Inżynieria obrazów - Laboratoria nr 1

Szymon Wichrowski 263960

30 marca 2024

1 Cele ćwiczenia

Celem ćwiczenia jest nauka programowego przetwarzania obrazów oraz zapoznanie się z podstawowymi modelami barw i ich praktycznym zastosowaniem. Dodatkowo do wykonania zadań wykorzystywana jest biblioteka *OpenCV*, która ma szerokie zastosowanie w rozpoznawaniu i przetwarzaniu obrazów, a ponadto jest dostępna w wielu językach programowania.

2 Zadanie nr 1 - Wykonać filtr górnoprzepustowy dla dowolnego obrazka wykorzystując podaną maskę

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Rysunek 1: Maska podana w poleceniu do zadania

Na początku należy wczytać wybrany obraz z pliku. Można to zrobić np. za pomocą funkcji *imread()* z biblioteki *OpenCV*, która wczytuje obraz w formacie BGR, czyli kanały kolorów są ułożone w kolejności: niebieski, zielony, czerwony. Natomiast do wyświetlenia obrazów służy funkcja *imshow()* z biblioteki *matplotlib*, która zakłada, że dane wejściowe są w formacie RGB, tzn. kanały kolorów są ułożone w odwrotnej kolejności niż we wczytanym obrazie. Można zmienić format obrazu na RGB za pomocą funkcji *cvtColor()* z biblioteki *OpenCV*.

Listing 1: Wczytanie i przygotowanie obrazu do dalszego przetwarzania

```
primaryImageBGR = cv2.imread("-- ścieżka do obrazu --")
primaryImageRGB = cv2.cvtColor(primaryImageBGR, cv2.COLOR_BGR2RGB)
```

Do wykonania filtru górnoprzepustowego można wykorzystać funkcję *filter2D()* z biblioteki *OpenCV*, która wykonuje operację splotu: dla każdego piksela obrazu obliczana jest jego nowa wartość na podstawie wartości pikseli sąsiadujących i wag filtru z podanej w argumencie maski. Dodatkowo maskę dzieli się przez sumę wartości wag filtru, aby zapobiec znacznemu rozjaśnieniu lub pociemnieniu obrazu.

Listing 2: Filtracja obrazu

```
kernel = numpy.array([[-1, -1, -1],
                      [-1, 8, -1],
                      [-1, -1, -1]])
sumOfWeights = kernel.sum()
if (sumOfWeights == 0):
    sumOfWeights = 1
kernel = kernel / sumOfWeights
filteredImage = cv2.filter2D(primaryImageRGB, -1, kernel)
```

Rysunek 2: Przykładowe działanie filtru

3 Zadanie nr 2 - Przekształcić kolory obrazu

Dla dowolnego obrazka kolorowego RGB [0-255;0-255;0-255] należy dokonać konwersji na format zmiennoprzecinkowy RGB [0-1.0;0-1.0;0-1.0] oraz wyznaczyć nowe wartości pikseli obrazu według podanego wzoru. Dodatkowo trzeba założyć, że jeśli któraś z nowych wartości przekroczy 1.0, należy ją przyjąć jako 1.0.

$$\begin{bmatrix} R_{new} \\ G_{new} \\ B_{new} \end{bmatrix} = \begin{bmatrix} 0.393 & 0.769 & 0.189 \\ 0.349 & 0.689 & 0.168 \\ 0.272 & 0.534 & 0.131 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Rysunek 3: Wzór podany w poleceniu do zadania

Na początek należy podzielić wartości pikseli obrazu przez wartość 255.0 - pozwala to na pracę z wartościami zmiennoprzecinkowymi, z zakresu od 0 do 1.

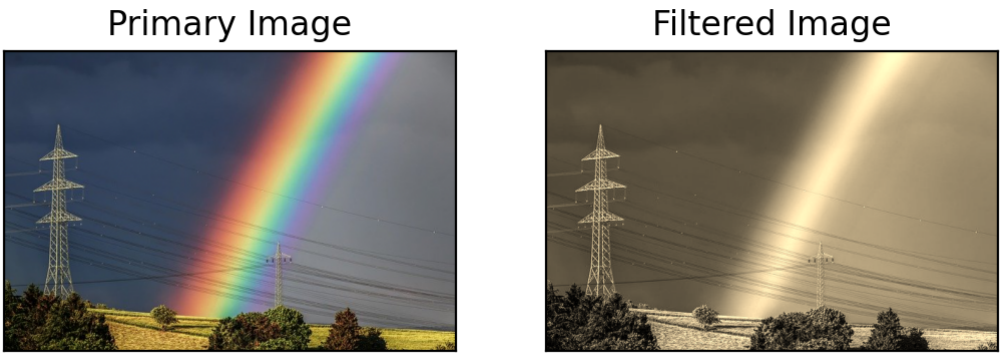
Listing 3: Konwersja na format zmiennoprzecinkowy

```
primaryImageRGB = primaryImageRGB / 255.0
```

Do mnożenia dwóch macierzy zgodnie ze wzorem podanym w poleceniu można wykorzystać funkcję *dot()* z biblioteki *NumPy*. Dla sytuacji, gdy wykonujemy operację między macierzą 3-wymiarową, a macierzą 2-wymiarową, tak naprawdę następuje mnożenie ostatniej osi macierzy 3-wymiarowej i przedostatniej osi macierzy 2-wymiarowej, czyli w tym przypadku wartości pikseli wczytanego obrazu oraz kolejnych kolumn transponowanej macierzy podanej w poleceniu. Następnie można zastosować funkcję *clip()*, również z biblioteki *NumPy*, która ograniczy wartości macierzy do podanego zakresu, tzn. przy podanym zakresie od 0 do 1, jeśli któraś z wartości pikseli powstałego obrazu przekroczy wartość 1.0, funkcja zmieni ją na wartość równą 1.

Listing 4: Wyznaczenie nowych wartości pikseli obrazu

```
multiplier = numpy.array([[0.393, 0.769, 0.189],
                          [0.349, 0.689, 0.168],
                          [0.272, 0.534, 0.131]])
newRGB = numpy.dot(primaryImageRGB, multiplier.T)
newRGB = numpy.clip(newRGB, 0, 1)
```



Rysunek 4: Przykładowe przekształcenie kolorów obrazu

Na rysunku nr 4 można zauważyć, że zaimplementowane przekształcenie obrazów tworzy *sepię*, czyli popularny filtr, który nadaje zdjęciom starodawny wygląd, poprzez taką modyfikacje ich kolorów, aby otworzyć brązową tonację.

4 Zadanie nr 3 - Konwersja obrazu do modelu barw YCrCb

Dla dowolnego obrazka kolorowego RGB [0-255;0-255;0-255] należy dokonać konwersji do modelu barw YCrCb. Następnie wyświetlić oryginalny obraz, składowe Y, Cr oraz Cb w odcieniach szarości, a także obraz po konwersji odwrotnej.

Konwersji obrazu RGB na YCrCb można dokonać za pomocą poniższego wzoru:

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0.229 & 0.587 & 0.114 \\ 0.500 & -0.418 & -0.082 \\ -0.168 & -0.331 & 0.500 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Rysunek 5: Równanie konwersji z RGB na YCrCb

Listing 5: Implementacja konwersji na model YCrCb

```
primaryImageRGB = primaryImageRGB / 255.0
multiplier = numpy.array([[0.229, 0.587, 0.114],
                          [0.500, -0.418, -0.082],
                          [-0.168, -0.331, 0.500]])

addend = numpy.array([0, 128, 128])
addend = addend / 255.0
Y = multiplier[0][0] * primaryImageRGB[:,:,0] + multiplier[0][1] * primaryImageRGB[:, :, 1] \
    +  multiplier[0][2] * primaryImageRGB[:, :, 2]
Cr = addend[1] + multiplier[1][0] * primaryImageRGB[:,:,0] + multiplier[1][1] \
    * primaryImageRGB[:, :, 1] +  multiplier[1][2] * primaryImageRGB[:, :, 2]
Cb = addend[2] + multiplier[2][0] * primaryImageRGB[:,:,0] + multiplier[2][1] \
    * primaryImageRGB[:, :, 1] +  multiplier[2][2] * primaryImageRGB[:, :, 2]
Y = numpy.clip(Y, 0, 1)
Cr = numpy.clip(Cr, 0, 1)
Cb = numpy.clip(Cb, 0, 1)
```

Po wyznaczeniu wartości składowych Y, Cr i Cb, ponownie należy zastosować funkcję *clip()*, z biblioteki *NumPy*, do ograniczenia wartości pikseli do zakresu [0, 1]. Do wyświetlenia tych składowych w odcieniach szarości można dodać parametr *cmap*, do funkcji *imshow()*, określający mapę kolorów używaną do wyświetlenia obrazu - w tym przypadku będzie to *'gray'*. Wyświetlenie obrazu w odcieniach szarości pozwoli na interpretowanie kolorów poprzez różnice jasności.

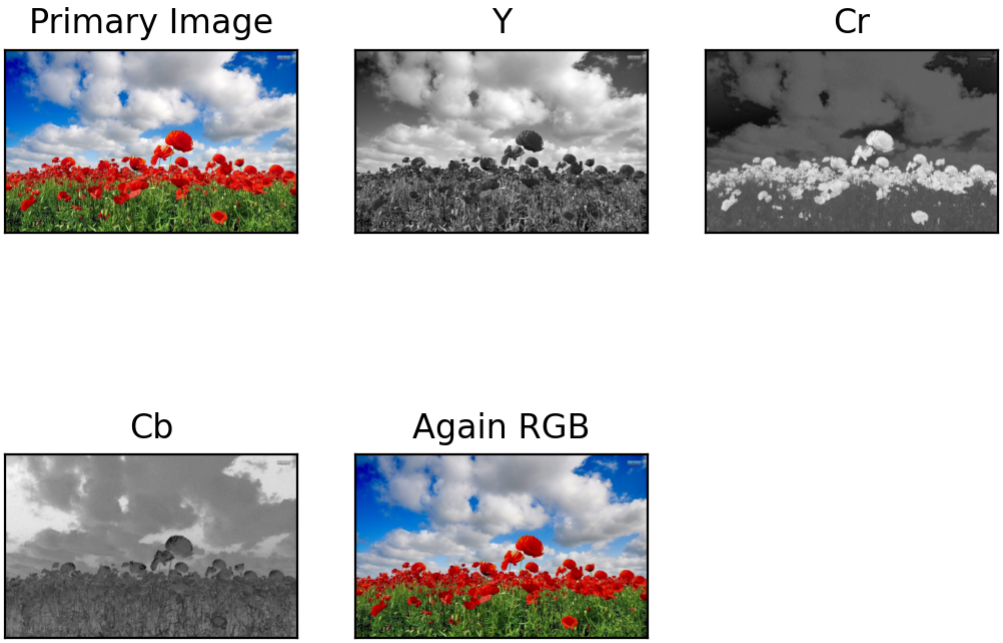
Do konwersji odwrotnej można wykorzystać poniższy wzór:

$$
\begin{aligned}
R &= Y + 1.402 \cdot (Cr - 128) \\
G &= Y - 0.34414 \cdot (Cb - 128) - 0.71414 \cdot (Cr - 128) \\
B &= Y + 1.772 \cdot (Cb - 128)
\end{aligned}
$$

Rysunek 6: Równanie odwrotne - konwersja z YCrCb na RGB

Listing 6: Implementacja konwersji na model RGB

```
R = Y + 1.402 * (Cr - addend[1])
G = Y - 0.34414 * (Cb - addend[2]) - 0.71414 * (Cr - addend[1])
B = Y + 1.772 * (Cb - addend[2])
R = numpy.clip(R, 0, 1)
G = numpy.clip(G, 0, 1)
B = numpy.clip(B, 0, 1)
againRGB = cv2.merge([R, G, B])
```



Rysunek 7: Przykładowa konwersja na YCrCb w odcieniach szarości oraz konwersja odwrotna

5 Źródła

- dokumentacja biblioteki OpenCV
- dokumentacja biblioteki NumPy
- Wikipedia - filtracja obrazów
- Wikipedia - YCbCr