VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF SCIENCE

FACULTY OF INFORMATION TECHNOLOGY

PROJECT REPORT

Image Processing

Presented by:	**Lecturers**
Trương Công Thiên Phú 23127455 23CLC05	Vu Quoc Hoang, MSc Tran Thi Thao Nhi, MSc Nguyen Ngoc Toan

Ho Chi Minh City, July 25th 2025

Contents

I. Introduction

This project focuses on implementing core image processing techniques using Python. In particular, the libraries that this project can uses are restricted, only ***Image*** from **PIL** (for reading, writting image), **numpy** (for matrix computing), ***pyplot*** from **matplotlib** (for showing image), and **colorsys** (for converting RGB to HSL) are allowed.

There are seven main techniques including:

- Brightness processing: change the image brightness
- Contrast processing: change the image contrast
- Flipping processing: flip the image horizontally/vertically
- Grayscale/Sepia conversion: convert image to grayscale/seapia image
- Blur/Sharpening processing: blur/sharpen the image
- Center cropping processing: crop a quarter of the image (center crop)
- Frame-based cropping processing: crop image on circular/elliptical frame

The image used here is the image of Lenna (512 x 512px). This is a standard test image used in the field of digital image processing.

Image 1: *Image of Lenna*

II. Implementation Concepts

An image can be stored as a matrix of pixels. The color image commonly used is RGB image which each pixel stores three channels of color: R (red), G(green), B(blue). Base on that, the image processing can be implemented easily.

1. **Image Brightness Processing**

   In essence, image brightness is a combination of red, green, and blue values. So if the brightness is modified (e.g, increasing the brightness) via RGB directly, the image can be distorted and doesn't look natural.

   Instead, converting the image from RGB to HSL is a better approach. HSL stands for Hue (color type), Saturation (color intensity), and Lightness (perceived brightness). The image brightness can be changed by modifying the Lightness (L) value.

2. **Image Contrast Processing**

   Contrast of the image is the difference in luminance between the light and the dark areas that make the image visible against a background of different luminance. Thus, changing the image contrast involves working with the Lightness (L) value in HSL.

3. **Image Flipping Processing**

   Two types of flipping image that will be implemented are vertical and horizontal image flipping.

   In simple term, flipping image horizontally is reversing the pixel order of each row while flipping image vertically is reversing the order of the pixel rows.

4. **Image Grayscale/Sepia Conversion**

   a. **Grayscale Conversion**

      A grayscale image is an image which is in black and white and brings out shades of gray from the blackest black to the whitest white. While colored image is made up by different color channels (e.g, RGB image), grayscale image consists of only one channels that depict intensity. Thus, to convert an RGB image, every color channel of each pixel has the same value.

   b. **Sepia Conversion**

      Sepia is a reddish-brown color. To convert RGB image to sepia image, a typical algorithm uses a linear transformation on the RGB channels will be applied.

5. **Image Blur/Sharpening Processing**

   In image processing, a kernel, convolution matrix, or mask is a small matrix used for blurring, sharpening. This technique is implemented by doing a convolution between **kernel** and **image.** In simple term, each pixel from the output image is a function of the nearby pixels (including itself) in the input image, the kernel is that function.

6. **Image Cropping Processing**

a. Center Cropping Processing

In simple term, a quarter cropped (center cropped) image is one in which the center portion of the image is extracted. The height and width of the cropped one will be half of the original one.

b. Frame-based Cropping Processing

The easy way to approach is finding the "frame areas", which means if the pixel is outside the frame, its value will be changed to 0 for three color channels.

III. Function Descriptions

1. Available Functions:

There are five available functions:

- *read_img* function receives the image path that leads to the target image then read it by using ***Image*** from **PIL**. The image after reading is converted to matrix by **numpy**. For further convenience, each color channel is divided into 255.0, hence, each value is between 0 and 1.
- *show_img* function receives the image as numpy array and uses ***pyplot*** from **matplotlib** to display on the screen. Note that the function treats each pixel as RGB value.
- *convert_rgb_to_hsl* function converts rgb image to hsl image by using ***rgb_to_hls*** in **colorsys**. After conversion, each pixel stores the hls value.
- *convert_hsl_to_rgb* function converts hsl image to rgb image by using ***hls_to_rgb*** in **colorsys**. This is an essential function before showing the result image because *show_img* can not distinguish hsl and rgb images. If we persist to show the image without converting it to rgb, the result might be different than the real one. (The following images illustrate the consequences of not converting from hsl to rgb image)

Image 2: *The original image*

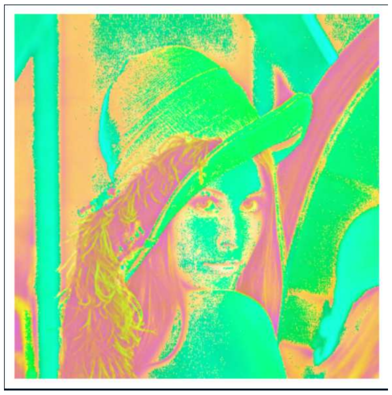


Image 3: *The hsl image (without converting to rgb)*

- *process_image* function receives the target image (as numpy array), and a list of number. Each number in the function list stands for which image processing function will be executed.
    - 0: Every result image will be saved.
    - 1: Brightness processing
    - 2: Contrast processing
    - 3: Flipping processing
    - 4: Grayscale/Sepia conversion
    - 5: Blur/Sharpen processing
    - 6: Crop on quarter (crop at center)
    - 7: Crop based on frame

2. Image Brightness Processing

There are three functions to execute the brightness processing:

- *increase_image_brightness* function takes the hsl image as parameter, it generally increases the Lightness (L) value of each pixel and then stores the new hsl image in another variable and return that new image. The function sets the increasement value of Lightness (L) is 0.3, which means that the new image has the Lightness is higher than the old one 0.3. ***min*** function is used to limit the Lightness to not fall outside the range of 1.
- *decrease_image_brightness* function takes the hsl image as parameter, it generally does the opposite thing of ***increase_image_brightness*** function. ***max*** function is used to limit the Lightness to not be smaller than 0.
- *change_image_brightness* function takes the rgb image (as numpy array) as parameter. It asks user to type the option, "0" means reduces the brightness, and "1" do the opposite (if user input does not match to both, the function increases the brightness). Base on the user input, equivalent function is invoked (*decrease_image_brightness* or *increase_image_brightness*) and the new image with new brightness is returned.

3. Image Contrast Processing

There is one function to handle the contrast processing:

- *change_image_contrast* function takes the RGB image (as numpy array) as parameter. It firstly asks user to type the contrast value which is float number or integer, then the Lightness value is modified based on the formula: L = 0.5 + <contrast value> * (L – 0.5). Let's break down the formula:

   - L obviously stands for Lightness value in HSL image,
   - Contrast value greater than 1: increases contrast (L is increased),
   - Contrast value between 0 and 1: decreases contrast (L is reduced),
   - Contrast value is 1: no change.

- After modifiying the contrast, the function converts the new HSL image to RGB image and returns it.

4. **Image Flipping Processing**

   For this technique, convert from RGB to HSL image is not necessary. There are three functions:

   - *flip_image_horizontally* function takes the RGB image (as numpy array) as parameter, it litterally inverts the order of pixels in each pixel row. For example, the first row having the pixel list [A, B, C, D] will be inverted to [D, C, B, A]. The new image (as numpy array) is returned.
   - *flip_image_vertically* function takes the RGB images (as numpy array) as parameter, it reverses the order of image rows. For instance, the image having the image row [row1, row2, row3, row4] will be reversed to [row4, row3, row2, row1]. The new image is returned.
   - *flip_image* function invokes ***flip_image_horizontally*** and ***flip_image_vertically*** and return two new images respectively.

5. **Image Grayscale/Sepia Conversion**

   change_image_to_grayscale_and_sepia function executes two conversion, grayscale and sepia conversion.

   a. **Grayscale Conversion**

   change_image_to_grayscale bases on the formula: *gray = 0.299 * R + 0.587 * G + 0.114 * B* to convert RGB to grayscale image. In particular, each color channel of every pixel is '*gray*'.

   The question is why not HSL? Luminance component of the HSL color system can not be used in this technique. The L component of the HSL descrie the brightness of a color relative to a base color, if this base color is blue, the image will get a darker color than if this base color is yellow. HSL is very useful if we want to create a lighter or darker version of a color but isn't useful if we want to know how bright a color is.

   Another approach is calculate the average of RGB, but in some cases the result has less accuracy.

b. Sepia Conversion

change_image_to_sepia function solve the problem by applying the following formula:

- New Red = 0.393 * R + 0.769 * G + 0.189 * B
- New Green = 0.349 * R + 0.686 * G + 0.168 * B
- New Blue = 0.272 * R + 0.534 * G + 0.131 * B

If any of these values greater than 1.0, they will be set to be 1.0

6. **Image Blur/Sharpening Processing**

The general expression of a convolution is:

$$g_{x,y} = \omega * f_{x,y} = \sum_{i=-a}^{a} \sum_{j=-b}^{b} \omega_{i,j} f_{x-i,y-j}$$

where g(x, y) is the filtered image, f(x, y) is the original image, ω is the filter kernel. Every element of the filter kernel is considered by -a ≤ i ≤ a and -b ≤ j ≤ b.

For blurring image, *box blur* is applied ("A **box blur** (also known as a box linear filter) is a spatial domain linear filter in which each pixel in the resulting image has a value equal to the average value of its neighboring pixels in the input image. It is a form of low-pass ("blurring") filter. A 3 by 3 box blur ("radius 1") can be written as matrix" - Wikipedia)

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

For sharpening image, *Sharpen* is applied:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

kernel_processing function performs **2D convolution filtering** on a given image using a specified kernel. It takes the image (as numpy array) and the kernel as parameters. Firstly, it calculates the amount of padding needed on each side of the image. For a standard convolution (where the output size is the same as the input size), the padding is typically half the kernel size. Then it pads on the input image. Padding is crucial in convolution to correctly handle pixels near the image borders. Without padding, applying the kernel to border pixels would require accessing pixels outside the image boundaries. After that, convolution is executed by using loop. For each pixel and each color channel, a region of the same size as the kernel is extracted from the padded image; element-wise multiplication is performed between the kernel and the region;

The sum of the result is assigned to the output pixel. Finally, to ensure the pixel values remain within the range from 0 to 1, the output image is clipped. The function return the output image (as numpy array).

blur_and_sharpen_image function simply invokes the ***kernel_processing*** function for blurring image and sharpenning image.

7. Image Cropping Processing

a. Center Cropping Processing

crop_quarter_image function firstly reduced the height, width to half. After that, a new cropped image is created by reading the original image from the new top left pixel to the new bottom right pixel. To get the new top left pixel position, the left edge moves to the right by quarter of the width, the top edge moves down by quarter of the height. To get the new bottom right pixel position, the right edge moves to the left by quarter of the width, the bottom edge moves up by quarter of the height.

b. Frame-based Cropping Processing

i. Circular Frame Cropping Processing

crop_image_on_circular_frame function firstly finds the radius (for circular frame) where its value is calculated by choosing the smaller value between the half height or half width of the image, then dividing to the radius_scale parameter which is set to 1.0 as default. Then it calculates the distance from the center pixel to every pixel by using the Euclidean distance. After that, it checks whether the distance is greater than radius or not, if greater, that pixel will be set to [0, 0, 0] (black color).

ii. Elliptical Frame Cropping Processing

crop_image_on_elliptical_frame function works similarly to the ***crop_image_on_circular_frame*** function, the objective is finding which pixel is in the elliptical area. The function invokes *find_ellipse_mask* twice to find two elliptical areas which each of them lies on the main diagonal or the secondary diagonal. Finally, the “black area” is colored by eliminating the pixels which are in two found frames the remaining is black.

IV. Result

1. Image Brightness Processing

Image 4: *Decrease image brightness*

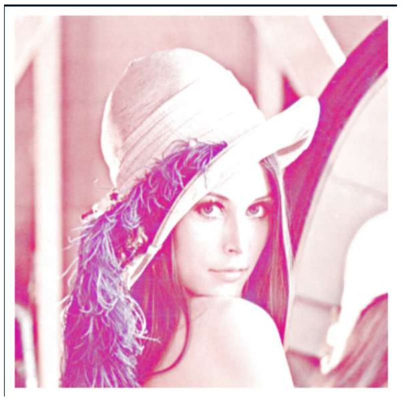


Image 5: *Increase image brightness*

2. **Image Contrast Processing**

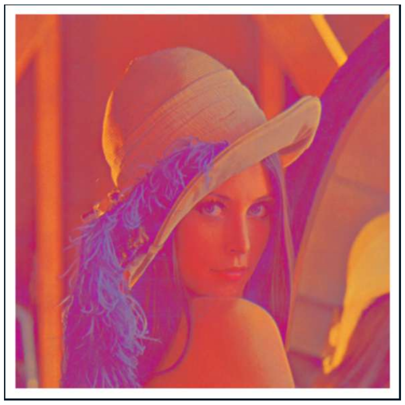
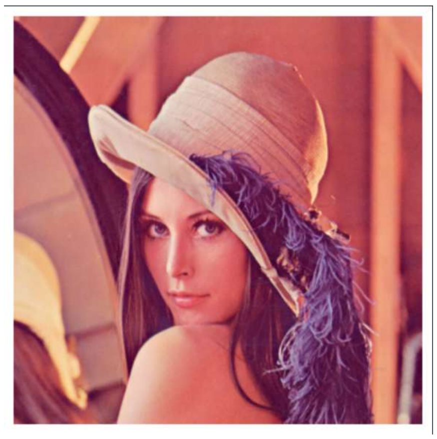


Image 6: *Deacrease image contrast (user input: 3)*

Image 7: *Increase image contrast (user input:0.3)*

3. **Image Flipping Processing**



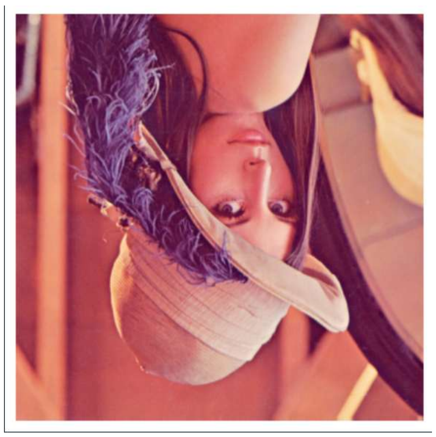


Image 8: Flip image horizontally

Image 9: Flip image vertically

4. Image Grayscale/Sepia Conversion

a. Grayscale Conversion

Image 10: *Grayscale image*

b. Sepia Conversion

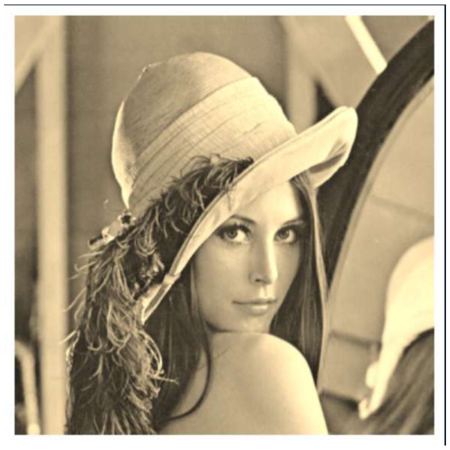


Image 11: *Sepia image*

5. Image Blur/Sharpening Processing

a. Blur Processing

Image 12: *Blur image*

b. Sharp Processing

Image 13: *Sharpen image*

6. Image Cropping Processing

a. Center Cropping Processing

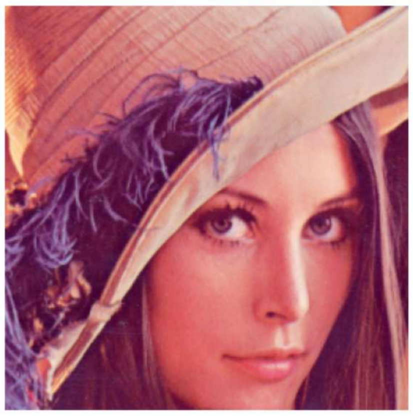


Image 14: *Crop on quarter (center cropping)*

b. **Frame-based Cropping Processing**

i. **Circular Frame Cropping Processing**

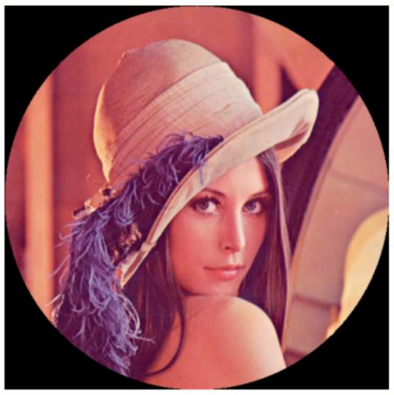


Image 15: *Circular frame cropping image*

ii. **Elliptical Frame Cropping Processing**

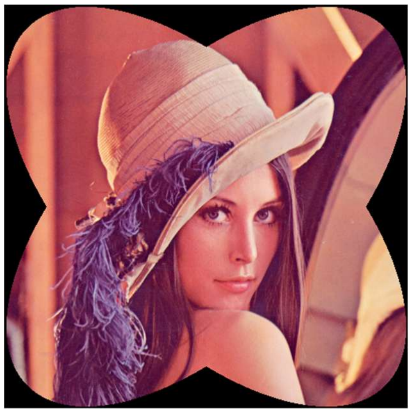


Image 16: *Elliptical frame cropping image*

V. Analysis and Evaluation

1. Image Brightness Processing

By converting from RGB to HSL image, the program can easily handle the brightness modification by adding or subtracting the Lightness (L) value. The larger the added value, the greater the brightness, and vice versa.

2. Image Contrast Processing

One more time, converting from RGB to HSL image is good way to easily handle the contrast modification by changing the Lightness (L) value. The larger the added value, the greater the contrast, and vice versa.

3. Image Flipping Processing

Obviously, reversing the order of each pixel in one row for flipping image horizontally and reversing the order of each row of the image for flipping image vertically are the best and easiest techniques.

4. **Image Grayscale/Sepia Conversion**

   a. **Grayscale Conversion**

   - **Correctness**: Accurate if luminance is computed using proper weights (e.g., 0.299 R + 0.587 G + 0.114 B).
   - **Clarity**: Clear separation from sepia.
   - **Complexity**: Low. Mainly linear transformation per pixel.

   b. **Sepia Conversion**

   - **Correctness:** Correct if applying the sepia transformation matrix properly.
   - **Value:** Adds artistic variation; demonstrates understanding of linear color shifts
   - **Complexity:** Moderate. Involves channel-weight mixing.

5. **Image Blur/Sharpening Processing**

   a. **Blur Processing**

   - **Correctness:** Good at blurring image, but less accurate for larger image
   - **Suggestion:** Gaussian blur (3x3) and (5x5) can do better, but still less accurate for large image.

   b. **Sharpen Processing**

   - **Complexity:** The complexity can be moderate to high

6. **Image Cropping Processing**

   a. **Center Cropping Processing**

   - The solution is easy to approach and handle.

   b. **Frame-based Cropping Processing**

   - **Correctness:** By using the built-in function from numpy, the technique is easily implemented and the result has high accuracy.
   - **Convenience:** For any edition (such as the circular frame's radius) we can pass the argument for many different size of frame.

VI. Reference

https://www.askpython.com/python-modules/matplotlib/convert-rgb-to-grayscale

For grayscale conversion: ***https://alienryderflex.com/hsp.html***

For sepia conversion: ***https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image***

For kernel processing:
https://en.wikipedia.org/wiki/Kernel_(image_processing)

ChatGPT: https://chatgpt.com/