Họ và tên	**Kiều Công Hậu**
MSSV	**18127259**
Lớp	18CLC1

TOÁN ỨNG DỤNG VÀ THỐNG KÊ

Đồ án 2: Image Processing

1. Các chức năng đã hoàn thành:

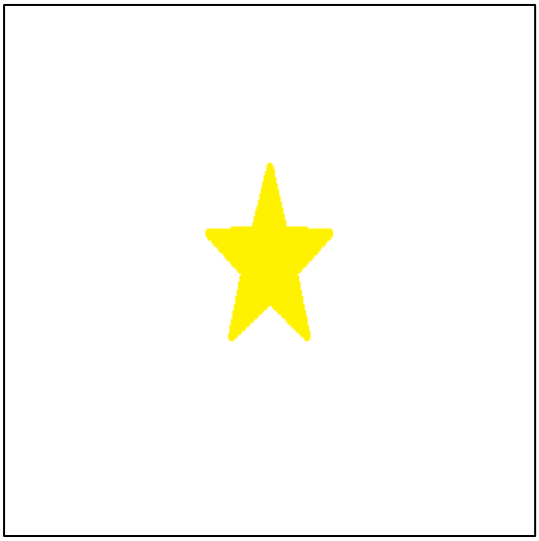
Em đã hoàn thành tất cả các yêu cầu.

STT	Yêu cầu	Hoàn thành
1	Thay đổi độ sáng cho ảnh	100%
2	Thay đổi độ tương phản	100%
3	Chuyển đổi ảnh RGB thành ảnh xám	100%
4	Lật ảnh (ngang – dọc)	100%
5	Chồng 2 ảnh cùng kích thước: ảnh xám	100%
6	Làm mờ ảnh	100%

2. Ý tưởng thực hiện, mô tả các hàm và hình ảnh kết quả:

Ảnh gốc:

Lenna



Star

<u>Lưu ý:</u> ảnh Star gốc dùng cho các output dưới đây không có viền đen, em đã cố tình thêm viền đen vào để thấy rõ giới hạn khung hình của ảnh (do ảnh có nền trắng).

2.0. Đọc ảnh

```python
# Đọc ảnh
def read_image_as_nparray(img_path):
    return np.array(Image.open(img_path))
```

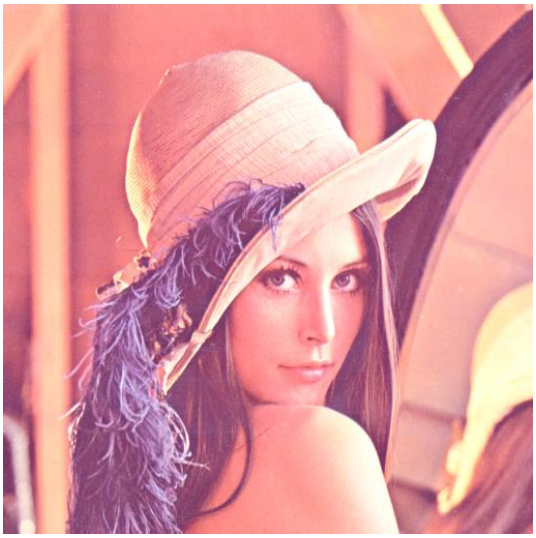
Đọc một ảnh dựa vào đường dẫn của ảnh muốn đọc (*img_path*) và trả về một numpy array các điểm ảnh (pixel) của ảnh đó.

2.1. Thay đổi độ sáng cho ảnh

```python
# 1. Thay đổi độ sáng cho ảnh
def adjust_brightness(img_path, alpha):
    img = read_image_as_nparray(img_path) + float(alpha)
    img = np.clip(img, 0, 255)
    return Image.fromarray(img.astype(np.uint8))
```

Đầu vào là đường dẫn của ảnh muốn thay đổi độ sáng (*img_path*) kèm với tham số thay đổi độ sáng (*alpha*) và trả về nội dung bức ảnh kiểu *Image*. Nếu *alpha* càng dương thì ảnh sẽ càng sáng, và ngược lại, nếu *alpha* càng âm thì ảnh sẽ càng tối. Tuy nhiên, nếu alpha lớn quá thì ảnh sẽ bị cháy trắng.

Ý tưởng là cộng *alpha* vào từng kênh màu của từng pixel, nếu giá trị kênh màu nào vượt quá 255 thì lấy 255.



Lenna – thay đổi độ sáng với alpha = 50

Tham khảo: https://ie.nitk.ac.in/blog/2020/01/19/algorithms-for-adjusting-brightness-and-contrast-of-an-image/

2.2. Thay đổi độ tương phản

```
# 2. Thay đổi độ tương phản
def adjust_contrast(img_path, alpha):
    f = (259 * (255 + alpha)) / (255 * (259 - alpha))
    img = read_image_as_nparray(img_path) * float(f) - 128 * f + 128
    img = np.clip(img, 0, 255)
    return Image.fromarray(img.astype(np.uint8))
```

Đầu vào là đường dẫn của ảnh muốn thay đổi độ sáng (*img_path*) kèm với tham số thay đổi độ tương phản (*alpha*) và trả về nội dung bức ảnh kiểu *Image*. Nếu alpha càng dương thì ảnh sẽ độ tương phản càng cao, và ngược lại.

Ý tưởng là ứng với giá trị đầu vào alpha, ta sẽ lần lượt thay đổi lần lượt tất cả các kênh màu của từng pixel theo công thức:

$$kenh\_mau\_moi = kenh\_mau\_cu * F - 128 * F + 128$$

với *kenh_mau_moi* (kênh màu mới) là giá trị sau khi áp dụng công thức của *kenh_mau_cu* (kênh màu cũ) và F được tính từ alpha

$$\mathrm{F} = \frac{259 * (255 + \mathrm{alpha})}{255 * (259 - alpha)}$$

Nếu giá trị kênh màu nào vượt quá 255 thì lấy 255.

Lenna – thay đổi tương phản với alpha = 100

Tham khảo: https://ie.nitk.ac.in/blog/2020/01/19/algorithms-for-adjusting-brightness-and-contrast-of-an-image/

2.3. Chuyển đổi ảnh RGB thành ảnh xám

```
# 3. Chuyển đổi ảnh RGB thành ảnh xám
def convert_to_grayscale(img_path):
    img = read_image_as_nparray(img_path) @ np.array([0.3, 0.59, 0.11])
    return Image.fromarray(img.astype(np.uint8))
```

Đầu vào là đường dẫn của ảnh muốn đổi thành ảnh xám (*img_path*) và trả về nội dung bức ảnh xám kiểu *Image*.

Ý tưởng là đối với từng pixel, ta sẽ gộp lại 3 kênh màu thành 1 giá trị duy nhất trong khoảng từ 0 đến 255 để biến ảnh màu RGB thành ảnh xám. Mỗi kênh màu sẽ có 1 trọng số tương ứng là 0.3, 0.59 và 0.11 ứng với lần lượt 3 kênh màu Red, Green và Blue. Vậy giá trị của mỗi pixel lúc này sẽ là:

$$GrayscalePixel = Red * 0.3 + Green * 0.59 + Blue * 0.11$$

Thay vì chạy vòng for để duyệt từng pixel, em tận dụng sức mạnh của Numpy để nhân luôn vector trọng số [0.3, 0.59, 0.11] với ma trận của ảnh (nhân ma trận trọng số này lần lượt các pixel của ảnh [R, G, B]).

$$[0.3, 0.59, 0.11] @ [R, G, B] = 0.3 * R + 0.59 * G + 0.11 * B$$

Lenna – ảnh RGB thành ảnh xám

Tham khảo: https://www.tutorialspoint.com/dip/grayscale_to_rgb_conversion.htm

2.4. Lật ảnh (ngang – dọc)

2.4.1. Ngang

```
# 4.1. Lật ảnh ngang
def flip_horizontally(img_path):
    return Image.fromarray(np.flip(read_image_as_nparray(img_path), axis=1))
```

Đầu vào là đường dẫn của ảnh muốn lật ngang (*img_path*) và trả về nội dung bức ảnh đã được lật ngang kiểu *Image*.

Ý tưởng là hoán đổi vị trí của các cột của bức ảnh, cột *0* hoán đổi với cột *width – 1*, cột *1* hoán đổi với cột *width – 2*, tổng quát là cột *i* hoán đổi với cột *width – 1 – i*. Em sử dụng hàm *flip* của Numpy với *axis = 1* để hoán đổi các cột của ma trận ảnh như ý tưởng trên.

Lenna – lật ảnh ngang

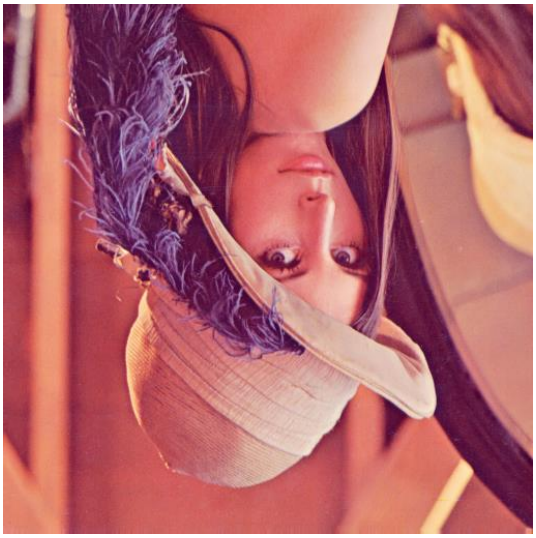
Tham khảo: https://numpy.org/doc/stable/reference/generated/numpy.flip.html

2.4.2. Dọc

```
# 4.2. Lật ảnh dọc
def flip_vertically(img_path):
    return Image.fromarray(np.flip(read_image_as_nparray(img_path), axis=0))
```

Đầu vào là đường dẫn của ảnh muốn lật dọc (*img_path*) và trả về nội dung bức ảnh đã được lật dọc kiểu *Image*.

Ý tưởng là hoán đổi vị trí của các dòng của bức ảnh, dòng *0* hoán đổi với dòng *height – 1*, dòng *1* hoán đổi với dòng *height – 2*, tổng quát là dòng *i* hoán đổi với dòng *height – 1 – i*. Em sử dụng hàm *flip* của Numpy với *axis = 0* để hoán đổi các dòng của ma trận ảnh như ý tưởng trên.



Tham khảo: https://numpy.org/doc/stable/reference/generated/numpy.flip.html

2.5. Chồng 2 ảnh cùng kích thước: ảnh xám

```
# 5. Chồng 2 ảnh cùng kích thước: chỉ làm trên ảnh xám
def blend(img_path_1, img_path_2, weight_1=0.5):
    weight_2 = 1 - weight_1
    img = np.array(convert_to_grayscale(img_path_1)) * weight_1\
        + np.array(convert_to_grayscale(img_path_2)) * weight_2
    return Image.fromarray(img.astype(np.uint8))
```

Đầu vào là đường dẫn của 2 ảnh muốn chồng lên nhau (*img_path_1, img_path_2*) kèm với trọng số của bức ảnh 1 (*weight_1*) và trả về nội dung bức ảnh sau khi đã chồng 2 ảnh đầu vào lên nhau theo trọng số kiểu *Image*.

Lưu ý: kích thước của 2 ảnh đầu vào là như nhau.

Ý tưởng là đổi 2 ảnh màu thành ảnh xám hết rồi tiến hành cộng 2 ảnh lại với trọng số tương ứng. Cụ thể là, ứng với từng pixel tương ứng của 2 ảnh:

$$pixel = pixel_1 * w_1 + pixel_2 * w_2$$

Với w_1 là trọng số của ảnh 1 và w_2 là trọng số của ảnh 2. w_1 và w_2 nằm trong khoảng giá trị từ *0* đến *1*, với $w_1 = 1 - w_2$. Nếu muốn ảnh nào rõ hơn thì cho trọng số của ảnh đó cao hơn, giá trị mặc định của *weight_1* là *0.5* đồng nghĩa với việc trọng số của 2 ảnh là như nhau.

Tham khảo: https://www.geeksforgeeks.org/addition-blending-images-using-opencv-python/

2.6. Làm mờ ảnh

```
# 6. Làm mờ ảnh (Gaussian blur 3x3)
def blur(img_path):
    kernel = np.array([[[1], [2], [1]],
                       [[2], [4], [2]],
                       [[1], [2], [1]]]) / 16

    img = read_image_as_nparray(img_path)
    temp = np.zeros((img.shape[0] + 2, img.shape[1] + 2, img.shape[2]))
    temp[1:-1,1:-1,:] = img

    for col in range(0, img.shape[0]):
        for row in range(0, img.shape[1]):
            img[col][row] = (temp[col:col+3, row:row+3] * kernel).sum(axis=1).sum(axis=0)

    return Image.fromarray(img.astype(np.uint8))
```

Đầu vào là đường dẫn của ảnh muốn làm mờ (*img_path*) và trả về nội dung bức ảnh đã được làm mờ kiểu *Image*.

Ý tưởng là sử dụng ma trận *kernel* kiểu *Gaussian blur 3x3* làm ma trận trọng số. Ta duyệt từng pixel và làm mờ từng pixel một bằng cách lấy trung bình các giá trị pixel xung quanh pixel đang xét ứng với trọng số tương ứng trong ma trận *kernel*. Để áp dụng công thức trên một cách tổng quát, em đã thêm padding các pixel [0, 0, 0] vào ma trận ảnh trước. Sau khi làm mờ tất cả các pixel, ta loại bỏ padding đi và trả về nội dung ảnh.

Giải thích code một tí:

```
    img = read_image_as_nparray(img_path)
    temp = np.zeros((img.shape[0] + 2, img.shape[1] + 2, img.shape[2]))
    temp[1:-1,1:-1,:] = img
```

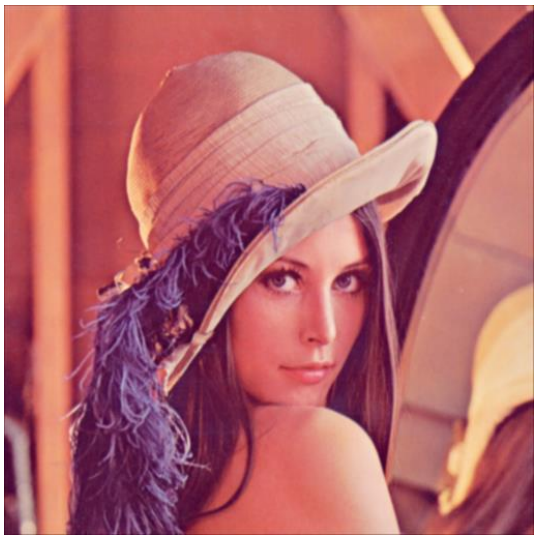
\

Đoạn code trên là dùng để thêm padding vào ma trận ảnh (*img*). Ý tưởng là tạo một ma trận ảnh toàn các pixel [0, 0, 0] có kích thước bằng đúng kích thước của ma trận ảnh sau khi thêm padding (*temp*). Sau đó ta gán phần "ruột" của ma trận *temp* này bằng ma trận ảnh *img*.

```
for col in range(0, img.shape[0]):
    for row in range(0, img.shape[1]):
        img[col][row] = (temp[col:col+3, row:row+3] * kernel).sum(axis=1).sum(axis=0)
```

Đoạn code trên là dùng để làm mờ từng pixel bằng cách duyệt 2 vòng for lồng nhau. Ứng với từng pixel của ma trận ảnh, ta tiến hành làm mờ theo ý tưởng đã nêu trên.

- *temp[col:col+3, row:row+3]* là ma trận *3x3* ứng với điểm ảnh mà ta muốn làm mờ, điểm ảnh thứ *(col, row)* của ma trận ảnh *img*.
- *kernel* là ma trận trọng số, cũng có kích thước *3x3*.
- Nhân 2 ma trận trên ta được một ma trận mới có kích thước *3x3 (9* pixels*)*, trong đó mỗi phần tử thứ *(i, j)* của ma trận kết quả là tích của *temp[col:col+3, row:row+3][i][j] và kernel[i][j]*, mỗi phần tử thứ *(i, j)* này là một pixel (mảng 3 phần tử ứng với 3 kênh màu RGB).
- *sum(axis=1)* là cộng 3 pixel trong cùng một hàng lại, ma trận tích vừa tính ở trên có 3 dòng 3 cột nên *(temp[col:col+3, row:row+3][i][j]*kernel).sum(axis=1)* trả về một mảng gồm 3 pixel.
- *sum(axis=0)* là cộng 3 pixel của ma trận kết quả trên lại, kết quả trả về còn 1 pixel. Pixel này là kết quả của thuật toán làm mờ điểm ảnh thứ *(col, row)*.



Lenna – làm mờ ảnh

Tham khảo: https://en.wikipedia.org/wiki/Kernel_(image_processing)

HẾT