Instrukcja wprowadzająca do pracy z obrazem

Słowem wstępu

Zadania w tej instrukcji są obowiązkowe do wykonania, ale nie są oceniane. Jeżeli ktoś ich nie wykona, nie będzie miał ocenianych dalszych zadań. Są to proste zadania, mające na celu, przygotowanie środowiska do pracy z materiałami oraz dania wam umiejętności potrzebnych do pracy na zajęciach. Część kodu będzie wykorzystywana na dalszych zajęciach, więc proszę ich nie usuwać po ukończeniu.

Termin wykonania: następne zajęcia.

Podstawowe informacje dotyczące pracy z obrazem

Przed rozpoczęciem pracy proszę pobrać archiwum z plikami wykorzystywanymi podczas dzisiejszych zajęć.

PLIKI

Podstawowe operacje na obrazach

Do tego zadania dołączone są obrazy służące przetestowaniu różnych funkcji. Zaczynamy od stworzenia nowego skryptu. Nazwijmy go np.: *Lab1_img.py*. Każdy ze skryptów będziemy zaczynali od zaimportowania odpowiednich bibliotek. W tym przypadku będą to:

```
01. import numpy as np
02. import matplotlib.pyplot as plt
03. import cv2
```

Część pierwsza — wczytywanie obrazów

Zacznijmy od wczy

```
01. img1 = plt.imread('nazwa_pliku.roz')
```

Teraz przeanalizujmy, co właściwie zawiera wczytany przez nas plik. Za pomocą poniższych poleceń wyświetlimy sobie typ danych oraz rozmiar naszego obrazu.

```
01. print(img1.dtype)
02. print(img1.shape)
03. print(np.min(img1),np.max(img1))
```

Teraz należy odpowiedzieć sobie na pytania, wczytując wszystkie pliki z kategorii A (*A1*,*A2*,...):

- Co oznacza nasz typ zmiennych? Jaki zakres wartości możemy spotkać?
- Jaki jest rozmiar naszego obrazu? Co oznacza ostatni parametr?
- Czy dla obu plików (*png*, *jpg*) otrzymujemy te same odpowiedzi?

Alternatywnie możemy również wykorzystać w tym celu *OpenCV*.

```
01. img2 = cv2.imread('nazwa_pliku.roz')
```

Jak wyglądają odpowiedzi na te pytania dla obrazu po otwarciu funkcją z OpenCV?

Zadanie Obraz 1

Jak już zauważyliście różne rozszerzenia plików i funkcje wczytywania dostarczają nam dane w różnych typach, zawierające informacje o różnej rozdzielczości bitowej. Rozdzielczość bitowa określa dokładność, z jaką zapisana jest jedna próbka danych, określając ilość dostępnych dla niej bitów. Większość obrazów, na jakich pracujemy jest zwykle zapisana w formacie *uint8*, czyli jako 8-bitowa liczba całkowita bez znaku, więc zawierająca wartości pikseli o wartościach całkowitych z zakresu $< 0, 255 >$. Możemy spotkać inne formaty *int* (16,32 lub 64 -bitowe) lub *float* o wartościach z przedziału $< 0, 1 >$. W zależności od zadania łatwiej będzie obsługiwać jeden typ danych *uint8* lub *float*. Celem zadania będzie napisanie dwóch funkcji, które przeskalują wam wartości na jeden z tych typów danych.

```
01. def imgToUInt8(img):
02.     pass
03.
04. def imgToFloat(img):
05.     pass
```


Obie funkcje na początku powinny sprawdzać czy na wejściu jest typ inny, od którego tego oczekujemy na wyjściu,

czyli czy gdy na wyjściu oczekujemy *float* to na wejściu również jest *uint8*. Można to zrobić, sprawdzając, czy tym naszych danych pochodzi z odpowiedniej rodziny:

```
01. np.issubdtype(data.dtype,np.integer)
02. np.issubdtype(data.dtype,np.unsignedinteger)
03. np.issubdtype(data.dtype,np.floating)
```

Jeżeli nasze typy są zgodne, to zwracamy wejście bez modyfikacji. Jeżeli nie są zgodne to, należy przeliczyć obraz do nowego zakresu:

- Do przejścia z *uint8* na *float* cały obraz należy podzielić przez *255.0* (nie potrzeba pętli pakiet NumPy wykona dla was operację *img/255.0* na całym obrazie),
- Do przejścia z *float* na *uint8* potrzeba już trochę, więcej kroków. Na początek należy pomnożyć cały obraz przez 255 (również bez pętli), a następnie zrzutować cały obraz przy wykorzystaniu komendy *.astype('uint8')*

Część druga — wyświetlanie obrazów

Teraz zajmiemy się wyświetleniem obrazu. **Na razie wykorzystajmy w tym celu obraz wczytany za pomocą *plt*.** Żeby wyświetlić obraz kolorowy, wystarczy wykorzystać komendę:

```
01. plt.imshow(img)
02. plt.show()
```

Teraz spróbujmy przekształcić obraz do skali odcieni szarości. Mamy 2 sposoby, w jakie możemy przekształcić obraz kolorowy w obraz w skali odcieni szarości:

1. Najprostsza i najmniej dokładna polega po prostu na wzięciu jednej warstwy obrazu, dlatego teraz: Wyświetlcie na ekranie jedną warstwę koloru — dowolną czerwoną, zielona lub niebieską (przykład czerwony).

   ```
   01. R=img[:,:,0]
   02. plt.imshow(R)
   03. plt.show()
   ```

   Jak widać, nie wyświetla się nam on w sposób poprawny. W jaki sposób możemy to poprawić? Tłumacząc komputerowi, że chcemy wyświetlić nasz obraz właśnie w skali odcieni szarości. Służy do tego komenda:

   ```
   01. plt.imshow(R, cmap=plt.cm.gray, vmin=??, vmax=??)
   ```

Kontakt

Materiały

Przetwarzanie Obrazów

Systemy Multimedialne

Interaktywne Systemy Multimedialne

Przetwarzanie i analiza danych

FAQ - najczęściej zadawane pytania

Parametry *vmin* i *vmax* służą skalowaniu wartości obrazu. W domyślnym przypadku dla pełnego zakresu danych można je pominąć, ale może to utrudniać prawidłową ocenę tego, co wyświetlamy. Teraz pytanie: Jakie wartości powinny przyjąć w zależności od naszych typów danych?

2. Teraz przechodzimy do dokładniejszej metody zamiany obrazu kolorowego na obraz w skali odcieni szarości. Są na to dwa sposoby i oba wymagają przeliczenia wartości dla każdego piksela według wzorów poniższych, więcej na ten temat można przeczytać na Wikipedii.
    1. $Y_1 = 0.299 * R + 0.587 * G + 0.114 * B$
    2. $Y_2 = 0.2126 * R + 0.7152 * G + 0.0722 * B$

    Teraz bardzo ważne pytanie: W jaki sposób zrobić to bez użycia ani jednej pętli?

    Ponieważ Python obsługuje operacje macierzowe bardzo łatwo (przy założeniu, że mamy już obraz rozłożony na warstwy *R,G,B*) przy użyciu poniższej linijki kodu:

```
01. Y2=0.2126 * R + 0.7152 * G + 0.0722 * B
```

Sprawdźmy teraz jak wygląda wyświetlony obraz wczytany za pomocą *OpenCV*, w ten sam sposób jak na początku. Po wykonaniu tej operacji spróbujmy odpowiedzieć na poniższe pytania:

- Co nam się wyświetliło?
- Co jest nie tak?

Powodem tego jest sposób, w jaki *OpenCV* wczytuje obrazy. Domyślenie w tym formacie warstwa niebieska *B* jest pierwsza, a wartości *R* umieszczane są w ostatniej warstwie, czyli spotykamy się tutaj z formatem *BRG*, a nie *RGB*. Można to rozwiązać ręcznie, zamieniając warstwy samemu lub poprzez wykorzystanie funkcji z pakietu *OpenCV*:

```
01. img_RGB = cv2.cvtColor(img_BGR, cv2.COLOR_BGR2RGB)
02. img_BGR = cv2.cvtColor(img_RGB, cv2.COLOR_RGB2BGR)
```

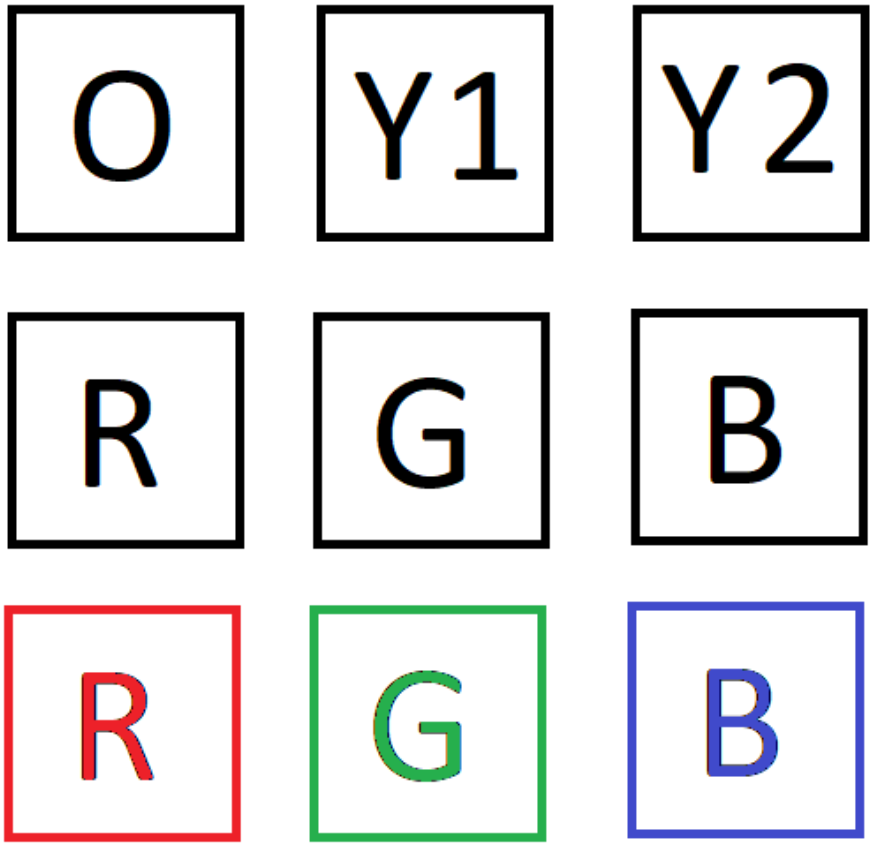
Te funkcje można również wykorzystać w późniejszych zajęciach do przekształcania obrazów kolorowych na obrazy w skali odcieni szarości wykorzystując w tym celu flagi *cv2.COLOR_RGB2GRAY* lub *cv2.COLOR_BGR2GRAY* w zależności od wykorzystywanej do wczytywania funkcji.

Część trzecia — wstęp przejścia do przechodzenia pomiędzy typami danych

Jeżeli wykonaliście poprzednie kroki, to powinniście wiedzieć, że w przypadku obrazu będziemy spotykali się z dwoma typami danych.

Zadanie Obraz 2

Zadaniem jest (przy wykorzystaniu wszystkich informacji podanych już w tej części instrukcji) napisanie funkcji wyświetlającej na jednym ekranie obrazu kolorowego z klasy B (*B1*,*B2*) w sposób zaprezentowany poniżej:



Wizualizacja zadania

W pierwszym rzędzie powinny po kolei wyświetlić się: *O* - kolorowy obraz oryginalny, *Y1* - obraz w skali odcieni przeliczony przy użyciu pierwszego wzoru oraz *Y2* - obraz w skali odcieni przeliczony przy użyciu drugiego wzoru. W drugim rzędzie wyświetlamy poszczególne warstwy obrazu (*R*,*G*,*B* - czerwona, zielona i niebieska). Ostatni wiersz powinien zawierać obrazy kolorowe, w których wartości pikseli we wszystkich warstwach poza jedną (*R*,*G*,*B* - czerwoną, zieloną lub niebieską) zostały ustawione na *0*. Należy to wykonać bez użycia żadnych pętli tak jak w

przypadku wyliczania wartości obrazu w skali odcieni szarości. **UWAGA** W Pythonie większość zmiennych przechowuje referencję do obiektu i operacja przypisania nie tworzy kopii danych tylko kopię referencji. Jeżeli chcemy wykonać kopię obiektu należy wykorzystać funkcję `.copy()`:

```
01. kopia = orginal.copy()
```

Do wyświetlenia wszystkiego w jednym oknie wykorzystujemy funkcję:

```
01. plt.subplot(w, k, n)
```

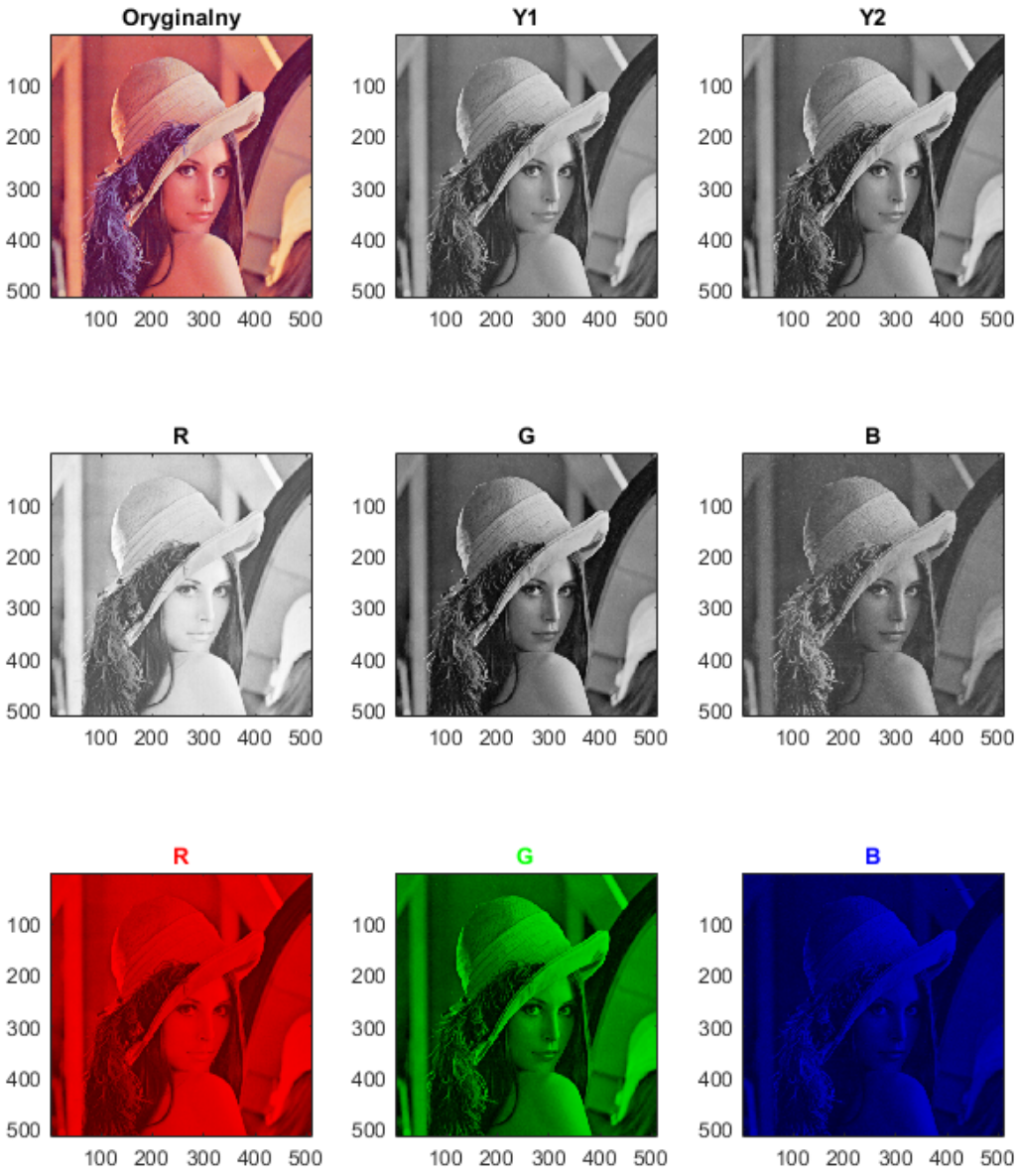
Wywołujemy ją przed wywołaniem funkcji `.imshow()`. Sama funkcja jako parametry przyjmuje: `w` - ilość wierszy, `k` - ilość kolumn, `n` - indeks elementu. Więcej informacji na ten temat można znaleźć w dokumentacji.

Uwaga: Jeżeli nie chcecie, żeby wszystkie wykresy wyświetlały się wewnątrz Spyder, a w osobnym oknie należy to zmienić w opcjach programu lub wykorzystać do tego komendę:

```
01. %matplotlib qt
```

Proces odwrotny realizujemy również komendą:

```
01. %matplotlib inline
```



Kontakt

Materiały

Przetwarzanie Obrazów

Systemy Multimedialne

Interaktywne Systemy Multimedialne

Przetwarzanie i analiza danych

FAQ - najczęściej zadawane pytania

Zadanie Obraz 3

W tym zadaniu wykorzystujemy funkcję, która powstała w Zadaniu 2 (rysującą cały *subplot* z 9 elementami, proszę nic nie usuwać). Ze zdjęć z kategorii *B*, mamy wybrać ***fragmenty*** zdjęcia o rozmiarze *200x200 px* zawierające różne fragmenty tęczy i wyświetlić go w ten sam sposób co poprzednio, a następnie zapisać automatycznie, każdy z nich. Do wycięcia fragmentu można wykorzystać poniższy kod zastępując zmienne *w1*, *k1*, *w2* oraz *k2* odpowiednimi wybranymi przez was współrzędnymi.

```
01. fragment = img[w1:w2,k1:k2].copy()
```

Do zapisania wyników do pliku można również wykorzystać funkcję *plt.savefig*. Funkcja zapisuje bieżące aktywne okno do pliku, warto zmodyfikować nazwę pliku, żeby nie nadpisywać poprzednich plików. Dodatkowe parametry mogą przydać wam się na dalszych zajęciach, żeby na przykład zmniejszyć marginesy.

```
01. plt.savefig("nazwa{}".format(parametr))
```

My jednak wykorzystamy fragmenty kodu z poprzednich zajęć (zadanie o automatycznym zapisywaniu do pliku *.docx*). Przebudujemy funkcję w taki sposób, żeby generowała wykresy dla odpowiednich fragmentów. Więcej informacji jak to wykorzystać znajdziecie również w dziale FAQ.

Do przetwarzania wycinków należy wykorzystać pętlę *for*, a informacje o punktach zapisać w tablicy może być to zarówno tablica tablic Pythona, jak i dwuwymiarowa tablica *numpy*.

Na początek musimy zadeklarować jakąś strukturę danych do przechowywania informacji o naszych plikach. Tu macie pewną dowolność, Ja dla przykładu wykorzystałem strukturę *DataFrame* z pakietu *Pandas*. Przechowuje ona informacje na temat pliku, czy obraz należy skonwertować do skali odcieni szarości oraz listę fragmentów (w formacie *[w1,k1,w2,k2]* - współrzędne pierwszego i ostatniego wierzchołka), jakie chcemy analizować.

```
01. import pandas as pd
02. import cv2
03. import numpy as np
04. import matplotlib.pyplot as plt
05.
06. df = pd.DataFrame()
07.
08. df = pd.DataFrame(data={'Filename':[ file.png'],'Grayscale':[False],
```

Kontakt

Materiały

Przetwarzanie Obrazów

Systemy Multimedialne

Interaktywne Systemy Multimedialne

Przetwarzanie i analiza danych

FAQ - najczęściej zadawane pytania

```
                        'Fragments':[[[20,20,60,60],[20,20,60,60]]]
                        })

print(df)
```

W jaki sposób przetworzyć taką listę? Poniżej przykład:

```
for index, row in df.iterrows():
    img = plt.imread(row['Filename'])
    if row['Grayscale']:
        # GS image - teraz nas nie intersuje
    else:
        # Obraz kolowowy
    if row['Fragments'] is not None:
        # mamy nie pustą listę fragmentów
        for f in row['Fragments']:
            fragment = img[f[0]:f[2],f[1]:f[3]].copy()
            # tu wykonujesz operacje i inne wyświetlenia na fragmencie
```

Jeżeli funkcja działa Wam dla pojedynczego pliku i fragmentu, można wtedy dodać kolejne informacje (pliki, fragmenty). Tylko w późniejszych instrukcjach, zanim odpalicie cały zbiór testowy, upewnijcie się, że wyniki działają, chociaż dla kilku plików (czy wyglądają poprawnie/podobnie do instrukcji). Jeżeli algorytm pomyślnie zadziałał dla podstawowych danych to, dla pozostałych również powinien działać. Z tej części zadania proszę też wygenerować plik *PDF*.

Źródła danych dołączonych do instrukcji (nie pobierać)

- https://pixy.org/518540/
- https://pixnio.com/media/rainbow-sun-atmosphere-weather-clouds