Uniwersytet Zielonogórski
Instytut Sterowania i Systemów Informatycznych

Rozpoznawanie Obrazów

Semestr VII, Informatyka

mgr inż. Marcin Skobel

2021

Laboratorium nr 4: Operatory punktowe.

I. Zagadnienia teoretyczne

Wstęp

W procesie przetwarzania obrazów cyfrowych istnieje grupa przekształceń polegających na zastępowaniu wartości pojedynczego piksela inną wartością. Metody te posiadają zbiorczą nazwę operatorów punktowych. Wbrew pozorom jednak operacje punktowe mogą przyjmować różne formy, począwszy od przekształceń za pomocą prostych działań arytmetycznych poprzez nieco bardziej skomplikowane funkcje a skończywszy na algorytmach przetwarzania punktowego obrazów.

Podstawowe transformacje pikseli

Transformacje pikseli na obrazie można zdefiniować za pomocą ogólnego wzoru:

$$g(x, y) = T(f(x, y)) \tag{1}$$

Dodawanie
Najprostszy rodzaj transformacji punktowej piksela to operacje z użyciem skalarów. Przykładem może być operacja dodawania polega na zmianie wartości pojedynczych pikseli przez dodanie do każdego piksela jednakowego skalaru:

$$T(p) = p + s, \tag{2}$$

gdzie p oznacza wartość pojedynczego piksela, natomiast s to skalar.
Dodawanie skalaru który zawiera ujemną wartość jest równoznaczne z odejmowaniem.
Mnożenie
Oprócz dodawania można też przy użyciu skalaru dokonać mnożenia/dzielenia wartości pojedynczych pikseli:

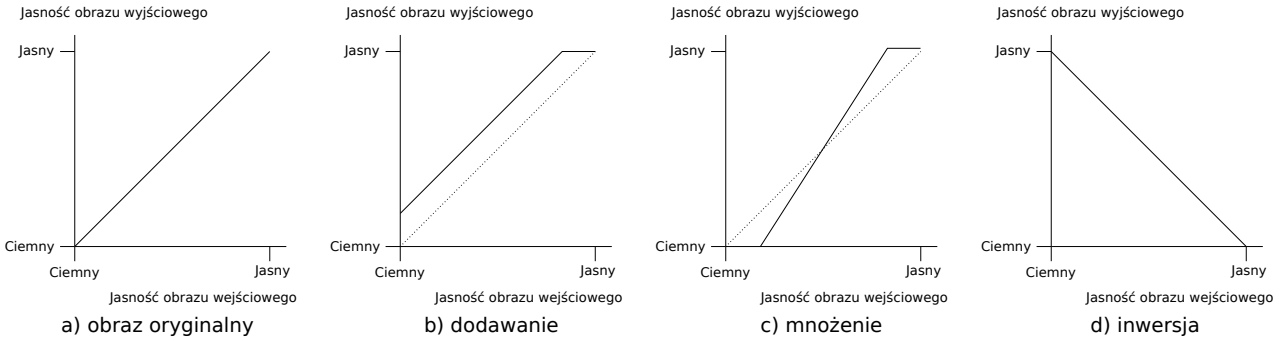
$$T(p) = p * s, \tag{3}$$

Inwersja

Operacja inwersji polega na odwróceniu barw pikseli na obrazie, wówczas jasne fragmenty obrazu stają się ciemne i na odwrót. Efekt jaki uzyskuje się dzięki inwersji nosi nazwę negatywu:

$$g(x, y) = L - 1 - I(x, y), \tag{4}$$

gdzie g oznacza nową wartość piksela, L - 1 oznacza górną wartość maksymalnej intensywności (dla obrazów uint8 = 255), I(x,y) oznacza wartość intensywności piksela obrazu wejściowego.



Rysunek 1: Podstawowe punktowe transformacje pikseli

Korekcja gamma

Korekcja gamma wykorzystywana jest do poprawy jakości obrazu. Korekcja ta może być przedstawiona przy użyciu wzoru:

$$g(x, y) = c\ I(x, y)^{\gamma} \tag{5}$$

gdzie c oraz γ są stałymi dodatnimi, natomiast I(x,y) oznacza wartość intensywności piksela. W większości przypadków wartość stałej c przyjmuje się jako równą 1. W przypadku gdy wartość $\gamma = 1$ to odwzorowanie jest liniowe, a to z kolei oznacza, że obraz wyjściowy nie zmienia się względem obrazu wejściowego. Z kolei gdy $\gamma < 1$, wąski zakres ciemnych (o niskiej intensywności) wartości pikseli w obrazie wejściowym jest odwzorowywany na szeroki zakres intensywności w obrazie wyjściowym, podczas gdy szeroki zakres jasnych (o wysokiej intensywności) wartości pikseli w obrazie wejściowym jest odwzorowywany na wąski zakres wysokich intensywności obrazu wyjściowego. Efekt z wartości $\gamma > 1$ jest odwrotny do wartości $\gamma < 1$. Korekcja gamma wykorzystywana jest do poprawy jakości wyświetlanych obrazów w monitorach i ekranach telewizyjnych.

Normalizacja histogramu

Normalizacja obrazu polega na zwiększeniu zakresu intensywności danego obrazu. Histogram przekształcanego obrazu podlega rozciągnięciu i przesunięciu w taki sposób aby obejmował pełen zakres dostępnych poziomów intensywności na przykład od 0 do 255. Jeśli oryginalny histogram starego obrazu O zaczyna się od wartości O_{min} i rozciąga się do O_{max} poziomów jasności, wtedy możemy przeskalować obraz tak, aby piksele na nowym obrazie N znajdowały

się między minimalnym poziomem wyjściowym N_{min} a maksymalnym poziomem N_{max}, po prostu zwiększając poziomy intensywności wejściowej zgodnie ze wzorem:

$$N_{x,y} = \frac{N_{max} - N_{min}}{O_{max} - O_{min}} * (O_{x,y} - O_{min}) + N_{min} \qquad \forall x, y \in 1, N \tag{6}$$

Wyrównanie histogramu

Korekcja histogramu to nieliniowy proces mający na celu podkreślenie jasności obrazu w sposób szczególnie odpowiedni do analizy wizualnej człowieka. Korekcja histogramu ma na celu zmianę obrazu w taki sposób, aby uzyskać obraz z bardziej płaskim histogramem, na którym wszystkie poziomy są jednakowo prawdopodobne. Aby zastosować ten operator, możemy najpierw przyjrzeć się histogramom. Histogram obrazu jest funkcją dyskretną, której dane wejściowe to wartość poziomu intensywności, a wynik to liczba pikseli z tą intensywnością wartości poziomu, dodatkowo można ją podać jako $h(x_n) = y_n$. Na obrazie w skali szarości, intensywności obrazu przyjmują wartości pomiędzy [0, L-1]. Jak wspomniano wcześniej, niskie wartości intensywności na obrazie (lewa strona histogramu) odpowiadają ciemnym regionom i wartościom wysokiego poziomu intensywności obrazu (rozszerzenie prawej strony histogramu) odpowiadają jasnym obszarom.

Na obrazie o niskim kontraście histogram jest wąski, podczas gdy na obrazie o wysokim kontraście histogram jest rozłożony. Celem wyrównywania histogramu jest poprawa kontrastu obrazu poprzez przeskalowanie histogramu, tak aby histogram nowego obrazu był rozłożony, a intensywność pikseli obejmowała wszystkie możliwe wartości poziomu szarości. Przeskalowanie histogramu zostanie wykonane przy użyciu transformacji. Aby zapewnić, że dla każdej wartości poziomu szarości w obrazie wejściowym istnieje odpowiednie wyjście, wymagana jest transformacja jeden do jednego; to znaczy, że każde wejście ma unikalne wyjście. Oznacza to, że transformacja powinna być funkcją rosnącą. Zapewni to odwracalność transformacji. Przed zdefiniowaniem transformacji wyrównawczej histogramu należy obliczyć:
- Standaryzowany Histogram obrazu wejściowego, tak aby zakres znormalizowanego histogramu to [0, 1]
- Ponieważ obraz jest dyskretny, prawdopodobieństwo wystąpienia wartości poziomu szarości jest określone przez $p_x(i)$ to stosunek liczby pikseli o wartości szarości i do całkowitej liczby pikseli w obrazie.
- Dystrybuantę zdefiniowaną jako $C(i) = \sum_{j=0}^{i} p_x(j)$, gdzie $0 \geqslant i \geqslant L-1$ oraz L jest całkowitą liczbą poziomów szarości na tym obrazie. C (i) jest sumą całkowitego prawdopodobieństwa wartości poziomu szarości pikseli od 0 do i. Uwaga zwróć uwagę, że C jest funkcją rosnącą.

Transformację wyrównania histogramu można zdefiniować w następujący sposób:

$$h(u) = round\left(\frac{C(u) - C_{min}}{1 - C_{min}} * (L-1)\right), \tag{7}$$

gdzie C_{min} to minimalna skumulowana wartość dystrybuanty na obrazie. Dla obrazu w skali szarości z zakresem między [0, 255], jeśli $C(u) = C_{min}$ to $h(u) = 0$. Jeśli $C(u) = 1$, to $h(u) = 255$. Wartość całkowitą dla obrazu wyjściowego uzyskuje się poprzez zaokrąglenie Równania 7.

Rozciągnięcie kontrastu

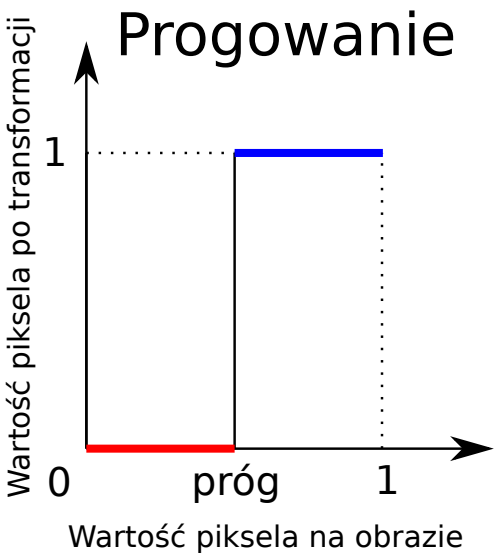
Zadanie polegające na rozciągnięciu kontrastu jest oparte na podobnej idei jak wyrównanie histogramu z tą różnicą, że jest przeprowadzane na wartościach pikseli a nie na dystrybuancie histogramu. W celu zwiększenia wartości intensywności piksela stosowane jest rozciąganie kontrastu zakresu wartości poprzez przeskalowanie wartości pikseli w obrazie wejściowym. Rozważmy 8-bitowy obraz z zakresem wartości pikseli $[a, b]$, gdzie $a > 0$ i $b < 255$. Jeśli a jest znacznie większe od zera lub jeśli b jest znacznie mniejsze niż 255, wówczas szczegóły obrazu mogą nie być widoczne. Ten problem można rozwiązać przez przeskalowanie zakresu wartości pikseli do $[0, 255]$, czyli znacznie większego zakresu pikseli. Transformację rozciągania kontrastu, $t(x, y)$ podaje poniższe równanie:

$$t(x, y) = 255 * \frac{I(x, y) - a}{b - a}, \tag{8}$$

gdzie $I(x, y)$, to intensywność pikseli w (x, y), a to odpowiednio minimalna wartość piksela oraz b to maksymalna wartość piksela w obrazie wejściowym.

Progowanie (binaryzacja)

Jest to operacja punktowa, która tworzy obrazy binarne z poziomu szarości, zmieniając wszystkie piksele poniżej pewnego progu na zero i wszystkie piksele powyżej tego progu na jeden, jak pokazano na poniższym rysunku: Jeśli g(x, y) jest progowaną wersją f(x, y) przy wybranym

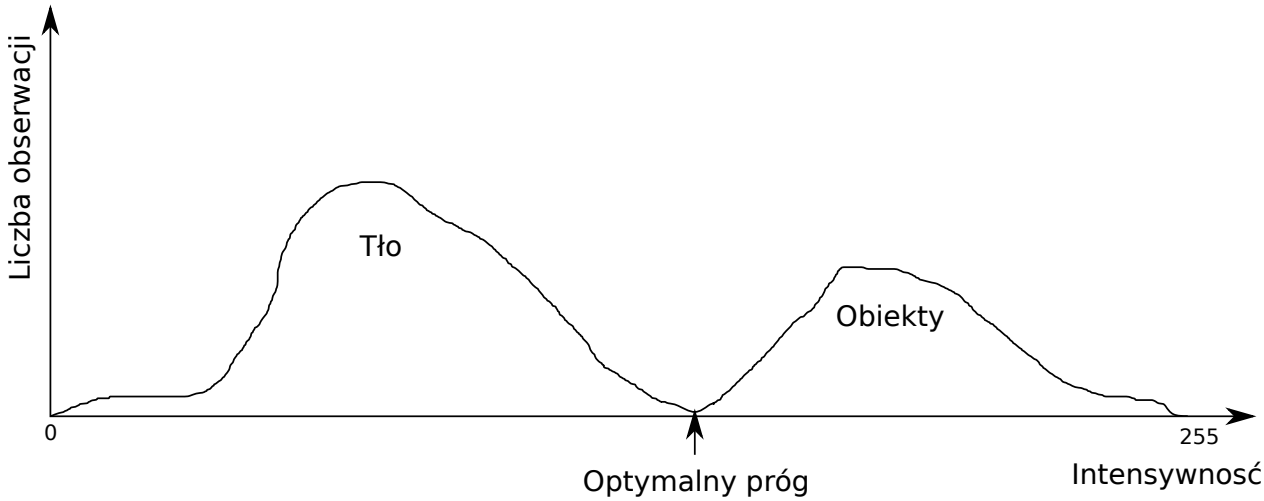


Rysunek 2: Schemat progowania

globalnym progu T, to można zastosować następujące zasady:

$$g(x, y) = \begin{cases} 1, & \text{dla f(x, y)} \geqslant \text{T} \\ 0, & \text{w przeciwnym razie} \end{cases} \tag{9}$$

Jak łatwo zauważyć w wyniki progowania powstaje obraz binarny (jedynie kolor czarny i biały). W najprostszej wersji wartość progu binaryzacji można dobrać ręcznie po uprzedniej analizie histogramu. W trakcie dobierania progu warto odszukać wartość optymalną która będzie się znajdować w punkcie siodłowym pomiędzy wartościami modalnymi. Istnieją jednak metody



Rysunek 3: Optymalne progowanie

globalnego automatycznego dobierania optymalnego progu binaryzacji. Jedną z najpopularniejszych metod binaryzacji optymalnej jest progowanie Otsu `https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4310076`, bazujące na minimalizacji sumy ważonej wariancji dwóch klas obiektów, które jest jednoznaczne z maksymalizacją wariancji międzyklasowej. Istnieje jednak wiele innych metod binaryzacji globalnej m. in.: algorytmy bazujące na klasteryzacji, metody bazujące na entropii. Istnieje również drugie podejście do zagadnienia progowania zwane adaptacyjnym. Binaryzacja adaptacyjna polega na progowaniu wybranych fragmentów obrazu. W praktyce oznacza to dobieranie różnych progów w zależności od aktualnej sytuacji na progowanym fragmencie obrazu.

II. Przykład praktyczny

Wstępna konfiguracja

Przykład praktyczny tradycyjnie rozpoczynamy od zaimportowania niezbędnych bibliotek oraz zamontowaniu Dysku Google:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import cv2
from google.colab import drive
drive.mount('/content/drive')
```

Wczytanie pliku

Następnie należy wczytać plik testowy i skonwertować do go odcieni szarości. Tym razem użyjemy gotowej funkcji do konwersji obrazu do odcieni szarości.

```
path = testImagePath = 'drive/My Drive/Colab Notebooks/Images/Test.tif'
img = Image.open(path)
imgGray = img.convert('LA')
```

Dodawanie

Dodawanie skalaru do wszystkich pikseli obrazu rozpoczynamy od wczytania go do tablicy nupmy deklarując ją jako typ o większym zakresie niż uint8. Wynika to z faktu, że w innym

przypadku nupmy domyślnie zaimportuje plik w formacie uint8 a to z kolei uruchomi algorytm chroniący przekroczenie wartości 255. Po przekroczeniu tej wartości licznik się kasuje, czyli jeśli była wartość 249 i dodaliśmy 21 otrzymamy wynik 14 zamiast 270. Trzecia linijka odpowiada za redukcję wszystkich dużych wartości do maksimum dla uint8 czyli 255.

```
npImageArray = np.asarray(imgGray,dtype="uint32")
imgGrayAdd = npImageArray + 100
imgGrayAdd[imgGrayAdd > 255] = 255
imgGrayAdd = Image.fromarray(np.uint8(imgGrayAdd))
plotGrayAdd = plt.imshow(imgGrayAdd)
```

Mnożenie

Mnożenie zostało wykonane analogicznie do dodawania, jedyną różnicą w kodzie jest działanie arytmetyczne. W trakcie mnożenia korzystamy nadal z tej samej zmiennej tablicowej o nazwie npImageArray.

```
imgGrayMul = npImageArray * 2
imgGrayMul[imgGrayMul > 255] = 255
imgGrayMul = Image.fromarray(np.uint8(imgGrayMul))
plotGrayMul = plt.imshow(imgGrayMul)
```

Inwersja

W pierwszej wersji dokonywana jest inwersja zgodnie ze wzorem z części teoretycznej bieżącej listy. Niestety matplotlib niespecjalnie dobrze wyświetla obraz po inversji. Lepiej zapisać go do pliku i podejrzeć efekt na dysku Google.

```
imgGrayInv = 255 - npImageArray
imgGrayInv = Image.fromarray(np.uint8(imgGrayInv))
plotGrayInv = plt.imshow(imgGrayInv)
imgGraySave = imgGrayInv.save('drive/My Drive/Colab Notebooks/Images/imgGrayInv.tif')
```

Druga metoda to użyciu gotowej funkcji inverse w pakiecie numpy. Metoda jest o tyle wygodna, że pozwala na konwersję zarówno obrazów w odcieniach szarości jaki i w RGB.

```
npImageArray = np.asarray(img,dtype="uint8")
imgInv = np.invert(npImageArray)
imgGrayAdd = Image.fromarray(np.uint8(imgInv))
plotInv = plt.imshow(imgInv)
```

Korekcja gamma

Korekcję gamma rozpoczynamy od wczytania obrazu do tablicy numpy typy uint8. Następnie deklarujemy kluczową w zadaniu zmienną o nazwie gamma. Następnie do zmiennej b1 konwertujemy tablicę do wartości zmiennoprzecinkowych typu float. zmienna b3 posłuży nam do znalezienia wartości maksymalnej intensywności na obrazie. W kolejnym kroku deklarujemy zmienną b2, która przechowa tablicę z wartościami intensywności podzielonymi przez maksymalną intensywność na obrazie. W języku Python zmiennej można użyć ponownie, zatem użyjemy zmiennej o nazwie b3 drugi raz tym razem logarytmując wartość b2 i mnożąc razy współczynnik gamma. W kolejnym kroku wykonamy korekcję gamma przy użyciu funkcji exp i pomnożeniu wartości razy 255. Następnie zmienna c1 posłuży do konwersji wartości do typu int. Ostatnie dwie linijki to zamiana do zmiennej obrazu i wyświetlenie wyniku na ekranie.

```
npImageArray = np.asarray(imgGray,dtype="uint8")
gamma = 1
b1 = npImageArray.astype(float)
b3 = np.max(b1)
b2 = b1/b3
b3 = np.log(b2)*gamma
c = np.exp(b3)*255.0
c1 = c.astype(int)
c2 = Image.fromarray(np.uint8(c1))
plotc2 = plt.imshow(c2)
```

Normalizacja histogramu

Normalizacja histogramu jest wykonywana wprost ze wzoru znajdującego się w części teoretycznej. Największa zatem trudność tego zadania to poprawne wpisanie działania arytmetycznego którego został zapisany do zmiennej nXY.

```
npImageArray = np.asarray(imgGray,dtype="uint8")
nMin = 0
nMax = 255
oMin = np.min(npImageArray)
oMax = np.max(npImageArray)
nXY = (nMax-nMin)/(oMax-oMin)*(npImageArray-oMin)+nMin
nXYImage = Image.fromarray(np.uint8(nXY))
plotnXYImage = plt.imshow(nXYImage)
```

Wyrównanie histogramu

Wyrównanie histogramu rozpoczynamy od wczytania tablicy npImageArray typu uint8. W kolejnym kroku konwertujemy obraz z poziomu 2d do 1D tym samym tworząc ciąg wartości. Następnie tworzymy histogram do zmiennej hist przy użyciu biblioteki numpy. Musimy pamiętać o zadeklarowaniu liczby słupków oraz przedziału wartości. W kolejnym kroku przy użyciu funkcji numpy.cumsum() tworzymy funkcję dystrybuanty histogramu. Następnie miejsca gdzie dystrybuanta = 0 są maskowane i przechowywane w zmiennej CDF_m. Kolejne trzy linijki kodu stanowią działania arytmetyczne wykonujące wyrównanie histogramu. W kolejnym kroku dokonujemy konwersji do wartości uint8 a wszystkie maskowane miejsca otrzymują wartość 0. W następnej linijce zmiennej im2 przypisywane są wartości dystrybuanty do spłaszczonej tablicy Image1D. Następnie przy użyciu funkcji reshape zmienna im2 jest przekształcana do tablicy dwuwymiarowej na kształt obrazu wejściowego. Ostatnie trzy linijki to przekształcenie tablicy do zmiennej obrazowej następnie wyświetlenie wyniku i zapisanie na dysk Google

```
npImageArray = np.asarray(imgGray,dtype="uint8")
Image1D = npImageArray.flatten()
hist, bins = np.histogram(npImageArray,256,[0,256])
CDF = hist.cumsum()
CDF_m = np.ma.masked_equal(CDF,0)
num_CDF_m = (CDF_m - CDF_m.min())*256
den_CDF_m = (CDF_m.max() - CDF_m.min())
CDF_m = num_CDF_m/den_CDF_m
cdf = np.ma.filled(CDF_m, 0).astype('uint8')
im2 = cdf[Image1D]
im3 = np.reshape(im2,npImageArray.shape)
```

```
Equalized = Image.fromarray(np.uint8(im3))
plotnEqualized = plt.imshow(Equalized)
EqualizedSave = Equalized.save('drive/My Drive/Colab Notebooks/Images/Equalized.tif')
```

Rozciąganie kontrastu

Rozciągnięcie kontrastu również zostało przeprowadzone bazując na wzorze z części teoretycznej.

```
npImageArray = np.asarray(imgGray,dtype="uint8")
a = npImageArray.min()
b = npImageArray.max()
c = npImageArray.astype(float)
tabStr = 255*(c-a)/(b-a)
imStr = Image.fromarray(np.uint8(tabStr))
plotimStr = plt.imshow(imStr)
```

Progowanie

Poniższa wersja progowania jest najprostszym możliwym przykładem tego zadania. W pierwszym kroku wczytujemy obraz do tablicy nupmy. Następnie wyznaczamy dowolny próg binaryzacji. W kolejnym kroku przekształcamy tablicę do wersji jednowymiarowej. Następnie dokonujemy modyfikacji pikseli do wartości minimalnej lub maksymalnej w zależności od przyjętego progu. W tym zadaniu użyto biblioteki numpy dlatego wartości binarne zostały zapisane w typie uint8. Mamy zatem dwie wartości 0 oraz 255. Obraz binarny powinien mieć wartości 0 oraz 1, lecz w przypadku numpy jest to problematyczne i wygodniej jest użyć 0 i 255 oraz typu uint8.

```
npImageArray = np.asarray(imgGray,dtype="uint8")
threshold = 123
Image1D = npImageArray.flatten()
imgThreshold = Image1D
imgThreshold[imgThreshold < threshold] = 0
imgThreshold[imgThreshold >= threshold] = 255
imgThreshold = np.reshape(imgThreshold,npImageArray.shape)
imgThreshold = Image.fromarray(np.uint8(imgThreshold))
plotimgThreshold = plt.imshow(imgThreshold)
```

Na szczęście istnieją również gotowe biblioteki służące do progowania obrazów. Przykładowo binaryzację Otsu można wykonać przy użyciu biblioteki OpenCV. Pakiet ten posiada możliwość wczytywania obrazu od razu w odcieniach szarości. Wystarczy w funkcji imread obok ścieżki do pliku wpisać po przecinku wartość 0.

```
path = 'drive/My Drive/Colab Notebooks/Images/Test.tif'
img = cv2.imread(path,0)
ret2,th2 = cv2.threshold(img,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
plotimgThresholdOtsu = plt.imshow(th2)
```

III. Uwagi

Przetwarzanie obrazów z użyciem operatorów punktowych można zweryfikować przy użyciu oprogramowania ImageJ. Szczególnie warto przejrzeć metody automatycznej binaryzacji wybierając w menu programu: Image - Adjust - Auto Threshold. Plik z zadaniami przykładowymi można pobrać ze strony: `http://staff.uz.zgora.pl/mskobel/lab4_py.py`

IV. Lista zadań

1. Zaprojektuj algorytm, który po wczytaniu obrazu sprawdzi czy jest w przestrzeni RGB czy w odcieniach szarości, następnie wykona inwersję obrazu do negatywu bez użycia gotowych bibliotek typu invert.
2. Wczytaj dowolny obrazek w formacie tif, dokonaj rozciągnięcia kontrastu oraz inwersji następnie wyświetl histogram, dobierz ręcznie próg binaryzacji i dokonaj progowania.
3. Wczytaj obrazek (RGB) cytologiczny `http://staff.uz.zgora.pl/mskobel/Test.tif`, następnie wyodrębnij kanały R G i B do oddzielnych zmiennych. Następnie dokonaj normalizacji histogramu dla każdego kanału a na koniec wykonaj progowanie metodą otsu dla każdego z kanałów oddzielnie. Porównaj wyniki i odpowiedz na pytanie na którym kanale binaryzacja najlepiej odseparowała obszar jąder komórkowych od tła.