Tên: Nguyễn Hoàng Minh
Mssv: 18127265
Lớp: 18CLC1
Môn: Toán ứng dụng và thống kê

Project 2: IMAGE PROCESSING

Các chức năng đã hoàn thành:

- Thay đổi độ sáng
- Thay đổi độ tương phản
- Chuyển RGB thành ảnh xám
- Chồng 2 ảnh (đối với ảnh xám)
- Lật ảnh theo chiều ngang
- Lật ảnh theo chiều dọc
- Làm mờ ảnh

1. Thay đổi độ sáng

- Ý tưởng và mô tả: để thay đổi độ sáng của 1 bức ảnh, t cộng hoặc trừ các giá trị RGB của từng điểm ảnh với 1 giá trị bất kỳ. Sau khi cộng hoặc trừ thì phải xét lại sao cho các giá trị RGB thuộc [0, 255]. Sử dụng numpy.array kiểu int để không bị tràn số.

```
def brightness(image, value):
    image = image + value
    image = np.clip(image,0,255)
    return image
```

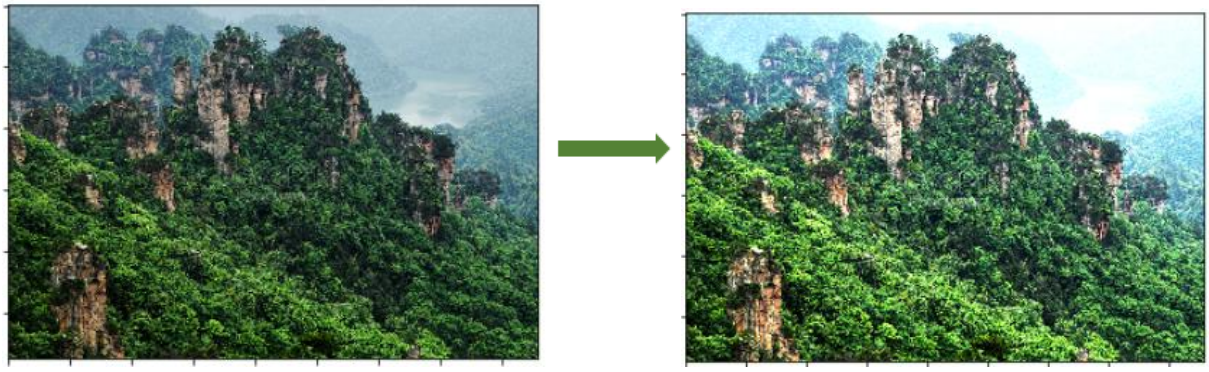
vd: với value = 50 ta được

2. Thay đổi độ tương phản

- Ý tưởng và mô tả: để thay đổi độ tương phản của 1 bức ảnh, t nhân các giá trị RGB của từng điểm ảnh với 1 giá trị bất kỳ. Sau khi cộng hoặc trừ thì phải xét lại sao cho các giá trị RGB thuộc [0, 255]. Sử dụng numpy.array kiểu int để không bị tràn số

```
def constrast(image, value):
    image = image*value
    image = np.clip(image,0,255)
    return image.astype(int)
```

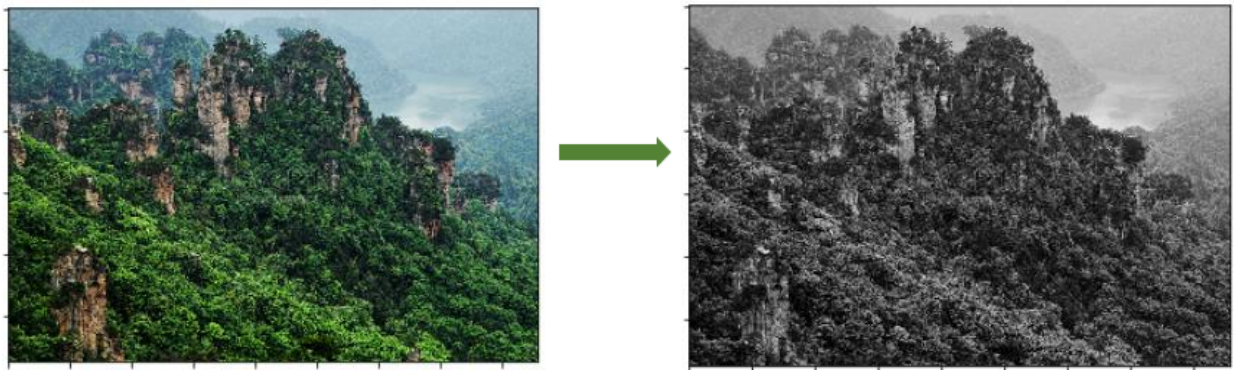
vd: với value = 1.5 ta được



3. Chuyển RGB thành ảnh xám

- Ý tưởng: sử dụng Weigted method với pixel = 0.3*Red + 0.59*Green + 0.11*Blue
- Mô tả: sử dụng numpy.dot để tích vô hướng chiều cuối cùng của mảng hình ảnh với vetor (0.3, 0.59, 0.11) rồi lưu vào temp. Gán lại vô mảng hình ảnh giá trị temp ở chiều cuối cùng với hàm numpy.newaxis. Ví dụ temp[::, ::, np.newaxis] sẽ gán cả giá trị ở chiều thứ 3 bằng với giá trị tương ứng ở temp.

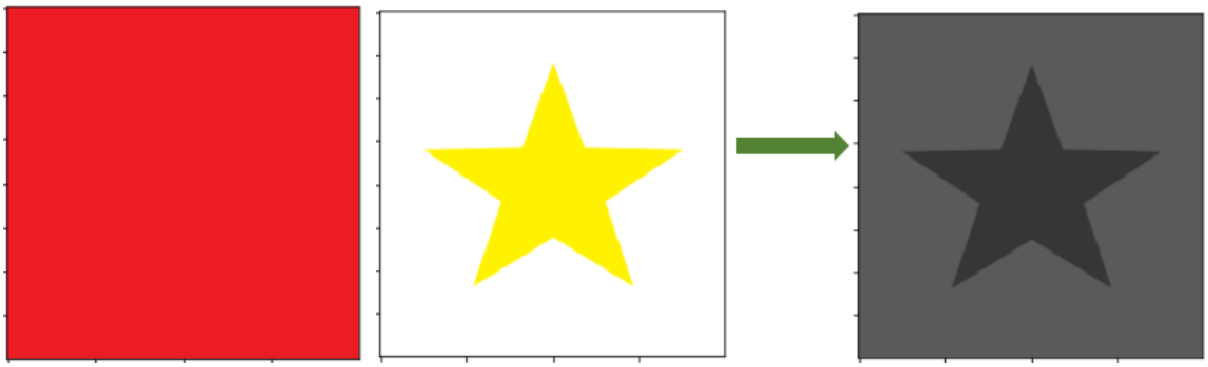
```
def grayscale(image):
    vector = [0.3, 0.59, 0.11]
    temp = np.dot(image.astype(float), vector)
    image[::] = temp[::, ::, np.newaxis]
    return image.astype('uint8')
```



4. Chồng 2 ảnh (đối với ảnh xám)

- Ý tưởng và mô tả: làm xám cả 2 ảnh bằng phương pháp trên, sau đó cộng 2 mảng hình ảnh với nhau bằng broadcasting. Để được mảng 2 ảnh lồng vào nhau, ta phải để ảnh ở dạng 'uint8' để ảnh có thể bị tràn số → ảnh không bị trắng vì giá trị RGB max là 255.

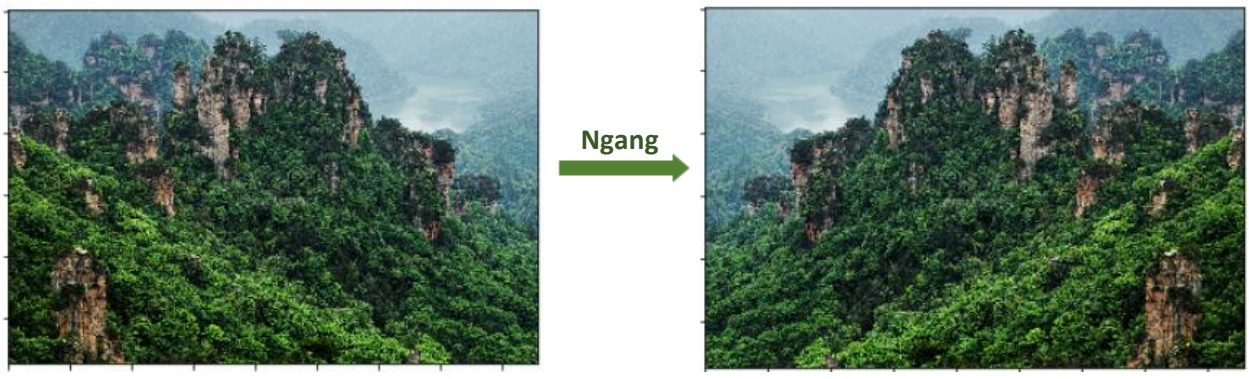
```
def overlap(img1,img2):
    img1 = grayscale(img1)
    img2 = grayscale(img2)
    img1 = img1 + img2
    return np.clip(img1,0,255)
```

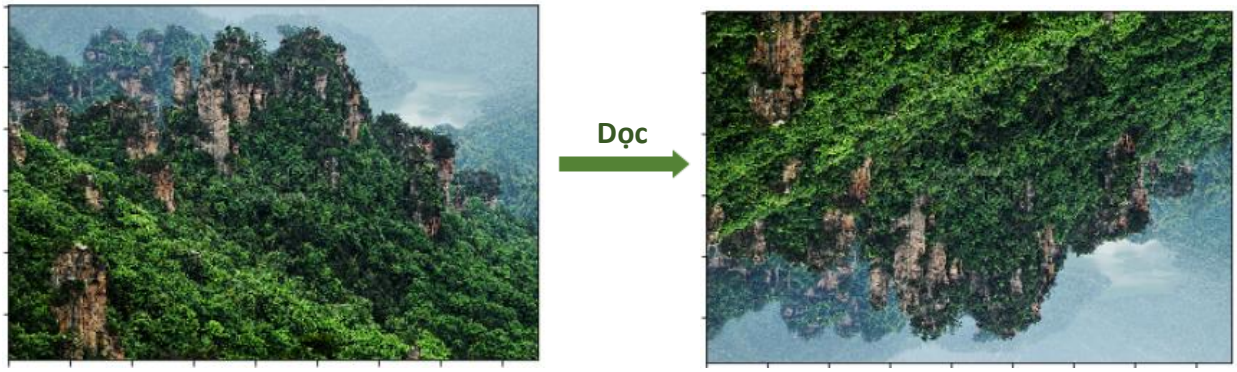


5. Lật ảnh theo chiều ngang / dọc

- Ý tưởng và mô tả: Ảnh RGB là mảng 3 chiều. Ta sử dụng slicing của numpy. Theo chiều ngang: slicing ngược lại ở chiều thứ 2, theo chiều dọc: slicing ngược lại ở chiều thứ 1. Các chiều còn lại giữ nguyên. Chú ý không làm ảnh hưởng đến chiều thứ 3 vì chiều này lưu giá trị RGB.

```
def flip(image, mode):
    if mode == 0: #ngang
        return image[:, ::-1]
    elif mode == 1: #doc
        return image[::-1]
```

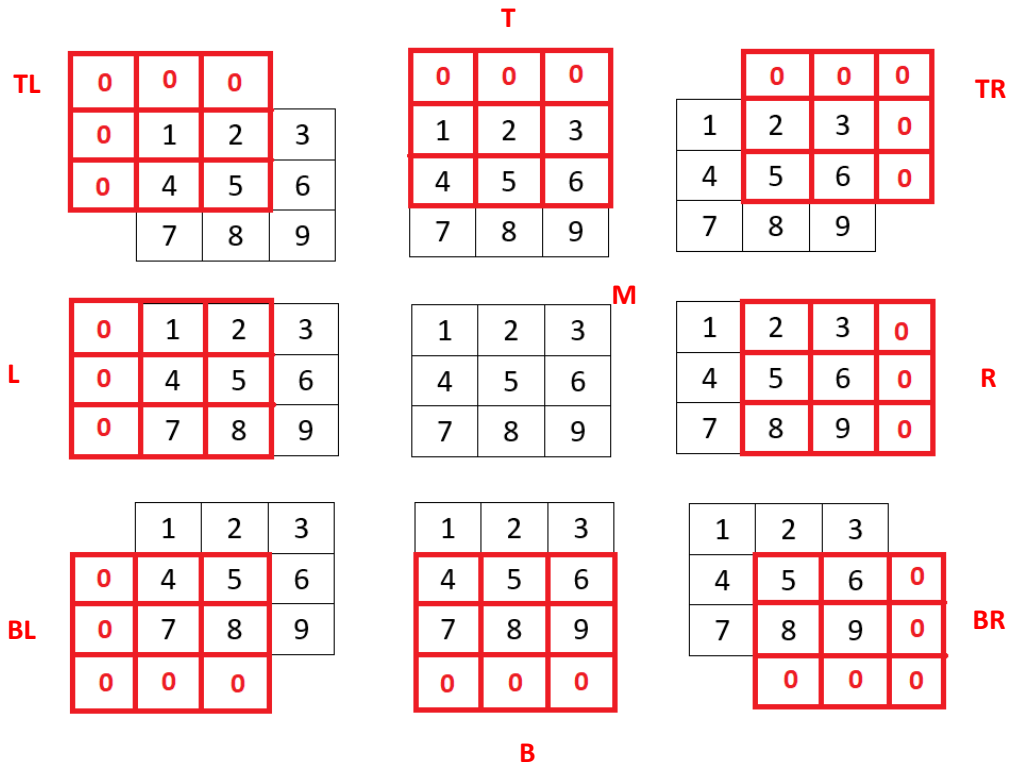




6. Làm mờ ảnh

Gaussian blur 3 × 3 (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$

- Ý tưởng: Làm mờ hình sử dụng kernel Gaussian blur 3x3 như ảnh trên. Tạo 9 mảng phụ với kích thước bằng mảng gốc, mỗi mảng là giá trị của mảng gốc lệch lên trên/qua trái/qua phải/xuống dưới 1 đơn vị. Mục đích là tạo ra các mảng con xung quanh mảng gốc để thực hiện nhân broadcasting từng mảng với phần tử tương ứng trong kernel rồi tính tổng các mảng bằng broadcasting lại với nhau ta sẽ được kết quả mong muốn. Hình minh họa:



- Mô tả: hàm tạo ra 9 mảng xung quanh mảng gốc ban đầu với giá trị bằng 0 kích thước bằng với kích thước mảng gốc. Sau đó, với mỗi vị trí mảng mà ta có cách duyệt để lấy giá trị tương ứng từ mảng gốc qua.

Ví dụ với mảng TopLeft: `TL[1:, 1:] = img[:img.shape[0]-1, :img.shape[1]-1]`

Mảng này lấy các giá trị nằm chếch lên bên trái từ mảng gốc và lưu vào vị trí chếch xuống bên phải của nó nên từ ở mảng gốc chỉ lấy ở chiều đầu tiên từ đầu đến sát cuối, chiều thứ 2 từ đầu đến sát cuối, trong khi đó ở mảng TL sẽ lưu vào chiều đầu tiên từ phần tử thứ 2 đến cuối, chiều thứ 2 từ phần tử thứ 2 đến cuối. Tương tự với các mảng con còn lại.

Sau đó chỉ cần cộng broadcasting (các tích của từng mảng với các phần tử tương ứng trong kernel). Và xét giá trị trong khoảng 0~255 rồi ép kiểu int là xong

```
def blur(img, kernel): #3x3
    TL,T,TR = np.zeros(img.shape),np.zeros(img.shape),np.zeros(img.shape)
    ML,M,MR = np.zeros(img.shape),np.zeros(img.shape),np.zeros(img.shape)
    BL,B,BR = np.zeros(img.shape),np.zeros(img.shape),np.zeros(img.shape)

    TL[1:, 1:] = img[:img.shape[0]-1, :img.shape[1]-1]
    T[1:, :] = img[:img.shape[0]-1, :]
    TR[1:, :img.shape[1]-1] = img[:img.shape[0]-1, 1:]
    ML[:, 1:] = img[:, :img.shape[1]-1]
    M[::] = img[::]
    MR[:, :img.shape[1]-1] = img[:, 1:]
    BL[:img.shape[0]-1, 1:] = img[1:, :img.shape[1]-1]
    B[:img.shape[0]-1, :] = img[1:, :]
    BR[:img.shape[0]-1, :img.shape[1]-1] = img[1:, 1:]

    blur = kernel[0][0]*TL + kernel[0][1]*T + kernel[0][2]*TR
    blur+= kernel[1][0]*ML + kernel[1][1]*M + kernel[1][2]*MR
    blur+= kernel[2][0]*BL + kernel[2][1]*B + kernel[2][2]*BR
    blur = np.clip(blur,0,255)
    return blur.astype(int)
```

Kết quả