Podstawy cyfrowego przetwarzania obrazów
- wstęp do OpenCV

Autorzy: dr inż. Jakub Markiewicz

dr inż. Michał Kowalczyk

Biblioteka OpenCV i działania z obrazami

Biblioteka **OpenCV** (*ang*. open computer vision) wykorzystywana jest przetwarzania obrazów, analiz przestrzennych i detekcji obiektów. Stanowi ona źródło funkcji tzw. widzenia maszynowego, będącego częścią składową sztucznej inteligencji.

1. Import biblioteki OpenCV

W celu rozpoczęcia pracy z biblioteką OpenCv niezbędny jest jej import przy wykorzystaniu funkcji import cv2 oraz biblioteki numpy. Warto również wykorzystać bibliotekę matplotlib do prezentacji wyników:

```
>>> import numpy
>>> import matplotlib
>>> import cv2 #import OpenCV
>>> print(cv2.__version__)
4.1.1
>>>
```

W przykładzie również zostało podane zapytanie o wersję biblioteki OpenCV. Dla skrócenia odwołań do biblioteki Numpy często ładuje się ją jako uchwyt (tutaj 'np'):

```
>>> del numpy   #usunięcie biblioteki numpy
>>> import numpy as np   #ponowne załadowanie biblioteki numpy jako np
>>>
```

2. Odczyt, wyświetlenie i zapis obrazu

Do elementarnych zadań biblioteki należy ładowanie i wyświetlenie obrazu. Jeżeli obraz jest zapisany w katalogu, w którym zapisany jest skrypt Python'a to nie ma potrzeby definiować pełnej ścieżki dostępu do pliku: Do wczytania (załadowania) służy funkcja cv2**.imread(Nazwa, parametr)**, gdzie: **Nazwa** oznacza nazwę pliku z ewentualną ścieżką dostępu (jeżeli plik jest zlokalizowany w innym katalogu niż skrypt Python'a), **parametr** oznacza tryb wczytania obrazu. Jeżeli pod Nazwą nie będzie pliku, to błąd nie będzie sygnalizowany.
Wartości parametru oznaczają:

1 - obraz barwny bez kanału Alfa,
0 - obraz monochromatyczny (w skali szarości)
-1 - obraz pełny z kanałem Alfa (oznaczeniem przezroczystości)

Można odpowiednio za wartości (1, 0, -1) wpisać predefiniowane stałe:

cv2.IMREAD_COLOR,
cv2.IMREAD_GRAYSCALE,
cv2.IMREAD_UNCHANGED

Przykład odczytu i wyświetlenia obrazu:

```
>>> import numpy as np
>>> import cv2
>>> obr = cv2.imread('IMGP0927.jpg',0)
>>> cv2.imshow('obraz', obr)   #wyświetlenie okna z monochromatyczną wersją obrazu
>>> cv2.waitKey(0)   #oczekiwanie na wciśnięcie przycisku przez użytkownika
13
>>> cv2.destroyAllWindows()   #usunięcie okna z obrazem i innych, jeżeli były stworzone
>>>
```

Do zapisu zdjęcia wykorzystywana jest funkcja: **cv2.imwrite(Nazwa, obraz)**, gdzie **Nazwa** jest nazwą tworzonego pliku, a **obraz** oznacza zmienną, pod którą jest zapisany obraz.

Jeżeli okno do wyświetlania ma być skalowane przez użytkownika to należy je zdefiniować poprzez wykorzystanie polecenienia cv2.namedWindow(Nazwa, parametr), gdzie Nazwa jest nazwą okna, a parametr oznacza predefiniowane stałe:

cv2.WINDOW_AUTOSIZE - okno skalowane do obrazu
cv2.WINDOW_NORMAL - obraz skalowany do okna, a okno przez użytkownika

Przykład:

```
>>> import cv2
>>> obr = cv2.imread('IMGP0927.jpg',0)
>>> cv2.namedWindow('obraz', cv2.WINDOW_NORMAL)
>>> cv2.imshow('obraz',obr)
>>> cv2.imwrite('zdj_szare.jpg',obr)
True
>>>
```

```
import numpy as np
import cv2

obr = cv2.imread('IMGP0927.jpg',0)
cv2.imshow('obraz',obr)
k = cv2.waitKey(0)
if k == 27:   # czeka na znak ESC i wychodzi
   cv2.destroyAllWindows()
elif k == ord('s'):   # czeka na znak 's' do zapisu i wychodzi
   cv2.imwrite('drzewo_szare.jpg',obr)
   cv2.destroyAllWindows()
```

W systemie 64-ro bitowym należy dodać **&0xFF** do funkcji oczekiwania na przycisk, argumentem **czas** jest liczba milisekund:

k = cv2.waitKey(czas) & 0xFF

W celu odczytu wymiarów obrazu, należy użyć funkcji **.shape()**:

```
>>> wiersze,kolumny,kanaly = obr.shape
>>> print(wiersze,kolumny,kanaly)
```

```
1944 2592 3
>>>
```

Jeżeli obraz jest monochromatyczny to wynikiem funkcji **.shape()** są tylko wymiary obrazu (bez kanałów).

Ogólny rozmiar obrazu w bajtach podaje funkcja **.size** (bez kompresji):

```
>>> obr.size
15116544
>>>
```

3. Barwa pikseli

Często zachodzi potrzeba odczytu i modyfikacji zarówno pojedynczych jak i grup pikseli obrazu. Służy do tego dostęp do tablicy obrazu.

Przykład odczytu i zapisu wartości piksela:

```
>>> import numpy as np
>>> import cv2
>>> obr = cv2.imread('IMGP0927.jpg')
>>> piksel = obr[50,50]
>>> print(piksel)
[140  83  58]
>>> piksel = [50,60,70]
>>> obr[50,50] = piksel
>>> print(obr[50,50])
[50 60 70]
>>>
```

Przykład odczytu i modyfikacji składowych piksela (po ponownym załadowaniu obrazu):

```
>>> czerwony = obr[50,50,2]; zielony = obr[50,50,1]; niebieski = obr[50,50,0]
>>> print(czerwony, zielony, niebieski)
58 83 140
>>> obr[50,50,2] = 100
>>> print(obr[50,50])
[140  83 100]
```

Lepszym sposobem dostępu do wartości pikseli jest wywołanie funkcji Numpy (uznawane za szybsze). Wykorzystuje się do tego funkcję **.item()** - odczytującą, oraz **.itemset()** - ustawiającą wartość składowej piksela:

```
>>> obr.item(50,50,2) #odczyt wartości składowej czerwonej piksela
100
>>> obr.itemset((50,50,2),90) #ustawienie wartości
>>> obr.item(50,50,2)
90
```

Jeżeli zachodzi potrzeba rozdzielenia całego obrazu na warstwę niebieską, zieloną i czerwoną to należy wykorzystać funkcję **.split()**.
W celu połączenia trzech składowych w jeden obraz, używa się funkcji **.merge()** :

```
>>> b,g,r = cv2.split(obr) #rozdzielenie
>>> obr = cv2.merge((b,g,r)) #połączenie
>>>
```

Jednak dla zapewnienia efektywnego dostępu do składowych obrazu, szybsze jest zastosowanie indeksowania z biblioteki Numpy:

```
>>> g = obr[:,:,1] #odczyt składowej zielonej na jedną warstwę

>>> obr[:,:,1] = 50 #przyporządkowanie wartości 50
                        do zielonych składowych wszystkich pikseli obrazu
```

Czasami zachodzi potrzeba skopiowania i wstawienia fragmentu obrazu. Można odnieść się do zakresu, przez odpowiednie indeksowanie:

```
>>> fragment = obr[0:50, 0:100] #skopiowanie (zapamiętanie fragmentu obrazu o zakresie

        xp:xk, yp:yk)
>>> obr[100:150, 100:200] = fragment #wstawienie
>>>
```

Dodanie obrazu do innego, z efektem nałożenia, uzyskuje się za pomocą polecenia:
.addWeighted(obr1, w1, obr2, w2, c)
gdzie: obr1, obr2 - obrazy, w1,w2 - wagi obu obrazów, c - dodanie wartości.
Wartość nowego obrazu uzyskuje się według wzoru:

nowy = obr1 * w1+ obr2 * w2 + c

Jeżeli rozmiary obrazów będą niezgodne, to pojawi się błąd.

Przykład:

```
>>> import numpy as np
>>> import cv2
>>> obr1 = cv2.imread('IMGP0927.jpg') #obraz o większych wymiarach
>>> obr2 = cv2.imread('IMGP4123_fr.jpg') #obraz o mniejszych wymiarach
>>> dj2, di2, k2 = obr2.shape
>>> obr1n = obr1[0:dj2, 0:di2] #wycięcie fragmentu z wiekszego obrazu, o rozmiarze
mniejszego
>>> wynik = cv2.addWeighted(obr1n ,0.6 ,obr2 ,0.4 ,20 )
>>> cv2.imshow('wynik',wynik)
>>> cv2.waitKey(0)
>>> cv2.destroyAllWindows()
>>>
```

4. Typy danych w OpenCV

Biblioteka OpenCV zawiera definicje wielu różnych typów danych.
Są to:

CV_8U - 8-bit unsigned integers (0..255)
CV_8S - 8-bit signed integers (-128..127)
CV_16U - 16-bit unsigned integers (0..65535)
CV_16S - 16-bit signed integers (-32768..32767)
CV_32S - 32-bit signed integers (-2147483648..2147483647)
CV_32F - 32-bit floating-point numbers (-FLT_MAX..FLT_MAX, INF, NAN)
CV_64F - 64-bit floating-point numbers (-DBL_MAX..DBL_MAX, INF, NAN)

Służą one do opisu punktów obrazu jako:

Unsigned 8bits uchar 0~255
Mat: **CV_8UC1, CV_8UC2, CV_8UC3, CV_8UC4**

Signed 8bits char -128~127
Mat: **CV_8SC1, CV_8SC2, CV_8SC3, CV_8SC4**

Unsigned 16bits ushort 0~65535
Mat:**CV_16UC1, CV_16UC2, CV_16UC3, CV_16UC4**

Signed 16bits short -32768~32767
Mat: **CV_16SC1, CV_16SC2, CV_16SC3, CV_16SC4**

Signed 32bits int -2147483648~2147483647
Mat: **CV_32SC1, CV_32SC2, CV_32SC3, CV_32SC4**

Float 32bits float -1.18*10-38~3.40*10-38
Mat: **CV_32FC1, CV_32FC2, CV_32FC3, CV_32FC4**

Double 64bits double
Mat: **CV_64FC1, CV_64FC2, CV_64FC3, CV_64FC4**

Gdzie: **C** - oznacza liczbę kanałów obrazu

5. Histogram

Histogram jest to wykres liczby pikseli z obrazu uzależnionej od wartości, przyporządkowanej w procesie kwantyzacji obrazu. Ta wartość może być podzialona na większe grupy, które stanowią przedziały wartości pikseli.
W bibliotece OpenCV do obliczenia histogramu służy funkcja:

cv2.calcHist([obraz], [kanały], maska, rozmiar, zakresy[histogram[, akumulacja]])

images : it is the source image of type uint8 or float32. it should be given in square brackets, ie, "[img]".

1. channels : it is also given in square brackets. It is the index of channel for which we calculate histogram. For example, if input is grayscale image, its value is [0]. For color image, you can pass [0], [1] or [2] to calculate histogram of blue, green or red channel respectively.
2. mask : mask image. To find histogram of full image, it is given as "None". But if you want to find histogram of particular region of image, you have to create a mask image for that and give it as mask. (I will show an example later.)
3. histSize : this represents our BIN count. Need to be given in square brackets. For full scale, we pass [256].
4. ranges : this is our RANGE. Normally, it is [0,256].

Przykład:

```
img = cv.imread('home.jpg',0)
hist = cv.calcHist([img],[0],None,[256],[0,256])
```

Przykład prezentacji histogramu za pomocą biblioteki matplotlib:

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt
img = cv.imread('home.jpg',0)
plt.hist(img.ravel(),256,[0,256]); plt.show()
```

6. Tworzenie rysunku i opis

Często zachodzi potrzeba narysowania w obrębie okna obrazu pewnych elementów, które są na przykład wynikiem przeprowadzonych obliczeń. Żeby zaznaczyć te elementy, można posłużyć się funkcjami dostępnymi w ramach biblioteki OpenCV.
Przykłady funkcji:

linia:	**cv2.line (obraz, (ip,jp), (ik,jk), (B,G,R), grubość)**
prostokąt:	**cv2.rectangle (obraz, (ip,jp), (ik,jk), (B,G,R), grubość)**
koło lub okrąg:	**cv2.circle (obraz, (isr,jsr), r, (B,G,R), grubość)**

gdzie:

ip, jp - oznaczają położenie początku
ik, jk - oznaczają położenie końca figury
isr, jsr - oznaczają położenie środka figury
r - promień okręgu
B, G, R - oznaczają składowe barwy (0..255)
Grubość o wartości -1 oznacza, że figura zamknięta ma być wypełniona.

Opis elementów rysunku może być standardowo wykonany za pomocą następujących, oznaczonych liczbami, krojów pisma:

FONT_HERSHEY_SIMPLEX = 0,
FONT_HERSHEY_PLAIN = 1,
FONT_HERSHEY_DUPLEX = 2,
FONT_HERSHEY_COMPLEX = 3,
FONT_HERSHEY_TRIPLEX = 4,
FONT_HERSHEY_COMPLEX_SMALL = 5,
FONT_HERSHEY_SCRIPT_SIMPLEX = 6,
FONT_HERSHEY_SCRIPT_COMPLEX = 7,
FONT_ITALIC = 16

Do opisu służy funkcja:

cv2.putText(obraz, tekst, (ip,jp), krój pisma, rozmiar, (R,G,B), grubość, styl linii)

gdzie:

ip, jp - oznaczają położenie początku (lewego dolnego narożnika)
rozmiar - oznacza wielkość znaków
styl linii - czy ma być wygładzona

predefiniowane stałe dla linii:
FILLED = -1 - wypełniona
LINE_4 = 4
LINE_8 = 8
LINE_AA = 16 - wygładzona

Przykład:

```
>>> import numpy as np
>>> import cv2
```

```
>>> obr = np.zeros((900,900,3), np.uint8) #założenie czarnego obrazu o podanych parametrach
>>> cv2.line(obr,(50,50),(300,100),(0,0,255),6) #narysowanie czerwonej linii
array([[[0, 0, 0],
...
        [0, 0, 0]]], dtype=uint8)
>>> cv2.imshow('obraz',obr) #wyświetlenie okna
>>>       cv2.circle(obr,(200,300),50,(255,0,0),-1)      #dorysowanie      niebieskiego      koła (wypełnionego)
array([[[0, 0, 0],
...
        [0, 0, 0]]], dtype=uint8)
>>> cv2.imshow('obraz',obr) #ponowne wyświetlenie okna
>>> cv2.putText(obr, 'Tekst', (300,600), 3, 12, (50,250,50), 2, cv2.LINE_AA) #umieszczenie tekstu
array([[[0, 0, 0],
...
        [0, 0, 0]]], dtype=uint8)
>>> cv2.imshow('obraz',obr) #ponowne wyświetlenie okna
```