TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM
KHOA CÔNG NGHỆ THÔNG TIN

---o0o---

BÁO CÁO ĐỒ ÁN THỰC HÀNH

IMAGE PROCESSING

Môn : **Toán ứng dụng và thống kê cho công nghệ thông tin**

Giảng Viên : **Phan Thị Phương Uyên**

Lớp : **21CLC05**

Tên : **Huỳnh Minh Quang**

MSSV : **21127149**

Thành phố Hồ Chí Minh, ngày 30 tháng 07 năm 2023

MỤC LỤC

I. Chức năng đã thực hiện

STT	Chức năng	Mức độ hoàn thành	Ghi chú
1	Thay đổi độ sáng ảnh	Hoàn thành 100%	
2	Thay đổi độ tương phản ảnh	Hoàn thành 100%	
3	Lật ảnh (ngang-dọc)	Hoàn thành 100%	Có 2 lựa chọn để xoay ảnh ngang hoặc dọc
4	Chuyển ảnh RGB thành ảnh xám và sepia	Hoàn thành 100%	Có 2 lựa chọn để xoay ảnh ngang hoặc dọc
5	Làm mờ ảnh	Hoàn thành 100%	
6	Làm sắc nét ảnh	Hoàn thành 100%	
7	Cắt ảnh theo kích thước	Hoàn thành 100%	
8	Cắt ảnh theo khung hình tròn	Hoàn thành 100%	
9	Cắt ảnh theo khung là hình ellip chéo nhau	Hoàn thành 100%	

II. Ý tưởng và mô tả các hàm chức năng

- Chương trình được tổ chức thành các hàm con để thực hiện từng chức năng xử lý ảnh. Các hàm chính bao gồm:

1. Thay đổi độ sáng - change_brightness(image, K):

```python
#Change the brightness
def change_brightness(image, K):
    newImage = np.uint8(np.clip(image + np.array([K], dtype=np.int16), 0, 255))
    return newImage
```

- Hàm này được sử dụng để điều chỉnh độ sáng của một bức ảnh bằng cách cộng một hằng số vào cường độ của mỗi điểm ảnh.
  - **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.
- K: Hệ số điều chỉnh độ sáng (có thể là số dương hoặc số âm).

  - **Hàm thực hiện các bước sau:**

- Nó cộng giá trị K vào mỗi điểm ảnh của bức ảnh bằng các thao tác mảng NumPy.
- Kết quả có thể làm cho một số giá trị điểm ảnh vượt qua khoảng giá trị hợp lệ từ 0 đến 255, do đó sử dụng hàm np.clip để đảm bảo tất cả các giá trị điểm ảnh nằm trong khoảng này.

2. Thay đổi độ tương phản change_contrast(image, K):

```python
#Change Contrast
def change_contrast(image, K):
    factor = np.clip(float(K), -255, 255)
    contrast = np.clip(float(K), -255, 255)
    factor = (259 * (contrast + 255)) / (255 * (259 - contrast))
    newImage= np.uint8(np.clip(factor * (image.astype(float) - 128) + 128, 0, 255))
    return newImage
```

- Hàm này được sử dụng để điều chỉnh độ tương phản của một bức ảnh bằng cách sử dụng biến đổi tuyến tính của cường độ điểm ảnh.

  - **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.
- K: Hệ số điều chỉnh độ tương phản (có thể là số dương hoặc số âm).

  - **Hàm thực hiện các bước sau:**

- Nó tính toán factor để điều chỉnh độ tương phản dựa trên giá trị K đầu vào. Factor được tính bằng cách sử dụng công thức (259 * (contrast + 255)) / (255 * (259 - contrast)).
- Độ tương phản được giữ trong khoảng từ -255 đến 255 bằng cách sử dụng hàm np.clip.
- Các giá trị điểm ảnh được chuyển đổi thành số thực để cho phép điều chỉnh độ tương phản.
- Cường độ điểm ảnh sau đó được tỉ lệ với factor đã tính và cộng thêm giá trị 128 để căn chỉnh cường độ xung quanh 128.
- Cuối cùng, các giá trị điểm ảnh được cắt tỉa để nằm trong khoảng giá trị hợp lệ từ 0 đến 255 bằng cách sử dụng np.clip, và bức ảnh kết quả được trả về dưới dạng mảng numpy.

3. Lật ảnh ngang/ dọc - flip_image(image, direction):

```python
# Flip the image (horizontal - vertical)
def flip_image(image, direction):
    if direction == 'Vertical':
        newImage= np.flipud(image)
    elif direction=='Horizontal':
        newImage=np.fliplr(image)
    else:
        print("Error flip image ")
        exit()
    newImage = np.clip(newImage, 0, 255).astype(np.uint8)
    return newImage
```

- Hàm này được sử dụng để lật ảnh theo hướng ngang hoặc dọc.
  - **Tham số:**
- image: Bức ảnh đầu vào dưới dạng mảng numpy.
- direction: Hướng lật ảnh ('Vertical' hoặc 'Horizontal').
  - **Hàm thực hiện các bước sau:**
- Nếu hướng lật là 'Vertical', hàm sẽ sử dụng np.flipud để lật ảnh theo chiều dọc (đảo ngược từ trên xuống dưới).
- Nếu hướng lật là 'Horizontal', hàm sẽ sử dụng np.fliplr để lật ảnh theo chiều ngang (đảo ngược từ trái sang phải).
- Nếu hướng lật không hợp lệ, hàm sẽ in thông báo lỗi và kết thúc chương trình bằng exit().

4. Chuyển đổi ảnh RGB thành ảnh xám - rgb_gray(image):

```python
#Convert RGB image to gray image
def rgb_gray(image):
    weight =[0.2126, 0.7152, 0.0722]
    newImage=np.uint8(np.dot(image[..., :3], weight))
    newImage = np.clip(newImage, 0, 255).astype(np.uint8)
    return newImage
```

- Hàm này được sử dụng để chuyển đổi một bức ảnh RGB thành ảnh xám bằng cách tính tổng trọng số của các kênh màu.
  - **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.
  - **Hàm thực hiện các bước sau:**
- Nó nhân các kênh màu của ảnh (được lựa chọn bằng cách cắt [..., :3]) với các trọng số được xác định trước [0.2126, 0.7152, 0.0722].
- Kết quả là tổng trọng số của các kênh màu đã nhân, tạo thành bức ảnh xám mới.

5. Chuyển đổi ảnh RGB thành ảnh sepia - rgb_sepia(image):

```python
#Convert RGB image to gray image
def rgb_gray(image):
    weight =[0.2126, 0.7152, 0.0722]
    newImage=np.uint8(np.dot(image[..., :3], weight))
    newImage = np.clip(newImage, 0, 255).astype(np.uint8)
    return newImage
```

- Hàm này được sử dụng để chuyển đổi một bức ảnh RGB thành ảnh sepia bằng cách áp dụng bộ lọc sepia.
  - **Tham số:**
- image: Bức ảnh đầu vào dưới dạng mảng numpy.
  - **Hàm thực hiện các bước sau:**
- Nó nhân ma trận image với ma trận sepia đã được xác định trước để tạo ra bức ảnh mới ở dạng sepia.

6. Làm mờ ảnh - blur_image(image, kernel_size):

```python
#Blur image
def blur_image(image, kernel_size):
    kernel = calc_Gaussian_kernel(kernel_size, sigma=(kernel_size-1)/6)
    newImage = convolution(image, kernel)
    newImage = np.clip(newImage, 0, 255).astype(np.uint8)
    return newImage
```

- Hàm này được sử dụng để làm mờ một bức ảnh bằng cách sử dụng bộ lọc Gaussian, và có các hàm hỗ trợ tính toán sau:
  - **Gaussian_func(x, y, sigma)**

```python
# Gaussian function
def Gaussian_func(x, y, sigma):
    return np.exp(-(x**2 + y**2) / (2 * sigma**2))
```

- Hàm này tính giá trị của hàm Gaussian 2 chiều (2D Gaussian function) tại các điểm (x, y) trong không gian 2D.
    - **Tham số:**
- x, y: Các giá trị x và y trong không gian 2D.
- sigma: Tham số đại diện cho độ rộng của phân phối Gaussian.
    - **Hàm thực hiện các bước sau:**
- Sử dụng công thức Gaussian: G(x, y) = exp(-(x**2 + y**2) / (2 * sigma**2)).
- Trả về giá trị Gaussian tại các điểm (x, y) đã cho.

- **calc_Gaussian_kernel(kernel_size, sigma)**

```
# Calculate 2D Gaussian filter matrix.
def calc_Gaussian_kernel(kernel_size, sigma):
    x, y = np.meshgrid(np.arange(-kernel_size // 2, kernel_size // 2 + 1),
                       np.arange(-kernel_size // 2, kernel_size // 2 + 1))
    kernel = Gaussian_func(x, y, sigma)
    kernel /= np.sum(kernel)
    return kernel
```

- Hàm này tính ma trận bộ lọc Gaussian 2 chiều có kích thước kernel_size x kernel_size với độ rộng Gaussian là sigma.
    - **Tham số:**
- kernel_size: Kích thước của bộ lọc Gaussian (một số lẻ).
- sigma: Tham số đại diện cho độ rộng của phân phối Gaussian.
    - **Hàm thực hiện các bước sau:**
- Tạo lưới các điểm (x, y) bằng cách sử dụng np.meshgrid trong khoảng từ -kernel_size // 2 đến kernel_size // 2 + 1.
- Tính toán giá trị của hàm Gaussian 2D tại các điểm lưới đã tạo bằng cách gọi hàm Gaussian_func với các giá trị (x, y) và sigma tương ứng.
- Chuẩn hóa ma trận bộ lọc bằng cách chia các giá trị của bộ lọc cho tổng của chúng, đảm bảo tổng của bộ lọc bằng 1.
- Trả về ma trận bộ lọc Gaussian.

- **convolve_layer(layer, kernel)**

```
# Convolution on 2 dimensional matrices
def convolve_layer(layer, kernel):
    view = kernel.shape + tuple(np.subtract(layer.shape, kernel.shape) + 1)
    submatrices = np.lib.stride_tricks.as_strided(layer, shape = view, strides = layer.strides * 2)
    return np.einsum('ij,ijkl->kl', kernel, submatrices)
```

- Hàm này thực hiện phép tích chập giữa ma trận layer và ma trận kernel trên các ma trận 2 chiều.
    - **Tham số:**
- layer: Ma trận 2 chiều đầu vào.
- kernel: Ma trận bộ lọc.
    - **Hàm thực hiện các bước sau:**
- Tạo các ma trận con (submatrices) của layer có kích thước bằng kích thước của kernel và dịch chuyển theo từng bước bằng cách sử dụng hàm np.lib.stride_tricks.as_strided. Điều này tạo ra một số lượng ma trận con có thể trùng lắp.
- Tích chập giữa kernel và các ma trận con bằng cách sử dụng np.einsum.
- Trả về ma trận kết quả sau khi tích chập.

- **convolution(img, kernel):**

```
# Convolution on a color image consisting of 3 RGB channels
def convolution(img, kernel):
    return np.dstack((convolve_layer(img[:,:,0], kernel),
                      convolve_layer(img[:,:,1], kernel),
                      convolve_layer(img[:,:,2], kernel)))
```

- Hàm này thực hiện phép tích chập giữa bức ảnh màu có 3 kênh (RGB) và ma trận bộ lọc.
    - **Tham số:**
- img: Bức ảnh đầu vào dưới dạng mảng numpy.
- kernel: Ma trận bộ lọc.
    - **Hàm thực hiện các bước sau:**
- Tách lấy các kênh màu R, G và B từ bức ảnh và thực hiện phép tích chập giữa mỗi kênh màu và kernel bằng cách gọi hàm convolve_layer.
- Kết hợp các kết quả tích chập của các kênh màu thành một bức ảnh màu mới bằng cách sử dụng np.dstack.

- Trả về bức ảnh mới sau khi thực hiện phép tích chập.

- **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.
- kernel_size: Kích thước của bộ lọc Gaussian (một số lẻ). (Khi kích thước của bộ lọc Gaussian là một số lẻ, trung tâm của bộ lọc có thể được định nghĩa rõ ràng, không có số nguyên trung tâm nào ở giữa nó).

- **Hàm thực hiện các bước sau:**

- Tính ma trận bộ lọc Gaussian bằng cách gọi hàm calc_Gaussian_kernel(kernel_size, sigma) với kích thước kernel_size và độ lệch chuẩn được tính dựa trên sigma=(kernel_size-1)/6.
- Áp dụng phép tích chập bộ lọc Gaussian với bức ảnh ban đầu bằng cách gọi hàm convolution(img, kernel), với ảnh đầu vào và ma trận lọc Gaussian để thực hiện phép tích chập.
- Trả về bức ảnh đã được làm mờ.

7. Làm sắc nét ảnh - sharpen_image(image):

```
# sharpen image
def sharpen_image(image):
    kernel = sharpen_kernel() # Create a Gaussian filter with sharpen matrix
    newImage = convolution(image, kernel)
    newImage = np.clip(newImage, 0, 255).astype(np.uint8)
    return newImage
```

- Đây là hàm được sử dụng để làm sắc nét cho ảnh bằng phép tích chập với một ma trận lọc sắc nét, và có các hàm hỗ trợ tính toán sau:

- **convolve_layer(layer, kernel)**

```
# Convolution on 2 dimensional matrices
def convolve_layer(layer, kernel):
    view = kernel.shape + tuple(np.subtract(layer.shape, kernel.shape) + 1)
    submatrices = np.lib.stride_tricks.as_strided(layer, shape = view, strides = layer.strides * 2)
    return np.einsum('ij,ijkl->kl', kernel, submatrices)
```

- Hàm này thực hiện phép tích chập giữa ma trận layer và ma trận kernel trên các ma trận 2 chiều.

  - **Tham số:**

  - layer: Ma trận 2 chiều đầu vào.

- kernel: Ma trận bộ lọc.
  - **Hàm thực hiện các bước sau:**
- Tạo các ma trận con (submatrices) của layer có kích thước bằng kích thước của kernel và dịch chuyển theo từng bước bằng cách sử dụng hàm np.lib.stride_tricks.as_strided. Điều này tạo ra một số lượng ma trận con có thể trùng lắp.
- Tích chập giữa kernel và các ma trận con bằng cách sử dụng np.einsum.
- Trả về ma trận kết quả sau khi tích chập.

- **sharpen_kernel():**

```
# Sharpen matrix
def sharpen_kernel():
    return np.array([[0, -1, 0],
                     [-1, 5, -1],
                     [0, -1, 0]])
```

- Đây là hàm đơn giản trả về một ma trận 3x3 được gọi là ma trận lọc sắc nét.
- Ma trận lọc sắc nét được sử dụng trong quá trình tích chập với ảnh để tăng độ tương phản và làm sắc nét hơn.

- **convolution(img, kernel):**

```
# Convolution on a color image consisting of 3 RGB channels
def convolution(img, kernel):
    return np.dstack((convolve_layer(img[:,:,0], kernel),
                      convolve_layer(img[:,:,1], kernel),
                      convolve_layer(img[:,:,2], kernel)))
```

- Hàm này thực hiện phép tích chập giữa bức ảnh màu có 3 kênh (RGB) và ma trận bộ lọc.
  - **Tham số:**
- img: Bức ảnh đầu vào dưới dạng mảng numpy.
- kernel: Ma trận bộ lọc.
  - **Hàm thực hiện các bước sau:**
- Tách lấy các kênh màu R, G và B từ bức ảnh và thực hiện phép tích chập giữa mỗi kênh màu và kernel bằng cách gọi hàm convolve_layer.

- Kết hợp các kết quả tích chập của các kênh màu thành một bức ảnh màu mới bằng cách sử dụng np.dstack.
- Trả về bức ảnh mới sau khi thực hiện phép tích chập.

- **Tham số:**

- image: Ảnh đầu vào dưới dạng mảng numpy.

- **Hàm thực hiện các bước sau:**

- Hàm bắt đầu bằng việc gọi hàm **sharpen_kernel()** để tạo ra ma trận lọc sắc nét. Ma trận lọc này chứa các trọng số được cấu hình để tăng độ tương phản và làm sắc nét cho ảnh.
- Sau đó, hàm gọi hàm **convolution(image, kernel)** để thực hiện phép tích chập giữa ảnh đầu vào và ma trận lọc sắc nét. Quá trình tích chập này giúp tăng độ tương phản và làm sắc nét cho ảnh.
- Cuối cùng, hàm trả về ảnh mới sau khi đã thực hiện phép làm sắc nét.

8. Cắt ảnh theo kích thước - crop_center(image, target_height, target_width):

```python
#Crop the image from the center to the target size
def crop_center(image, target_height, target_width):
    height, width = image.shape[:2]
    #center image
    start_y = (height - target_height) // 2
    start_x = (width - target_width) // 2
    end_y = start_y + target_height
    end_x = start_x + target_width
    newImage = image[start_y:end_y, start_x:end_x]
    return newImage
```

- Hàm này cắt ảnh từ trung tâm để có kích thước mục tiêu target_height x target_width.

- **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.
- target_height: Chiều cao mục tiêu của ảnh cắt.
- target_width: Chiều rộng mục tiêu của ảnh cắt.

- **Hàm thực hiện các bước sau:**

- Lấy kích thước (chiều cao và chiều rộng) của ảnh đầu vào.
- Tính toán vị trí bắt đầu (start_y, start_x) và vị trí kết thúc (end_y, end_x) để cắt ảnh từ

trung tâm.

- Thực hiện cắt ảnh từ vị trí bắt đầu đến vị trí kết thúc.

9. Cắt ảnh theo khung hình tròn - crop_circle(image):

```
# crop image circle
def crop_circle(image):
    height, width = image.shape[:2]
    center_y, center_x = height // 2, width // 2  #center
    y, x = np.ogrid[:height, :width]
    # equation of the circle
    mask = (x - center_x)**2 + (y - center_y)**2 <= min(center_y, center_x)**2
    newImage=image * mask[:, :, np.newaxis]
    return newImage
```

- Hàm này cắt ảnh thành một khung hình tròn bằng cách tạo một mặt nạ hình tròn và áp dụng nó lên ảnh đầu vào.

- **Tham số:**

- image: Bức ảnh đầu vào dưới dạng mảng numpy.

- **Hàm thực hiện các bước sau:**

- Lấy kích thước (chiều cao và chiều rộng) của ảnh đầu vào.
- Xác định tâm của ảnh (center_y, center_x) là trung tâm của ảnh.
- Tạo một lưới các điểm (y, x) bằng np.ogrid với khoảng từ 0 đến chiều cao và từ 0 đến chiều rộng.
- Xác định phương trình của hình tròn: **(x - center_x)**2 + (y - center_y)**2 <= min(center_y, center_x)**2**.
- Tạo mặt nạ hình tròn bằng cách gán giá trị True (1) cho các điểm nằm trong hình tròn và giá trị False (0) cho các điểm nằm ngoài hình tròn.
- Áp dụng mặt nạ lên ảnh đầu vào để cắt ảnh thành hình tròn.

10. Cắt ảnh theo khung hình ellip chéo nhau - crop_ellip_frame(image):

```python
# Crop the image in the frame of 2 diagonal ellip
def crop_ellip_frame(image):
    # Parameters of two diagonal ellipses
    height, width = image.shape[:2]
    center_y, center_x = height // 2, width // 2
    major_axis = int(width * 0.5)
    minor_axis = int(height * 0.3)
    angle = 30 # The inclination of the ellipse (unit is degrees)
    mask1 = create_elliptical_mask(image.shape[:2], center_x, center_y, major_axis, minor_axis, angle)
    mask2 = create_elliptical_mask(image.shape[:2], center_x, center_y, major_axis, minor_axis, -angle)
    mask = mask1 | mask2
    # Apply mask to crop iamge
    newImage=image.copy()
    newImage[mask == 0] = 0
    y, x = np.nonzero(mask)
    min_y, max_y = np.min(y), np.max(y)
    min_x, max_x = np.min(x), np.max(x)
    newImage = newImage[min_y:max_y + 1, min_x:max_x + 1]
    return newImage
```

- Hàm này được sử dụng để cắt ảnh theo khung của hai hình elip đối chéo, dựa trên hàm tạo mặt nạ hình ellip sau:
  - **create_elliptical_mask(shape, center_x, center_y, major_axis, minor_axis, angle):**

```python
#create ellip mask
def create_elliptical_mask(shape, center_x, center_y, major_axis, minor_axis, angle):
    mask = np.zeros(shape, dtype=np.uint8)
    y, x = np.ogrid[:shape[0], :shape[1]]
    angle_rad = np.deg2rad(angle)
    cos_a = np.cos(angle_rad)
    sin_a = np.sin(angle_rad)
    # Equation of the ellipse
    ellipse = (((x - center_x) * cos_a + (y - center_y) * sin_a) ** 2) / (major_axis ** 2) + \
              (((x - center_x) * sin_a - (y - center_y) * cos_a) ** 2) / (minor_axis ** 2) <= 1
    mask[ellipse] = 255
    return mask
```

- Đây là hàm dùng để tạo ma trận lọc hình elip.
  - **Tham số:**

- shape: Kích thước của ma trận lọc.
- center_x, center_y: Tọa độ tâm của hình elip.
- major_axis: Trục chính của hình elip (độ dài nửa trục lớn).
- minor_axis: Trục phụ của hình elip (độ dài nửa trục nhỏ).
- angle: Góc nghiêng của hình elip (đơn vị là độ).

- **Hàm thực hiện các bước sau:**

- Tạo ma trận lọc (mask) có kích thước shape ban đầu và đặt tất cả các giá trị bằng 0.
- Tạo một lưới các điểm (x, y) trong ma trận lọc bằng hàm np.ogrid.
- Chuyển đổi góc angle từ độ sang radian bằng hàm np.deg2rad.
- Tính cosin và sin của angle_rad.
- Tính toán điều kiện để tạo hình elip bằng cách kiểm tra xem mỗi điểm trong ma trận có nằm trong hình elip hay không.

- **Tham số:**

- image: Ảnh đầu vào dưới dạng mảng numpy.

- **Hàm thực hiện các bước sau:**

- Xác định kích thước của ảnh image.
- Xác định tọa độ tâm của hình elip (center_x, center_y) là trung điểm của ảnh.
- Xác định chiều dài nửa trục lớn major_axis và chiều dài nửa trục nhỏ minor_axis của hình elip, ở đây major_axis là một nửa chiều rộng của ảnh và minor_axis là một nửa chiều cao của ảnh.
- Xác định góc nghiêng của hình elip angle.
- Tạo hai ma trận lọc hình elip mask1 và mask2 bằng cách sử dụng hàm create_elliptical_mask.
- Gộp hai ma trận lọc hình elip mask1 và mask2 lại với toán tử OR để tạo thành ma trận lọc cuối cùng mask.
- Áp dụng ma trận lọc mask lên ảnh image để cắt ảnh theo khung của hai hình elip.

11. Các hàm xử lý khác:

- **Input():**

```
# Input funtion
def Input():
    input_File_Name= input("Enter the image file name: ")
    image = Image.open(input_File_Name)
    # Convert the image to a numpy array
    img = np.array(image)
    return img,  input_File_Name
```

- Hàm này yêu cầu người dùng nhập tên tệp hình ảnh, sau đó mở tệp và chuyển đổi hình ảnh thành một mảng numpy, sau đó trả về mảng numpy và tên tệp hình ảnh.

- **Display(image, newImage, key):**

```
# Display Image_original and newImage
def Display(image, newImage, key):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(image)
    axes[0].set_title("Image_original")
    axes[0].axis("off")
    axes[1].imshow(newImage)
    axes[1].set_title("Image_"+key)
    axes[1].axis("off")
    plt.show()
```

- Hàm này hiển thị hình ảnh gốc và hình ảnh đã được xử lý (được tham số hóa bởi key) bên cạnh nhau.

- **Display_Gray(image, newImage, key):**

```
# Display Image_original and newImage --gray image
def Display_Gray(image, newImage, key):
    fig, axes = plt.subplots(1, 2, figsize=(10, 5))
    axes[0].imshow(image)
    axes[0].set_title("Image_original")
    axes[0].axis("off")
    axes[1].imshow(newImage, cmap='gray')
    axes[1].set_title("Image_"+key)
    axes[1].axis("off")
    plt.show()
```

- Tương tự như hàm Display(), nhưng hiển thị hình ảnh xám thay vì màu.

- **Save_img(newImage, input_File_Name, key):**

```python
#save the changed image file
def Save_img(newImage, input_File_Name, key):
    output_format = 'png' #format ảnh có thể thay đổi thành jpg/pdf/..
    fileName = input_File_Name.split('.')
    fileName = fileName[0] +'_'+ key +'.'+ output_format
    image=Image.fromarray(newImage.astype(np.uint8))
    image.save(fileName)
```

- Hàm này lưu hình ảnh mới vào tệp với tên được định nghĩa bởi key.

- **Menu():**

```python
# Menu to select functions
def Menu():
    print("\nNhập phím chọn các chức năng xử lý ảnh:")
    print("[0] Chọn tất cả các chức năng và file ảnh")
    print("[1] Thay đổi độ sáng")
    print("[2] Thay đổi độ tương phản")
    print("[3] Lật ảnh (ngang - dọc)")
    print("[4] Chuyển đổi ảnh RGB thành ảnh xám hoặc sepia")
    print("[5] Làm mờ ảnh")
    print("[6] Làm sắc nét ảnh")
    print("[7] Cắt ảnh từ trung tâm với kích thước")
    print("[8] Cắt ảnh thành một khung hình tròn")
    print("[9] Cắt ảnh thành một khung hình elip chéo")
    print("[q] Thoát")
```

- Hàm này hiển thị menu chọn các chức năng xử lý hình ảnh.

- **Menu_flip():**

- Hàm này hiển thị menu để chọn hướng lật ảnh (ngang hoặc dọc).

- **Menu_gray_sepia():**

- Hàm này hiển thị menu để chọn chuyển đổi ảnh RGB thành ảnh xám hoặc sepia.

- **Choice(choice, image):**

- Hàm này chọn chức năng xử lý hình ảnh dựa trên lựa chọn của người dùng (biểu diễn bởi choice). Nó gọi các hàm xử lý hình ảnh tương ứng và trả về hình ảnh mới và khóa của hàm xử lý (biểu diễn bởi key).

- **main():**

- Hàm chính của chương trình. Nó yêu cầu người dùng nhập tên tệp hình ảnh và hiển thị menu chọn các chức năng xử lý hình ảnh. Sau đó, nó lặp lại việc thực hiện chức năng

chọn bởi người dùng cho đến khi người dùng chọn thoát.

III. Hình ảnh kết quả

- **Menu cho các chức năng:**

```
Enter the image file name: test_image.jpg

Nhập phím chọn các chức năng xử lý ảnh:
[0] Chọn tất cả các chức năng và file ảnh
[1] Thay đổi độ sáng
[2] Thay đổi độ tương phản
[3] Lật ảnh (ngang - dọc)
[4] Chuyển đổi ảnh RGB thành ảnh xám hoặc sepia
[5] Làm mờ ảnh
[6] Làm sắc nét ảnh
[7] Cắt ảnh từ trung tâm với kích thước
[8] Cắt ảnh thành một khung hình tròn
[9] Cắt ảnh thành một khung hình elip chéo
[q] Thoát

Nhập lựa chọn của bạn:
```

```
Nhập lựa chọn của bạn: 3

 Nhập phím để chọn hướng lật ảnh
[v] Vertical - dọc
[h] Horizontal - ngang
[q] Thoát
Chọn hướng lật ảnh: h
```

```
Nhập lựa chọn của bạn: 4

 Nhập phím để chọn đổi ảnh RGB thành ảnh xám/sepia
[g] Gray - xám
[s] Spepia
[q] Thoát
xám/sepia: s
```

- **Hình ảnh thay đổi độ sáng (tăng):**

Image_original

Image_brightness

- **Hình ảnh thay đổi độ tương phản (tăng):**

Image_original

Image_contrast

- **Hình ảnh khi lật ảnh (ngang**

Image_original

Image_flip

- **Hình ảnh khi lật ảnh (dọc):**

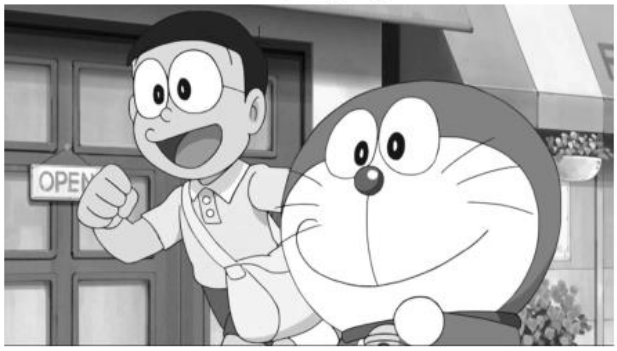
Image_original

Image_flip

- **Hình ảnh chuyển RGB sáng ảnh xám:**

Image_original

Image_gray



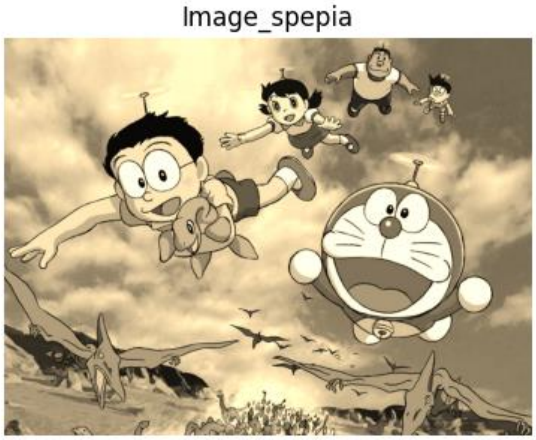
- **Hình ảnh chuyển RGB sáng ảnh sepia:**

Image_original

Image_spepia

- **Hình ảnh khi làm mờ:**

- **Hình ảnh khi làm sắt nét:**



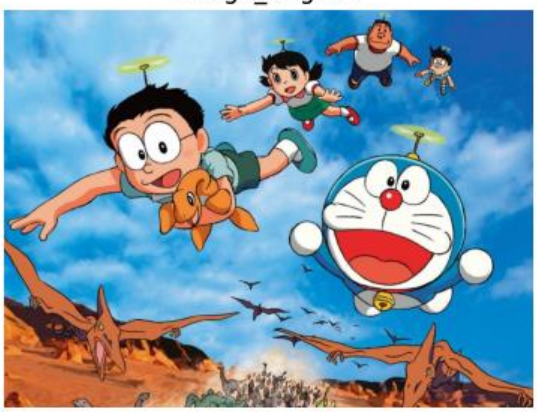
- **Cắt ảnh từ trung tâm với kích thước:**

Image_original

Image_crop(500x400)

- **Cắt ảnh hình tròn:**

Image_original



Image_circle

- **Cắt ảnh với khung hình ellip chéo:**

IV. Tài liệu tham khảo

- https://nttuan8.com/bai-5-gioi-thieu-ve-xu-ly-anh/
- https://en.wikipedia.org/wiki/Kernel_(image_processing)
- https://codelearn.io/sharing/ma-tran-co-ban-voi-thu-vien-numpy
- https://mmlab.uit.edu.vn/tutorials/cv/basic/pixel_wised_transform

___Hết___