

**TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN TP.HCM**  
**KHOA CÔNG CÔNG THÔNG TIN**



**BÁO CÁO ĐỒ ÁN 2**

**Image Processing**

**Chủ đề: Toán ứng dụng – thống kê**

**Lớp: 22CLC01**

**Sinh viên:                      Trương Thuận Kiệt – 22127224**

**TP. Hồ Chí Minh, tháng 6 năm 2024**

## Mục lục

<b>1. Ý tưởng thực hiện .....</b>	<b>5</b>
<b>a. Thay đổi độ sáng .....</b>	<b>5</b>
<b>b. Thay đổi độ sắc nét .....</b>	<b>5</b>
<b>c. Lật ảnh (Ngang, Dọc).....</b>	<b>6</b>
<b>d. RGB thành ảnh xám .....</b>	<b>7</b>
<b>e. RGB thành ảnh sepia.....</b>	<b>7</b>
<b>f. Làm mờ ảnh.....</b>	<b>7</b>
- <b>Ma trận Gaussian Kernel: .....</b>	<b>7</b>
- <b>Convolve: .....</b>	<b>7</b>
<b>g. Làm sắc nét ảnh .....</b>	<b>9</b>
<b>h. Cắt ảnh theo kích thước .....</b>	<b>10</b>
<b>i. Cắt ảnh theo khung .....</b>	<b>10</b>
- <b>Hình tròn: .....</b>	<b>10</b>
- <b>2 Hình ellipse chéo nhau:.....</b>	<b>10</b>
<b>j. Phóng to/ thu nhỏ x2.....</b>	<b>11</b>
<b>2. Mô tả các hàm của chương trình .....</b>	<b>11</b>
<b>a. Change_brightness .....</b>	<b>11</b>
- <b>Mã giả:.....</b>	<b>11</b>
- <b>Mô tả hàm:.....</b>	<b>12</b>
<b>b. Adjust_contrast.....</b>	<b>12</b>
- <b>Mã giả:.....</b>	<b>12</b>
- <b>Mô tả hàm:.....</b>	<b>13</b>
<b>c. Flip_image:.....</b>	<b>13</b>
- <b>Mã giả:.....</b>	<b>13</b>
- <b>Mô tả hàm:.....</b>	<b>14</b>
<b>d. Grayscale_image .....</b>	<b>14</b>

- Mã giả:.....	14
- Mô tả hàm .....	15
e. Sepia_image.....	15
- Mã giả:.....	15
- Mô tả hàm: .....	16
f. Gaussian_kernel.....	16
- Mã giả:.....	16
- Mô tả hàm .....	17
g. Convolve2d .....	17
- Mã giả:.....	17
- Mô tả hàm: .....	18
h. Blur_image .....	18
- Mã giả:.....	18
- Mô tả hàm: .....	19
i. Sharpen_image .....	19
- Mã giả:.....	19
- Mô tả hàm: .....	20
j. Center_crop .....	20
- Mã giả:.....	20
- Mô tả hàm: .....	21
k. Circle_crop .....	22
- Mã giả:.....	22
- Mô tả hàm: .....	22
l. Create_ellipse_mask .....	23
- Mã giả:.....	23
- Mô tả hàm .....	24
m. Zoom_image.....	25
- Mã giả:.....	25
- Mô tả hàm: .....	25

n.	Main.....	26
-	Mô tả hàm:.....	26
3.	Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng .....	26
	Ảnh ban đầu .....	31
a.	Làm mờ ảnh với Gaussian Kernel size = 5 .....	31
b.	Làm sắc nét ảnh .....	32
c.	Phóng to/thu nhỏ 2x.....	33
-	Phóng nhỏ 2x .....	33
-	Phóng to 2x.....	34
4.	Tài liệu tham khảo .....	35
-	Change contrast .....	35
-	Gray scale image .....	35
-	Sharpen image .....	35
-	Sepia image.....	35
-	Ellipse shaped image.....	35

## 1. Ý tưởng thực hiện

### a. Thay đổi độ sáng

- Để thay đổi độ sáng của ảnh, ta cần tăng lên giá trị của mỗi ảnh điểm lên (tối đa 255 và tối thiểu -255) để tinh chỉnh độ sáng
- Giá trị điểm ảnh càng về -255 thì càng mờ, giá trị điểm ảnh càng tới 255 thì càng sáng
- Phương pháp sử dụng ở đây sẽ là **cộng k scalar (trong chương trình sẽ là alpha) vào ma trận**
- Với mỗi vector màu  $v = [a, b, c]$  thì ta cần cộng vào một giá trị  $\alpha$ , khi đó  $v = [a + \alpha, b + \alpha, c + \alpha]$
- Ngoài ra cần phải đảm bảo các giá trị điểm phải nằm trong khoảng  $[0, 255]$

### b. Thay đổi độ sắc nét

- Tương tự như thay đổi độ sáng, nhưng thay vì cộng, chúng ta sẽ nhân với một k scalar
- Trong đó **factor =  $(259 * (\alpha + 255)) / (255 * (259 - \alpha))$**   
**(alpha là k scalar)**
- Sau đó điều chỉnh giá trị từng điểm màu bằng công thức:  **$img = 128 + factor * (img - 128)$**  (128 là giá trị trung bình của khoảng  $[0, 255]$ )
- **Việc trừ 128:** Trước khi áp dụng hệ số điều chỉnh tương phản (factor), trừ 128 từ mỗi giá trị pixel giúp dịch chuyển điểm trung bình của dải giá trị pixel về 0. Điều này tạo điều kiện thuận lợi cho việc áp dụng phép nhân với hệ số tương phản, vì hệ số tương phản được thiết kế để khuếch đại hoặc giảm các giá trị xung quanh 0.

- **Việc cộng 128:** Sau khi áp dụng hệ số tương phản, cộng lại 128 vào mỗi giá trị pixel để dịch chuyển các giá trị trở lại khoảng giá trị ban đầu (0-255). Điều này đảm bảo rằng các giá trị pixel được điều chỉnh vẫn nằm trong khoảng hợp lệ của ảnh 8-bit.

### c. Lật ảnh (Ngang, Dọc)

- ~~Lật ngang:~~ đảo ngược các dòng từ trái qua phải và đồng thời đảo các vector ngược từ dưới lên

Vd:  
 [[[172, 172, 172],  
 [172, 172, 172],  
 [171, 171, 171],  
 [176, 175, 181],  
 [176, 175, 181],  
 [169, 170, 175]],  
 [[172, 172, 172],  
 [172, 172, 172],  
 [171, 171, 171],  
 [177, 176, 182],  
 [177, 176, 182],  
 [170, 171, 176]]]



[[[169 170 175]  
 [176 175 181],  
 [176 175 181],  
 [171 171 171],  
 [172 172 172],  
 [172 172 172]],  
 [[170 171 176],  
 [177 176 182],  
 [177 176 182],  
 [171 171 171],  
 [172 172 172],  
 [172 172 172]]]

- Lật dọc: Chỉ cần lật theo chiều dọc từ dưới lên  
 Vd:

[[[172, 172, 172],  
 [172, 172, 172],  
 [171, 171, 171],  
 [176, 175, 181],  
 [176, 175, 181],  
 [169, 170, 175]],  
 [[172, 172, 172],  
 [172, 172, 172],  
 [171, 171, 171],  
 [177, 176, 182],  
 [177, 176, 182],  
 [170, 171, 176]]]



[[[172 172 172],  
 [172 172 172],  
 [171 171 171],  
 [177 176 182],  
 [177 176 182],  
 [170 171 176]],  
 [[172 172 172],  
 [172 172 172],  
 [171 171 171],  
 [176 175 181],  
 [176 175 181],  
 [169 170 175]]]

#### d. RGB thành ảnh xám

- Ở đây, chúng ta sẽ sử dụng phương pháp weighted thông qua công thức  $\text{Gray} = 0.2989 \times R + 0.5870 \times G + 0.1140 \times B$
- Để có hệ số sau thì chúng ta cần một số thông tin như sau:
  - Mắt người nhạy cảm với màu xanh lá nhất nên hệ số của màu xanh sẽ là cao nhất 0.587
  - Mắt người nhạy cảm trung bình với màu đỏ nên hệ số của màu đỏ sẽ là cao nhì 0.2989
  - Mắt người nhạy cảm ít nhất với màu xanh dương nên hệ số của màu xanh dương thấp nhất 0.114

#### e. RGB thành ảnh sepia

- Tương tự như cách chuyển RGB thành ảnh xám, nhưng ở đây chúng ta sẽ chỉnh sửa giá trị R, G, B
  - $R = 0.393 \times R + 0.769 \times G + 0.189 \times B$
  - $G = 0.349 \times R + 0.686 \times G + 0.168 \times B$
  - $B = 0.272 \times R + 0.534 \times G + 0.131 \times B$

#### f. Làm mờ ảnh

- Chúng ta, sẽ có 2 bước để thực hiện lần lượt: Tạo ma trận Gaussian Kernel và Convolve
- **Ma trận Gaussian Kernel:**
  - Công thức:  $f(x) = \frac{1}{2\pi\sigma^2} e^{-\frac{(x-x_0)^2 + (y-y_0)^2}{2\sigma^2}}$
- **Convolve:**
  - Đây là quá trình thêm từng phần tử của ảnh vào các phần tử lân cận cục bộ của nó và được tính theo trọng số của kernel bằng từng lớp của kênh màu, sau đó chúng ta gộp chung lại
  - Công thức:

$$\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix} * \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1n} \\ y_{21} & y_{22} & \cdots & y_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{m1} & y_{m2} & \cdots & y_{mn} \end{bmatrix} = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} x_{(m-i)(n-j)} y_{(1+i)(1+j)}$$

- **Ví dụ:**

**Matrix**

```
[[10, 10, 10, 10, 10],  
[10, 20, 20, 20, 10],  
[10, 20, 30, 20, 10],  
[10, 20, 20, 20, 10],  
[10, 10, 10, 10, 10]]
```

**Gaussian Kernel**

```
[[0.07511361, 0.1238414, 0.07511361],  
[0.1238414, 0.20417996, 0.1238414],  
[0.07511361, 0.1238414, 0.07511361]]
```

**Output:**

```
[[16.02089978 18.49772784 16.02089978],  
[18.49772784 22.04179956 18.49772784],  
[16.02089978 18.49772784 16.02089978]]
```

+ Trong đó:

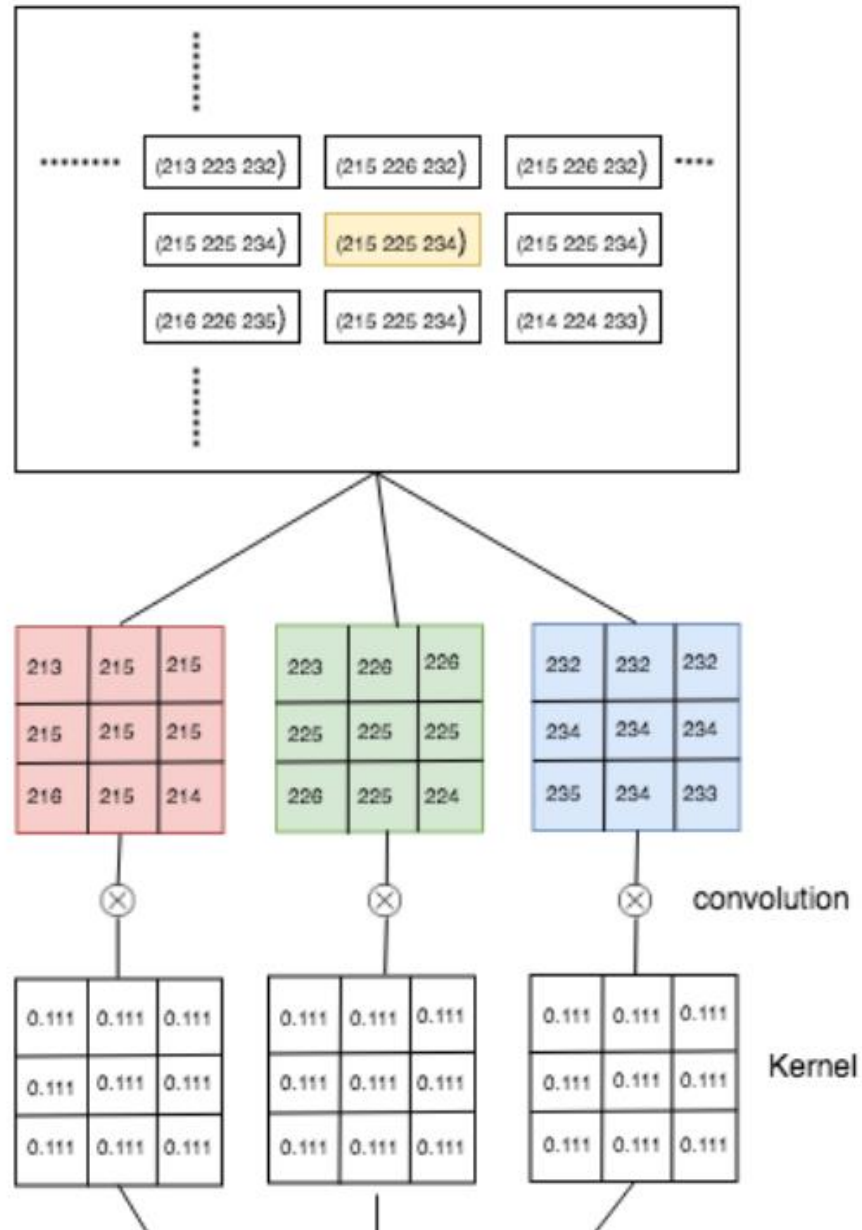
Phần tử đầu của Output sẽ bằng

$$\begin{bmatrix} 10 & 10 & 10 \end{bmatrix} \cdot \begin{bmatrix} 0.07511361 & 0.1238414 & 0.07511361 \\ 0.1238414 & 0.20417996 & 0.1238414 \\ 0.07511361 & 0.1238414 & 0.07511361 \end{bmatrix} = 16.02$$

Và các phần tử khác cũng sẽ lần lượt như vậy

+ Sau đó tương tự với từng kênh màu ta sẽ có được kết quả như ví dụ sau:





### g. Làm sắc nét ảnh

- Tương tự như cách làm mờ ảnh nhưng thay vì dùng Gaussian Kernel mà sẽ dùng ma trận

[[0, -1, 0],
[-1, 5, -1],
[0, -1, 0]]

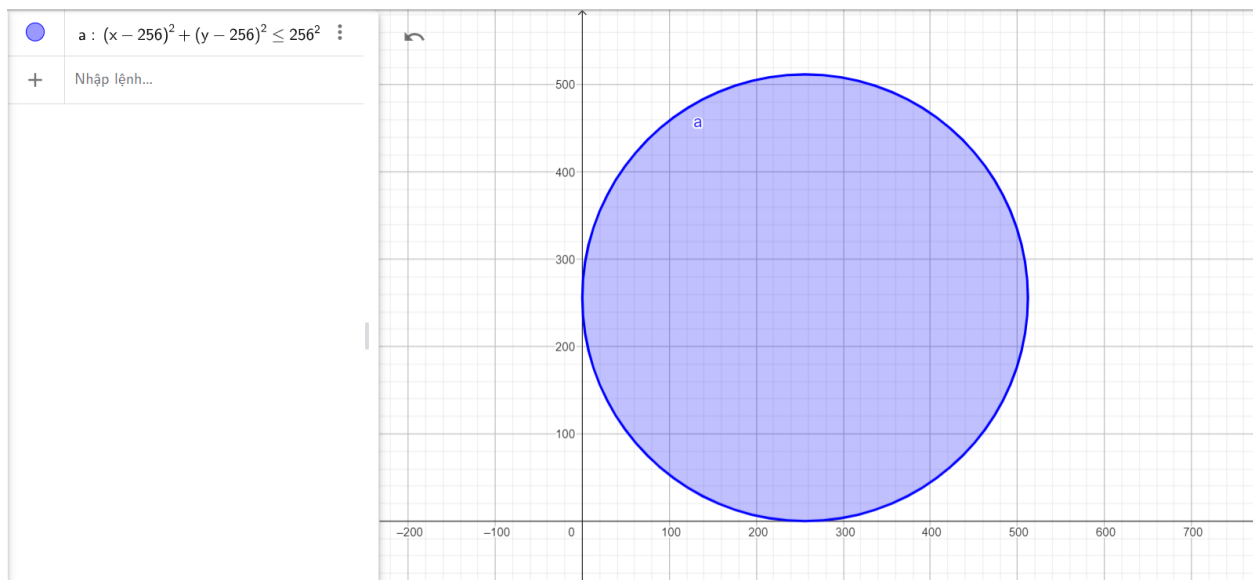
- Tuy nhiên cần phải đảm bảo các điểm ảnh phải nằm trong khoảng [0,255]

#### h. Cắt ảnh theo kích thước

- Đầu tiên, chúng ta cần tính toán chiều dài và chiều rộng mới theo kích thước mong muốn
- Sau đó, cắt ảnh theo độ dài mà mình mong muốn

#### i. Cắt ảnh theo khung

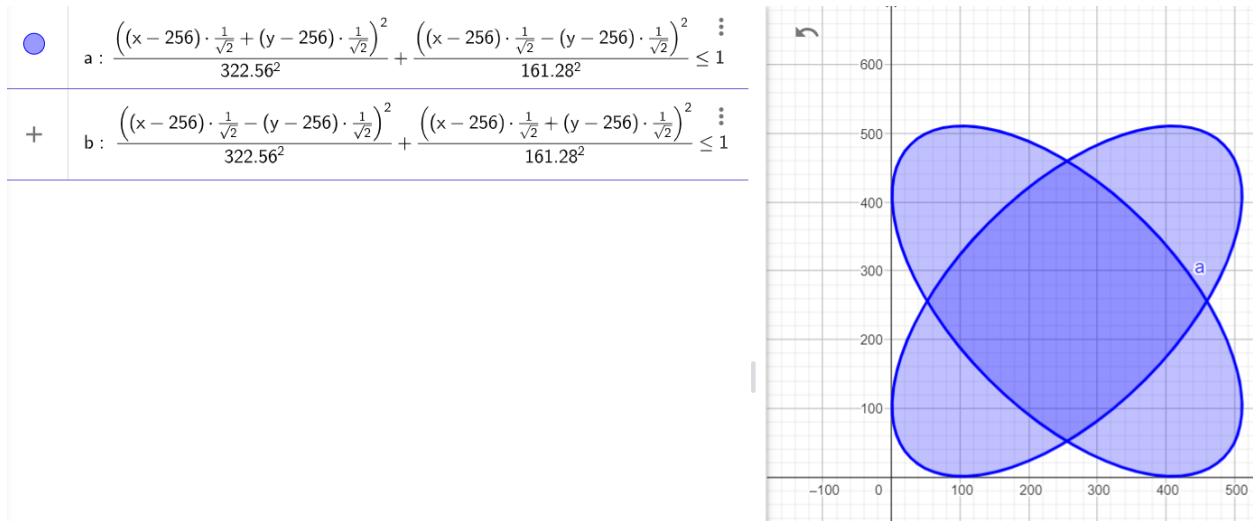
- Ý tưởng chính cho cả cắt ảnh hình tròn và hình ellipse chéo nhau chính là tạo ra mask để có thể cắt hình theo mask đó
- Trong đó, chúng ta sẽ sử dụng mask thông qua **Boolean array** và **phương trình của hình tròn và ellipse**
- **Hình tròn:**
  - Công thức:  $f(x, y) = (x - x_0)^2 + (y - y_0)^2 \leq r^2$
  - Ví dụ (Hình kích cỡ 512 x 512):



- **2 Hình ellipse chéo nhau:**
  - Công thức:

$$a: \left( \frac{((x - x_0) * \cos(45) + (y - y_0) * \sin(45))^2}{(0.63 * d)^2} \right) + \left( \frac{((x - x_0) * \sin(45) - (y - y_0) * (\cos(45)))^2}{(0.315 * d)^2} \right) \leq 1$$

• Ví dụ (Hình kích cỡ 512 x 512):



j. Phóng to/ thu nhỏ x2

- Ý tưởng chung của hai chức năng trên là tính toán chiều dài và rộng mới của hình và sau đó
- Sau đó, sẽ dùng phương pháp nội suy gần nhất (Nearest Neighbor Interpolation) để gán giá trị của từng điểm ảnh gần nhất tương ứng từ hình ảnh gốc
- Phóng to x2:
  - Lấy chiều dài và chiều rộng nhân cho mức độ phóng to
- Thu nhỏ x2:
  - Lấy chiều dài và rộng chia cho mức độ phóng nhỏ

2. Mô tả các hàm của chương trình

a. Change\_brightness

- Mã giả:

```
FUNCTION change_brightness(img, alpha):
```

```

# img: Input image as a NumPy array
# alpha: Alpha value for brightness adjustment (float)

array_alpha = Create NumPy array from alpha with data type
float32

# Adjust the brightness of the image by adding alpha to
eachpixel
img = Add array_alpha to img

# Clip the pixel values to be in the range [0, 255]
img = Clip the values of img to be within [0, 255]

# Convert the data type of img to uint8 (8-bit unsigned integer)
img = Convert img to data type uint8

# Return the new image as a PIL Image object
RETURN PIL Image object created from img
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Alpha: hệ số điều chỉnh độ sáng

- **Output:**

Img: Hình ảnh sau khi chỉnh độ sáng

- **Bước thực hiện:**

B1: Tạo mảng numpy alpha với kiểu float32

B2: Tăng độ sáng của ảnh bằng cách thêm alpha vào mỗi pixel ảnh

B3: Giới hạn giá trị mỗi pixel ảnh bằng hàm **np.clip** trong khoảng **[0,255]** để tránh hiện tượng tràn số

B4: Chuyển lại sang kiểu dữ liệu unit8 bằng **astype(np.uint8)** và chuyển mảng lại thành hình ảnh và trả về ảnh sau khi chỉnh sửa bằng **Image.fromarray**

**b. Adjust\_contrast**

- **Mã giả:**

```

FUNCTION adjust_contrast(img, alpha):
# img: Input image as a NumPy array
# alpha: Contrast adjustment factor (float)

# Convert image to float32 data type

```

```

img = Convert img to float32

# Clip the alpha value to be within the range [-255, 255]
alpha = Clip alpha to be within [-255, 255]

# Calculate the contrast adjustment factor
factor = (259 * (alpha + 255)) / (255 * (259 - alpha))

# Adjust the contrast of the image
img = 128 + factor * (img - 128)

# Clip the pixel values to be within the range [0, 255]
img = Clip the values of img to be within [0, 255]

# Convert the image back to uint8 data type
img = Convert img to data type uint8

# Return the adjusted image as a PIL Image object
RETURN PIL Image object created from img
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Alpha: hệ số điều chỉnh độ sắc nét

- **Output:**

Img: Hình ảnh sau khi chỉnh sửa độ sắc nét

- **Bước thực hiện:**

B1: Chuyển ma trận đầu vào thành kiểu dữ liệu float32

B2: Giới hạn ma trận alpha trong khoảng [-255,255]

B3: Tính toán factor thay đổi độ sắc và theo điểm giữa của mỗi pixel ảnh

B4: Giới hạn lại các giá trị điểm ảnh trong khoảng [0,255] để phù hợp ảnh 8 bit

B5: Chuyển ma trận về dạng dữ liệu unit8 và về lại ảnh

**c. Flip\_image:**

- **Mã giả:**

```

FUNCTION flip_image(img, direction):
    # img: Input image as a NumPy array
    # direction: Direction to flip the image ('horizontal' or
'vertical')

```

```

IF direction == 'horizontal':
    # Flip the image horizontally (left-to-right)
    img = np.fliplr(img)
    PRINT "Horizontal: " and the NumPy array representation of
img
ELSE IF direction == 'vertical':
    # Flip the image vertically (top-to-bottom)
    img = np.flipud(img)
    PRINT "Vertical: " and the NumPy array representation of img

# Convert the NumPy array back to a PIL Image and return it
RETURN pil.Image.fromarray(img)
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Direction: Phương hướng để xoay (horizontal/ vertical)

- **Output:**

Img: Ảnh sau khi xoay

- **Bước thực hiện:**

B1: Nếu là **horizontal**, chúng ta sẽ dùng **np.fliplr**, nếu là **vertical**, chúng ta sẽ dùng **np.flipud**

B2: Sau khi thực hiện xoay chuyển ma trận về lại hình

#### d. Grayscale\_image

- **Mã giả:**

```

FUNCTION grayscale_image(img):
    # img: Input image as a NumPy array

    # Convert the image to grayscale using the luminance formula
    # Weighted sum of the RGB values: 0.2989 * R + 0.5870 * G +
0.1140 * B
    grayscale = np.dot(img[..., :3], [0.2989, 0.5870, 0.1140])

    # Convert the grayscale values to uint8 data type
    grayscale = Convert grayscale to uint8

    # Replicate the grayscale values across the three RGB channels
    img = Stack grayscale values along a new axis to form an RGB
image

```

```
# Return the grayscale image as a PIL Image object
RETURN PIL Image object created from img
END FUNCTION
```

- **Mô tả hàm**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

- **Output:**

Img: Hình ảnh sau khi chuyển sang kênh xám

- **Bước thực hiện:**

B1: Dùng **np.dot** để thực hiện nhân từng điểm màu tương ứng với từng kênh màu, nhân **0.2989** vào **giá trị màu đỏ của mỗi kênh màu**, nhân **0.5870** vào **giá trị màu xanh lá của mỗi kênh màu**, nhân **0.1140** vào **giá trị màu xanh dương của mỗi kênh màu**

B2: Sau đó chuyển về kiểu dữ liệu uint8

B3: Dùng **np.stack** để tạo ra ảnh có ba kênh màu với kênh màu xám được tạo trước đó

B4: Chuyển ma trận trở lại về hình ảnh và trả về

e. **Sepia\_image**

- **Mã giả:**

```
FUNCTION sepia_image(img):
    # img: Input image as a NumPy array with RGB channels

    # Check if the input image is an RGB image
    IF len(img.shape) == 2 OR img.shape[2] != 3:
        RAISE ValueError("Input image must be an RGB image")

    # Convert the image to float for precision before applying the
    sepia filter
    img = Convert img to float

    # Define the sepia filter matrix
    sepia_filter = [[0.393, 0.769, 0.189],
                    [0.349, 0.686, 0.168],
                    [0.272, 0.534, 0.131]]

    # Apply the sepia filter to each pixel
    sepia_img = img @ Transpose of sepia_filter

    # Clip the pixel values to be within the range [0, 255]
    sepia_img = Clip sepia_img to range [0, 255]
```

```

# Convert the image back to uint8 data type
sepia_img = Convert sepia_img to data type uint8

# Return the sepia-toned image as a PIL Image object
RETURN PIL Image object created from sepia_img
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

- **Output:**

Img: Hình ảnh sau khi được chỉnh về dạng sepia

- **Bước thực hiện:**

B1: Chuyển ma trận ảnh về lại kiểu dữ liệu float32

B2: Áp ma trận `sepia_filter` đã được khai báo trước vào ma trận bằng phép toán `@` để nhân ma trận và **`sepia_filter.T`** để biểu thị sự chuyển vị của ma trận `sepia_filter`

B3: Giới hạn các giá trị điểm ảnh trong khoảng [0,255]

B4: Chuyển ma trận trở lại về kiểu dữ liệu uint8 và hình ảnh để trả về

#### f. **Gaussian\_kernel**

- **Mã giả:**

```

FUNCTION gaussian_kernel(size, sigma):
    # size: The size of the Gaussian kernel (NxN matrix)
    # sigma: The standard deviation of the Gaussian distribution

    # Initialize the kernel using the Gaussian formula
    kernel = np.fromfunction(
        FUNCTION (x, y):
            # Calculate the Gaussian value for each element (x, y)
            # Normalize the result by (1 / (2 * π * sigma^2))
            gaussian_value = (1 / (2 * π * sigma^2)) * exp(
                - ((x - (size - 1) / 2)^2 + (y - (size - 1) / 2)^2)
            / (2 * sigma^2)
            )
            RETURN gaussian_value
        END FUNCTION,
        shape=(size, size) # The shape of the kernel
    )

```



```

    # Normalize the kernel by dividing each element by the sum of
    all elements
    normalized_kernel = kernel / sum of all elements in kernel

    RETURN normalized_kernel
END FUNCTION

```

- **Mô tả hàm**

- **Input:**

Size: Kích cỡ của ma trận Gaussian Kernel

Sigma: Độ lệch chuẩn phân bố Gaussian

- **Output:**

Normalized\_kernel: Ma trận Gaussian Kernel

- **Bước thực hiện:**

B1: Sử dụng **np.fromfunction** để tạo ra ma trận kernel, với mỗi vị trí (x,y) trong ma trận được tính toán theo công thức đã nói ở trên

B2: Sau khi tạo kernel, chuẩn hóa nó bằng cách chia từng phần tử cho tổng của tất cả các phần tử trong kernel. Điều này đảm bảo tổng các phần tử kernel bằng 1, duy trì cường độ hình ảnh khi kernel được sử dụng để convolve2d

**g. Convolve2d**

- **Mã giả:**

```

FUNCTION convolve2d(image, kernel):
    # image: The input image as a 2D NumPy array
    # kernel: The convolution kernel as a 2D NumPy array

    # Get the dimensions of the kernel
    kernel_height = height of the kernel
    kernel_width = width of the kernel

    # Get the dimensions of the image
    image_height = height of the image
    image_width = width of the image

    # Calculate the dimensions of the output image
    output_height = image_height - kernel_height + 1
    output_width = image_width - kernel_width + 1

    # Create a strided view of the image

```

```

    strided_image = CREATE strided view of the image using
np.lib.stride_tricks.as_strided
    with shape (output_height, output_width, kernel_height,
kernel_width)
    and strides image.strides + image.strides

    # Perform the convolution using Einstein summation
    output = np.einsum('ijkl,kl->ij', strided_image, kernel)

    RETURN output
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Kernel: Ma trận Gaussian Kernel

- **Output:**

Output: Ma trận convolve2d

- **Bước thực hiện:**

B1: Đầu tiên tính toán kích thước của hình ảnh đầu ra dựa trên kích thước ảnh ban đầu và kernel

B2: Sau đó, dùng hàm **np.lib.stride\_tricks.as\_strided** để tạo ra strided view của hình ảnh để có thể cho phép trích xuất mảng con từ hình ảnh tương ứng với kích thước kernel tại mỗi vị trí. Tham số hình dạng xác định kích thước của chế độ xem mới, trong khi các bước kiểm soát kích thước bước khi di chuyển sang bản vá tiếp theo. Sự kết hợp **image.strides + image.strides** đảm bảo rằng các miếng vá trượt trên hình ảnh một cách chính xác

B3: Sử dụng **np.einsum('ijkl,kl->ij',strid\_image, kernel)** để thực hiện convolve2d. Hoạt động này tính toán một cách hiệu quả tổng các tích số theo từng phần tử giữa kernel và từng bản vá được trích xuất từ hình ảnh, dẫn đến kết quả đầu ra được ma trận covolve2d

## h. Blur\_image

- **Mã giả:**

```

FUNCTION blur_image(img, size, sigma):
    # img: The input RGB image as a 3D NumPy array
    # size: The size of the Gaussian kernel
    # sigma: The standard deviation of the Gaussian distribution

```

```

# Generate the Gaussian kernel
kernel = gaussian_kernel(size, sigma)

# Initialize the output blurred image array with the same shape
as the input image
blurred_img = CREATE zero array with the same shape as img

# Apply convolution to each channel (R, G, B) separately
FOR each channel in img (e.g., R, G, B):
    # Apply the convolve2d function to the current channel with
the Gaussian kernel
    blurred_channel = convolve2d(channel, kernel)

    # Stack the blurred channels to form the final blurred image
    blurred_img = np.dstack(blurred_channel for each channel)

# Convert the blurred image to uint8 data type
blurred_img = Convert blurred_img to uint8

# Return the blurred image as a PIL Image object
RETURN PIL Image object created from blurred_img
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Size: Kích thước của Gaussian Kernel

Sigma: Độ lệch chuẩn của phân bố Gaussian

- **Output:**

Img: Hình ảnh sau khi làm mờ

- **Bước thực hiện:**

**B1: Tạo ma trận Gaussian Kernel**

**B2:** Áp dụng convolve2d cho từng kênh màu (R, G, B), sau đó dùng hàm **np.dstack** để gộp lại thành ảnh có ba kênh màu RGB như cũ

**B3:** Hình ảnh mờ được chuyển sang kiểu dữ liệu uint8, đảm bảo giá trị pixel nằm trong khoảng [0, 255], phù hợp để hiển thị và lưu dưới dạng định dạng ảnh chuẩn và trả ảnh về.

**i. Sharpen\_image**

- **Mã giả:**

```

FUNCTION sharpen_image(img):

```

```

# img: The input RGB image as a 3D NumPy array

# Define the sharpening kernel
kernel = ARRAY([[0, -1, 0],
                [-1, 5, -1],
                [0, -1, 0]])

# Initialize the output sharpened image array with the sameshape
as the input img
sharpened_img = CREATE zero array with the same shape as img

# Apply convolution to each channel (R, G, B) separately
FOR each channel in img (e.g., R, G, B):
    # Apply the convolve2d function to the current channel with
the sharpening kernel
    sharpened_channel = convolve2d(channel, kernel)

    # Stack the sharpened channels to form the final sharpened
image
    sharpened_img = np.dstack(sharpened_channel for each
channel)

# Clip the pixel values to be in the range [0, 255]
sharpened_img = CLIP values in sharpened_img to [0, 255]

# Convert the sharpened image to uint8 data type
sharpened_img = CONVERT sharpened_img to uint8

# Return the sharpened image as a PIL Image object
RETURN PIL Image object created from sharpened_img
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

- **Output:**

Img: Hình ảnh sau khi làm mờ

- **Bước thực hiện:**

Các bước thực hiện phần lớn tương tự như hàm trên tuy nhiên thay vì dùng Gaussian Kernel thì chúng ta sẽ dùng Kernel được khai báo trước để làm sắc nét ảnh

**j. Center\_crop**

- **Mã giả:**

```

FUNCTION center_crop(img, new_height, new_width):
    # img: The input image as a NumPy array
    # new_height: The height of the cropped image
    # new_width: The width of the cropped image

    # Get the original dimensions of the image
    height = height of img
    width = width of img

    # Calculate the top and left coordinates for the crop
    top = (height - new_height) // 2
    left = (width - new_width) // 2

    # Calculate the bottom and right coordinates for the crop
    bottom = top + new_height
    right = left + new_width

    # Crop the image from the center using the calculated
coordinates
    cropped_image = img[top:bottom, left:right]

    # Return the cropped image as a PIL Image object
    RETURN PIL Image object created from cropped_image
END FUNCTION

```

- **Mô tả hàm:**

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

New\_height: Chiều cao của ảnh đã cắt mong muốn.

New\_width: Chiều rộng của ảnh đã cắt mong muốn.

- **Output:**

Img: Hình ảnh sau khi cắt

- **Bước thực hiện:**

B1: Tính toán các tọa độ cho ảnh mới trong đó:

+ Tọa độ trên cùng được tính làm điểm bắt đầu cắt xén theo hướng dọc, dựa trên chênh lệch giữa chiều cao ban đầu và chiều cao mới, chia cho 2.

+ Tọa độ bên trái được tính tương tự cho chiều ngang, dựa trên sự chênh lệch giữa chiều rộng ban đầu và chiều rộng mới, chia cho 2.

Tọa độ dưới cùng được xác định bằng cách thêm new\_height vào trên cùng.

+ Tọa độ bên phải được xác định bằng cách thêm new\_width vào bên trái.

B2: Hình ảnh được cắt bằng cách sử dụng tọa độ được tính toán, trích xuất phần trung tâm của hình ảnh.

B3: Trả lại hình ảnh dạng hình

#### k. Circle\_crop

- Mã giả:

```
FUNCTION circle_crop(img):  
    # img: The input image as a NumPy array  
  
    # Get the height and width of the image  
    h = height of img  
    w = width of img  
  
    # Calculate the center of the image  
    center = (w // 2, h // 2)  
  
    # Calculate the radius of the circle  
    radius = MIN(center[0], center[1], w - center[0], h - center[1])  
  
    # Create a circular mask  
    Y, X = CREATE grid arrays using np.ogrid for the dimensions of  
the image (h, w)  
    dist_from_center = sqrt((X - center[0])^2 + (Y - center[1])^2)  
    circular_mask = dist_from_center <= radius  
  
    # Apply the mask to the image, setting non-mask areas to black  
    img = img * circular_mask[:, :, NEWAXIS]  
  
    # Return the circular-cropped image as a PIL Image object  
    RETURN PIL Image object created from img  
END FUNCTION
```

- Mô tả hàm:

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

- **Output:**

Img: Hình ảnh sau khi cắt

- **Bước thực hiện:**

B1: Tính toán bán kính và tâm của hình tròn

B2: Tạo một grid chỉ số Y và X được tạo ra bằng hàm **np.ogrid**, **dis\_from\_center** là bán kính hình tròn được tính bằng **khoảng cách Euclide**

B3: Tạo mask bằng công thức hình tròn được nói ở trong phần ý tưởng

B4: Mask được áp dụng cho hình ảnh bằng cách nhân nó với mask. Các pixel bên ngoài vùng hình tròn được đặt thành màu đen (0) do phép nhân với các giá trị sai

B5: Trả về hình ảnh với dạng hình

## I. Create\_ellipse\_mask

### - Mã giả:

```
FUNCTION create_ellipse_mask(img):
    # img: The input image as a NumPy array

    # Get the height and width of the image
    h = height of img
    w = width of img

    # Calculate the semi-major axis (a) and semi-minor axis (b) for
the ellipse
    d = min(h, w) / 2
    a = d * 1.26
    b = a / 2

    # Calculate the center of the ellipse
    center = (w // 2, h // 2)

    # The ellipse equation is:
    #  $(x - center[0])^2 / a^2 + (y - center[1])^2 / b^2 \leq 1$ 

    # Create grid arrays for the coordinates of the image
    Y, X = CREATE grid arrays using np.ogrid for the dimensions of
the image (h, w)

    # Define sin(45 degrees) and cos(45 degrees)
    sin_45 = sin(pi / 4)
    cos_45 = cos(pi / 4)

    # Calculate the numerator and denominator for the ellipse
equation
    numerator_1 = ((X - center[0]) * cos_45 + (Y - center[1]) *
sin_45)^2
```

```

        numerator_2 = ((X - center[0]) * sin_45 - (Y - center[1]) *
cos_45)^2
        denominator_1 = a^2
        denominator_2 = b^2

        # Create boolean masks to check if the point is inside the
ellipse
        mask1 = numerator_1 / denominator_1 + numerator_2 /
denominator_2 <= 1
        mask2 = numerator_2 / denominator_1 + numerator_1 /
denominator_2 <= 1

        # Combine the two masks
        mask = mask1 OR mask2

        # Apply the mask to the image, setting non-mask areas to black
        img = img * mask[:, :, NEWAXIS]

        # Return the masked image as a PIL Image object
        RETURN PIL Image object created from img
END FUNCTION

```

#### - Mô tả hàm

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

- **Output:**

Img: Hình ảnh sau khi cắt theo 2 hình ellipse chéo

- **Bước thực hiện:**

B1: Tính bán trục lớn (a) và bán trục nhỏ (b) của hình elip được tính bằng cách sử dụng các kích thước của hình ảnh. Tâm của hình elip được đặt ở điểm giữa của hình ảnh.

B2: Sử dụng phương trình tổng quát của một hình elip quay 45 độ để tạo ra hai masks, mask1 và mask2.

B3: Tạo Y và X là các mảng lưới biểu thị tọa độ pixel trong ảnh.

B4: Boolean mask được tạo để xác định pixel nào nằm bên trong hình elip dựa trên phương trình hình elip. Các mặt nạ được kết hợp để tạo thành một mặt nạ duy nhất.

B5: Mask được áp dụng cho hình ảnh, đặt các pixel bên ngoài hình elip thành màu đen (0).

B6: Trả về hình ảnh với dạng hình



### m. Zoom\_image

#### - Mã giả:

```
FUNCTION zoom_image(image, zoom_factor, zoom_in=True):
    # image: The input image as a NumPy array
    # zoom_factor: The factor by which to zoom the image
    # zoom_in: Boolean flag to determine zoom in or zoom out

    # Get the height and width of the input image
    height = height of image
    width = width of image

    IF zoom_in:
        # Calculate the new dimensions for zooming in
        new_width = int(width * zoom_factor)
        new_height = int(height * zoom_factor)
    ELSE:
        # Calculate the new dimensions for zooming out
        new_width = int(width / zoom_factor)
        new_height = int(height / zoom_factor)

    # Create a new array for the zoomed image
    zoomed_image = CREATE array with shape (new_height, new_width)

    FOR each i from 0 to new_height:
        FOR each j from 0 to new_width:
            # Map the new coordinates to the original image
            coordinates
            original_i = int(i * height / new_height)
            original_j = int(j * width / new_width)
            # Assign the pixel value from the original image
            zoomed_image[i, j] = image[original_i, original_j]
        END FOR
    END FOR

    # Convert the zoomed array back to an image format
    RETURN PIL Image object created from zoomed_image
END FUNCTION
```

#### - Mô tả hàm:

- **Input:**

Img: Hình ảnh đầu vào dưới dạng mảng numpy

Zoom\_factor: Giá trị float biểu thị hệ số thu phóng

Zoom\_in: cho biết phóng to (True) hay thu nhỏ (False)

- **Output:**

Img: Hình ảnh sau khi phóng to thu nhỏ

- **Bước thực hiện:**

B1: Dựa trên lựa chọn zoom in hay out mà kích cỡ sẽ được tính toán, nếu là zoom in thì kích thước mới sẽ được nhân với zoom\_factor còn zoom out thì sẽ chia

B2: Ánh xạ các tọa độ này trở lại tọa của hình gốc

B3: Trả về hình ảnh với dạng hình

**n. Main**

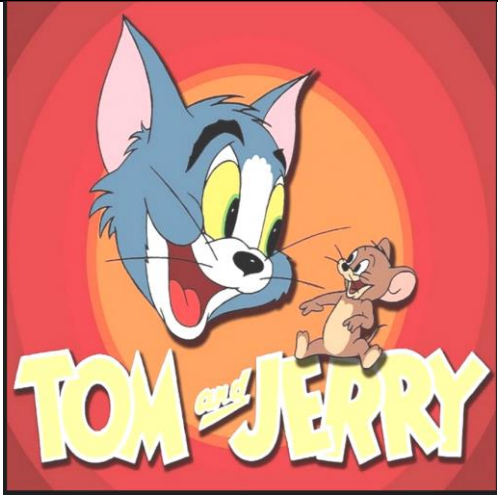
- **Mô tả hàm:**




- **Bước thực hiện:**




B1: Nhận vào link ảnh


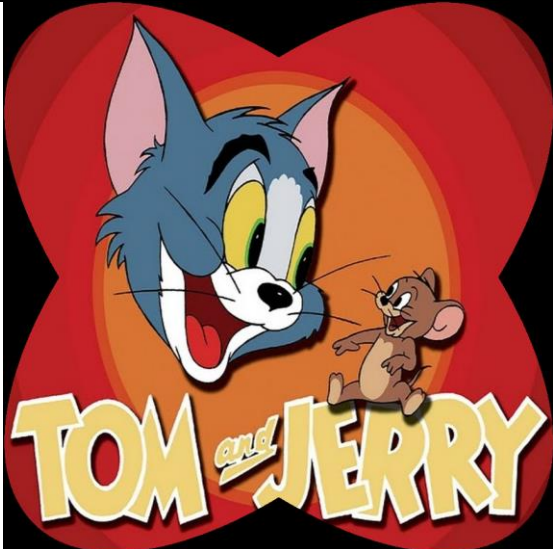
B2: Nhận vào lựa chọn dựa vào đó thao tác trên ảnh

**3. Bảng đánh giá mức độ hoàn thành và hình ảnh kết quả cho từng chức năng**

STT	Chức năng/Hàm	Mức độ hoàn thành	Ảnh kết quả
1	Thay đổi độ sáng	100%	 Độ sáng 50

2	Thay đổi độ tương phản	100%	 <p>Độ tương phản 50</p>
3.1	Lật ảnh ngang	100%	
3.2	Lật ảnh dọc	100%	

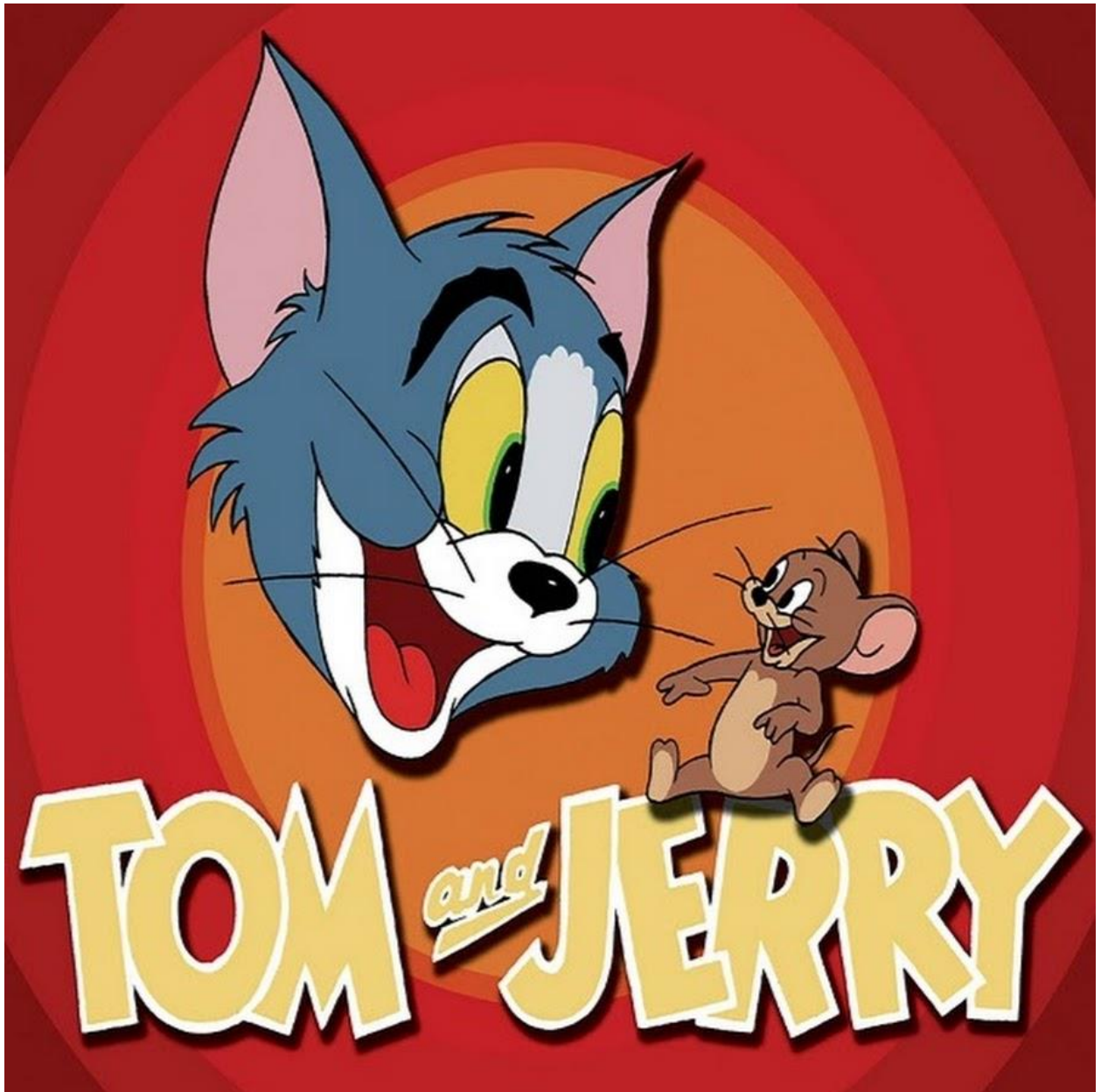
4.1	RGB – Xám	100%	
4.2	RGB – Sepia	100%	
5.1	Làm mờ ảnh	100%	Xem ảnh bên dưới để thấy sự khác biệt
5.2	Làm sắc nét ảnh	100%	Xem ảnh bên dưới để thấy sự khác biệt
6	Cắt ảnh theo kích thước	100%	 <p>Kích thước 100x100</p>

7.1	Cắt ảnh theo khung tròn	100%	
7.2	Cắt ảnh theo khung ellipse	100%	

8	Hàm main	100%	<p>Choose a function:</p> <ul style="list-style-type: none"> <li>0. All functions</li> <li>1. Change brightness</li> <li>2. Adjust contrast</li> <li>3. Flip image</li> <li>4. Grayscale image</li> <li>5. Sepia image</li> <li>6. Blur image</li> <li>7. Sharpen image</li> <li>8. Center crop</li> <li>9. Circle crop</li> <li>10. Create ellipse mask</li> <li>11. Zoom at</li> </ul>
9	Phóng to/Thu nhỏ 2x	100%	Xem ảnh dưới để thấy sự khác biệt



Ảnh ban đầu

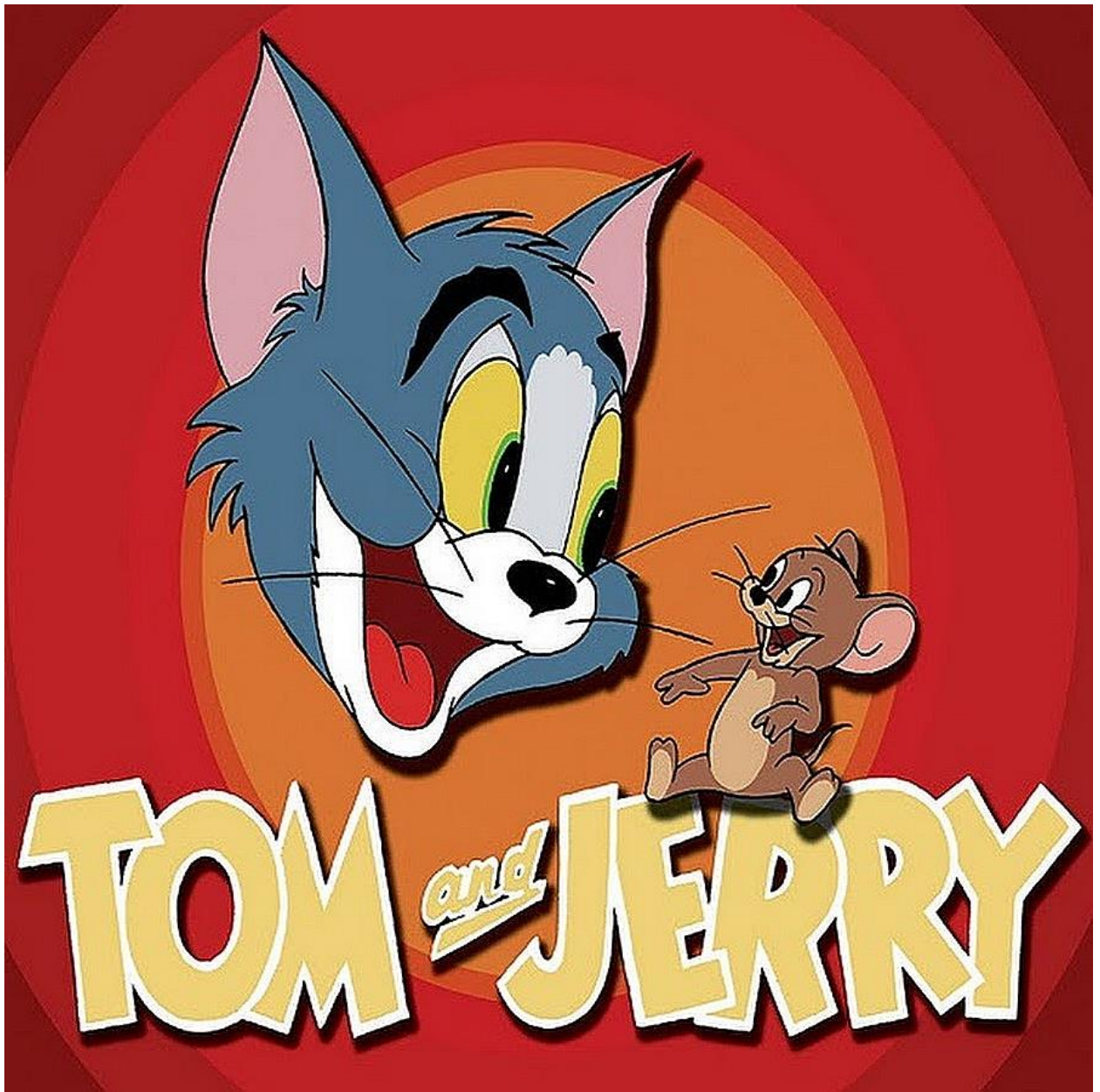


a. Làm mờ ảnh với Gaussian Kernel size = 5

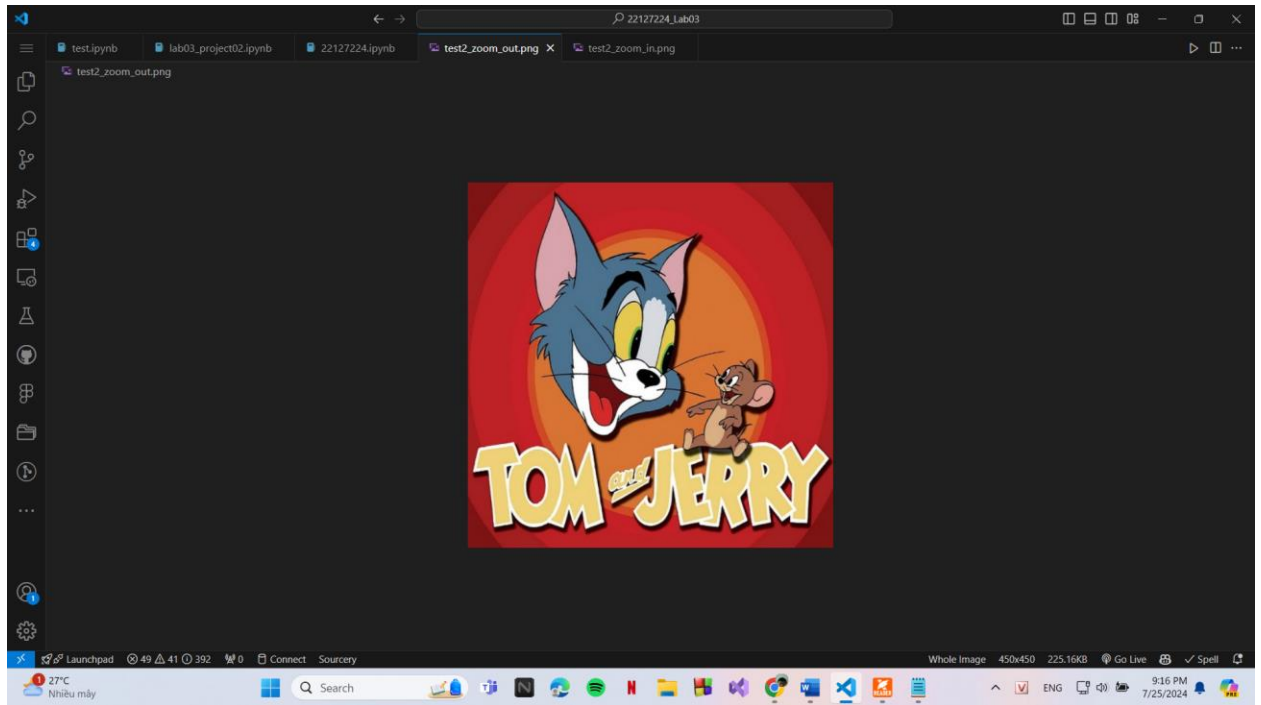


**b. Làm sắc nét ảnh**

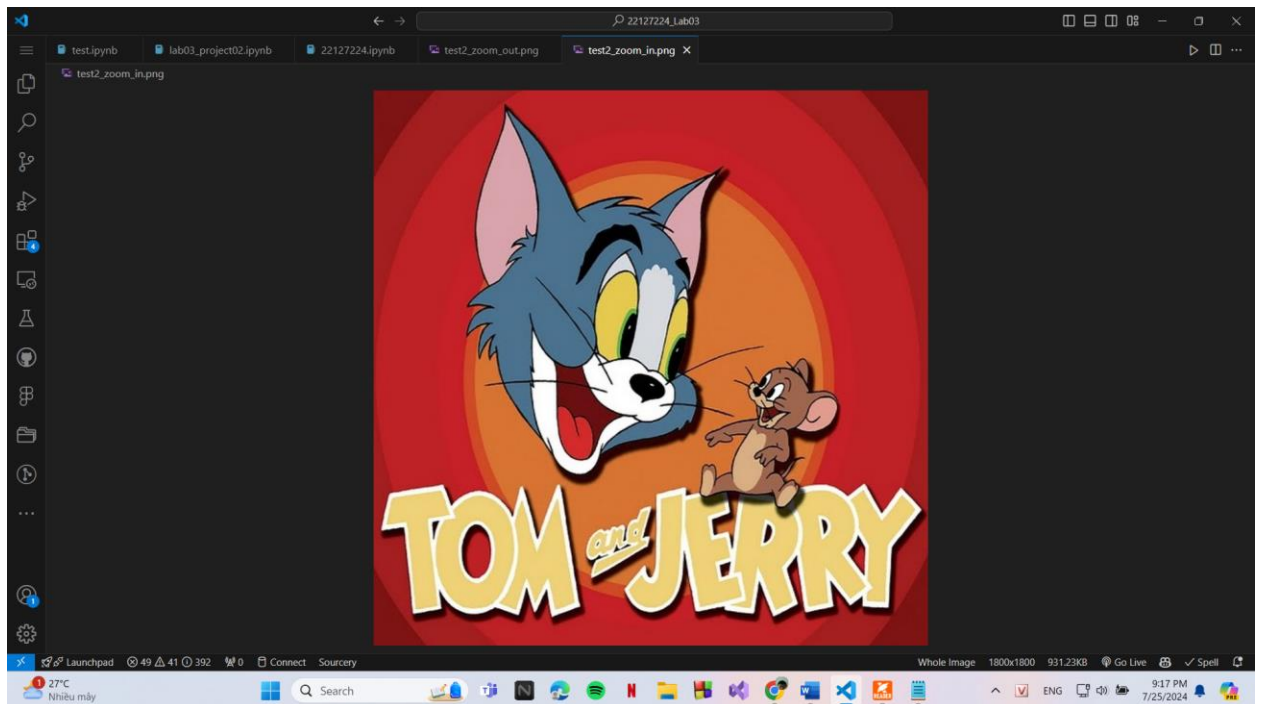




- c. Phóng to/thu nhỏ 2x  
- Phóng nhỏ 2x



- Phóng to 2x



#### 4. Tài liệu tham khảo

- **Change contrast**  
<https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/>
- **Gray scale image**  
<https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python>  
[https://e2eml.school/convert\\_rgb\\_to\\_grayscale](https://e2eml.school/convert_rgb_to_grayscale)
- **Sharpen image**  
<https://pages.stat.wisc.edu/~mchung/teaching/MIA/reading/diffusion.gaussian.kernel.pdf.pdf>  
<https://stackoverflow.com/questions/37095783/how-is-a-convolution-calculated-on-an-image-with-three-rgb-channels>  
<https://llego.dev/posts/boolean-indexing-masking-in-numpy/>
- **Sepia image**  
<https://stackoverflow.com/questions/36434905/processing-an-image-to-sepia-tone-in-python>
- **Ellipse shaped image**  
<https://www.cuemath.com/geometry/ellipse/>