6_Implementacja_Konwolucji

January 27, 2019

1 Implementacja operacji konwolucji w Pythonie

Czasem zdarza się, że chcemy sami stworzyć jakiś filtr by analizować obraz – na przykład podczas wykorzystywania obrazów w uczeniu maszynowym. Wtedy warto wiedzieć, czym jest konwolucja i jak można ją zaimplementować w łatwy sposób. Poniżej pokażemy Ci przykładowe wywołania. A czym jest **konwolucja**? Najprościej mówiąc, jest to złożenie dwóch funkcji, w nową, trzecią funkcję. Matematycznych podstaw nie będziemy tu prezentować, polecamy za to gorąco artykuł pod tym adresem: https://ksopyla.com/python/operacja-splotu-przetwarzanie-obrazow/. W przetwarzaniu obrazów cyfrowych, konwolucję można rozumieć jako używanie wartości z jednej funkcji, jako maski filtrującej, która przesuwa się następnie po całym obrazie.

Tym razem będziemy potrzebować większego zestawu bibliotek – jeśli nie pamiętasz jak się je instaluje, wróć do wstępu tego poradnika.

In [2]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import numba as nb
from numba import cuda
from scipy.ndimage.filters import convolve
import cv2
```

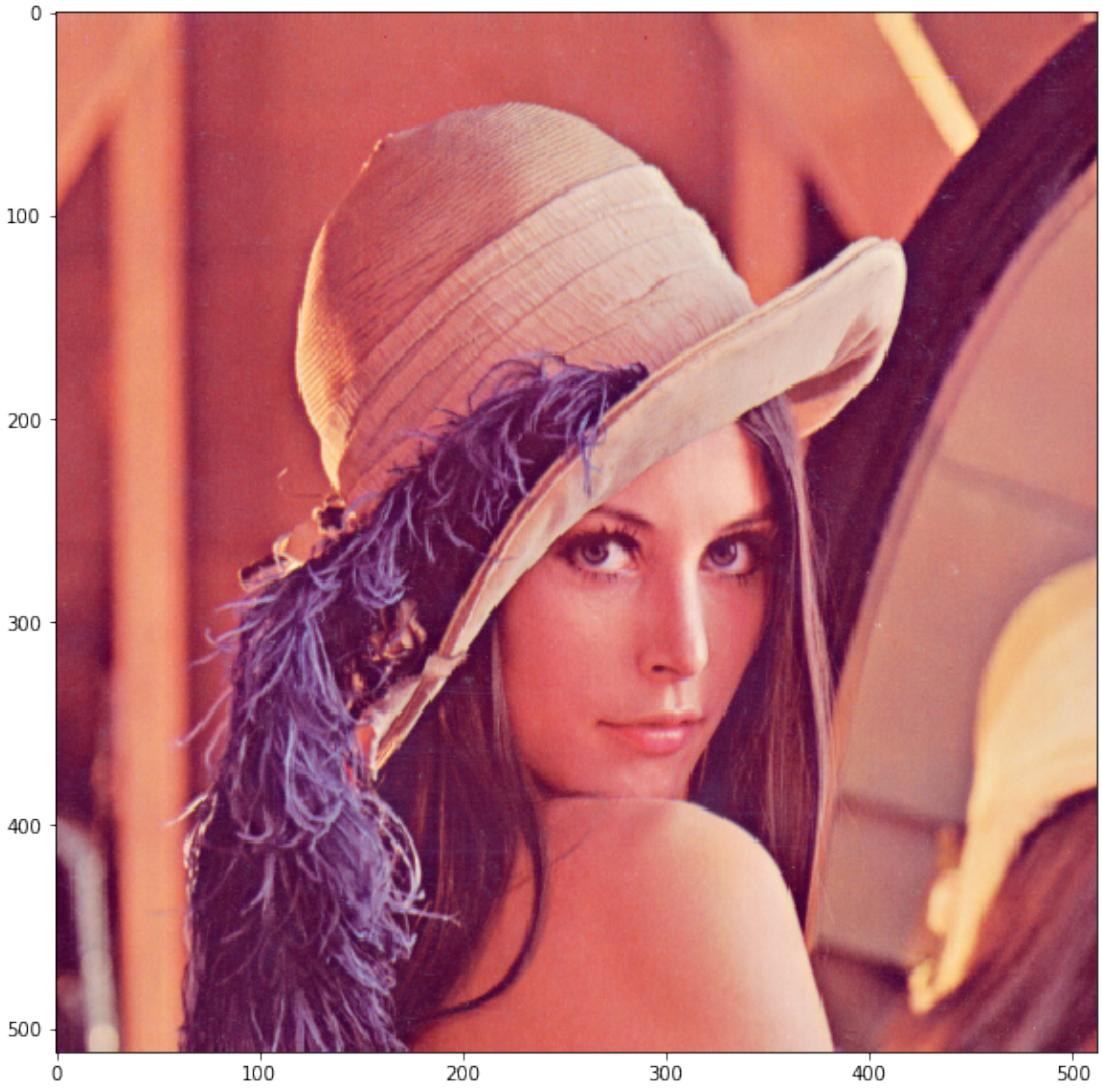
```
---------------------------------------------------------------------------

ModuleNotFoundError                       Traceback (most recent call last)

<ipython-input-2-7ec0cc26164c> in <module>
      2 import matplotlib.pyplot as plt
      3 import numpy as np
----> 4 import numba as nb
      5 from numba import cuda
      6 from scipy.ndimage.filters import convolve


ModuleNotFoundError: No module named 'numba'
```

In [4]:
```
img_pil = Image.open('../obrazy_testowe/lena_512x512.png')
img_pil
```

Out[4]:

In [5]:
```
img_np = np.array(img_pil)/255
```

In [4]:
```
plt.figure(figsize=(10,10))
plt.imshow(img_np)
plt.show()
img_np.shape
```




```
Out[4]: (512, 512, 3)
```

1.1 Filtr średniej 3x3

Pokażemy Ci działanie bardzo prostej maski, która jest po prostu średnią wartością z maski 3x3. Pamiętaj, że przy bardziej skomplikowanych przekształceniach, czas wykonywania obliczeń także się wydłuży, dlatego warto wiedzieć, jak zaoszczędzić sobie trochę czasu ;)

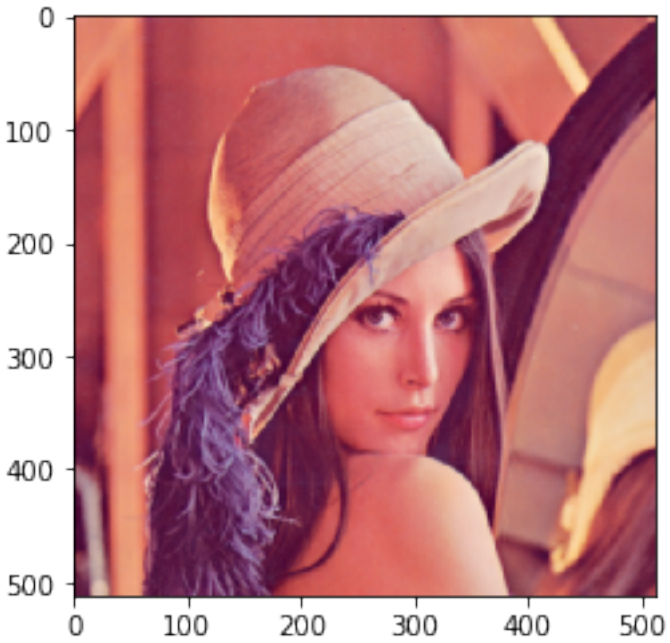
```
In [1]: # tworzymy maskę
        mean_3_3 = np.array([[1, 1, 1],
                             [1, 1, 1],
                             [1, 1, 1],],
                            dtype='float')/9
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-1-622e4f183bdc> in <module>
  1 # tworzymy maskę
----> 2 mean_3_3 = np.array([[1, 1, 1],
  3                  [1, 1, 1],
  4                  [1, 1, 1],],
  5                dtype='float')/9


NameError: name 'np' is not defined
```

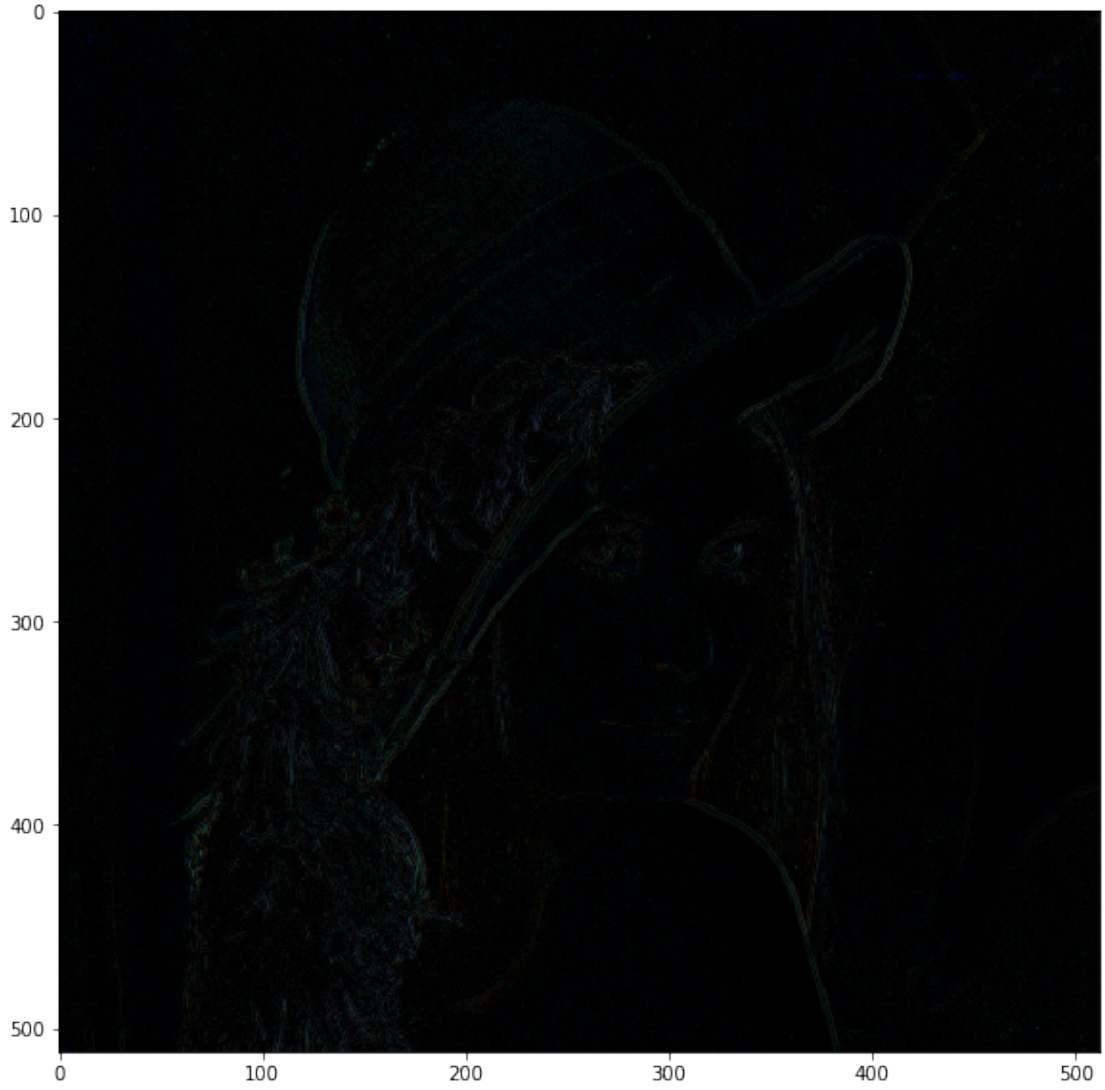
- **funkcja wbudowana w bibliotekę *scipy***

```
In [15]: %%timeit
         img_mean = np.stack([convolve(np.squeeze(img_np[...,i]),mean_3_3) for i in range(3)],ax

11.7 ms ± 51.5 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
In [18]: img_mean = np.stack([convolve(np.squeeze(img_np[...,i]),mean_3_3) for i in range(3)],ax
         plt.imshow(img_mean)
         plt.show()

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] f
```



```
In [19]: plt.figure(figsize=(10,10))
         plt.imshow(np.abs(img_mean-img_np))
         plt.show()
```



- **tym razem funkcja z *OpenCV***

```
In [17]: %timeit cv2.filter2D(img_np,-1,mean_3_3)

4.24 ms ± 58.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

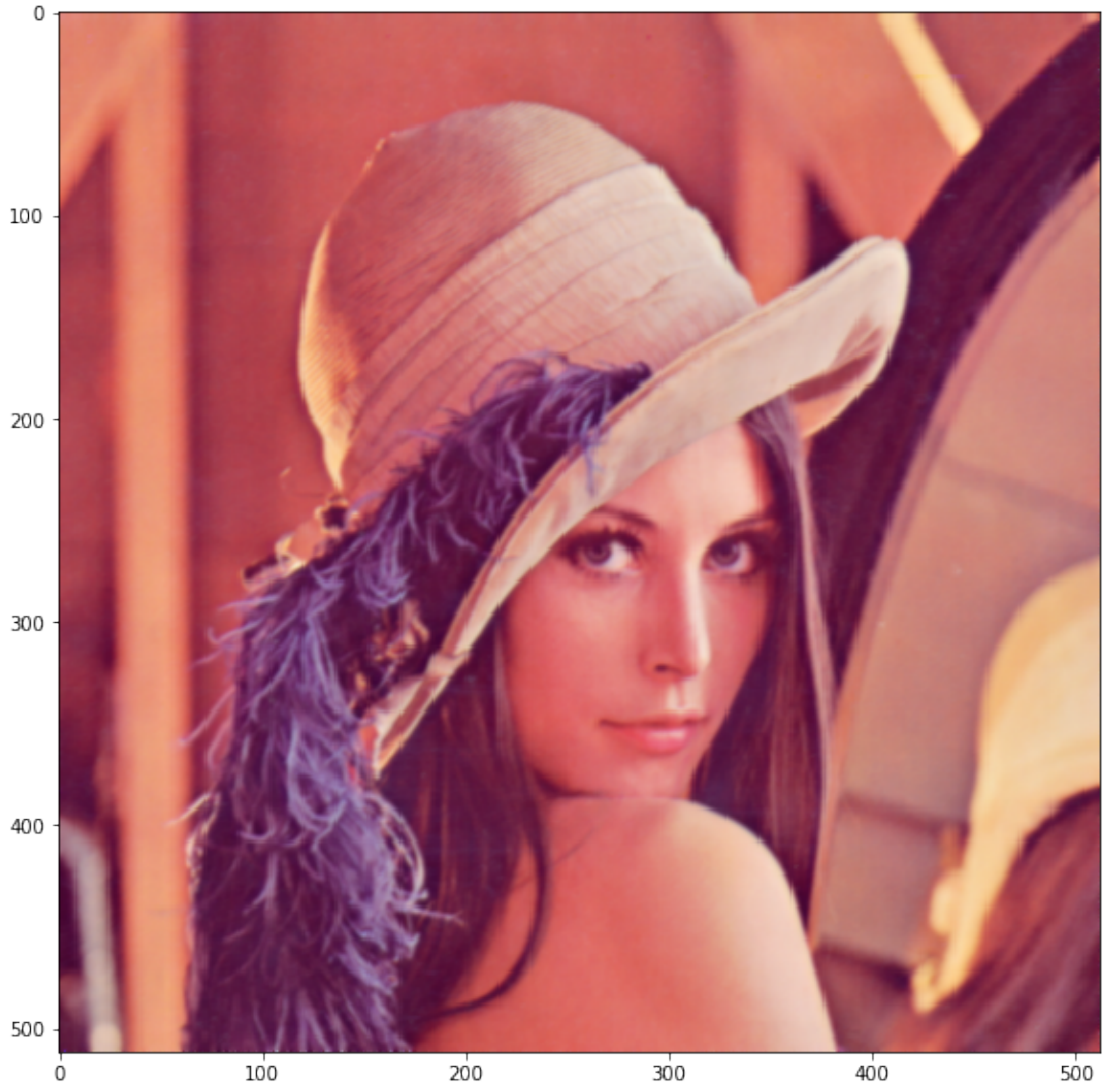
- **jak zrobić to własnoręcznie z użyciem pętli**

```
In [20]: %%timeit
         img_mean_fl = img_np.copy()
         for i in range(1,img_mean_fl.shape[0]-1):
             for j in range(1,img_mean_fl.shape[1]-1):
                 for k in range(3):
                     val = 0
                     for m in range(-1,2):
                         for n in range(-1,2):
                             val+= img_mean_fl[i+m,j+n,k]*mean_3_3[m+1,n+1]
                     img_mean_fl[i,j,k] = val
```

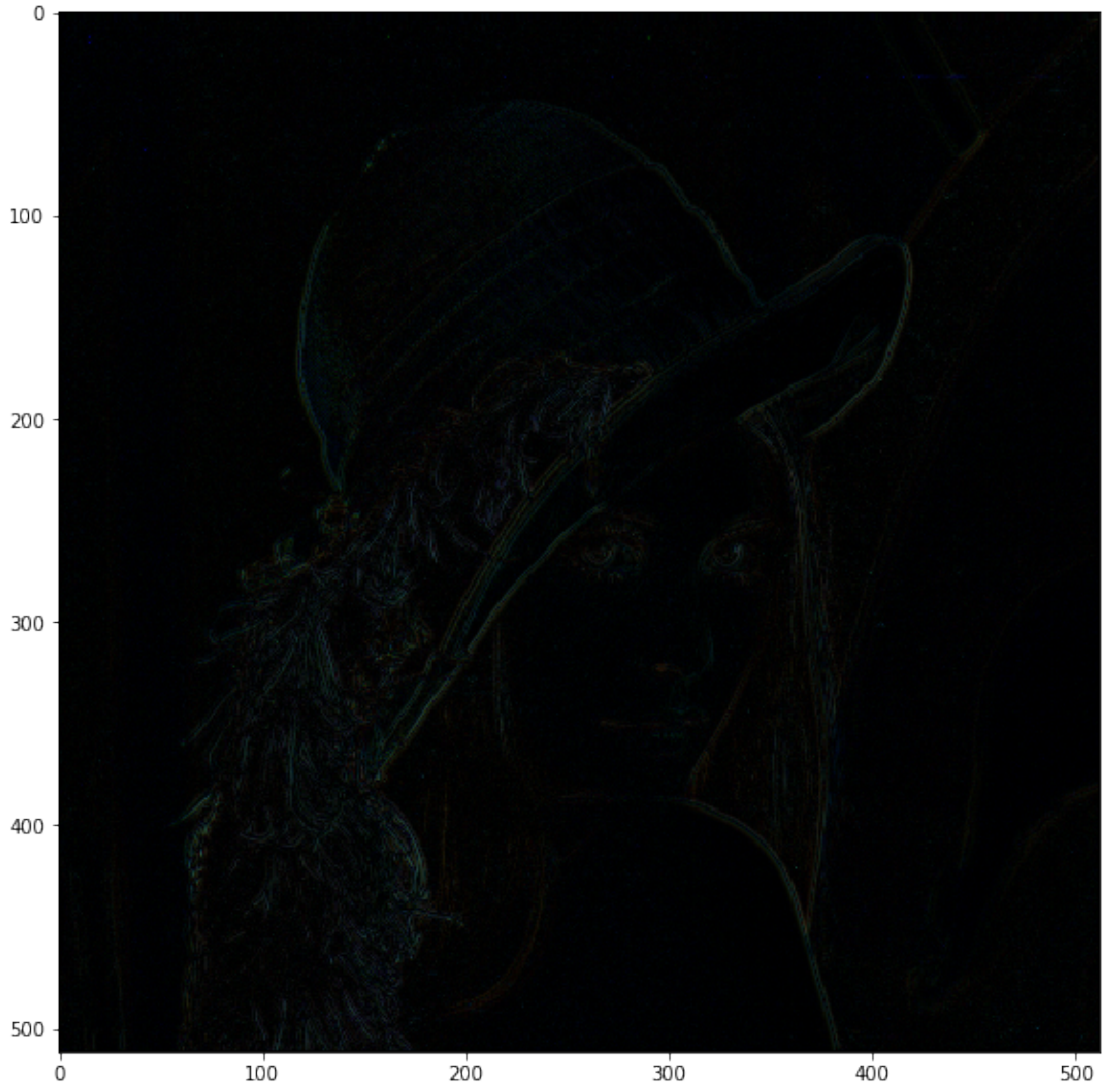
```
5.08 s ± 234 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
In [22]: img_mean_fl = img_np.copy()
         for i in range(1,img_mean_fl.shape[0]-1):
             for j in range(1,img_mean_fl.shape[1]-1):
                 for k in range(3):
                     val = 0
                     for m in range(-1,2):
                         for n in range(-1,2):
                             val+= img_mean_fl[i+m,j+n,k]*mean_3_3[m+1,n+1]
                     img_mean_fl[i,j,k] = val
```

```
In [23]: plt.figure(figsize=(10,10))
         plt.imshow(img_mean_fl)
         plt.show()
```



In [24]:
```
plt.figure(figsize=(10,10))
plt.imshow(np.abs(img_mean_fl-img_np))
plt.show()
```



- **i jak to wygląda po przyspieszeniu z użyciem pakietu *numba***

```
In [32]: img_mean_fl = img_np.copy()
         @nb.jit()
         def wolna_funkcja_duzo_petli(img_mean_fl):
             for i in range(1,img_mean_fl.shape[0]-1):
                 for j in range(1,img_mean_fl.shape[1]-1):
                     for k in range(3):
                         val = 0
                         for m in range(-1,2):
                             for n in range(-1,2):
                                 val+= img_mean_fl[i+m,j+n,k]*mean_3_3[m+1,n+1]
                         img_mean_fl[i,j,k] = val
             return img_mean_fl
```

```
In [33]: %time img_mean_fl = wolna_funkcja_duzo_petli(img_mean_fl)

Wall time: 291 ms
```