ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

MÔN : TOÁN ỨNG DỤNG VÀ THỐNG KÊ

CHO CÔNG NGHỆ THÔNG TIN

BÁO CÁO ĐỒ ÁN 2 :

IMAGE PROCESSING

GIẢNG VIÊN: NGUYỄN ĐÌNH THÚC

NGÔ ĐÌNH HY

NGUYỄN VĂN QUANG HUY

SINH VIÊN THỰC HIỆN: NGUYỄN THÁI ĐAN SÂM – 21127414 – 21CLC01

Mục lục

Phần I. Giới thiệu đồ án

Một bức ảnh có thể lưu trữ dưới ma trận của các điểm ảnh. Có nhiều loại ảnh được sử dụng trong thực tế, ví dụ: ảnh xám, ảnh màu,...

Đối với ảnh xám, một điểm ảnh sẽ là được biểu diễn bằng giá trị không âm.
Ảnh màu được sử dụng phổ biến là ảnh RGB, trong đó, mỗi điểm ảnh sẽ lưu trữ 3 thông tin kênh màu (mỗi kênh màu 1 byte) là: R (red - đỏ), G (green - xanh lá), B (blue - xanh dương).

Xử lý ảnh (tiếng Anh: digital image processing) hay xử lý ảnh kỹ thuật số là một phân ngành trong xử lý số tín hiệu với tín hiệu xử lý là ảnh. Đây là một phân ngành khoa học mới rất phát triển trong những năm gần đây. Xử lý ảnh gồm 4 lĩnh vực chính: xử lý nâng cao chất lượng ảnh, nhận dạng ảnh, nén ảnh và truy vấn ảnh. Sự phát triển của xử lý ảnh đem lại rất nhiều lợi ích cho cuộc sống của con người.

Ngày nay xử lý ảnh đã được áp dụng rất rộng rãi trong đời sống như: photoshop, nén ảnh, nén video, nhận dạng biển số xe, nhận dạng khuôn mặt, nhận dạng chữ viết, xử lý ảnh thiên văn, ảnh y tế,....

Trong đồ án 2 – Image Processing, các kỹ thuật xử lí hình ảnh sau sẽ được thực hiện và trình bày :

- Thay đổi độ sáng cho ảnh
- Thay đổi độ tương phản
- Lật ảnh (ngang – dọc)
- Chuyển đổi ảnh RBG thành ảnh xám/sepia
- Làm mờ / sắc nét ảnh
- Cắt ảnh theo kích thước (cắt ở trung tâm)
- Cắt theo khung tròn
- Cắt theo khung ellip

Phần II. Ý tưởng và cài đặt thuật toán

1. Các thư viện hỗ trợ và hàm hỗ trợ

- **Thư viện**

`numpy :` tính toán trên ma trận, dùng nhiều trong cài đặt thuật toán K - Means
`matplotlib.pyplot :` Đọc, xuất hình ảnh và đặt title cho ảnh
`PIL :` Mở file ảnh và biến đổi hình ảnh thành ma trận pixel.

- **Hàm**

`def remake_img(img) :` Chuyển đổi hình ảnh trước khi xử lí, hàm này trả về 1 mảng đa chiều NumPy (trong trường hợp tham số đầu vào là hình ảnh màu hệ RBG thì đầu ra sẽ là mảng 3 chiều với định dạng (height, width, channels). Hàm này sẽ được gọi đầu tiên, trước khi gọi các hàm xử lí ảnh.
`Image.fromarray(new_img.astype('uint8'), 'RGB') :` Chuyển đổi mảng ảnh thành dạng hình ảnh PIL với kiểu dữ liệu của từng phần tử trong mảng ảnh (pixel) là số nguyên dương 8 bit và giá trị của từng pixel nằm trong khoảng 0 đến 255, đối số **`'RGB'`** biểu thị cho hệ màu của ảnh. Hàm này được gọi ngay sau các hàm xử lí hình ảnh, việc gọi hàm này nhằm mục đích phục vụ cho các thao tác có liên quan đến ảnh PIL (hiển thị ảnh, lưu ảnh, tải ảnh, ...).

2. Các hàm chính trong chương trình

2.1. Hàm thay đổi độ sáng

```
def adjust_brightness(img, brightness):
```

Ý tưởng : Để điều chỉnh độ sáng của bức ảnh, ta tiến hành thao tác trên các giá trị màu RBG của từng pixel trong ảnh gốc bằng cách cộng giá trị brightness (do người dùng nhập vào). Giá trị **`brightness`** tỉ lệ thuận với độ sáng của bức ảnh, nếu **`brightness`** < 0, độ sáng của bức ảnh sẽ giảm xuống, bức ảnh sẽ tối lại.

Cài đặt : Đầu vào sẽ là hình ảnh đã qua bước chuyển đổi (có dạng NumPy array), thay đổi giá trị pixel trong mảng ảnh bằng cách cộng mảng ảnh với mảng giá trị **`brightness`**. Mảng pixel thu được sau đó được cắt bớt trong phạm vi giá trị từ 0 đến 255 để đảm bảo không có pixel nào vượt ra ngoài phạm vi

hợp lệ. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh **PIL** bằng hàm **Image.fromarray**

2.2. Hàm thay đổi độ tương phản

def adjust_contrast(img, contrast):

Ý tưởng : Để điều chỉnh độ tương phản của bức ảnh, ta nhân mỗi pixel trong ảnh với hệ số *factor* được tính với công thức $factor = \frac{259(contrast+255)}{255(259-contrast)}$ (1). Ảnh lúc này được biến đổi bằng cách lấy ảnh gốc trừ 128 tất cả nhân với số *factor* và cuối cùng là cộng 128, với công thức như sau: $I' = factor(I - 128) + 128$ (2).

Cài đặt : Để đảm bảo thuật toán hoạt động chạy chính xác, giá trị **contrast** và mảng hình ảnh sẽ được chuyển về kiểu dữ liệu '**float**' và **contrast** được đảm bảo nằm trong phạm vi từ -255 đến 255 bằng hàm '**np.clip**'. *factor* được tính như công thức (1) đã nêu trên sau đó áp dụng công thức (2) để chuyển đổi ảnh. Mảng pixel thu được sau đó được cắt bớt trong phạm vi giá trị từ 0 đến 255 để đảm bảo không có pixel nào vượt ra ngoài phạm vi hợp lệ. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh **PIL** bằng hàm **Image.fromarray**

2.3. Hàm lật ảnh (ngang – dọc)

def flip_horizontally(img): / **def flip_vertically(img):**

Ý tưởng : Để lật ảnh, ta chỉ cần thực hiện đảo thứ tự các pixel trong mảng ảnh đã cho theo hướng yêu cầu. Đối với lật ngang (horizontal), ta đổi chiều thứ tự các hàng trong mảng hình ảnh. Đối với lật dọc (vertical), ta đổi chiều thứ tự các cột trong mảng hình ảnh. Được minh họa bởi ví dụ sau :

Ma trận gốc :	Lật ngang :	Lật dọc:
[[1 2 3]	[[3 2 1]	[[7 8 9]
[4 5 6]	[6 5 4]	[4 5 6]
[7 8 9]]	[9 8 7]]	[1 2 3]]

Cài đặt : Hàm lật ảnh ngang trả về **img[:,::-1]**, nghĩa là mảng ảnh được xử lí bằng phương pháp cắt mảng, phần ' **:** 'trước dấu phẩy nghĩa là các hàng (các mảng 1 chiều) trong mảng ảnh được giữ nguyên thứ tự, phần ' **::-1** ' với tham số -1 cho biết các cột sẽ lùi về trước, nghĩa là các phần tử của các mảng 1 chiều sẽ bị đảo ngược thứ tự. Hàm lật ảnh dọc trả về **img[::-1]**, nghĩa là mảng ảnh

cũng được xử lí bằng phương pháp cắt ảnh, nhưng ở đây chỉ cần đảo ngược thứ tự của các phần tử (các mảng 1 chiều) trong mảng ảnh là được ảnh lật dọc như yêu cầu. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh **PIL** bằng hàm **Image.fromarray**

2.4. Hàm chuyển đổi ảnh RBG thành ảnh xám

```
def convert_grayscale(img):
```

Ý tưởng : Để chuyển đổi ảnh RBG thành ảnh xám, ta cần tính giá trị màu (ảnh xám) trung bình của từng hệ màu trong pixel, gọi là *grayscale_filter*, được tính như sau: $grayscale\_filter = \frac{R+B+G}{3}$, tuy nhiên để màu của ảnh xám được ổn định và chuẩn hơn, ta sẽ dùng công thức này:

$grayscale\_filter = 0.299R + 0.587G + 0.114B$

Nhân từng hệ màu RBG của từng pixel trong mảng ảnh với *grayscale_filter* vừa tính được thì sẽ thu được ảnh xám.

Cài đặt : Đầu tiên áp dụng các trọng số [0.299, 0.587, 0.114] cho các giá trị RBG của từng pixel trong mảng hình ảnh bằng cách sử dụng hàm '**np.dot**'. Giá trị thu được sau đó được lưu vào biến '**avg**'. Sau đó tiến hành tạo 1 mảng hình ảnh **adjusted_img** chứa số 0 có cùng kích thước với mảng hình ảnh ban đầu, mảng hình ảnh này sẽ được dùng để lưu trữ ảnh xám nhưng chỉ với 1 channel. Khởi tạo vòng lặp với i có giá trị lần lượt là 0, 1 và 2. Mỗi lần lặp, giá trị **avg** được tính toán trước đó sẽ được gán vào cho cả 3 channel của mảng ảnh **adjusted_img**. Vì giá trị '**avg**' lúc này chính là giá trị 'độ xám' cho mỗi điểm ảnh nên việc gán **avg** cho cả 3 channels sẽ cho ra được hình ảnh xám. Mảng pixel thu được sau đó được cắt bớt trong phạm vi giá trị từ 0 đến 255 để đảm bảo không có pixel nào vượt ra ngoài phạm vi hợp lệ. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh **PIL** bằng hàm **Image.fromarray**

2.5. Hàm chuyển đổi ảnh RBG thành ảnh sepia

```
def convert_sepia(img):
```

Ý tưởng : Tương tự như kĩ thuật chuyển đổi thành ảnh xám, đối với chuyển đổi thành ảnh sepia ta cũng tính giá trị RBG mới của từng pixel bằng công thức :

$$R' = 0.393R + 0.769G + 0.189B$$
$$G' = 0.349R + 0.686G + 0.168B$$
$$B' = 0.272R + 0.534G + 0.131B$$

Cài đặt : Đầu tiên khởi tạo 1 ma trận 3x3 `sepia_array` chứa các trọng số như trên để điều chỉnh RBG của từng pixel. Tiếp theo tiến hành nhân các giá trị RBG của từng pixel trong mảng hình ảnh đã cho với ma trận `sepia_array` đã khởi tạo bằng cách sử dụng hàm '`np.dot`'. Mảng pixel thu được sau đó được cắt bớt trong phạm vi giá trị từ 0 đến 255 để đảm bảo không có pixel nào vượt ra ngoài phạm vi hợp lệ. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh `PIL` bằng hàm `Image.fromarray`

2.6. Hàm làm mờ ảnh

```
def blur_image(img):
```

Ý tưởng : Hạt nhân Gaussian là một công cụ trong xử lý ảnh được dùng để làm mờ hình ảnh. Nó giảm cường độ của từng điểm ảnh trong ảnh dựa trên khoảng cách từ điểm ảnh đó đến trung tâm của hạt nhân. Điều này làm cho các điểm ảnh ở xa trung tâm bị ảnh hưởng ít hơn, trong khi các điểm ảnh gần trung tâm bị ảnh hưởng mạnh hơn. Hàm Gaussian 1D là công thức toán học đại diện cho hạt nhân Gaussian và được sử dụng để tính toán cường độ mờ cho mỗi điểm ảnh. Công thức hàm Gaussian 1D là :

$$g_\sigma(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

Đầu tiên, cần tạo một bộ lọc Gaussian có hình dạng là một ma trận có kích thước và phạm vi xác định. Bộ lọc này chứa các trọng số được tính toán dựa trên hàm Gaussian, với trung tâm làm tròn và phân bố đối xứng. Bộ lọc Gaussian xác định mức độ mờ được áp dụng cho ảnh.

Tiếp theo, áp dụng bộ lọc Gaussian lên từng điểm ảnh trong ảnh gốc. Đối với mỗi điểm ảnh, bộ lọc sẽ tính toán trung bình có trọng số của các điểm ảnh lân cận, sử dụng các trọng số từ bộ lọc Gaussian. Điều này làm mờ điểm ảnh và các điểm ảnh lân cận xung quanh, tạo ra hiệu ứng mờ cho ảnh.

Kích thước của bộ lọc Gaussian ảnh hưởng trực tiếp đến mức độ làm mờ của ảnh. Một bộ lọc lớn hơn sẽ tạo ra hiệu ứng làm mờ mạnh hơn, trong khi một bộ lọc nhỏ hơn sẽ làm mờ ít hơn.

Cài đặt : Đầu tiên khởi tạo giá trị '`sigma`', giá trị này chính là độ lệch chuẩn cho hạt nhân Gaussian, nó dùng để xác định độ lan rộng của hạt nhân và ảnh hưởng đến độ mờ nhất định của ảnh. Nghĩa là '`sigma`' này có thể mang có giá trị khác (1, 3, 5, 7,...) ưu tiên các số lẻ vì như thế sẽ dễ dàng xác định tâm (kernel) cho các phần tử trong mảng. Giá trị '`sigma`' càng lớn thì ảnh càng mờ. Tiếp theo, tính toán kích thước của kernel dựa trên giá trị '`sigma`', vị trí của kernel và giá trị của kernel, giá trị của kernel được tính bởi công thức trên với hàm '`np.exp`' và chuẩn hóa bằng hàm '`np.sum`'. Kernel được áp dụng cho từng hàng và cột trong mảng hình ảnh bằng cách sử dụng hàm '`np.convolve`'. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh `PIL` bằng hàm `Image.fromarray`

2.7. Hàm làm sắc nét ảnh

```
def sharpen_image(img):
```

Ý tưởng : Dựa trên ý tưởng làm mờ ảnh, kỹ thuật làm sắc nét ảnh bằng hạt nhân Gaussian 1D là áp dụng bộ lọc làm mờ cho hình ảnh để loại bỏ nhiễu và các chi tiết không mong muốn. Quá trình làm mờ này giúp giảm sự nổi bật của các cạnh và chi tiết trong ảnh gốc. Tiếp theo, ta áp dụng phép trừ giữa ảnh ban đầu và ảnh đã bị mờ. Phép trừ này tạo ra một ảnh khác biệt chỉ tập trung vào các cạnh và chi tiết trong ảnh gốc, làm nổi bật chúng. Sau đó, ảnh khác biệt này được thêm lại vào ảnh ban đầu để thu được ảnh sắc nét.
Cài đặt : Tương tự như cách cài đặt hàm làm mờ ảnh, nhưng thêm vào đó một vài bước nữa như sau. Sau khi tạo ra hình ảnh mờ, ta tính toán sự khác biệt giữa ảnh gốc và ảnh mờ để thu về ảnh khác biệt. Mức độ làm sắc nét được xác định bằng cách sử dụng biến số lượng và hình ảnh khác biệt được nhân với biến số lượng và được cộng trở lại ảnh gốc để thu được hình ảnh được làm sắc nét. Các giá trị pixel được cắt bớt để đảm bảo chúng nằm trong phạm vi hợp lệ từ 0 đến 255. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh `PIL` bằng hàm `Image.fromarray`

2.8. Hàm cắt ảnh theo kích thước (cắt ở trung tâm)

```
def crop_center(img, Height, Width):
```

Ý tưởng : Để có thể cắt được ảnh ở trung tâm, trước hết cần xác định được vị trí tâm của bức ảnh, cùng với `Height, Width`(chiều dài và rộng mong muốn của ảnh sau khi cắt) do người dùng yêu cầu, ta có công thức tính kích thước các

cạnh của bức ảnh như sau với **`width, height`** lần lượt là chiều dài và rộng của ảnh gốc :

$$\text{left} = \frac{width - Width}{2} \qquad \text{right} = \frac{width + Width}{2}$$

$$\text{up} = \frac{height - Height}{2} \qquad \text{down} = \frac{height + Height}{2}$$

Cài đặt : Cho **`height`** và **`width`** lần lượt là chiều dài và rộng của ảnh gốc, kiểm tra xem chiều dài và rộng mong muốn của ảnh sau khi cắt có vượt quá kích thước của ảnh gốc không, nếu có thì trả về ảnh gốc. Các cạnh của ảnh cắt được tính theo công thức như trên. Mảng hình ảnh sau đó được cắt bằng cách sử dụng phép cắt mảng với các thông số kích thước đã tính trên. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh **`PIL`** bằng hàm **`Image.fromarray`**

2.9. Hàm cắt ảnh theo khung tròn

```
def crop_circle(img):
```

Ý tưởng : Để có thể cắt được ảnh theo khung tròn, trước hết cần xác định tâm của hình tròn (a,b) và đường kính C của hình tròn. Tâm của hình tròn sẽ là chính giữa ảnh và kích thước của đường kính sẽ là giá trị nhỏ hơn của 2 trong 4 cạnh của hình ảnh (chiều dài hoặc chiều rộng). Công thức để cắt ảnh sẽ là :

$$(x - a)^2 + (y - b)^2 \leq (\frac{C}{2})^2$$

Cài đặt : Cho height và width lần lượt là chiều dài và rộng của ảnh gốc, đường kính diameter được lấy từ giá trị nhỏ nhất giữa height và width vừa được khởi tạo. Lần lượt tính ra được bán kính D (đường kính C chia 2), tọa độ x, y của tâm hình tròn (x bằng nửa chiều rộng, y bằng nửa chiều dài, điều này đảm bảo tâm hình tròn nằm ngay giao điểm của chiều rộng và chiều dài, tức là tâm bức ảnh). Tiến hành tạo 1 mảng phần tử 0 có cùng kích thước và kiểu dữ với ảnh gốc, cho chạy 2 vòng lặp lần lượt y x, tính khoảng cách từ điểm ảnh tại hàng y cột x đến tâm của hình tròn bằng công thức Pythago, kiểm tra xem điểm ảnh vừa tính có nằm trong hình tròn bán kính D hay không. Nếu đúng, lưu điểm ảnh vào mảng mới vừa tạo. Hình ảnh thu được sau đó được chuyển đổi thành hình ảnh PIL bằng hàm **`Image.fromarray`**

2.10. Hàm cắt ảnh theo khung ellip

```
def crop_ellipse(img):
```

Ý tưởng : Để có thể cắt được ảnh theo khung ellipse, trước hết cần xác định tâm (x_0, y_0) của ảnh. Góc xoay α của ellipse ta đặt là 45° để được ảnh như yêu cầu. Sau đó tiến hành cắt ảnh theo công thức :

$$\frac{[(x-x_0)cos\ \alpha-(y-y_0)sin\ \alpha]^2}{a^2}+\frac{[(x-x_0)sin\ \alpha-(y-y_0)cos\ \alpha]^2}{b^2}\leq 1$$

Giá trị a, b chính là chiều rộng và chiều cao của mask hình ellipse, ta cần xác định 2 giá trị này bằng cách thiết lập hệ phương trình 2 ẩn để tìm ra 2 nghiệm a và b. Hệ phương trình đó có dạng như sau :

$$\left(\begin{array}{cc|c} \left(-x_0\cos\alpha+\frac{y_0\sin\alpha}{2}\right)^2 & \left(-x_0\sin\alpha-\frac{\cos\alpha}{2}\right)^2 & 1 \\ \left(-\frac{x_0\cos\alpha}{2}+y_0\sin\alpha\right)^2 & \left(-\frac{x_0\sin\alpha}{2}-y_0\cos\alpha\right)^2 & 1 \end{array}\right)$$

Tuy nhiên, trong trường hợp chúng ta cần xử lý ảnh với điều kiện $x_0 = y_0$ và $\alpha = 45°$, hệ phương trình sẽ có vô số nghiệm. Để giải quyết vấn đề này, ta sử dụng một phép chia lấy phần nguyên trong hệ phương trình để thu được một nghiệm duy nhất mà kết quả gần như không có sự khác biệt. Sau khi giải ra chiều rộng và chiều cao, ta sử dụng kết quả này để tạo ra một mask. Mask còn lại được tạo ra bằng cách hoán đổi chiều rộng và chiều cao. Sau đó, ta sử dụng phép logic OR để kết hợp hai mask thành một mask cuối cùng.
Bằng cách áp dụng mask vừa tính, ta thu được một ma trận mới, là hình ảnh được cắt theo khung là hai hình elip chéo nhau.

Cài đặt :

Cho height và width lần lượt là chiều dài và rộng của ảnh gốc, diameter được lấy từ giá trị nhỏ nhất giữa height và width vừa được khởi tạo. Tọa độ x_0 y_0 lúc này được tính từ diameter đã xác định trên chia 2. Đặt góc xoay $\alpha = 45°$. Các hệ số của phương trình cho hình elip xoay được tính bằng cách sử dụng tọa độ tâm và góc quay. Điều này thực hiện thông qua việc chuyển đổi tọa độ thành một hệ tọa độ xoay bằng cách sử dụng phép nhân ma trận, sau đó giải hệ

phương trình để tìm các hệ số. Chiều cao và chiều rộng của hình ellipse được tính dựa trên các hệ số của phương trình.

Mask hình ellipse được tạo bằng cách sử dụng hàm **`ellipse_mask`**, trong đó hàm này nhận kích thước của hình ảnh, tọa độ tâm, chiều cao và chiều rộng của hình elip cũng như góc xoay. Hai mask được tạo và chúng được kết hợp bằng phép toán OR logic để tạo thành mask cuối cùng.

Một mảng NumPy mới với hình dạng giống như hình ảnh đầu vào, nhưng với tất cả các phần tử được đặt thành 0, được tạo và gọi là **`cropped_img`**. Các pixel trong cropped_img tương ứng với các giá trị True trong mask được đặt bằng các giá trị pixel tương ứng trong hình ảnh đầu vào, kết quả là ta thu được một mảng mới có hình dạng giống như hình ảnh ban đầu, nhưng chỉ chứa hình ảnh được cắt theo hình ellipse.

2.11. Hàm main

Cho phép người dùng lựa chọn chức năng xử lí ảnh theo ý muốn, hiển thị ảnh gốc và ảnh đã qua xử lí, gọi hàm xử lí hình ảnh trước và sau, ngoài ra còn lưu ảnh đã qua xử lí về máy.

Phần III. Các test case

1.Thay đổi độ sáng cho ảnh

Độ phân giải: 1440 x 1440 pixel

Ảnh gốc

Tăng 100

Ảnh gốc

Giảm 50

2.Thay đổi độ tương phản cho ảnh

Độ phân giải: 1440 x 1440 pixel

Ảnh gốc

Giảm 50

Ảnh gốc

Tăng 50

3.Lật ảnh ngang - dọc

Độ phân giải: 1440 x 1440 pixel

Ảnh gốc

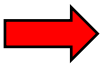
Lật dọc

Ảnh gốc

Lật ngang

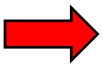
4.Chuyển đổi ảnh RBG thành ảnh xám / sepia

Độ phân giải: 1440 x 1440 pixel



Ảnh gốc

Ảnh xám



Ảnh gốc

Ảnh sepia

5.Làm mờ / sắc nét ảnh

Độ phân giải: 1440 x 1440 pixel

Ảnh gốc

Ảnh mờ

Ảnh gốc

Ảnh sắc nét

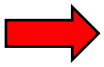
Trường hợp làm cho ảnh sắc nét từ ảnh đã làm mờ :

Giả sử giá trị sigma ở cả 2 hàm đều = 9

Ảnh mờ

Ảnh sắc nét

Trường hợp làm mờ ảnh sắc nét từ ảnh xám :



Ảnh xám

Ảnh mờ

6.Cắt ảnh theo kích thước (cắt từ trung tâm)

Độ phân giải: 1440 x 1440 pixel

Ảnh gốc

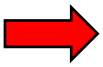
800 x 800

400 x 600

7.Cắt ảnh theo khung tròn

<u>Trường hợp cắt từ ảnh hình vuông</u> :

Ảnh 1 :
Độ phân giải: 1440 x 1440 pixel



Ảnh gốc

Ảnh khung tròn

<u>Trường hợp cắt từ ảnh hình chữ nhật</u> :

Ảnh 2 :
Độ phân giải : 3480 x 2160 pixel

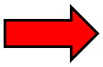
Ảnh gốc

Ảnh khung tròn

8.Cắt ảnh theo khung ellipse

<u>Trường hợp cắt từ ảnh hình vuông</u> :

Ảnh 1 :
Độ phân giải: 1440 x 1440 pixel



Ảnh gốc

Ảnh khung ellipse

<u>Trường hợp cắt từ ảnh hình chữ nhật</u> :

Ảnh 2 :
Độ phân giải : 3480 x 2160 pixel

Ảnh gốc

Ảnh khung ellipse

PHẦN IV. Kết luận và đánh giá

1. Chất lượng hình ảnh

Tùy vào độ phân giải của ảnh, độ phân giải càng cao thì thời gian xử lí của các hàm sẽ tăng lên nhưng không đáng kể, vẫn đảm bảo ảnh thời gian xử lí nhanh (dưới 1s) đối với các ảnh có độ phân giải từ 1500 x 1500 pixel trở xuống.

2. Thời gian thực hiện thuật toán

Tất cả các hàm đều đảm bảo thời gian xử lí ảnh nằm trong mức cho phép, không để người dùng đợi quá lâu. Ngoài ra tất cả đều hoạt động tốt với các loại ảnh đầu vào (png, jpg) với hệ màu RBG.

3. Hạn chế

Hàm cắt ảnh theo khung ellipse sẽ cho ra ảnh với khung ellipse bị lệch nếu hình ảnh đầu vào có dạng hình chữ nhật (chiều dài và chiều rộng của ảnh không bằng nhau). Độ chênh lệch của chiều dài và chiều rộng càng lớn thì độ lệch của khung ellipse sẽ càng lớn, điều này sẽ không thể hiện rõ nếu kích thước của chiều dài và chiều rộng chênh lệch nhau không đáng kể.

PHẦN V. Tham khảo

[1] Lab02 – Ma trận trong Python, Ngô Đình Hy, Nguyễn Văn Quang Huy

[2] Computer Vision - Image Processing - Point Processes – Viblo
https://viblo.asia/p/computer-vision-image-processing-point-processes-Az45bAROlxY

[3] Image Processing Using Numpy: With Practical Implementation And Code – Analytics Vidhya
https://www.analyticsvidhya.com/blog/2021/05/image-processing-using-numpy-with-practical-implementation-and-code/

[4] How do I increase the contrast of an image in Python OpenCV – stackoverflow
https://stackoverflow.com/questions/39308030/how-do-i-increase-the-contrast-of-an-image-in-python-opencv

[5] How can I convert an RBG image into grayscale in Python – stackoverflow
https://stackoverflow.com/questions/12201577/how-can-i-convert-an-rgb-image-into-grayscale-in-python

[6] Creating a sepia filter with python – Yabir's blog
https://yabirgb.com/sepia_filter/

[7] Kernel (image processing) – wikipedia
https://en.wikipedia.org/wiki/Kernel_(image_processing)

[8] Kernels / Convolution / Image Filtering – wordpress
https://cvexplained.wordpress.com/2020/04/30/kernels/

[9] OpenCV - Gaussian Blur – Tutorialspoint
https://www.tutorialspoint.com/opencv/opencv_gaussian_blur.htm

[10] 2.6. Image manipulation and processing using Numpy and Scipy – Scipy lecture notes
https://scipy-lectures.org/advanced/image_processing/

[11] Image Processing using Numpy – Youtube
https://www.youtube.com/watch?v=bI2raX05vlY&ab_channel=Autonise

[12] image-processing – Github Topics
https://github.com/topics/image-processing

[13] Image Processing - Towards Data Science
https://towardsdatascience.com/tagged/image-processing

[14] Nguyễn Lê Tấn Phúc – Create_ellipse_mask

[15] Image Processing – GeeksforGeeks
https://www.geeksforgeeks.org/image-processing/