1. Podstawowe przetworzenia obrazów

Operacje arytmetyczne polegają na wyznaczeniu nowych wartości pikseli bez zmiany ich położenia w przestrzeni. Zazwyczaj operacje arytmetyczne punktowe wykonywane są na pojedynczym obrazie (np. poprzez zmianę jasności, rozciągnięcie histogramu, binaryzację, itp.) lub między kilkoma obrazami (np. suma, różnica, mnożenie, itp.). W operacjach punktowych nowa wartość intensywności obliczana jest na podstawie jej poprzedniej wartości w oparciu o równanie:

$$I(x_2, y_2) = f(I(x_1, y_1)) \quad (1)$$

gdzie: f - oznacza operację arytmetyczną, $I(x_2, y_2)$ - intensywność piksela po przetworzeniu, $I(x_1, y_1)$ - intensywność piksela przed operacjami arytmetycznymi.

1.1. Analiza i przekształcanie histogramów

Jednym z podstawowych zadań przetwarzania obrazów jest ich poprawa poprzez analizę i przetwarzanie histogramów. Przetwarzanie (modelowanie) histogramu polega na statystycznej analizie obrazu. W celu wyświetlenia histogramu przy wykorzystaniu biblioteki OpenCV oraz Numpy wykorzystywane są następujące funkcje:

```
#Plotting Histograms Using Matplotlib (for one channel) in OpenCV
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('moje_zdjecie.png')
histogram = cv2.calcHist([img], # Obraz wywoływany w twardych nawiasach [] - format: uint8 lub float32
                         [0], #Kanał, dla którego wyznaczany jest histogram, 0-blue, 1-green, 2-red
                         None, #Maska, gdy wyznaczony ma być histogram dla fragmentu
                         [256], #Wielkość histogramu
                         [0,256]) #Zakres wartości
plt.plot(histogram, color='b')
plt.xlim([0,256])
plt.show()
```

(2)

```
#Plotting Histograms Using Matplotlib (for all channels) in OpenCV
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('moje_zdjecie.png')
for i, col in enumerate(['b', 'g', 'r']):
    histogram = cv2.calcHist([img], [i], None, [256], [0, 256])
    plt.plot(histogram, color = col)
    plt.xlim([0, 256])
plt.show()
```

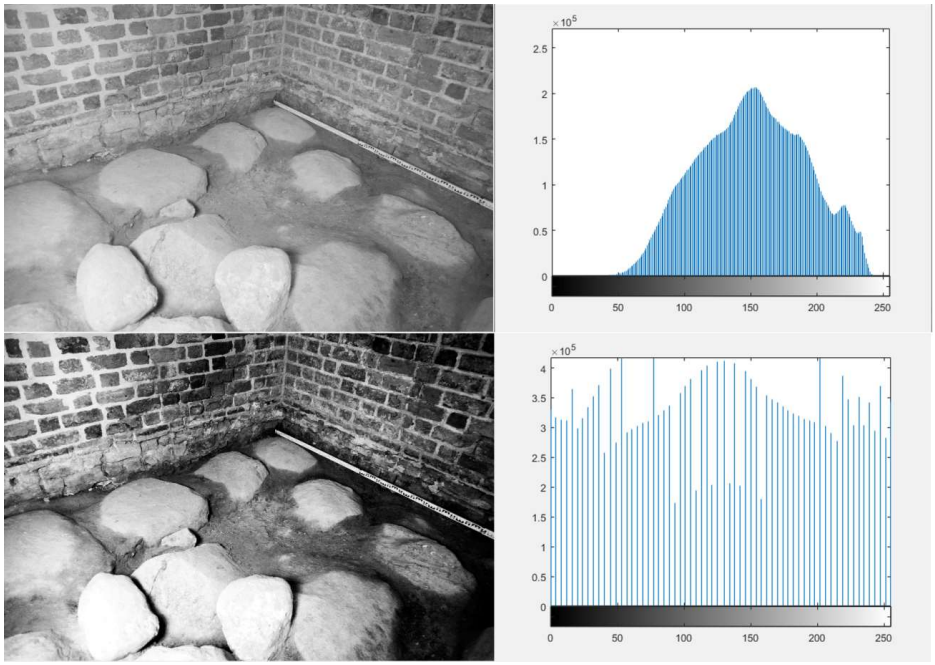
```
#Plotting Histograms Using Matplotlib in NumPy
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('moje_zdjecie.png',0) # 0 – greyscale image
plt.hist(img.ravel(),256,[0,256]);
plt.xlim([0,256])
plt.show()
#histogram is basically the same as one calculated using OpenCV. But bins will have 257 elements, because Numpy calculates bins as 0-0.99, 1-1.99, 2-2.99 etc. So final range would be 255-255.99. To represent that, they also add 256 at end of bins. But we don't need that 256. Upto 255 is sufficient.
```

(3)

Zalecane jest wykorzystanie funkcji OpenCV do wyznaczenia histogramu, gdyż jest ona około **40-krotnie** szybsza.

Zastosowanie korekcji histogramu umożliwia poprawę jakości obrazu np. gdy jego fragmenty są niedoświetlone lub prześwietlone. Złe warunki oświetleniowe wpływają na niepełne wykorzystanie wszystkich poziomów stopnia szarości z przedziału 0 - 255. Korekcja histogramu polega na takim przekształceniu jego elementów, aby wynikowy histogram był płaski i równomiernie wypełniał cały zakres jasności.



Rysunek 1 Przykład normalizacji obrazu

Zakłada się, że maksymalna wartość stopnia szarości jest mniejsza od 255, a minimalna większa od 0. Oznacza to, że najjaśniejsze punkty obrazu nie są białe, a najciemniejsze czarne. Normalizacja (rozciąganie) obrazu (Rys. 1) polega na przekształceniu wartości stopnia szarości każdego piksela funkcją liniową za pomocą wzoru:

$$g_{n,m} = 255 * (\frac{f_{n,m} - f_{min}}{f_{max} - f_{min}}) \quad (4)$$

gdzie: f_{max} - maksymalna wartość intensywności obrazu wejściowego, f_{min} - minimalna wartość intensywności obrazu wejściowego, $f_{m,n}$ - intensywność piksela wejściowego, $g_{m,n}$ - wartość piksela po normalizacji.

Wyrównanie histogramu obrazu ma na celu poprawę globalnego kontrastu , szczególnie w przypadku gdy wartości stopni szarości są reprezentowane przez wartości z niewielkiego obszaru. Dzięki wyrównaniu histogramu czyli jego „rozciągnięciu" na większy zakres możliwe jest zwiększenie kontrastu na niskich poziomach. W celu wyznaczenia nowych wartości pikseli wykorzystywana jest skumulowana funkcja gęstości zaimplementowana w funkcji 5:

```
#Histogram Equalization
import cv2
import numpy as np

img = cv2.imread('moje_zdjecie.png')
img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #or read image in greyscale using 0
equ = cv2.equalizeHist(img)

# stacking images side-by-side
res = np.hstack((img, equ))
# show image input vs output
```

(5)

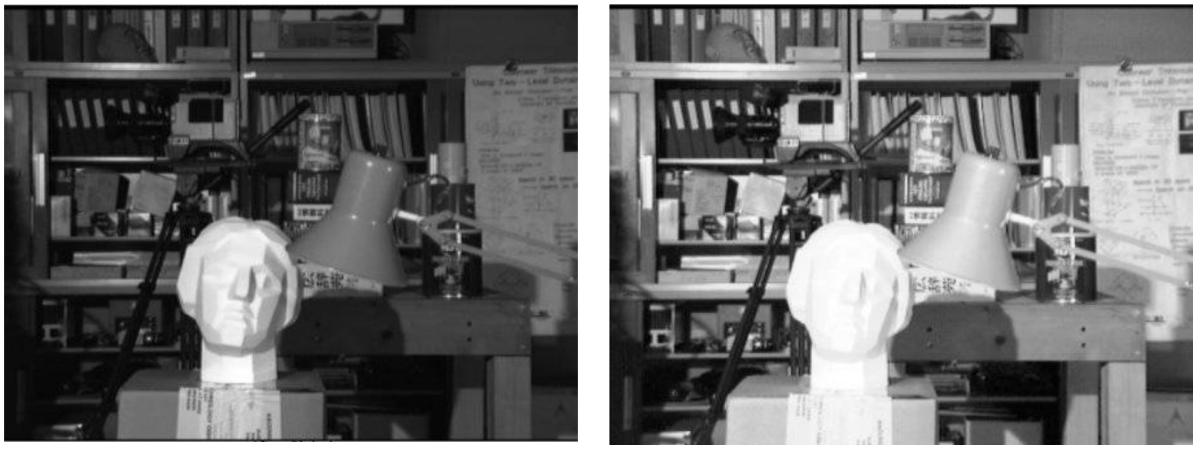
```
cv2.imshow('img', res)

# or show image in two separate windows
cv2.imshow('Oryginalne zdjecie', img)
cv2.imshow('Zdjecie po wyrownaniu histogramu', equ)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
   cv2.waitKey(1)
```

Wyrównanie histogramu jest dobre, gdy histogram obrazu jest ograniczony do określonego regionu. Nie będzie działać dobrze w miejscach, gdzie występują duże różnice w intensywności, gdzie histogram obejmuje duży obszar, tzn. występują zarówno jasne, jak i ciemne piksele.

Chociaż globalne wyrównywanie histogramu może być użyteczne, w przypadku niektórych obrazów korzystniejsze może być zastosować różne rodzaje wyrównania w różnych regionach. Na przykład, rysunek 2 przedstawia obraz oryginalny i jego wynik po globalnym wyrównaniu histogramu.



(**a**) (**b**)

Rysunek 2 Przykład (a) zdjęcia oryginalnego, (b) po globalnym wyrównaniu histogramu

Z przedstawionego przykładu jednoznacznie wynika, że kontrast tła poprawił się po wyrównaniu histogramu. Jednak analizując fragment głowy, można zauważyć, że wykorzystanie globalnego wyrównania histogramu przyczyniło się do utraty większości informacji z powodu nadmiernej jasności. Z tego też powodu należy rozważyć podział obrazu na mniejsze bloki o rozmiarze M×M pikseli i wykonanie osobne wyrównywanie histogramu w każdym podbloku. W celu rozwiązania tego problemu, stosuje się adaptacyjne wyrównywanie histogramów. Obraz jest dzielony na małe bloki zwane "kafelkami" (domyślny rozmiar 8x8). Następnie, w każdym z bloków wyznaczana jest lokalna funkcja poprawy kontrastu, która definiuje nową wartość barwy dla centralnego piksela wewnątrz bloku. Tak więc na małym obszarze, histogram będzie ograniczony do małego obszaru (chyba, że jest tam szum). Jeśli jest tam szum, zostanie on wzmocniony. Aby tego uniknąć, stosowane jest ograniczenie kontrastu. Jeśli któryś z przedziałów histogramu jest powyżej określonej granicy kontrastu (domyślnie 40 w OpenCV), piksele te są przycinane i rozdzielane równomiernie do innych przedziałów przed zastosowaniem wyrównania histogramu. W CLAHE wzmocnienie kontrastu w sąsiedztwie danej wartości piksela jest dane przez nachylenie funkcji transformacji. Jest ono proporcjonalne do nachylenia funkcji skumulowanego rozkładu gęstości funkcji prawdopodobieństwa (CDF), a zatem do wartości histogramu przy tej wartości piksela. CLAHE ogranicza wzmocnienie poprzez obcięcie histogramu przy predefiniowanej wartości przed obliczeniem CDF. Ogranicza to nachylenie CDF, a tym samym funkcji transformacji. Zazwyczaj przyjmuje się te wartości z przedziału 3 a 4. Po wyrównaniu, aby usunąć artefakty na granicach kafelków, stosowana jest interpolacja bilinearna:

```
import cv2
from matplotlib import pyplot as plt

img = cv2.imread('moje_zdjecie.png',0)
claheAlgorytm = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
adaptacyjneWyrownanieHistogramow = claheAlgorytm.apply(img)

# stacking images side-by-side
res = np.hstack((img, adaptacyjneWyrownanieHistogramow))
# show image input vs output
cv2.imshow('img', res)

# or show image in two separate windows
cv2.imshow('Oryginalne zdjecie', img)
cv2.imshow('Zdjecie po wyrownaniu histogramu', adaptacyjneWyrownanieHistogramow)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)
```

(6)

1.2. Poprawa jakości obrazu

Poprawa jakości obrazu polega na przetworzeniu obrazu poprzez zmianę jasności obrazu J i kontrastu K:

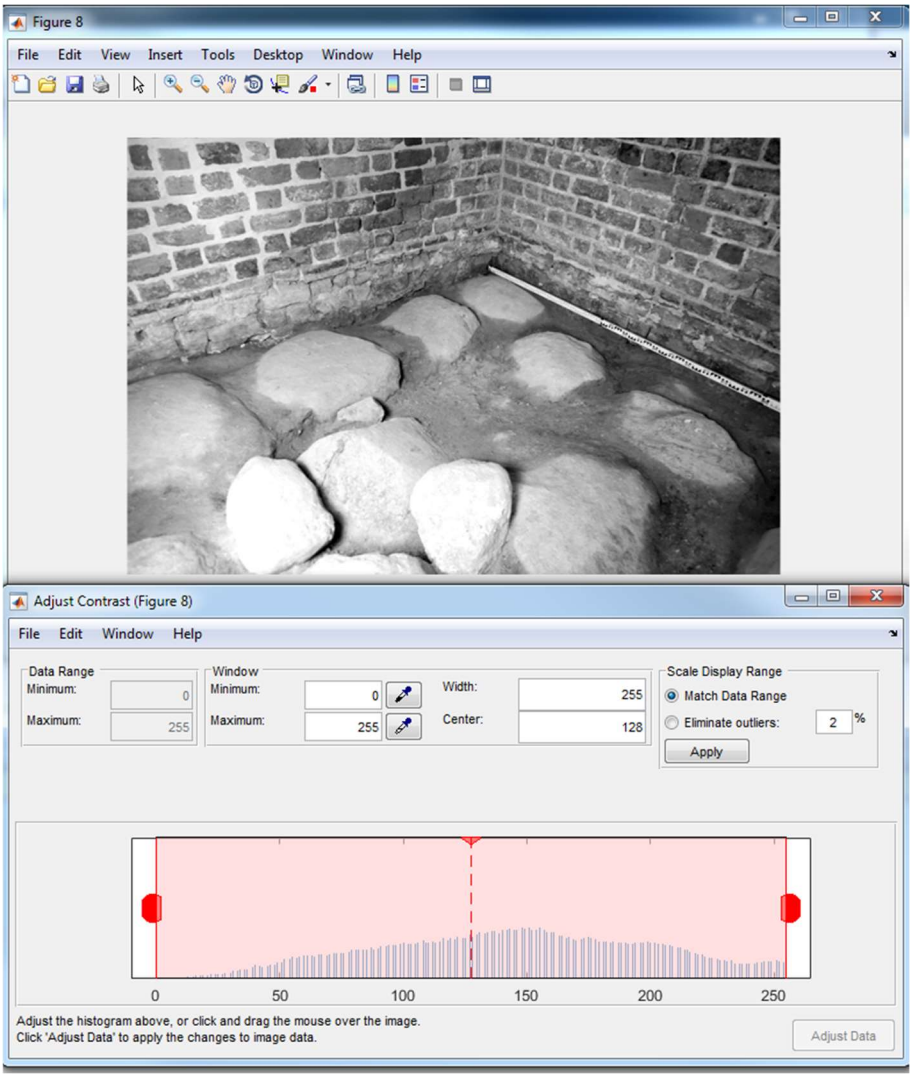
$$I(x_2, y_2) = J + K * I(x_1, y_1)$$

gdzie: $I(x_2, y_2)$ - intensywność piksela po przetworzeniu, $I(x_1, y_1)$ - intensywność piksela przed operacjami arytmetycznymi, J - współczynnik poprawy jasności obrazu, K - współczynnik poprawy kontrastu.

Jedną z metod poprawy jakości obrazów jest ograniczenie jasności (ograniczenie liczby barw na obrazie). Zazwyczaj wykonuje się poprzez dobór odpowiedniego progu, powyżej lub poniżej którego piksele są lub nie są przetwarzane (Rys. 8).

Rysunek 3 Przykład ograniczania jasności przy założeniu progu 150: (A) oryginalnego zdjęcia (stopnie szarości 15, 248) (B) gdy zmieniony został stopień szarości dla pikseli mniejszych od przyjętego progu (C) gdy zmieniony został stopień szarości dla pikseli większych od przyjętego progu.

Kolejnymi metodami poprawy jakości obrazów jest dodawanie lub odejmowanie stałej wartości, która wpływa na rozjaśnienie lub przyciemnianie obrazu, bądź poprawa kontrastu (Rys. 9)



Rysunek 4 Przykład poprawy kontrastu

Przykładowe funkcje zmiany jakości obrazu przedstawione zostały w (7):

```
#image saturation (brighter images)
import cv2

img = cv2.imread('moje_zdjecie.png')

#Create image with the same size as original and multiply as 50 - add 50 to all pixels.
scalar = np.ones(img.shape, dtype = 'uint8')*50
new_image = cv2.add(img, scalar)
cv2.imshow('Add 50 to image', new_image)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)
```

(7)

```
#image saturation (darker images)
import cv2

img = cv2.imread('moje_zdjecie.png')

#Create image with the same size as original and multiply as50 - add 50 to all pixels.
scalar = np.ones(img.shape, dtype = 'uint8')*50
new_image = cv2. subtract (img, scalar)

cv2.imshow('Add 50 to image', new_image)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
```

```
for i in range(2):
    cv2.waitKey(1)

#changing the contrast and brightness of the image using cv2.convertScaleAbs() method
import cv2

img = cv2.imread('moje_zdjecie.png')

# define the alpha and beta
alpha = 1.5 # Contrast control - to lower the contrast, use 0 < alpha < 1. And for higher contrast use alpha > 1.
beta = 10 # Brightness control – for example a good range for brightness value is [-127, 127]

# call convertScaleAbs function
adjusted = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

cv2.imshow('adjusted', adjusted)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)

#changing the contrast and brightness of the image using cv2.addWeighted() method
import cv2

img = cv2.imread('moje_zdjecie.png')

# define the contrast and brightness value
contrast = 5 # Contrast control ( 0 to 127)
brightness = 2 # Brightness control (0-100)

# call addWeighted function. use beta = 0 to effectively only operate on one image
out = cv2.addWeighted(img, contrast, img, 0, brightness)

cv2.imshow('adjusted', out)

cv2.startWindowThread()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)
```

1.3. Binaryzacja

Binaryzacja polega na zmianie obrazów RGB lub w stopniach szarości na obrazy binarne (zero-jedynkowe). To przekształcenie jest wykorzystywane w wielu zagadnieniach związanych m.in.:

- określeniem liczebności elementów na obrazie,
- pomiarem pola powierzchni, obwodu czy długości linii,
- automatyczną wektoryzacją,
- analizą i modyfikacją kształtu obiektów,
- klasyfikacją lub w detekcji punktów charakterystycznych, wykorzystywanych w procesie orientacji zdjęć.

Biblioteka OpenCV umożliwia wykonanie binaryzacji na dwa sposoby tj. prostej binaryzacji uwzględniającej globalny próg lub metody adaptacyjnej bazującej na obszarach otaczających przetwarzany piksel. **Prosta (globalna) binaryzacja** polega na wykorzystaniu przetworzeniu każdego z pikseli przy wykorzystaniu tej samej wartość progowa. Jeśli wartość piksela jest mniejsza od progu, to nowa wartość piksela ustawiana jest na 0, w przeciwnym razie ustawiana jest na wartość maksymalną.

```
zdjecieBinarne = cv2. threshold(zdjecie, próg, wartość maksymalna, typ_progowania)
```

gdzie:

- zdjęcie - tablica wejściowa (wielokanałowa, 8-bitowa lub 32-bitowa zmiennoprzecinkowa typu float),
- próg – wartość progowa, która jest używana do klasyfikacji wartości pikseli
- wartość maksymalna - maksymalna wartość do użycia z typami progowania THRESH_BINARY i THRESH_BINARY_INV,
- typ progowania – wykorzystywana metoda progowania.

Podstawowe progowanie odbywa się przy użyciu typu cv2.THRESH_BINARY. Wszystkie proste typy progowania to:

THRESH_BINARY Python: cv2.THRESH_BINARY	$wynik(x,y) = \begin{cases} maksymala_{wartość} & if \quad zdjęcie(x,y) > próg \\ 0 & else \end{cases}$
THRESH_BINARY_INV Python: cv2.THRESH_BINARY_INV	$wynik(x,y) = \begin{cases} 0 & if \quad zdjęcie(x,y) > próg \\ maksymala_{wartość} & else \end{cases}$
THRESH_TRUNC Python: cv2.THRESH_TRUNC	$wynik(x,y) = \begin{cases} próg & if \quad zdjęcie(x,y) > próg \\ zdjęcie(x,y,) & else \end{cases}$
THRESH_TOZERO Python: cv2.THRESH_TOZERO	$wynik(x,y) = \begin{cases} zdjęcie(x,y,) & if \quad zdjęcie(x,y,) > próg \\ 0 & else \end{cases}$
THRESH_TOZERO_INV Python: cv2.THRESH_TOZERO_INV	$wynik(x,y) = \begin{cases} 0 & if \quad zdjęcie(x,y,) > próg \\ zdjęcie(x,y,) & else \end{cases}$

Przykładowa funkcja dla binaryzacji pokazana została w funkcji (8):

```
import cv2 as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('moje_zdjecie.png', cv.IMREAD_GRAYSCALE)

ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)
ret,thresh2 = cv.threshold(img,127,255,cv.THRESH_BINARY_INV)
ret,thresh3 = cv.threshold(img,127,255,cv.THRESH_TRUNC)
ret,thresh4 = cv.threshold(img,127,255,cv.THRESH_TOZERO)
ret,thresh5 = cv.threshold(img,127,255,cv.THRESH_TOZERO_INV)

titles = ['Original Image','BINARY','BINARY_INV','TRUNC','TOZERO','TOZERO_INV']
images = [img, thresh1, thresh2, thresh3, thresh4, thresh5]

#plot images in one window
for i in range(6):
    plt.subplot(2,3,i+1),plt.imshow(images[i],'gray',vmin=0,vmax=255)
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

(8)

Podobnie jak w przypadku wyrównania histogramów, zastosowanie globalnego progowania nie przyniesie oczekiwanych efektów w przypadku występowania np. znaczących zmian oświetlenia w różnych fragmentach obrazów. Z tego też powodu niezbędne jest wykorzystanie metod lokalnej binaryzacji nazywanej również *adaptive thresholding*, w której wartość progu jest automatycznie wyznaczana w oparciu zdefiniowany obszar wokół przetwarzanego punktu. Otrzymujemy więc różne progi dla różnych regionów tego samego obrazu, co daje lepsze rezultaty dla obrazów o zmiennym oświetleniu. W celu implementacji tej metody wykorzystywana jest funkcja:

zdjecieBinarne = cv2. adaptiveThreshold(zdjecie, wartość maksymalna, metod adaptacyjnego przetwarzania danych, typ_progowania, wielkość maski, stała)

gdzie:

- zdjęcie - tablica wejściowa (8-bitowa),
- wartość maksymalna - Wartość niezerowa przypisana do pikseli, dla których warunek jest spełniony,
- metod adaptacyjnego przetwarzania danych – adaptacyjny algorytm progowania:
  - ADAPTIVE_THRESH_MEAN_C / Python: cv.ADAPTIVE_THRESH_MEAN_C - wartość progowa to średnia z obszaru sąsiedztwa
  - wartość progowa jest sumą ważoną wartości sąsiedztwa, gdzie wagi wyliczane przy wykorzystaniu funkcji Gaussa z wartością sgima = `0.3*((wielkość maski-1)*0.5 - 1) + 0.8` (więcej na ten temat opisane zostało w rozdziale dotyczącym przetwarzania kontekstualnego)
- typ_progowania -. typ progowania, który musi być zdefiniowany jako THRESH_BINARY lub THRESH_BINARY_INV

Przykładowa funkcja dla lokalnej binaryzacji ma postać (9):

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('moje_zdjecie.png', cv2.IMREAD_GRAYSCALE)

img = cv2.medianBlur(img,5)
th = cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C, \ cv2.THRESH_BINARY,11,2)

cv2.imshow('Adaptive Mean Thresholding',th)

print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
   cv2.waitKey(1)
```

(9)

W zależności od sposobu zdefiniowania progu doboru pikseli wykorzystywanych w procesie zamiany na postać binarną obrazów cyfrowych rozróżniamy m.in.:

- binaryzację z dolnym progiem, dla której wartości intensywności mniejsze od założonego progu zostają zmienione na czarny (0), a reszta na kolor biały (1);
- binaryzacja z górnym progiem, dla której wartości intensywności większe od założonego progu zostają zmienione na czarny (0), a reszta na kolor biały (1),
- binaryzacja z podwójny progiem, która bazuje na dwóch progach - dolnym i górnym. Przyjmuje się, że piksele, które mieszczą się w zdefiniowanym zakresie, przyjmują kolor biały (1), a pozostałe czarny (0).

Poniżej (funkcja 10) przedstawiono przykład binaryzacji dla trzech progów odcięcia – 25%, 50% i 75% maksymalnej wartości (w tym przypadku 255):

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('moje_zdjecie.png', cv2.IMREAD_GRAYSCALE)

imgArray = np.asarray(img)
ret1, imgThresholded1 = cv2.threshold(imgArray, 64, 255, cv2.THRESH_BINARY)
ret2, imgThresholded2 = cv2.threshold(imgArray, 127, 255, cv2.THRESH_BINARY)
ret3, imgThresholded3 = cv2.threshold(imgArray, 191, 255, cv2.THRESH_BINARY)
```

(10)

```
plt.figure(1, figsize = (15, 10))

plt.subplot(131)
plt.title('Próg 64')
plt.axis('off')
plt.imshow(imgThresholded1, cmap = 'gray')

plt.subplot(132)
plt.title('Próg 127')
plt.axis('off')
plt.imshow(imgThresholded2, cmap = 'gray')

plt.subplot(133)
plt.title('Próg 191')
plt.axis('off')
plt.imshow(imgThresholded3, cmap = 'gray')

plt.show()
print(cv2.waitKey(0))
cv2.destroyAllWindows()
for i in range(2):
    cv2.waitKey(1)
```