

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



BÁO CÁO ĐỒ ÁN 2
TOÁN ỨNG DỤNG VÀ THỐNG KÊ
< IMAGE PROCESSING >

Lâm Thanh Ngọc - 21127118
Lớp: 21CLC02

Giảng viên:
Vũ Quốc Hoàng
Nguyễn Văn Quang Huy
Lê Thanh Tùng
Phan Thị Phương Uyên

Ngày 31 tháng 7 năm 2023

Mục lục

1	Đánh giá mức độ hoàn thành các chức năng	2
2	Mô tả các hàm	4
2.1	Các hàm hỗ trợ	4
2.2	Các hàm xử lý ảnh	5
3	Hình ảnh kết quả với từng chức năng	13
4	Tài liệu tham khảo	17

1 Đánh giá mức độ hoàn thành các chức năng

Chức năng	Mức độ hoàn thành	Đánh giá	Hình ảnh kết quả
Thay đổi độ sáng cho ảnh	100%	Hoàn thành	
Thay đổi độ tương phản	100%	Hoàn thành	
Lật ảnh ngang	100%	Hoàn thành	
Lật ảnh dọc	100%	Hoàn thành	
Chuyển đổi ảnh RGB thành ảnh xám	100%	Hoàn thành	
Chuyển đổi ảnh RGB thành ảnh sepia	100%	Hoàn thành	

Chức năng	Mức độ hoàn thành	Đánh giá	Hình ảnh kết quả
Làm mờ ảnh		Bỏ qua việc xử lý các pixel ở viền	
Làm sắc nét ảnh		Bỏ qua việc xử lý các pixel ở viền	
Cắt ảnh theo kích thước (cắt ở trung tâm)	100%	Hoàn thành	
Cắt ảnh theo khung hình tròn	100%	Hoàn thành	
Cắt ảnh theo khung 2 hình ellip chéo nhau	100%	Hoàn thành	

2 Mô tả các hàm

2.1 Các hàm hỗ trợ

`print_img(imgs, row, col)`

- Ý tưởng: Hiển thị các hình ảnh cùng lúc với số dòng và cột được truyền vào.
- Mô tả: Hàm `print_img(imgs, row, col)` hiển thị các ảnh trong mảng `imgs` theo dạng lưới với kích thước `row x col`. Sau khi điều chỉnh kích thước mỗi ảnh hiển thị là 20×15 bằng hàm `plt.figure()`, từng hình ảnh trong mảng danh sách các hình `imgs` được vẽ vào 1 subplot với chỉ số tương ứng bằng 2 hàm `plt.subplot()` và `plt.imshow()` và lưới hình ảnh được hiển thị bằng hàm `plt.show()`.
- Input:
 - + `imgs`: mảng chứa các hình ảnh cần in.
 - + `row, col`: số dòng và cột để hiển thị các ảnh theo dạng lưới.
- Output: Các hình ảnh từ mảng `imgs` được in ra ở dạng lưới (`row x col`)

`save_img(imgs, file_name, function)`

- Ý tưởng: Lưu nhiều hình ảnh cùng lúc với tên file được lưu với phần mở rộng `".png"`. Các ảnh được lưu là các ảnh đã được xử lý theo chức năng được chọn, không bao gồm ảnh gốc.
- Mô tả: Hàm `save_img(imgs, file_name, function)` lưu các hình ảnh trong mảng `imgs` thành các tập tin theo công thức: "tên ảnh gốc (đã bỏ phần mở rộng)_tên chức năng.png".
- Input:
 - + `imgs`: mảng chứa các hình ảnh cần lưu.
 - + `file_name`: tên tập tin ảnh gốc.
 - + `function`: mảng chứa tên các chức năng xử lý hình ảnh
- Output: Các ảnh được lưu dưới dạng tập tin png với dạng "tên ảnh gốc (đã bỏ phần mở rộng)_tên chức năng.png" (ví dụ: hình ảnh "Lenna.png" đã được làm mờ được lưu với tên `Lenna_blur.png`).

main()

- Ý tưởng: Cho phép người dùng nhập vào tên tập tin ảnh mỗi khi hàm main được thực thi và lựa chọn các chức năng xử lý ảnh (từ 1 đến 8). Lựa chọn 0 cho phép thực hiện tất cả chức năng với tên file đầu ra tương ứng với từng chức năng.
- Mô tả: Hàm main() cho phép người dùng nhập vào tên tập tin ảnh cần xử lý, sau đó tiến hành mở và đọc tập tin. Nếu tập tin hợp lệ thì cho phép người dùng nhập vào chức năng mong muốn từ 0 đến 8 dựa vào bảng chức năng được in ra và lưu vào biến function. Hàm xét từng trường hợp của biến function và trả về chức năng cùng tập tin ảnh được lưu tương ứng. Trường hợp chức năng khác những con số từ 0 đến 8, người dùng sẽ nhận được thông báo chức năng không phù hợp.

2.2 Các hàm xử lý ảnhbrighten(img_d, brightvalue)

- Ý tưởng: Tăng giá trị các pixels trong ảnh bằng cách cộng ma trận các pixels cho một tham số giá trị độ sáng và giới hạn giá trị các pixels mới trong khoảng từ 0 đến 255.
- Mô tả: Hàm brighten(img_d, brightvalue) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy và một giá trị brightvalue dưới dạng số nguyên. Hàm sẽ tăng độ sáng của ảnh bằng cách cộng thêm giá trị brightvalue cho mỗi điểm ảnh trong ảnh. Kết quả trả về là một ảnh mới có cùng kích thước và kiểu dữ liệu với ảnh ban đầu với độ sáng cao hơn.

Để đảm bảo rằng các giá trị điểm ảnh không vượt quá phạm vi cho phép của kiểu dữ liệu uint8, hàm sẽ sử dụng hàm np.clip để giới hạn các giá trị trong khoảng từ 0 đến 255, và hàm astype được dùng để chuyển đổi kiểu dữ liệu của điểm ảnh về uint8 để đảm bảo ảnh trả về có cùng kiểu dữ liệu với ảnh gốc.

- Input:
 - + img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
 - + brightvalue: giá trị độ sáng được tăng thêm của hình ảnh. Nếu giá trị brightvalue là âm thì hàm sẽ giảm độ sáng của ảnh.
- Output: Ma trận numpy mới chứa các giá trị pixels đã được tăng với giá trị brightvalue tương ứng và được giới hạn trong khoảng từ 0 đến 255.

`contrast(img_d, contrastvalue)`

- Ý tưởng: Tăng độ chênh lệch của các giá trị các pixels trong ảnh bằng cách nhân ma trận các pixels cho một tham số theo công thức:

$$R' = F(R - 128) + 128$$

Trong đó, R tương ứng với thành phần màu đỏ của một pixel màu, F tương ứng với giá trị của hệ số hiệu chỉnh độ tương phản được lưu dưới dạng số dấu phẩy động theo công thức:

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

Trong đó, C là giá trị mức độ tương phản mong muốn, tương ứng với tham số contrastvalue được truyền vào.

- Mô tả: Hàm contrast(img_d, contrastvalue) nhận vào một ảnh img_d dưới dạng mảng 2 chiều numpy và một giá trị contrastvalue. Kết quả trả về là một ảnh mới có cùng kích thước và kiểu dữ liệu với ảnh ban đầu với độ tương phản được điều chỉnh theo công thức R'.

Công thức R' được tính bằng cách sử dụng hệ số hiệu chỉnh độ tương phản F theo tham số contrastvalue và áp dụng cho từng điểm ảnh của img_d. Hàm np.clip được dùng để giới hạn các giá trị của điểm ảnh trong khoảng từ 0 đến 255, và hàm astype được dùng để chuyển đổi kiểu dữ liệu của điểm ảnh về uint8 để đảm bảo ảnh trả về có cùng kiểu dữ liệu với ảnh gốc.

- Input:

- + img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.
- + contrastvalue: giá trị độ tương phản được sử dụng để tính toán hệ số tỉ lệ được áp dụng cho mỗi pixel trong ảnh. Nếu giá trị độ tương phản càng cao thì ảnh sẽ có độ tương phản càng cao và ngược lại. Giá trị độ tương phản nằm trong khoảng từ 0 đến 255.
- Output: Ma trận numpy mới chứa các giá trị pixels đã được thay đổi với giá trị contrastvalue tương ứng theo công thức và được giới hạn trong khoảng từ 0 đến 255.

hflip(img_d), vflip(img_d)

- Ý tưởng: Lật ảnh theo chiều ngang hoặc đọc bằng cách đảo ngược vị trí các cột hoặc dòng của ảnh.
- Mô tả: Hàm hflip(img_d) và vflip(img_d) sử dụng cú pháp cắt mảng của Python (Python List Slicing) để đảo ngược thứ tự của các cột (để lật ảnh ngang) hoặc dòng (để lật ảnh dọc) trong img_d, tạo ra một bản sao ngược của ảnh ban đầu. Cụ thể, cú pháp cắt mảng của Python được định nghĩa bằng

$$\text{Lst}[\text{Initial} : \text{End} : \text{IndexJump}]$$
.

Dựa theo cú pháp được định nghĩa, cú pháp `[:, ::-1]` của hàm hflip được hiểu với dấu `:` trước dấu phẩy được sử dụng để chọn tất cả các hàng của ma trận ảnh, dấu `:` sau dấu phẩy được sử dụng để chọn tất cả các cột và `-1` được sử dụng để các cột được chọn theo thứ tự đảo ngược.

Tương tự, cú pháp `[::-1]` của hàm vflip được hiểu với dấu `:` được sử dụng để chọn tất cả các hàng và `-1` được sử dụng để các hàng được chọn theo thứ tự đảo ngược.

- Input:
 - + `img_d`: ma trận numpy chứa các giá trị pixels của hình ảnh.
- Output: Ma trận numpy mới với các cột/dòng được hoán đổi tương ứng với hình ảnh được lật ngang/dọc có cùng kích thước, kiểu dữ liệu và nội dung với hình ảnh ban đầu.

to_grayscale(img_d)

- Ý tưởng: Tính trung bình cộng các trọng số của mỗi kênh màu để được giá trị xám tương ứng. Có thể sử dụng công thức trung bình cộng thông thường là $(R + G + B)/3$. Khi đó, mỗi kênh màu đều sẽ mang giá trị như nhau. Tuy nhiên, dựa trên nghiên cứu về thị giác con người cho thấy mắt người sẽ nhạy hơn với màu xanh lá, tiếp đến là màu đỏ và cuối cùng là xanh dương. Vì vậy, phương pháp Luminosity được lựa chọn sử dụng để tính trung bình trọng số nhằm hướng đến kết quả trả về có nội dung gần với ảnh gốc hơn.
- Mô tả: Hàm `to_grayscale(img_d)` sử dụng phương pháp Luminosity biểu diễn công thức để tính giá trị xám tương ứng với 3 màu R, G, B như sau:

$$\text{grayscale} = 0.3 * R + 0.59 * G + 0.11 * B$$

Sau khi nhân trọng số các kênh màu của từng pixel cho mảng grayscale chứa các tỉ lệ tương ứng cho 3 kênh màu R, G, B [0.3, 0.59, 0.11] và cộng theo trục cuối cùng (axis = -1) hay cộng các trọng số các kênh màu trong 1 pixel, ta được giá trị xám tương ứng.

Lúc này, kết quả trả về là ảnh với kênh màu đỏ được đổi thành các giá trị xám, còn kênh màu xanh lá và xanh dương vẫn được giữ lại do ma trận numpy chứa các giá trị pixels mới không chứa giá trị cho 2 kênh màu này.

Để giải quyết vấn đề này, hàm np.reshape() được sử dụng để khởi tạo thêm 1 phần tử trong shape lưu số lượng kênh màu là 1. Kết quả trả về là ma trận numpy kết quả res được nhân bản lên để cùng kích thước với ảnh ban đầu và gán các giá trị xám cho 2 kênh màu xanh lá và xanh dương.

- Input:

- + img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.

- Output: Ma trận numpy mới chứa các giá trị xám mang kiểu dữ liệu uint8 tương ứng với các kênh màu có cùng kích thước và kiểu dữ liệu với ảnh ban đầu.

`to_sepia(img_d)`

- Ý tưởng: Tương tự với hàm to_grayscale(img_d), tính trung bình các trọng số của mỗi pixel ảnh để được các màu R, G, B mới.
- Mô tả: Hàm to_sepia(img_d) sử dụng công thức để tính 3 màu R, G, B mới (new_red, new_green, new_blue) dựa trên các giá trị màu R, G, B của ảnh ban đầu như sau:

$$\text{new_red} = 0.393 * R + 0.769 * G + 0.189 * B$$

$$\text{new_green} = 0.349 * R + 0.686 * G + 0.168 * B$$

$$\text{new_blue} = 0.272 * R + 0.534 * G + 0.131 * B$$

Sau khi nhân trọng số các kênh màu của từng pixel cho mảng sepia [[0.393, 0.769, 0.189], [0.349, 0.686, 0.168], [0.272, 0.534, 0.131]] chứa các phần tử là các hệ số biến đổi tương ứng lần lượt cho từng kênh màu đỏ, xanh lá và xanh dương và cộng các phần tử cộng các trọng số các kênh màu trong 1 pixel, ta được các giá trị R, G, B mới.

Các giá trị kênh màu mới được giới hạn từ 0 đến 255 với kiểu dữ liệu uint8. Hàm np.shape() có công dụng tương tự như ở hàm to_grayscale(img_d) được

áp dụng cho từng kênh màu. Kết quả trả về là ma trận numpy được tạo bằng cách nối các ma trận kênh màu theo trục cuối cùng (axis = -1) chứa các giá trị màu sepia bằng hàm np.concatenate().

- Input:

+ img_d: ma trận numpy chứa các giá trị pixels của hình ảnh.

- Output: Ma trận numpy mới chứa các giá trị màu sepia mang kiểu dữ liệu uint8 tương ứng với các kênh màu có cùng kích thước và kiểu dữ liệu với ảnh ban đầu.

blur(img_d)

- Ý tưởng: Tọa độ pixel xác định trên ảnh kết quả chính là việc giao thoa màu với các điểm ảnh lân cận. Ảnh được làm mờ bằng cách tính giá trị trung bình của các pixel và các pixel lân cận để giảm sự chênh lệch rõ ràng của các điểm ảnh. Có hai kiểu làm mờ ảnh phổ biến là Box blur và Gaussian blur được phân biệt nhau bởi kernel dùng để nhân vô hướng với các giá trị trong ảnh. Hàm blur(img_d) trong chương trình được thực thi dựa trên kernel của bộ lọc Gaussian.

- Mô tả: Hàm blur(img_d) được dùng để làm mờ một ảnh màu RGB bằng bộ lọc gaussian biểu diễn bởi 1 ma trận có trọng số cao ở giữa và giảm dần về hai bên. Bộ lọc Gaussian được khởi tạo là 1 ma trận numpy kernel chứa các thành phần: $1/16 * [[1, 2, 1], [2, 4, 2], [1, 2, 1]]$.

Sau khi ma trận numpy mới được tạo ra bằng cách sao chép từ ma trận numpy img_d ban đầu bằng hàm np.copy() theo kiểu dữ liệu float64 để tăng tính chính xác cho tính toán, từng kênh màu của pixel được duyệt và tính lại bằng kỹ thuật convolution.

Kỹ thuật convolution là một kỹ thuật xử lý ảnh sử dụng phép tích chập trong toán học. Thành phần không thể thiếu của phép tích chập là ma trận kernel (bộ lọc). Trong hàm này, bộ lọc Gaussian được lựa chọn để sử dụng. Điểm neo (anchor point) của kernel sẽ quyết định vùng ma trận tương ứng trên ảnh để tích chập, thông thường anchor point được chọn là tâm của kernel. Phép tích chập được hình dung thực hiện bằng việc dịch chuyển ma trận kernel lần lượt qua tất cả các điểm ảnh trong ảnh, bắt đầu từ góc trên bên trái của ảnh và đặt anchor point tương ứng tại điểm ảnh đang xét. Ở mỗi lần dịch chuyển, ma trận numpy img_d được cắt lại theo kích thước của ma trận kernel

với điểm neo là điểm ảnh đang xét và nhân vô hướng với ma trận kernel, sau đó gán kết quả cho

Kết quả trả về là bản sao được ép kiểu thành uint8 là giá trị được giới hạn trong khoảng từ 0 đến 255 chứa các pixel cho hình ảnh đã được làm mờ. Với cách làm này hình ảnh sẽ được làm mờ theo bộ lọc Gaussian mà bỏ qua việc xử lý phần viền của ảnh.

- Input:

- + img_d: ma trận numpy chứa các giá trị pixel của hình ảnh.

- Output: Ma trận numpy mới chứa các giá trị của ảnh được làm mờ với kích thước và kiểu dữ liệu giống với ảnh ban đầu.

`sharpen(img_d)`

- Ý tưởng: Làm rõ ảnh bằng cách tính lại giá trị các pixel trong ảnh để tăng sự phân biệt rõ rệt và sắc nét giữa các điểm ảnh.

- Mô tả: Hàm sharpen(img_d) được triển khai tương tự với hàm blur(img_d) với kernel được thay đổi thành: [[0, -1, 0], [-1, 5, -1], [0, -1, 0]].

- Input:

- + img_d: ma trận numpy chứa các giá trị pixel của hình ảnh.

- Output: Ma trận numpy mới chứa các giá trị của ảnh được làm sắc nét với kích thước và kiểu dữ liệu giống với ảnh ban đầu.

`crop_center(img_d)`

- Ý tưởng: Cắt ảnh ở phần trung tâm bằng cách tính độ dài phần bị bỏ ở mỗi cạnh và chỉ hiển thị phần trung tâm của hình ảnh ban đầu.

- Mô tả: Hàm crop_center(img_d) khởi tạo 2 biến width và height tương ứng với phần cần bỏ lần lượt theo chiều rộng và chiều cao bằng cách chia chiều rộng và chiều cao của ảnh gốc cho 4. Trả về ảnh kết quả res bằng cách lấy các phần tử của ảnh gốc từ vị trí width đến vị trí -width theo chiều ngang và từ vị trí height đến vị trí -height theo chiều cao.

- Input:

- + img_d: ma trận numpy chứa các giá trị pixel của hình ảnh.

- Output: Ma trận numpy mới đã được cắt 1/4 ở mỗi cạnh xung quanh của ma trận ban đầu và hiển thị hình ảnh ở phần trung tâm của hình ảnh ban đầu.

`crop_circle(img_d)`

- Ý tưởng: Tạo 1 khung ảnh hình tròn bằng việc vẽ 1 đường tròn nội tiếp hình ảnh ban đầu. Các pixels bên ngoài đường tròn được gán bằng 0 và các pixels bên trong đường tròn là hình ảnh ban đầu được giữ lại.
- Mô tả: Hàm `crop_circle(img_d)` áp dụng phương trình đường tròn trong hệ tọa độ Descartes:

$$(x - a)^2 + (y - b)^2 = r^2$$

Trong đó, (x, y) là tọa độ điểm thuộc đường tròn, (a, b) là tâm đường tròn và r là bán kính đường tròn.

Đối với thuật toán của hàm `crop_circle(img_d)`, hình ảnh được hiểu là 1 hình vuông và tâm đường tròn được xác định tại giao điểm 2 đường chéo của ảnh. Cụ thể, (a, b) lần lượt là 1/2 chiều rộng và chiều cao của ảnh. Bán kính r được xác định bởi 1/2 chiều rộng (hoặc chiều cao). Để có thể tính toán, các giá trị a, b và r được ép kiểu thành các giá trị nguyên.

Duyệt từng điểm ảnh của ma trận numpy mới được tạo ra bằng cách sao chép từ ma trận numpy `img_d` bằng hàm `np.copy()`, mỗi điểm ảnh tương ứng với 1 tọa độ (i, j). Nếu điểm ảnh nằm ngoài đường tròn (phương trình $> r^2$) thì gán điểm ảnh bằng 0.

- Input:
 - + `img_d`: ma trận numpy chứa các giá trị pixel của hình ảnh.
- Output: Ma trận numpy mới biểu diễn ảnh được cắt theo khung hình tròn.

`crop_ellip(img_d)`

- Ý tưởng: Tạo 1 khung ảnh bằng việc vẽ ra 2 đường ellip chéo nhau. Các pixels bên ngoài khung ảnh được gán bằng 0 và các pixels bên trong khung ảnh là hình ảnh ban đầu được giữ lại.
- Mô tả: Hàm `crop_ellip(img_d)` áp dụng phương trình đường ellip được xoay với 1 góc A:

$$\frac{((x - h)\cos(A) + (y - k)\sin(A))^2}{a^2} + \frac{((x - h)\sin(A) - (y - k)\cos(A))^2}{b^2} = 1$$

Trong đó, (x, y) là tọa độ điểm thuộc đường ellip, (h, k) là tâm đường ellip và (a, b) lần lượt là bán kính lớn và bán kính nhỏ của đường ellip.

Đối với thuật toán của hàm `crop_circle(img_d)`, hình ảnh được hiểu là 1 hình vuông và tâm đường ellip được xác định tương tự với việc xác định tâm đường tròn như ở hàm `crop_circle(img_d)`. 2 bán kính a, b được xác định lần lượt bởi $20/47$ và $1/4$ lần đường chéo ảnh được tính bởi định lý Pythagoras. Con số $20/47$ (hay 2.35) được tìm ra sau nhiều lần thử nghiệm với các con số dao động từ 2 đến 3 và con số $1/4$ được tìm ra nhờ vào ý tưởng của hàm `crop_center(img_d)`. Để có thể tính toán, các giá trị h, k, a, b được ép kiểu thành các giá trị nguyên.

Các điểm ảnh được chuyển đổi thành dạng lưới tọa độ nhờ vào hàm `np.meshgrid()` với các giá trị x, y tương ứng. Từ đó, hệ tọa độ của các điểm ảnh được chuyển sang hệ tọa độ của 2 đường ellip chéo nhau với các góc 45 độ và 135 độ. Giá trị cos và sin của các góc được tính cụ thể là:

$$\cos(45) = \sin(45) = \sin(135) = \frac{\sqrt{2}}{2}$$

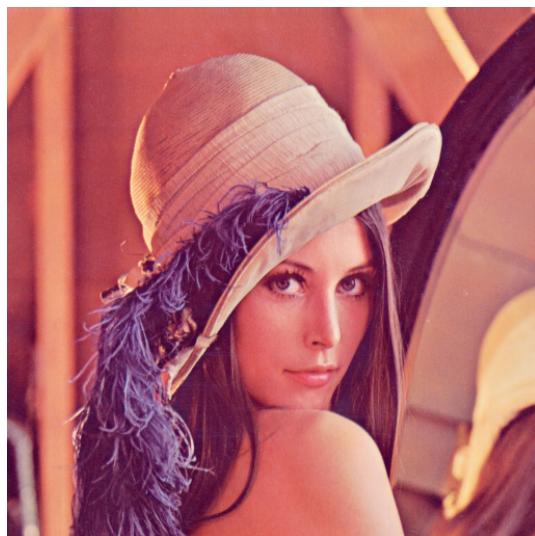
$$\cos(135) = -\frac{\sqrt{2}}{2}$$

Sau khi lọc ra các điểm ảnh nằm ngoài cả 2 đường ellip và lưu vào biến `mask`, sao chép ma trận numpy ảnh gốc bằng hàm `np.copy()` và gán điểm ảnh tại vị trí `mask` bằng 0 .

- Input:
 - + `img_d`: ma trận numpy chứa các giá trị pixel của hình ảnh.
- Output: Ma trận numpy mới biểu diễn ảnh được cắt theo khung tạo bởi 2 hình ellip chéo nhau.

3 Hình ảnh kết quả với từng chức năng

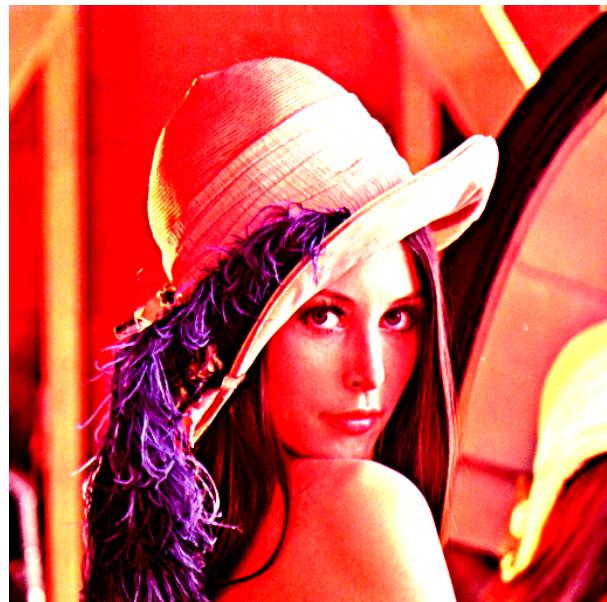
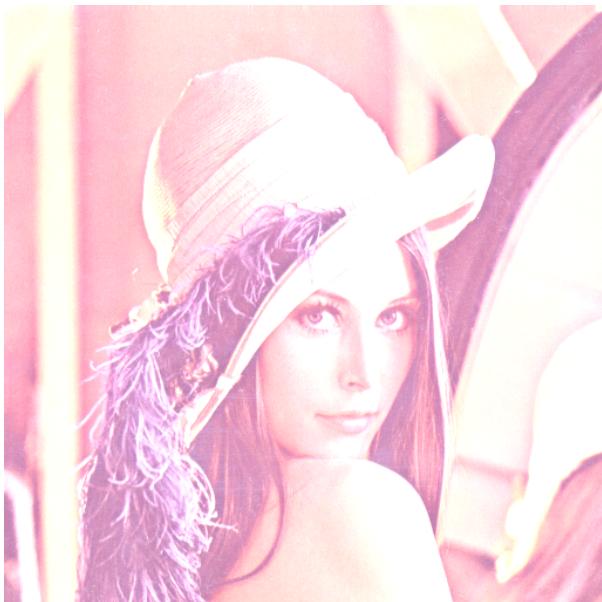
Ảnh gốc:



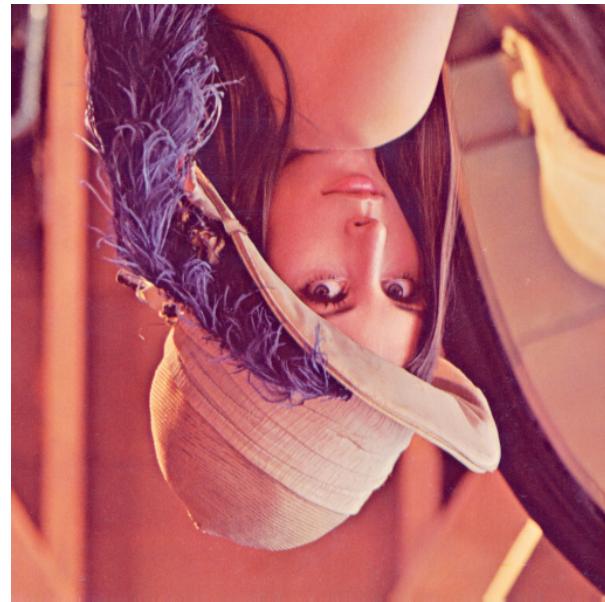
Các thông số:

- Kích thước: 462KB
- Chiều dài: 512 pixels
- Chiều rộng: 512 pixels

Các ảnh kết quả:



Hình 1: Thay đổi độ sáng và độ tương phản cho ảnh (giá trị thay đổi cộng thêm là 128)



Hình 2: Lật ảnh (ngang - dọc)



Hình 3: Chuyển đổi ảnh RGB thành ảnh xám/sepia



Hình 4: Làm mờ/sắc nét ảnh



Hình 5: Cắt ảnh theo kích thước (cắt ở trung tâm) và theo khung hình tròn



Hình 6: Cắt ảnh theo khung là 2 hình ellip chéo nhau

4 Tài liệu tham khảo

- File lab03_project02.ipynb
- Công thức thay đổi độ tương phản
- Kỹ thuật cắt mảng Python (Python List Slicing)
- Phương pháp Lumosity chuyển đổi ảnh màu sang ảnh xám
- Công thức chuyển đổi ảnh màu thành ảnh sepia
- Kernel và thuật toán sử dụng cho 2 hàm làm mờ và làm sắc nét ảnh
- Ý tưởng và giải thuật làm mờ ảnh
- Kỹ thuật convolution trong xử lý ảnh
- Phương trình đường tròn
- Phương trình đường ellip