = ceil((image1.imageData[i][j][2] -
        (image1.imageData[i][j][2] * X)) +
        (image2.imageData[i][j][2] - (image2.imageData[i][j][2] * X)))
    image1.imageData[i][j][2] = tempSumB

    if max([tempSumR, tempSumG, tempSumB]) > fmax:
        fmax = max([tempSumR, tempSumG, tempSumB])

    if min([tempSumR, tempSumG, tempSumB]) < fmin:
        fmin = min([tempSumR, tempSumG, tempSumB])

writeTiff('add_two_image_RGB', image1)

for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = ceil(maxBitsColor *
            ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][1] = ceil(maxBitsColor *
            ((image1.imageData[i][j][1] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][2] = ceil(maxBitsColor *
            ((image1.imageData[i][j][2] - fmin) / (fmax - fmin)))

writeTiff('normalization_add_two_image_RGB', image1)

```

4.3. Mnożenie obrazu przez zadaną liczbę

Opis ćwiczenia

Mnożenie obrazu f przez skalar α wykonuję się mnożąc każdy element obrazu $f_{i,j}$ przez skalar $f_{i,j} * \alpha$.

Opis realizowanych operacji

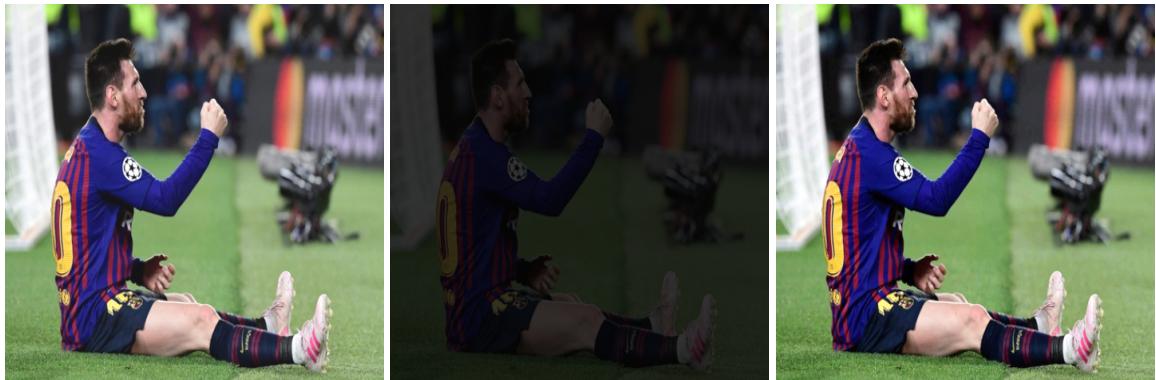
1. Dla wszystkich pikseli w obrazie wykonaj:
2. Jeśli składowa barwy piksla ma maksymalną głębię koloru to składowa wynikowa otrzymuje wartość odpowiadającą wartości stałej
3. W przeciwnym wypadku, jeśli składowa barwy pikslu ma wartość 0 to składowa wynikowa otrzymuje wartość 0
4. W przeciwnym wypadku mnóż odpowiednie sobie składowe, a wynik dziel przez maksymalną głębię koloru, zaokrąglając do najbliższej większej liczby całkowitej
5. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

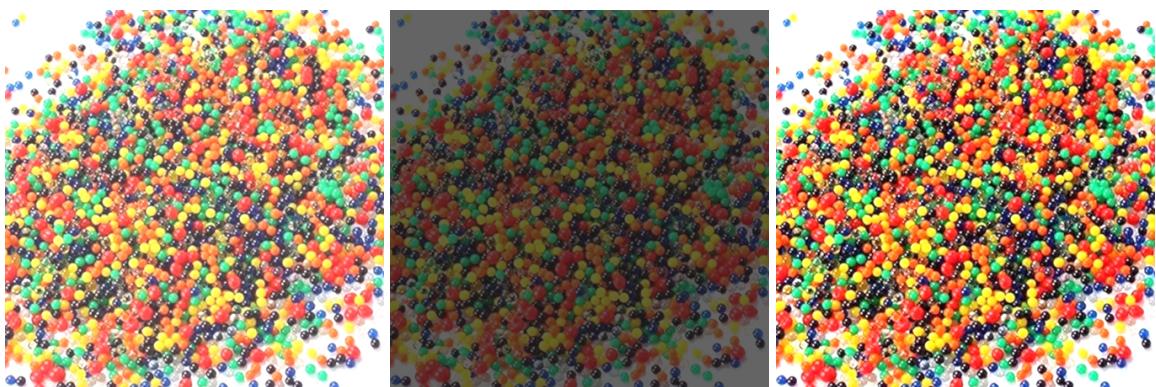
```
nameFileONE = 'img/RGMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
multiplication_const_RGB(imageOne, 50)

nameFileFOUR = 'img/RGBkulki.tif'
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
multiplication_const_RGB(imageFour, 100)
```

Przeprowadzone testy



Rysunek 4.7: (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji



Rysunek 4.8: (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji

Kod funkcji

Listing 4.3: Mnożenie obrazu przez zadaną liczbę

```
def multiplication_const_RGB(image1, const=1):

    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == 4:

        maxBitsColor = 15
        if not (0 < const <= 15):
            raise Exception("program do obrazow RGB 4 bitowych moze pomnozyc liczbe z zakresu 0-15, a podana liczba " "to %d." % const)
    else:
        if image1.imageBitsColor[0] == 8:

            maxBitsColor = 255
            if not (0 < const <= 255):
                raise Exception("program do obrazow RGB 8 bitowych moze pomnozyc liczbe z zakresu 0-255, a podana " "podana to %d." % const)
        else:
            raise Exception("program mnozy jedynie obrazy RGB 4, 8 bitowe ze stala")

    # mnozenie
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempMultR = image1.imageData[i][j][0]
            tempMultG = image1.imageData[i][j][1]
            tempMultB = image1.imageData[i][j][2]

            if tempMultR == maxBitsColor:
                tempMultR = const
            elif tempMultR == 0:
                tempMultR = 0
            else:
                tempMultR = ceil((image1.imageData[i][j][0] * const) / maxBitsColor)

            if tempMultG == maxBitsColor:
                tempMultG = const
            elif tempMultG == 0:
                tempMultG = 0
            else:
```

```

tempMultG = ceil((image1.imageData[i][j][1] * const) /
maxBitsColor)

if tempMultB == maxBitsColor:
    tempMultB = const
elif tempMultB == 0:
    tempMultB = 0
else:
    tempMultB = ceil((image1.imageData[i][j][2] * const) /
maxBitsColor)

image1.imageData[i][j][0] = tempMultR
image1.imageData[i][j][1] = tempMultG
image1.imageData[i][j][2] = tempMultB

if max([tempMultR, tempMultG, tempMultB]) > fmax:
    fmax = max([tempMultR, tempMultG, tempMultB])

if min([tempMultR, tempMultG, tempMultB]) < fmin:
    fmin = min([tempMultR, tempMultG, tempMultB])

writeTiff('multi_const_RGB', image1)

```

4.4. Mnożenie obrazu przez inny obraz

Opis ćwiczenia

Algebraiczne mnożenie obrazów $f \cdot f'$ jest określone jedynie dla obrazów o tych samych wymiarach $M \times N$ i strukturze ich macierzy i wykonuje się mnożąc każdy element obrazu P_1 przez odpowiadający piksel drugiego obrazu P_2 .

Opis realizowanych operacji

1. Weź dwa identycznych rozmiarów obrazy $P_1 \cdot P_2$
2. Dla wszystkich pikseli w obrazie wykonaj:
3. Jeśli składowa barwy piksla ma maksymalną głębię koloru to składowa wynikowa otrzymuje wartość odpowiadającą wartości składowej drugiego obrazu
4. W przeciwnym wypadku, jeśli składowa barwy pikslu ma wartość 0 to składowa wynikowa otrzymuje wartość 0
5. W przeciwnym wypadku mnóż odpowiednie sobie składowe, a wynik dziel przez maksymalną głębię koloru, zaokrąglając do najbliższej większej liczby całkowitej
6. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

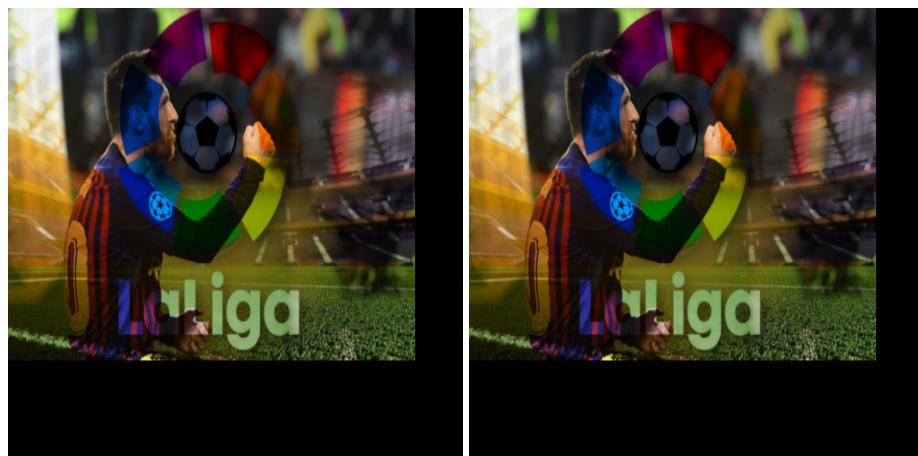
```
nameFileONE = 'img/RGMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Geometric(imageOne, imageTwo)
unification_RGB_Resolution(imageOne, imageTwo)
multiplication_two_images_RGB(imageOne, imageTwo)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Geometric(imageThree, imageFour)
unification_RGB_Resolution(imageThree, imageFour)
multiplication_two_images_RGB(imageThree, imageFour)
```

Przeprowadzone testy



Rysunek 4.9: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.10: Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji



Rysunek 4.11: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.12: Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 4.4: Mnożenie dwóch obrazów

```
def multiplication_two_images_RGB(image1, image2):

    global maxBitsColor
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

        else:
            raise Exception("program mnozy jedynie obrazy RGB 4, 8 bitowe
                             oraz obrazy musza miec takie same rozmiary.")

    # mnozenie
    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempMultR = image1.imageData[i][j][0]
            tempMultG = image1.imageData[i][j][1]
            tempMultB = image1.imageData[i][j][2]

            if tempMultR == maxBitsColor:
                tempMultR = image2.imageData[i][j][0]
            elif tempMultR == 0:
                tempMultR = 0
            else:
                tempMultR = ceil((image1.imageData[i][j][0] *
                                  image2.imageData[i][j][0]) / maxBitsColor)

            if tempMultG == maxBitsColor:
                tempMultG = image2.imageData[i][j][1]
            elif tempMultG == 0:
                tempMultG = 0
            else:
                tempMultG = ceil((image1.imageData[i][j][1] *
                                  image2.imageData[i][j][1]) / maxBitsColor)

            if tempMultB == maxBitsColor:
```

```

        tempMultB = image2.imageData[i][j][2]
elif tempMultB == 0:
    tempMultB = 0
else:
    tempMultB = ceil((image1.imageData[i][j][2] *
    image2.imageData[i][j][2]) / maxBitsColor)

image1.imageData[i][j][0] = tempMultR
image1.imageData[i][j][1] = tempMultG
image1.imageData[i][j][2] = tempMultB

if max([tempMultR, tempMultG, tempMultB]) > fmax:
    fmax = max([tempMultR, tempMultG, tempMultB])

if min([tempMultR, tempMultG, tempMultB]) < fmin:
    fmin = min([tempMultR, tempMultG, tempMultB])

writeTiff('multi_two_images_RGB', image1)

# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
            ((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][1] = round(maxBitsColor *
            ((image1.imageData[i][j][1] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][2] = round(maxBitsColor *
            ((image1.imageData[i][j][2] - fmin) / (fmax - fmin)))
writeTiff('normalization_multi_two_images_RGB', image1)

```

4.5. Mieszanie obrazów z określonym współczynnikiem

Opis ćwiczenia

Mieszanie dwóch obrazów polega na sumowaniu ich z wagami α i $(1 - \alpha)$, odpowiednio według wzoru: $f_m = f * \alpha + f' * (1 - \alpha)$, gdzie $\alpha \in [0, 1]$. Plynna zmiana parametru α w przedziale $[0, 1]$ powoduje efekt przechodzenia obrazu f' w obraz f

Opis realizowanych operacji

1. Weź dwa identycznych rozmiarów obrazy P_1 i P_2
2. Określ współczynnik mieszania α wyrażony jako liczba rzeczywista z zakresu $< 0, 1 >$;
0 reprezentuje pełną przezroczystość, 1 - nieprzezroczystości
3. Dla wszystkich pikseli w obrazach wejściowych wykonuj: $Q(i, j) = \alpha * P_1(i, j) + (1 - \alpha) * P_2(i, j)$

Kod do wykonania danego problemu

```
nameFileONE = 'img/RGBMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Geometric(imageOne, imageTwo)
unification_RGB_Resolution(imageOne, imageTwo)
mixing_images_RGB(imageOne, imageTwo, 0.5)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Geometric(imageThree, imageFour)
unification_RGB_Resolution(imageThree, imageFour)
mixing_images_RGB(imageThree, imageFour, 0.8)
```

Przeprowadzone testy



Rysunek 4.13: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.14: Obrazy wyjściowe (od lewej): obraz powstający w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.5$



Rysunek 4.15: Obrazy wejściowe (od lewej):pierwszy obraz wejściowy,drugi obraz wejściowy



Rysunek 4.16: Obrazy wyjściowe (od lewej): obraz powstający w wyniku mieszania obrazów ze współczynnikiem $\alpha = 0.8$

Kod funkcji

Listing 4.5: Mieszanie obrazów z określonym współczynnikiem

```

def mixing_images_RGB(image1, image2, scales=0.0):
    global maxBitsColor
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
        (image1.imageLength == image2.imageLength) and
        (image1.imageWidth == image2.imageWidth):

        if not (0.0 <= scales <= 1.0):
            raise Exception("program_miesza_obrazy_RGB_z_waga
zakresu_0.0-1.0, a podana liczba "
"to_%f ." % scales)

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

        else:
            raise Exception("program_miesza_jedynie_obrazy_RGB_4, 8_bitowe
oraz_obrazy_musza_miec_takie_same_rozmiary .")

        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                tempMixR = ceil(scales * image1.imageData[i][j][0] +
                    (1 - scales) * image2.imageData[i][j][0])
                tempMixG = ceil(scales * image1.imageData[i][j][1] +
                    (1 - scales) * image2.imageData[i][j][1])
                tempMixB = ceil(scales * image1.imageData[i][j][2] +
                    (1 - scales) * image2.imageData[i][j][2])

                image1.imageData[i][j][0] = tempMixR
                image1.imageData[i][j][1] = tempMixG
                image1.imageData[i][j][2] = tempMixB

                if max([tempMixR, tempMixG, tempMixB]) > fmax:
                    fmax = max([tempMixR, tempMixG, tempMixB])

                if min([tempMixR, tempMixG, tempMixB]) < fmin:
                    fmin = min([tempMixR, tempMixG, tempMixB])

```

4.6. Potęgowanie obrazu (z zadaną potęgą)

Opis ćwiczenia

Potęgowanie obrazu jest szczególnym przypadkiem operacji mnożenia obrazów.

Kod do wykonania danego problemu

```
nameFileONE = 'img/RGBMessi.tif'  
readFileOne = ReadTiff(nameFileONE)  
imageOne = Image(readFileOne)  
pow_image_RGB(imageOne, 2)  
  
nameFileFOUR = 'img/RGBkulki.tif'  
readFileFour = ReadTiff(nameFileFOUR)  
imageFour = Image(readFileFour)  
pow_image_RGB(imageFour, 3)
```

Przeprowadzone testy



Rysunek 4.17: (Od lewej):obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 2$, obraz po normalizacji



Rysunek 4.18: (Od lewej):obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 3$, obraz po normalizacji

Kod funkcji

Listing 4.6: Potęgowanie obrazu

```
def pow_image_RGB(image1, p=1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    if not (0 < p):
        raise Exception("program potęguje obraz SZARY z zadana potega uzyzakresu p>0, a podana liczba " "to %d." % p)

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPowR = image1.imageData[i][j][0]
            tempPowG = image1.imageData[i][j][1]
            tempPowB = image1.imageData[i][j][2]

            if max([tempPowR, tempPowG, tempPowB]) > fmaximage:
                fmaximage = max([tempPowR, tempPowG, tempPowB])

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPowR = image1.imageData[i][j][0]
            tempPowG = image1.imageData[i][j][1]
            tempPowB = image1.imageData[i][j][2]

            if tempPowR == maxBitsColor:
                tempPowR = maxBitsColor
            elif tempPowR == 0:
                tempPowR = 0
            else:
                tempPowR = ceil(pow(image1.imageData[i][j][0] / fmaximage, p) * maxBitsColor)

            if tempPowG == maxBitsColor:
                tempPowG = maxBitsColor
            elif tempPowG == 0:
                tempPowG = 0
```

```

else :
    tempPowG = ceil(pow(image1.imageData[i][j][1] /
fmaximage , p) * maxBitsColor)

    if tempPowB == maxBitsColor:
        tempPowB = maxBitsColor
    elif tempPowB == 0:
        tempPowB = 0
    else :
        tempPowB = ceil(pow(image1.imageData[i][j][2] /
fmaximage , p) * maxBitsColor)

image1.imageData[i][j][0] = tempPowR
image1.imageData[i][j][1] = tempPowG
image1.imageData[i][j][2] = tempPowB

if max([tempPowR , tempPowG , tempPowB]) > fmax:
    fmax = max([tempPowR , tempPowG , tempPowB])

if min([tempPowR , tempPowG , tempPowB]) < fmin:
    fmin = min([tempPowR , tempPowG , tempPowB])

writeTiff('pow_image_RGB' , image1)
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][1] = round(maxBitsColor *
((image1.imageData[i][j][1] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][2] = round(maxBitsColor *
((image1.imageData[i][j][2] - fmin) / (fmax - fmin)))

writeTiff('normalization_pow_image_RGB' , image1)

```

4.7. Dzielenie obrazu przez (zadaną) liczbę

Opis realizowanych operacji

1. Dla wszystkich piksli w tym obrazie wykonaj:
2. Policz sumę piksli ze stałą
3. Wybierz największą sumę $Q_{max} = \max(R_S; G_S; B_S)$ i policz równania:
$$Q_R[i, j] = (R_S * \text{maksymalna gęścia koloru}) / Q_{max}$$
$$Q_G[i, j] = (G_S * \text{maksymalna gęścia koloru}) / Q_{max}$$
$$Q_B[i, j] = (B_S * \text{maksymalna gęścia koloru}) / Q_{max},$$

Wynik zaokrąglaj do najbliższej górnej liczby całkowitej
4. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min}) / (f_{max} - f_{min})]$$

Kod do wykonania danego problemu

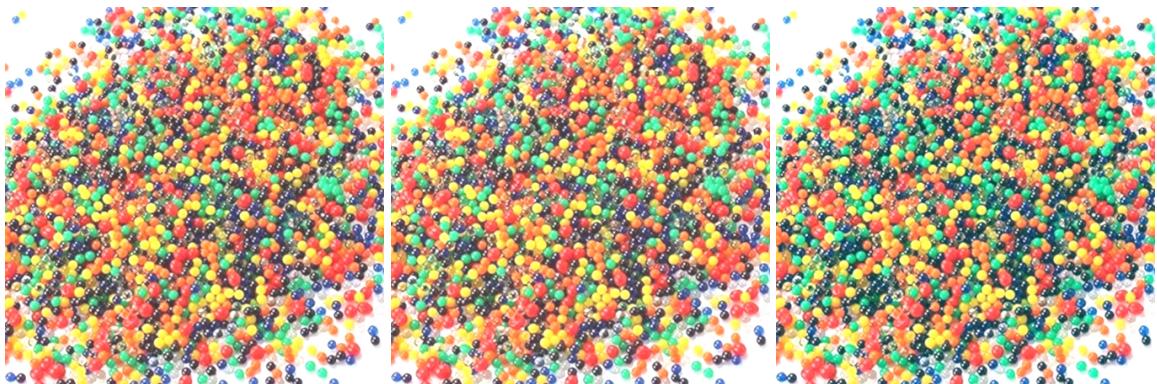
```
nameFileONE = 'img/RGBMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
division_const_RGB(imageOne, 15)

nameFileFOUR = 'img/RGBkulki.tif'
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
division_const_RGB(imageFour, 3)
```

Przeprowadzone testy



Rysunek 4.19: (Od lewej):obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji



Rysunek 4.20: (Od lewej):obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji

Kod funkcji

Listing 4.7: Dzielnie obrazu przez liczbę

```
def division_const_RGB(image1, const=1):

    Qmax = 0
    fmax = 0
    fmin = 256

    if image1.imageBitsColor[0] == 4:

        maxBitsColor = 15
        if not (0 < const <= 15):
            raise Exception("program do obrazow RGB 4 bitowych"
                            "moze dzielic liczbe z zakresu 0-15, a podana liczba"
                            "to %d." % const)
    elif image1.imageBitsColor[0] == 8:

        maxBitsColor = 255
        if not (0 < const <= 255):
            raise Exception("program do obrazow RGB 8 bitowych"
                            "moze dzielic liczbe z zakresu 0-255, a podana"
                            "podana to %d." % const)
    else:
        raise Exception("program dzieli jedynie obrazy RGB 4, 8 bitowe"
                        "ze stala")

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempDivR = image1.imageData[i][j][0] + const
            tempDivG = image1.imageData[i][j][1] + const
            tempDivB = image1.imageData[i][j][2] + const

            if Qmax < max([tempDivR, tempDivG, tempDivB]):
                Qmax = max([tempDivR, tempDivG, tempDivB])

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempDivR = image1.imageData[i][j][0] + const
            tempDivG = image1.imageData[i][j][1] + const
            tempDivB = image1.imageData[i][j][2] + const

            resultDivR = ceil((tempDivR * maxBitsColor) / Qmax)
            resultDivG = ceil((tempDivG * maxBitsColor) / Qmax)
            resultDivB = ceil((tempDivB * maxBitsColor) / Qmax)

            image1.imageData[i][j][0] = resultDivR
```

```

image1.imageData[ i ][ j ][ 1 ] = resultDivG
image1.imageData[ i ][ j ][ 2 ] = resultDivB

if max([ resultDivR , resultDivG , resultDivB ]) > fmax:
    fmax = max([ resultDivR , resultDivG , resultDivB ])

if min([ resultDivR , resultDivG , resultDivB ]) < fmin:
    fmin = min([ resultDivR , resultDivG , resultDivB ])

writeTiff( 'div_image_const' , image1 )
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[ i ][ j ][ 0 ] = round( maxBitsColor *
            ((image1.imageData[ i ][ j ][ 0 ] - fmin) / (fmax - fmin)))

writeTiff( 'normalization_div_image_const' , image1 )

```

4.8. Dzielenie obrazu przez inny obraz

Opis realizowanych operacji

1. Weź dwa identycznych rozmiarów obrazy P_1 i P_2
2. Dla wszystkich pikseli w obrazie wykonaj:
3. Policz sumy pikseli ze stałą
4. Wybierz największą sumę $Q_{max} = \max(R_s, G_s, B_s)$ i policz równanie:
$$Q_R[i, j] = (R_s - \text{maksymalna gęścia koloru})/Q_{max}$$
$$Q_G[i, j] = (G_s - \text{maksymalna gęścia koloru})/Q_{max}$$
$$Q_B[i, j] = (B_s - \text{maksymalna gęścia koloru})/Q_{max},$$

Wynik zaokrąglaj do najbliższej górnej liczby całkowitej.
5. Znormalizowanie obrazu wzorem:
$$f_{norm} = Z_{rep}[(f - f_{min})/(f_{max} - f_{min})]$$

Kod do wykonania danego problemu

```
nameFileONE = 'img/RGMessi.tif'
nameFileTWO = 'img/RGBLL.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
readFileTwo = ReadTiff(nameFileTWO)
imageTwo = Image(readFileTwo)
unification_RGB_Geometric(imageOne, imageTwo)
unification_RGB_Resolution(imageOne, imageTwo)
division_two_images_RGB(imageOne, imageTwo)

nameFileTHREE = 'img/RGBkamel.tif'
nameFileFOUR = 'img/RGBkulki.tif'
readFileThree = ReadTiff(nameFileTHREE)
imageThree = Image(readFileThree)
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
unification_RGB_Geometric(imageThree, imageFour)
unification_RGB_Resolution(imageThree, imageFour)
division_two_images_RGB(imageThree, imageFour)
```

Przeprowadzone testy



Rysunek 4.21: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.22: Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji



Rysunek 4.23: Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy



Rysunek 4.24: Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji

Kod funkcji

Listing 4.8: Dzielenie dwóch obrazów

```
def division_two_images_RGB(image1, image2):

    global maxBitsColor
    fmax = 0
    fmin = 256
    Qmax = 0

    if image1.imageBitsColor[0] == image2.imageBitsColor[0] and
       (image1.imageLength == image2.imageLength) and
       (image1.imageWidth == image2.imageWidth):

        if image1.imageBitsColor[0] == 4:
            maxBitsColor = 15

        elif image1.imageBitsColor[0] == 8:
            maxBitsColor = 255

        else:
            raise Exception("program dzieli jedynie obrazy RGB 4, 8 bitowe
                             oraz obrazy musza miec takie same rozmiary .")

        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                tempDivR = image1.imageData[i][j][0] + image2.imageData[i][j][0]
                tempDivG = image1.imageData[i][j][1] + image2.imageData[i][j][1]
                tempDivB = image1.imageData[i][j][2] + image2.imageData[i][j][2]

                if Qmax < max([tempDivR, tempDivG, tempDivB]):
                    Qmax = max([tempDivR, tempDivG, tempDivB])

        for i in range(image1.imageLength):
            for j in range(image1.imageWidth):
                tempDivR = image1.imageData[i][j][0] + image2.imageData[i][j][0]
                tempDivG = image1.imageData[i][j][1] + image2.imageData[i][j][1]
                tempDivB = image1.imageData[i][j][2] + image2.imageData[i][j][2]

                resultDivR = ceil((tempDivR * maxBitsColor) / Qmax)
                resultDivG = ceil((tempDivG * maxBitsColor) / Qmax)
                resultDivB = ceil((tempDivB * maxBitsColor) / Qmax)

                image1.imageData[i][j][0] = resultDivR
                image1.imageData[i][j][1] = resultDivG
                image1.imageData[i][j][2] = resultDivB
```

```

if max([ resultDivR , resultDivG , resultDivB ]) > fmax:
    fmax = max([ resultDivR , resultDivG , resultDivB ])

if min([ resultDivR , resultDivG , resultDivB ]) < fmin:
    fmin = min([ resultDivR , resultDivG , resultDivB ])

writeTiff( 'div_two_images_RGB' , image1 )
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[ i ][ j ][ 0 ] = round(maxBitsColor *
            ((image1.imageData[ i ][ j ][ 0 ] - fmin) / (fmax - fmin)))
        image1.imageData[ i ][ j ][ 1 ] = round(maxBitsColor *
            ((image1.imageData[ i ][ j ][ 1 ] - fmin) / (fmax - fmin)))
        image1.imageData[ i ][ j ][ 2 ] = round(maxBitsColor *
            ((image1.imageData[ i ][ j ][ 2 ] - fmin) / (fmax - fmin)))

writeTiff( 'normalization_div_two_images_RGB' , image1 )

```

4.9. Pierwiastkowanie obrazu

Opis ćwiczenia

Pierwiastkowanie obrazu jest szczególnym przypadkiem operacji potęgowania obrazów, gdzie wykładnikiem jest ułamek.

Kod do wykonania danego problemu

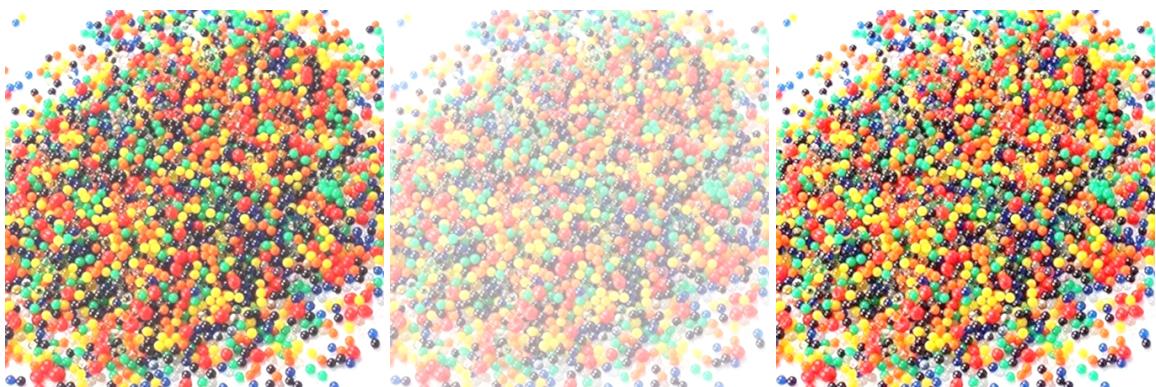
```
nameFileONE = 'img/RGBMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
sqrt_image_RGB(imageOne, 2)

nameFileFOUR = 'img/RGBkulki.tif'
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
sqrt_image_RGB(imageFour, 3)
```

Przeprowadzone testy



Rysunek 4.25: (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym, obraz po normalizacji



Rysunek 4.26: (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego, obraz po normalizacji

Kod funkcji

Listing 4.9: Pierwiastkowanie obrazu

```
def sqrt_image_RGB(image1, deg=1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    p = 1/deg

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPowR = image1.imageData[i][j][0]
            tempPowG = image1.imageData[i][j][1]
            tempPowB = image1.imageData[i][j][2]

            if max([tempPowR, tempPowG, tempPowB]) > fmaximage:
                fmaximage = max([tempPowR, tempPowG, tempPowB])

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempPowR = image1.imageData[i][j][0]
            tempPowG = image1.imageData[i][j][1]
            tempPowB = image1.imageData[i][j][2]

            if tempPowR == maxBitsColor:
                tempPowR = maxBitsColor
            elif tempPowR == 0:
                tempPowR = 0
            else:
                tempPowR =
                    ceil(pow(image1.imageData[i][j][0] / fmaximage, p) *
                         maxBitsColor)

            if tempPowG == maxBitsColor:
                tempPowG = maxBitsColor
            elif tempPowG == 0:
                tempPowG = 0
            else:
                tempPowG =
```

```

ceil(pow(image1.imageData[i][j][1] / fmaximage, p) *
maxBitsColor)

if tempPowB == maxBitsColor:
    tempPowB = maxBitsColor
elif tempPowB == 0:
    tempPowB = 0
else:
    tempPowB =
    ceil(pow(image1.imageData[i][j][2] / fmaximage, p) *
maxBitsColor)

image1.imageData[i][j][0] = tempPowR
image1.imageData[i][j][1] = tempPowG
image1.imageData[i][j][2] = tempPowB

if max([tempPowR, tempPowG, tempPowB]) > fmax:
    fmax = max([tempPowR, tempPowG, tempPowB])

if min([tempPowR, tempPowG, tempPowB]) < fmin:
    fmin = min([tempPowR, tempPowG, tempPowB])

writeTiff('root_image_RGB', image1)
# normalizacja
for i in range(image1.imageLength):
    for j in range(image1.imageWidth):
        image1.imageData[i][j][0] = round(maxBitsColor *
((image1.imageData[i][j][0] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][1] = round(maxBitsColor *
((image1.imageData[i][j][1] - fmin) / (fmax - fmin)))
        image1.imageData[i][j][2] = round(maxBitsColor *
((image1.imageData[i][j][2] - fmin) / (fmax - fmin)))

writeTiff('normalization_root_image_RGB', image1)

```

4.10. Logarytmowanie obrazu

Opis ćwiczenia

Przesunięcie funkcji obrazowej f do góry o 1 przed jej logarytmowaniem wynika z nieokreśloności logarytmu w zerze. Logarytmowanie obrazu powoduje rozjaśnienie i zróżnicowanie najciemniejszych obszarów obrazu.

Kod do wykonania danego problemu

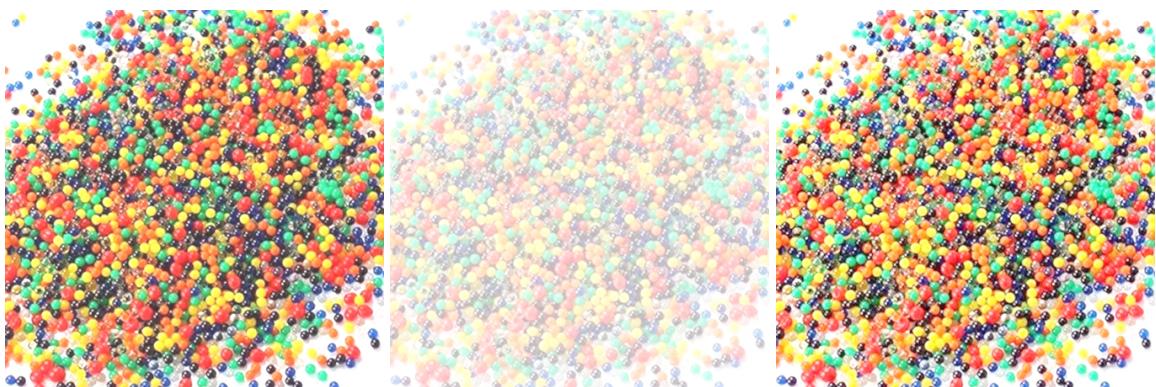
```
nameFileONE = 'img/RGBMessi.tif'
readFileOne = ReadTiff(nameFileONE)
imageOne = Image(readFileOne)
log_image_RGB(imageOne)

nameFileFOUR = 'img/RGBkulki.tif'
readFileFour = ReadTiff(nameFileFOUR)
imageFour = Image(readFileFour)
log_image_RGB(imageFour)
```

Przeprowadzone testy



Rysunek 4.27: (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji



Rysunek 4.28: (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji

Kod funkcji

Listing 4.10: Logarytmowanie obrazu

```
def log_image_RGB(image1):

    global maxBitsColor
    fmax = 0
    fmin = 256
    fmaximage = 0

    if image1.imageBitsColor[0] == 4:
        maxBitsColor = 15
    elif image1.imageBitsColor[0] == 8:
        maxBitsColor = 255

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempLogR = image1.imageData[i][j][0]
            tempLogG = image1.imageData[i][j][1]
            tempLogB = image1.imageData[i][j][2]

            if fmaximage < max([tempLogR, tempLogG, tempLogB]):
                fmaximage = max([tempLogR, tempLogG, tempLogB])

    for i in range(image1.imageLength):
        for j in range(image1.imageWidth):
            tempLogR = image1.imageData[i][j][0]
            tempLogG = image1.imageData[i][j][1]
            tempLogB = image1.imageData[i][j][2]

            if tempLogR == 0:
                tempLogR = 0
            else:
                tempLogR =
                    ceil((log(1 + tempLogR) / log(1 + fmaximage)) * maxBitsColor)

            if tempLogG == 0:
                tempLogG = 0
            else:
                tempLogG =
                    ceil((log(1 + tempLogG) / log(1 + fmaximage)) * maxBitsColor)

            if tempLogB == 0:
                tempLogB = 0
            else:
                tempLogB =
                    ceil((log(1 + tempLogB) / log(1 + fmaximage)) * maxBitsColor)
```

```

image1.imageData[ i ][ j ][ 0 ] = tempLogR
image1.imageData[ i ][ j ][ 1 ] = tempLogG
image1.imageData[ i ][ j ][ 2 ] = tempLogB

if max([ tempLogR , tempLogG , tempLogB ]) > fmax :
    fmax = max([ tempLogR , tempLogG , tempLogB ])

if min([ tempLogR , tempLogG , tempLogB ]) < fmin :
    fmin = min([ tempLogR , tempLogG , tempLogB ])

writeTiff( 'log_image_RGB' , image1 )
# normalizacja
for i in range( image1.imageLength ):
    for j in range( image1.imageWidth ):
        image1.imageData[ i ][ j ][ 0 ] = round( maxBitsColor *
            ((image1.imageData[ i ][ j ][ 0 ] - fmin) / (fmax - fmin)))
        image1.imageData[ i ][ j ][ 1 ] = round( maxBitsColor *
            ((image1.imageData[ i ][ j ][ 1 ] - fmin) / (fmax - fmin)))
        image1.imageData[ i ][ j ][ 2 ] = round( maxBitsColor *
            ((image1.imageData[ i ][ j ][ 2 ] - fmin) / (fmax - fmin)))

writeTiff( 'normalization_log_image_RGB' , image1 )

```

Spis rysunków

2.1. Obrazy wejściowe (od lewej): obraz 1 (320x361), obraz 2 (400x225)	5
2.2. Obrazy wyjściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)	5
2.3. Obrazy wejściowe (od lewej): obraz 1 (605x512), obraz 2 (332x250)	6
2.4. Obrazy wyjściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)	6
2.5. Obrazy wejściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)	10
2.6. Obrazy wyjściowe (od lewej): obraz 1 (400x361), obraz 2 (400x361)	10
2.7. Obrazy wejściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)	11
2.8. Obrazy wyjściowe (od lewej): obraz 1 (605x512), obraz 2 (605x512)	11
2.9. Obrazy wejściowe (od lewej): obraz 1 (475x355), obraz 2 (425x275)	16
2.10. Obrazy wyjściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)	16
2.11. Obrazy wejściowe (od lewej): obraz 1 (500x400), obraz 2 (450x450)	17
2.12. Obrazy wyjściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)	17
2.13. Obrazy wejściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)	21
2.14. Obrazy wyjściowe (od lewej): obraz 1 (475x355), obraz 2 (475x355)	21
2.15. Obrazy wejściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)	22
2.16. Obrazy wyjściowe (od lewej): obraz 1 (500x450), obraz 2 (500x450)	22
3.1. (Od lewej): obraz wejściowy szary, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	25
3.2. (Od lewej): obraz wejściowy szary, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	25
3.3. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy . . .	29
3.4. Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji	29
3.5. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy . . .	30
3.6. Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji	30
3.7. (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji	34
3.8. (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji	34
3.9. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy . . .	38
3.10. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji	38
3.11. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy . . .	39
3.12. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji	39
3.13. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy . . .	43

3.14. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mieszania obrazów ze wspólnym czynnikiem $\alpha = 0.5$	43
3.15. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	44
3.16. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mieszania obrazów ze wspólnym czynnikiem $\alpha = 0.8$	44
3.17. (Od lewej): obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 2$, obraz po normalizacji	46
3.18. (Od lewej): obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 3$, obraz po normalizacji	47
3.19. (Od lewej): obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji	49
3.20. (Od lewej): obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji	50
3.21. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	53
3.22. Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji	53
3.23. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	54
3.24. Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji	54
3.25. (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym, obraz po normalizacji	58
3.26. (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego, obraz po normalizacji	58
3.27. (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	62
3.28. (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	62
 4.1. (Od lewej): obraz wejściowy RGB, obraz po sumowaniu ze stałą = 50, obraz po normalizacji	65
4.2. (Od lewej): obraz wejściowy RGB, obraz po sumowaniu ze stałą = 100, obraz po normalizacji	65
4.3. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	70
4.4. Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji	70
4.5. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	71
4.6. Obrazy wyjściowe (od lewej): obraz powstały w wyniku sumowania dwóch obrazów, obraz wynikowy po normalizacji	71
4.7. (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=50, obraz po normalizacji	75
4.8. (Od lewej): obraz wejściowy, obraz po przemnożeniu przez liczbę=100, obraz po normalizacji	75
4.9. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	79
4.10. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji	79
4.11. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	80
4.12. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mnożenia dwóch obrazów, obraz wynikowy po normalizacji	80
4.13. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	84

4.14. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mieszania obrazów ze wspólnym czynnikiem $\alpha = 0.5$	84
4.15. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	85
4.16. Obrazy wyjściowe (od lewej): obraz powstały w wyniku mieszania obrazów ze wspólnym czynnikiem $\alpha = 0.8$	85
4.17. (Od lewej): obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 2$, obraz po normalizacji	88
4.18. (Od lewej): obraz wejściowy, obraz po podniesieniu do potęgi $\alpha = 3$, obraz po normalizacji	88
4.19. (Od lewej): obraz wejściowy, obraz po podzieleniu przez liczbę=15, obraz po normalizacji	92
4.20. (Od lewej): obraz wejściowy, obraz po podzieleniu przez liczbę=3, obraz po normalizacji	92
4.21. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	96
4.22. Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji	96
4.23. Obrazy wejściowe (od lewej): pierwszy obraz wejściowy, drugi obraz wejściowy	97
4.24. Obrazy wyjściowe (od lewej): obraz powstały w wyniku podzielenia obrazów, obraz wynikowy po normalizacji	97
4.25. (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem kwadratowym, obraz po normalizacji	101
4.26. (Od lewej): obraz wejściowy, obraz po spierwiastkowaniu pierwiastkiem stopnia trzeciego, obraz po normalizacji	101
4.27. (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	104
4.28. (Od lewej): obraz wejściowy, obraz po logarytmowaniu logarytmem naturalnym, obraz po normalizacji	104

Bibliografia

- [1] Wojciech S. Mokrzycki, *Wprowadzenie do przetwarzania informacji wizualnej Tom II*, Akademicka Oficyna Wydawnicza EXIT, 2012.
- [2] Oficjalna strona do dokumentacji TIFF w wersji 6.0, June 3, 1992
<https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf>