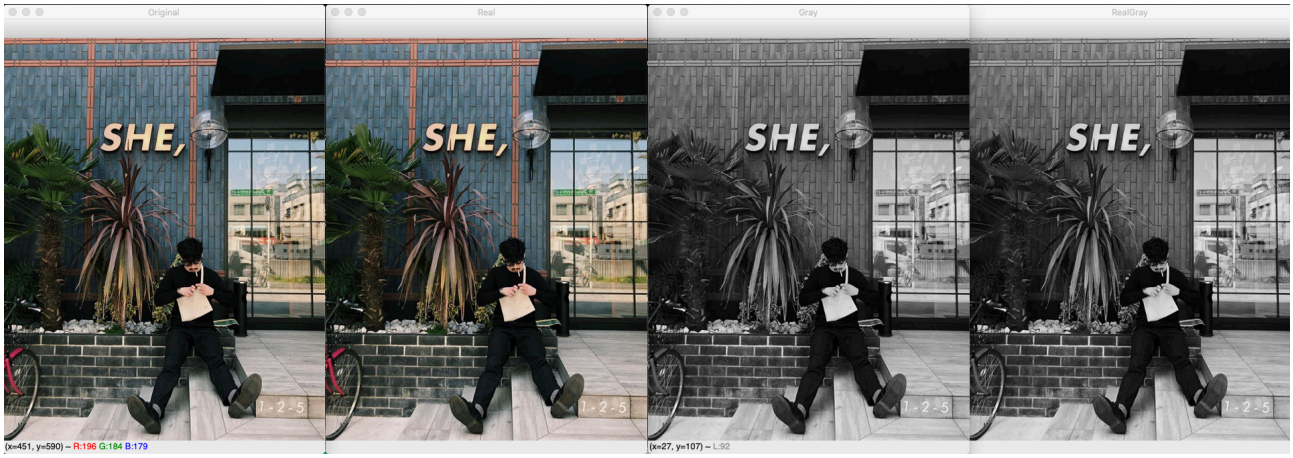
レポート1
08-172022
永田 諒

grayscale.py



左から
Original, Real, Gray, Real Grayである。

出力

```
original size = 864000
(height, width, channels) = (600, 480, 3)
channel depth = uint8

real number size = 864000
(height, width, channels) = (600, 480, 3)
channel depth = float64

grayscale size = 288000
(height, width, channels) = (600, 480)
channel depth =  uint8

real number size = 288000
(height, width, channels) = (600, 480)
channel depth = float64
```

コード

```python
#!/usr/bin/env python
# coding: utf-8

import sys
import cv2
import numpy as np

def readImage():
    """
    read an image file and return its data
    """
    if len(sys.argv) > 1:
        fname = sys.argv[1]
    else:
        fname = input('image file -> ')
    image = cv2.imread(fname)
    if image is None:
        print('no image file:', fname)
        sys.exit(1)
    return image

def main():
    img = readImage()
    print('original size =', img.size)
    print('(height, width, channels) =', img.shape)
    print('channel depth =', img.dtype)
    cv2.imwrite('image.jpg', img)
    wname = 'Original'
    #
    cv2.imshow(wname, img)

    real = img.astype(np.float64) / 255.0
    print('\nreal number size =', real.size)
    print('(height, width, channels) =', real.shape)
    print('channel depth =', real.dtype)
    cv2.imwrite('real.jpg', real)
    wname = 'Real'
```

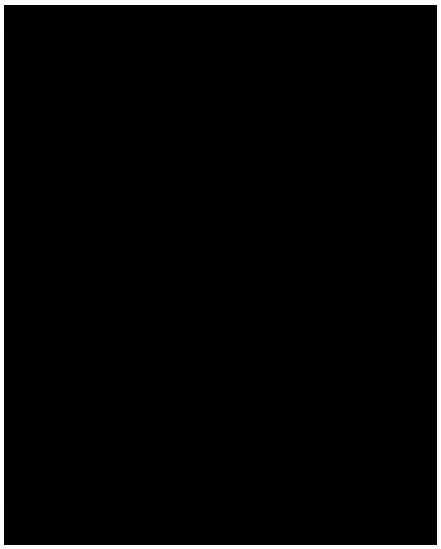
```
    cv2.imshow(wname, real)

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    print('\ngrayscale size =', gray.size)
    print('(height, width, channels) =', gray.shape)
    print('channel depth = ', gray.dtype)
    cv2.imwrite('gray.jpg', gray)
    wname = 'Gray'
    cv2.imshow(wname, gray)

    rGray = gray.astype(np.float64) / 255.0
    print('\nreal number size =', rGray.size)
    print('(height, width, channels) =', rGray.shape)
    print('channel depth =', rGray.dtype)
    cv2.imwrite('realGray.jpg', rGray)
    wname = 'RealGray'
    cv2.imshow(wname, rGray)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == '__main__':
    main()
```

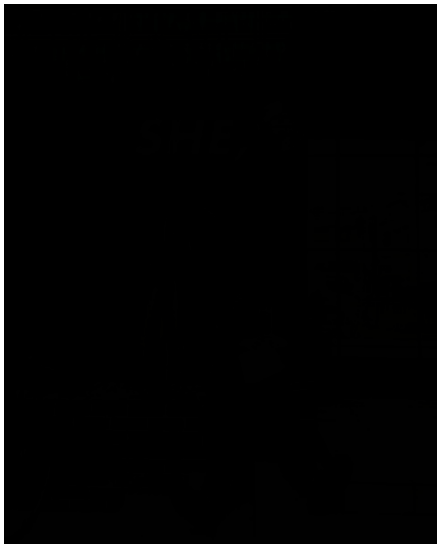
考察：
sysモジュールは標準入出力を管理してくれているモジュールで、cv2はOpenCVのPythonラッパーである。

cv2.imreadではnumpy.ndarray型の画像データを返す。
この配列のshapeはchannelで示されるように、
カラー画像では600x480x3の3次元配列。グレースケールの画像では、600x480の二次元配列。
ここからわかるように、カラー画像での3つの要素を持つ配列と、二次元配列での各要素はピクセルに対応しており、それぞれのピクセルの色を指定している。

サイズはカラー画像とグレースケール画像で比較すると、前者は後者の３倍になっている。この理由は3倍の表現力を持ったカラーコードを保持しているためである。

保存された画像は以下





左上から
(original, real
gray scale, real gray)

カラーコードを実数化したとき、保存された画像は表示されなくなった。
これは、実数として保存された値を、整数として解釈しているために真っ黒になってしまっているのだと考えられる。

binary.py

コード：

```
#!/usr/bin/env python
# coding: utf-8

# In[3]:


import cv2
from grayscale import readImage


# In[4]:


wname = 'Binary'
trackbar = 'Threshold'


# In[5]:


def onChange(val):
    """
    val - changed trackbar value
    binalize the image (gray) with the threshold value
    """
```

```
    global gray
    white = 255
    ret, binary = cv2.threshold(gray, val, white,
cv2.THRESH_BINARY)

    cv2.imshow(wname, binary)


# In[6]:


def main():
    global gray
    img = readImage()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    cv2.imshow('Original', gray)
    trackHalf = 127
    trackMax = trackHalf * 2
    cv2.namedWindow(wname)
    cv2.createTrackbar(trackbar, wname, 0, trackMax, onChange)

    cv2.setTrackbarPos(trackbar, wname, trackHalf)
    onChange(trackHalf)
    cv2.waitKey(0)
    cv2.destroyAllWindows()


# In[7]:


if __name__ == '__main__':
    main()


# In[ ]:
```

考察：
今回行った処理では、色の値に閾値を設けることで2値に分類している。
閾値の設定はGUIから実行後に変更することができるようになっている。

onChange関数をcallbackのような形でmain関数の中でcv2のインスタンスに渡すことで、イベント発生時（トラックバーのドラッグ時）に閾値の変更と画像の再描画を行なっている。

今回以降の例題を含めて、コードに数多く、global変数が出てきたが、
変数の変化が追いずらく、自分で書く場合には可能な限り避けるべきだと感じた。

今までjavaとtypescriptを書いていたので、静的型付でない言語は久しぶりであるが、jupyter notebookのように真に対話形式でコーディングを行うことができるためいろんな試行錯誤が高速でできるのでありがたいと思った。

今回の課題のコードに関しても、全てコマンドライン形式での実行になっているが、より多くのPDCAを高速に回すために、Jupiter Notebookでできたらいいのにと感じてしまった。