



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

Môn XỬ LÝ ẢNH SỐ & VIDEO SỐ

BÁO CÁO BTTH

SINH VIÊN THỰC HIỆN:

1712296 **Nguyễn Văn Tam Bình**

1712898 **Trần Việt Văn**

1712919 **Lê Văn Vũ**

LÝ THUYẾT/ HD THỰC HÀNH:

Thầy **Lý Quốc Ngọc**

Thầy **Phạm Minh Hoàng**

Thầy **Phạm Thanh Tùng**

MỤC LỤC:

Bài tập	Trang
LAB1	2/18
LAB2	9/18
LAB3	11/18
LAB4	13/18
LAB5	16/18
LAB6	18/18

Link drive:

https://drive.google.com/drive/folders/1Tin_suaIWkHtszQWiz45YC7iCXLug3jd?usp=sharing

A. LAB1:

SV Thực hiện: Tam Bình_1712296(câu 6-9), Việt Văn_1712898(câu 3-4),
Văn Vũ_1712919(câu 1a,b; 2a,b)

I. Phần BT 1.1 - 1.4:

Nguồn tham khảo:

<https://www.codeproject.com/Tips/314506/Convert-RGB-to-Gray-Scale-without-using-Pointers>

<https://stackoverflow.com/questions/54173733/convert-rgb-to-grayscale-in-bare-c>

Câu 1a:

Ảnh GrayScale (thang độ xám) có nhiều sắc thái khác nhau của màu xám (từ đen đến trắng) có trong hình ảnh. Mỗi pixel trong ảnh thang độ xám được lưu trữ dưới dạng một byte tức là 8 bit.

→ Do đó, giá trị pixel có thể nằm trong khoảng từ 0 đến 255.

→ Trong đó 0 đại diện cho màu đen và 255 đại diện cho màu trắng.

Tất cả các giá trị trung gian có màu xám với thành phần màu đen tăng dần khi giá trị pixel tiến đến 0. Do đó, các giá trị pixel chỉ mang thông tin cường độ trong đó màu đen biểu thị cường độ yếu nhất so với màu trắng có cường độ mạnh nhất.

Do đó hình ảnh thang độ xám được cho là chỉ có 1 kênh.

$$X = (a * R + b * G + c * B)$$

Tùy thuộc vào các giá trị của a, b, c, chúng ta có hai phương thức chính để chuyển đổi hình ảnh RGB thành GreyScale.

→ Bài này, em sẽ chuyển đổi RGB sang Grayscale bằng **PP độ chói**:

$$X = 0,3 * R + 0,59 * G + 0,11 * B$$

Câu 1b:

Chuyển đổi ảnh xám sang ảnh màu → ta đặt R=G=B=grayscale.

Câu 2a, 2b:

Nguồn tham khảo: <https://www.rapidtables.com/convert/color/>

Source code tham khảo:

<https://gist.github.com/kuathadianto/200148f53616cbd226d993b400214a7f>

<https://gist.github.com/ytomino/857105>

*RGB to HSV conversion formula:

The R, G, B values are divided by 255 to change the range from 0..255 to 0..1:

$$R' = R/255$$

$$G' = G/255$$

$$B' = B/255$$

$$Cmax = \max(R', G', B')$$

$$Cmin = \min(R', G', B')$$

$$\Delta = Cmax - Cmin$$

Hue calculation:

$$H = \begin{cases} 0^\circ & \Delta = 0 \\ 60^\circ \times \left(\frac{G' - B'}{\Delta} \bmod 6 \right) & , C_{max} = R' \\ 60^\circ \times \left(\frac{B' - R'}{\Delta} + 2 \right) & , C_{max} = G' \\ 60^\circ \times \left(\frac{R' - G'}{\Delta} + 4 \right) & , C_{max} = B' \end{cases}$$

















Saturation calculation:

$$S = \begin{cases} 0 & , C_{max} = 0 \\ \frac{\Delta}{C_{max}} & , C_{max} \neq 0 \end{cases}$$

Value calculation:

$$V = C_{max}$$

RGB to HSV color table

Color	Color name	Hex	(R,G,B)	(H,S,V)
	Black	#000000	(0,0,0)	(0°,0%,0%)
	White	#FFFFFF	(255,255,255)	(0°,0%,100%)
	Red	#FF0000	(255,0,0)	(0°,100%,100%)
	Lime	#00FF00	(0,255,0)	(120°,100%,100%)
	Blue	#0000FF	(0,0,255)	(240°,100%,100%)
	Yellow	#FFFF00	(255,255,0)	(60°,100%,100%)
	Cyan	#00FFFF	(0,255,255)	(180°,100%,100%)
	Magenta	#FF00FF	(255,0,255)	(300°,100%,100%)
	Silver	#BFBFBF	(191,191,191)	(0°,0%,75%)
	Gray	#808080	(128,128,128)	(0°,0%,50%)
	Maroon	#800000	(128,0,0)	(0°,100%,50%)
	Olive	#808000	(128,128,0)	(60°,100%,50%)
	Green	#008000	(0,128,0)	(120°,100%,50%)
	Purple	#800080	(128,0,128)	(300°,100%,50%)
	Teal	#008080	(0,128,128)	(180°,100%,50%)
	Navy	#000080	(0,0,128)	(240°,100%,50%)

*HSV to RGB conversion formula:

When $0 \leq H < 360$, $0 \leq S \leq 1$ and $0 \leq V \leq 1$:

$$C = V \times S$$










$$X = C \times (1 - |(H / 60^\circ) \bmod 2 - 1|)$$

$$m = V - C$$

$$(R', G', B') = \begin{cases} (C, X, 0) & , 0^\circ \leq H < 60^\circ \\ (X, C, 0) & , 60^\circ \leq H < 120^\circ \\ (0, C, X) & , 120^\circ \leq H < 180^\circ \\ (0, X, C) & , 180^\circ \leq H < 240^\circ \\ (X, 0, C) & , 240^\circ \leq H < 300^\circ \\ (C, 0, X) & , 300^\circ \leq H < 360^\circ \end{cases}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255)$$

HSV to RGB color table

Color	Color name	(H,S,V)	Hex	(R,G,B)
	Black	(0°,0%,0%)	#000000	(0,0,0)
	White	(0°,0%,100%)	#FFFFFF	(255,255,255)
	Red	(0°,100%,100%)	#FF0000	(255,0,0)
	Lime	(120°,100%,100%)	#00FF00	(0,255,0)
	Blue	(240°,100%,100%)	#0000FF	(0,0,255)
	Yellow	(60°,100%,100%)	#FFFF00	(255,255,0)
	Cyan	(180°,100%,100%)	#00FFFF	(0,255,255)
	Magenta	(300°,100%,100%)	#FF00FF	(255,0,255)
	Silver	(0°,0%,75%)	#BFBFBF	(191,191,191)
	Gray	(0°,0%,50%)	#808080	(128,128,128)
	Maroon	(0°,100%,50%)	#800000	(128,0,0)
	Olive	(60°,100%,50%)	#808000	(128,128,0)
	Green	(120°,100%,50%)	#008000	(0,128,0)
	Purple	(300°,100%,50%)	#800080	(128,0,128)
	Teal	(180°,100%,50%)	#008080	(0,128,128)
	Navy	(240°,100%,50%)	#000080	(0,0,128)

Câu 3, 4: Tăng giảm độ sáng, độ tương phản của ảnh:

Nguồn tham khảo: https://docs.opencv.org/3.4/d3/dc1/tutorial_basic_linear_transform.html
<https://www.stdio.vn/articles/xu-ly-anh-voi-opencv-do-sang-do-tuong-phan-va-bieu-do-tan-so-histogram-387>

Cơ sở lý thuyết:

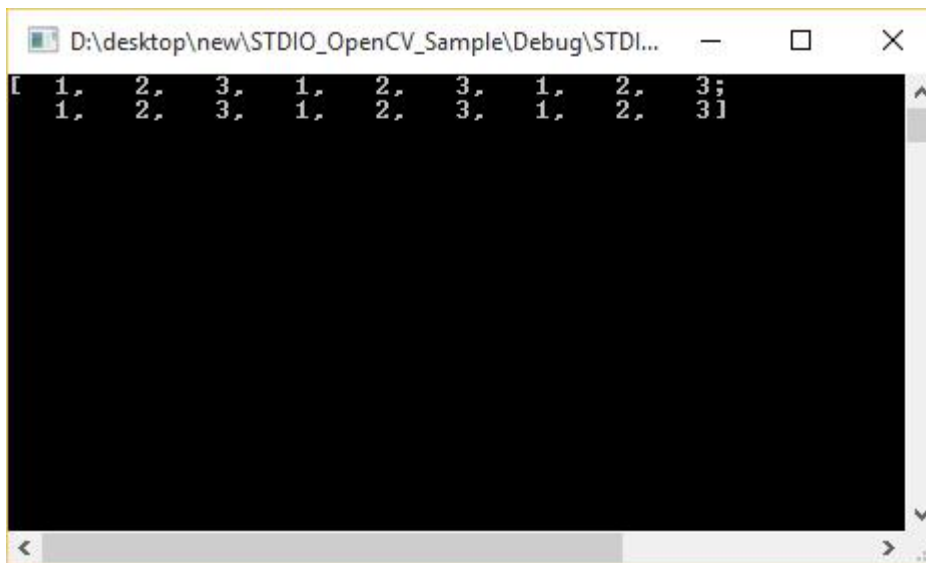
Một hình ảnh được lưu trữ là một ma trận các điểm ảnh (Pixel). Trong OpenCV nó được biểu diễn dưới dạng `cv::Mat`. Tôi có một đoạn code sau:

```
1. cv::Mat Example(2, 3, CV_8UC3, Scalar(1, 2, 3));
```

Phân tích

Đây là một cách khởi tạo đơn giản với kiểu dữ liệu `cv::Mat`. Bản chất của `cv::Mat` là một ma trận. Ví dụ này giống như cấu trúc một ảnh có không gian màu RGB.

- Hai tham số đầu tiên là kích thước của ma trận lần lượt là hàng (Rows) = 2 và cột (Columns) = 3 của Example.
 - CV_8UC3: Với mỗi điểm trên `cv::Mat Example` được lưu trữ dưới dạng: Loại unsigned char (0-255) có 8 bit và có 3 kênh.
- ```
1. CV_[Số bit cho mỗi kênh của điểm][Signed hoặc Unsigned][Kiểu dữ liệu]C[Số lượng kênh]
```
- Scalar(1, 2, 3): Với mỗi điểm trên Mat Example được khởi tạo các giá trị 1, 2 và 3 tương ứng với các kênh 1, kênh 2 và kênh 3.



Để có thể truy cập giá trị điểm ảnh (Pixel) với những kênh màu, bạn sử dụng mẫu sau:

```
1. image.at<cv::Vec3b>(row, col)[channel]
```

- row: Là hàng thứ row trong image.
- col: Là cột thứ col trong image.
- channel: Là kênh màu của image. Ví dụ: với không gian màu R-G-B, channel có giá trị lần lượt là channel = 1, channel = 2, channel = 3. Trong OpenCV kênh màu của không gian R-G-B được xếp theo thứ tự là B-G-R.

Để thay đổi độ sáng, độ tương phản của một tấm ảnh, bạn sử dụng công thức sau:

$$1. \quad G(x,y) = A \cdot F(x, y) + B$$

### Phân tích

- $F(x, y)$ : là giá trị pixel trong ảnh chưa được xử lý ở vị trí  $(x, y)$ .
- $G(x, y)$ : là giá trị pixel trong ảnh đã được xử lý ở vị trí  $(x, y)$ .
- $A$ : Là giá trị đặc trưng cho chênh lệch độ tương phản của  $G(x, y)$  và  $F(x, y)$ . Hay nói một cách khác chính là: Ảnh  $G$  tương phản gấp  $A$  lần với ảnh  $F$ .
- $B$ : Với  $B \neq 0$ , ảnh  $G$  có độ sáng thay đổi một lượng là  $B$ .

### Ví dụ

```
1. // www.stdio.vn
2. // www.stdio.vn/users/index/11/truong-dat
3. #include <stdio.h>
4. #include "opencv2/core/core.hpp"
5. #include "opencv2/highgui/highgui.hpp"
6. #include "opencv2/imgproc/imgproc.hpp"
7.
8. using namespace cv;
9.
10. int main()
11. {
12. // Read image
13. Mat image = imread("stdio.png", CV_LOAD_IMAGE_COLOR);
14. Mat imageDst = image.clone(); // sao chép 2 Mat
15. // Check for valid
16. if (!image.data)
17. {
18. printf("Could not open or find the image\n");
19. return -1;
20. }
21.
22. #define NUMBER_CHANNEL_RGB_COLOR_MODEL 3
23. double contrast = 2.0; // A
24. int brightness = 30; // B
25. for (int indexRow = 0; indexRow < image.rows; indexRow++) {
26. for (int indexCol = 0; indexCol < image.cols; indexCol++) {
27. for (int channel = 0; channel < NUMBER_CHANNEL_RGB_COLOR_MODEL; channel++) {
28. imageDst.at<Vec3b>(indexRow, indexCol)[channel] = saturate_cast<uchar>
```

```

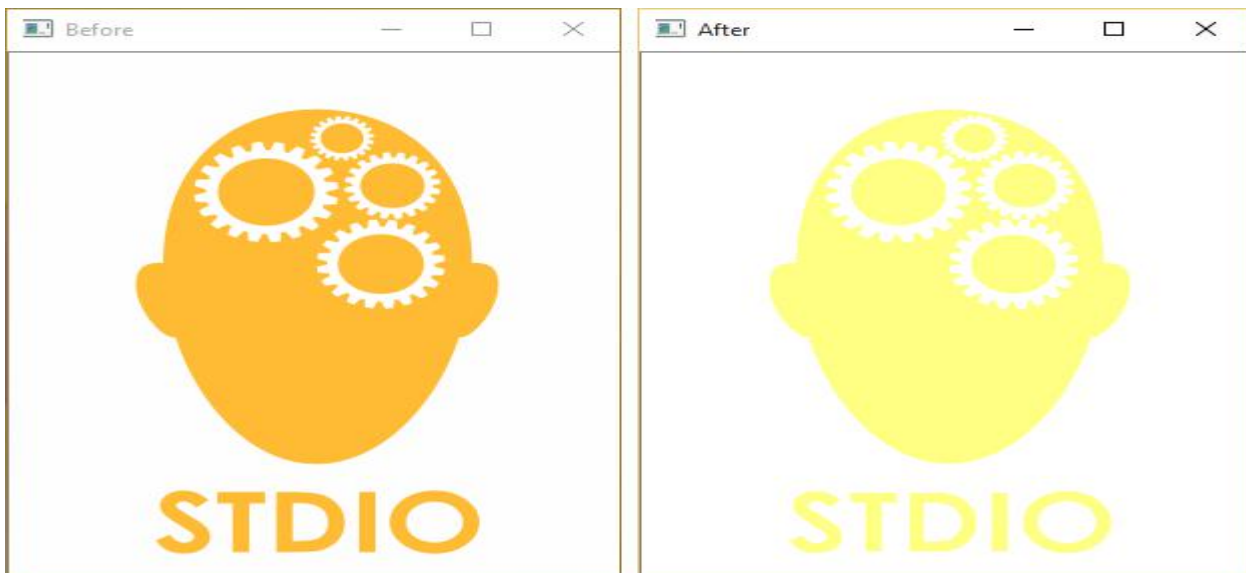
29. (contrast*(imageDst.at<Vec3b>(indexRow, indexCol)[channel]) + bright-
ness);
30. }
31. }
32. }
33.
34. imshow("Before", image);
35. imshow("After", imageDst);
36.
37. // Wait input and exit
38. waitKey(0);
39.
40. return 0;
41. }

```

## Phân tích

- Phương thức `saturate_cast()` giúp tránh tình trạng tràn số hay kiểu dữ liệu không tương thích. Vì giá trị điểm ảnh (Pixel) phải nằm trong khoảng 0-255 (unsigned char), trong khi theo công thức  $G(x,y) = A \cdot F(x,y) + B$  thì giá trị có thể là  $G(x,y) < 0$  hoặc  $G(x,y) > 255$ .

1. `uchar a = saturate_cast<uchar>(-100); // a = 0 (UCHAR_MIN)`
2. `short b = saturate_cast<short>(33333.33333); // b = 32767 (SHRT_MAX)`



## II. Phần BT 1.5 - 1.8:

### 1. Cách tiếp cận bài toán:

1.1) Tính histogram của ảnh màu, ảnh xám

#### 1.1.a) Cơ sở lý thuyết

Histogram vector thể hiện các pixel của cùng giá trị mức xám

#### 1.1.b) Thực hành

Duyệt qua tất cả pixel trên ảnh, đếm số pixel có cùng giá trị độ xám: 0-255

Đối với ảnh màu tính cả 3 kênh màu RGB

1.2) Cân bằng histogram ảnh màu, ảnh xám

#### 1.2.a) Cơ sở lý thuyết:

1. Từ ảnh ban đầu ta tính được histogram của ảnh cho từng kênh màu
2. Tính lược đồ xám tích lũy, chuẩn hóa về đoạn 0-255
3. Ánh xạ tạo ảnh kết quả

#### 1.2.b) Thực hành:



### 3.1. Phép biến đổi màu

#### Giải thuật (Cân bằng lược đồ xám)

**B1.** Khởi tạo mảng H chiều dài nG với giá trị 0

(giả sử ảnh f kích thước NxM có nG mức xám)

**B2.** Tính lược đồ độ xám của ảnh f, lưu vào H

$$H[f(x, y)] += 1$$

**B3.** Tính lược đồ độ xám tích lũy của f, lưu vào T

$$T[0] = H[0]$$

$$T[p] = T[p-1] + H[p], \quad p = 1, 2, \dots, nG-1$$

**B4.** Chuẩn hóa T về đoạn [0;nG-1]

$$T[p] = \text{round}((nG-1/NM)T[p])$$

**B5.** Tạo ảnh kết quả g:  $g(x, y) = T[f(x, y)]$

1.3) Vẽ histogram của ảnh màu, ảnh xám:

#### 1.3.a) Cơ sở lý thuyết:

Từ histogram của ảnh, tiến hành vẽ biểu đồ cột thể hiện giá trị ảnh và số lượng pixel có giá trị tương ứng.

#### 1.3.b) Thực hành

1. Tính được histogram từng kênh màu hay 1 kênh xám
2. Tạo ảnh kết quả tương ứng kích thước đủ cho số lượng mỗi kênh là 255 chiều rộng, và chiều cao ảnh là giá trị Max trong histogram
3. Chuẩn hóa giá trị histogram về đoạn 0-Max
4. Tạo ảnh kết quả

### 2. Chú thích hàm:

```
/*
 Hàm nhận vào một ảnh, thay đổi độ sáng của ảnh này và lưu kết quả vào ảnh mới
 Tham số:
 sourceImage : ảnh ban đầu
```



|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <pre> destinationImage: ảnh kết quả b      : giá trị số nguyên dùng để thay đổi độ sáng của ảnh Hàm trả về: 1: Nếu thành công thì trả về ảnh kết quả (ảnh gốc vẫn giữ nguyên giá trị) 0: Nếu không tạo được ảnh kết quả hoặc ảnh input không tồn tại */ int ChangeBrightness(const Mat&amp; sourceImage, Mat&amp; destinationImage, short b); </pre>                                                                                                                                                                       |
| <pre> /*     Hàm nhận vào một ảnh, thay đổi độ tương phản của ảnh này và lưu kết quả vào ảnh mới     Tham so :     sourceImage : ảnh ban đầu     destinationImage : ảnh kết quả     c          : giá trị số thực dùng để thay đổi độ tương phản của ảnh     Hàm trả về:     1: Nếu thành công thì trả về ảnh kết quả (ảnh gốc vẫn giữ nguyên giá trị)     0: Nếu không tạo được ảnh kết quả hoặc ảnh input không tồn tại     */ int ChangeContrast(const Mat&amp; sourceImage, Mat&amp; destinationImage, float c); </pre> |
| <pre> /*Hàm tính histogram     Tham số là ảnh     Giá trị trả về: Mảng histogram ảnh xám     Tính số giá trị cùng mức xám của các pixel     */ vector&lt;double&gt; CalculatorHistogram(const Mat&amp; sourceImage,vector&lt;double&gt;histMatrix); </pre>                                                                                                                                                                                                                                                                 |
| <pre> /*     Hàm cân bằng lược đồ màu tổng quát cho ảnh bất kỳ     Tham so :     sourceImage : ảnh ban đầu     destinationImage : ảnh kết quả     Hàm trả về:     1: Nếu thành công thì trả về ảnh kết quả (ảnh gốc vẫn giữ nguyên giá trị)     0: Nếu không tạo được ảnh kết quả hoặc ảnh input không tồn tại     */ int HistogramEqualization(const Mat&amp; sourceImage, Mat&amp; destinationImage); </pre>                                                                                                             |
| <pre> /*Vẽ histogram ảnh     Hàm cân bằng lược đồ màu tổng quát cho ảnh bất kỳ     Tham so :     histMatrix : vector histogram ảnh ban đầu     histImage : ảnh histogrma     Hàm trả về:     1: Nếu thành công     0: Nếu không tạo được ảnh kết quả hoặc ảnh input không tồn tại     */ int DrawHistogram(vector&lt;double&gt;histMatrix, Mat&amp; histImage); </pre>                                                                                                                                                     |

## LAB2: SV thực hiện: 1712898\_ Trần Việt Văn:

### 1.Đánh giá

| Câu | Yêu cầu                                 | Tên câu lệnh | Tham số câu lệnh | Mức độ hoàn thành |
|-----|-----------------------------------------|--------------|------------------|-------------------|
| 1   | Cài đặt lớp AffineTransform             |              |                  | 100%              |
| 2   | Cài đặt lớp nội suy láng giềng gần nhất |              |                  | 100%              |

|   |                                                      |           |             |      |
|---|------------------------------------------------------|-----------|-------------|------|
| 3 | Cài đặt lớp nội suy song tuyến tính                  |           |             | 0%   |
| 4 | Cài đặt hàm Transform trong lớp GeometricTransformer |           |             | 0%   |
| 5 | Phóng to, thu nhỏ ảnh                                | --zoom    | Hệ số zoom  | 50%  |
| 6 | Thay đổi kích thước ảnh                              | --resize  | Kích cỡ mới | 0%   |
| 7 | Xoay ảnh quanh tâm (bảo toàn nội dung)               | --rotate  | Góc xoay    | 0%   |
| 8 | Xoay ảnh quanh tâm (không bảo toàn nội dung)         | --rotateN | Góc xoay    | 0%   |
| 9 | Đổi xứng ảnh qua trục đứng (Oy) hay trục ngang (Ox)  | --flip    | Oy hay Ox   | 100% |

## 2. Các hàm

Hàm nhân ma trận với ma trận affine

```
Mat affineMultiply(Mat _matrixTransformer, Mat point)
```

Hàm tạo ma trận translate

```
void AffineTransform::Translate(float dx, float dy)
```

Hàm tạo ma trận rotate

```
void AffineTransform::Rotate(float angle)
```

Hàm tạo ma trận scale

```
void AffineTransform::Scale(float sx, float sy)
```

Hàm biến đổi 1 điểm

```
void AffineTransform::TransformPoint(float &x, float &y)
```

Hàm tính NNI

```
uchar* NearestNeighborInterpolate::Interpolate(float tx, float ty, uchar* pSrc, int srcWidthStep, int nChannels)
```

Hàm lấy scale ảnh

```
int GeometricTransformer::Scale(const Mat &srcImage,
 Mat &dstImage,
 float sx, float sy,
 PixelInterpolate* interpolator)
```

Hàm lật ngược ảnh

```
int GeometricTransformer::Flip(
 const Mat &srcImage,
 Mat &dstImage,
 bool direction,
 PixelInterpolate* interpolator)
```

## LAB3:

SV thực hiện:

| STT | NỘI DUNG CÔNG VIỆC          | PHÂN CÔNG    | HOÀN THÀNH |
|-----|-----------------------------|--------------|------------|
| 1   | Hàm chập ảnh                | Vũ_1712919   | 100%       |
| 2   | Lọc trung bình              | Bình_1712296 | 100%       |
| 3   | Lọc trung vị                | Bình_1712296 | 80%        |
| 4   | Lọc Gauss                   | Bình_1712296 | 100%       |
| 5   | Phát hiện biên cạnh Sobel   | Vũ_1712919   | 100%       |
| 6   | Phát hiện biên cạnh Prewitt | Vũ_1712919   | 100%       |
| 7   | Phát hiện biên cạnh Laplace | Vũ_1712919   | 100%       |

### Các nguồn tham khảo:

<https://www.programming-techniques.com/2013/02/calculating-convolution-of-image-with-c-2.html>

<https://github.com/heshanera/EdgeDetector>

<https://github.com/lakehanne/Savitzky-Golay>

<https://www.kancloud.cn/digest/imageproebow/122463>

[https://blog.csdn.net/lsh\\_2013/article/details/44938423](https://blog.csdn.net/lsh_2013/article/details/44938423)

<https://github.com/LariscusObscurus/EdgeDetection>

<https://github.com/mdclark01/EdgeDetection>

## I. Đánh giá chung

### 1. Phần làm được:

- Tất cả các yêu cầu hàm đề bài
- Xử lý được trên ảnh xám
- Ma trận kernel có kích thước 3x3

### 2. Phần chương làm được

- Không khái quát được kích thước ma trận kernel
- Tham khảo nhiều tài nguyên internet

Mức độ hoàn thành đồ án 100%

## II. Tổ chức và thiết kế đồ án

### 1. Cách tiếp cận bài toán:

#### a) Hàm tính chập ảnh

#### b) Lọc trung bình

b.1) Cơ sở lý thuyết:

Giá trị độ xám ảnh kết quả có được bằng giá trị độ xám của pixel tương ứng ảnh

#### 5.1. Toán tử trung bình

$$g(x, y) = \sum_i \sum_j f(x-i, y-j) \cdot h(i, j), \\ (i, j) \in O$$

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

nguồn và 8 pixel gần nhất, lấy trung bình 9 giá trị độ xám chính là giá trị ảnh kết quả

**b.2) Thực hành:**

Ảnh kết quả = ảnh gốc tích chập với ma trận kernel h

1. SetKernel là vector chứa số thực gồm 9 phần tử 1.0/9
2. Gọi hàm tính chập ảnh ban đầu và Kernel ta được ảnh kết quả

**c) Lọc trung vị:**

**c.1) Cơ sở lý thuyết:**

Giá trị độ xám ảnh kết quả có được bằng giá trị độ xám của pixel tương ứng ảnh nguồn và 8 pixel gần nhất, sắp xếp 9 giá trị độ xám tăng dần, giá trị trung vị của dãy tăng chính là giá trị độ xám ảnh kết quả tại pixel đang xét

**c.2) Thực hành**

Ảnh kết quả = med(sort(I1..I9))

1. Duyệt từng phần tử pixel ảnh ban đầu, bỏ qua các pixel biên
2. Tìm 8 giá trị láng giềng gần nhất đưa vào vector 9 phần tử (điểm đang xét)
3. Sắp tăng các phần tử trong vector bằng thuật toán selectionSort
4. Phần tử thứ 5 trong vector gán cho giá trị tọa độ tương ứng với điểm đang xét vào ảnh kết quả

**d) Lọc Gauss**

**d.1) Cơ sở lý thuyết:**

Giá trị độ xám ảnh kết quả có được bằng giá trị độ xám của pixel tương ứng ảnh nguồn và giá trị của các phần tử láng giềng gần nhất được tính theo công thức  $h(i,j)$ , tiến hành tính chập ảnh ta được ảnh kết quả

**d.2) Thực hành**

1. Nhập giá trị sigma
2. Set kernel theo công thức, và tính tổng các giá trị trong kernel
3. Chuẩn hóa các giá trị trong kernel theo sumker
4. Tích chập ảnh ban đầu với ma trận kernel

**e) Phát hiện biên bằng sobel và prewitt:**

| sobel                                                                                                                                                          | prewitt                                                                                                                                                        |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $h_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$ | $h_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad h_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ |

### 5.3. Toán tử Trung vị

$$g(x, y) = \text{med}\{f(x+i, y+j), (i, j) \in O\}$$

Giả sử  $\{f(x+i, y+j), (i, j) \in O\}$  được sắp thứ tự tăng dần và ký hiệu lại:

$$I_1 < I_2 < \dots < I_n, n = 2v + 1$$

$$\text{med}(I_i) = I_{v+1}$$

### 5.2. Toán tử Gaussian

$$g(x, y) = \sum_i \sum_j f(x-i, y-j) \cdot h(i, j),$$

$$(i, j) \in O$$

$$h(i, j) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{i^2+j^2}{2\sigma^2}}$$

## Magnitude

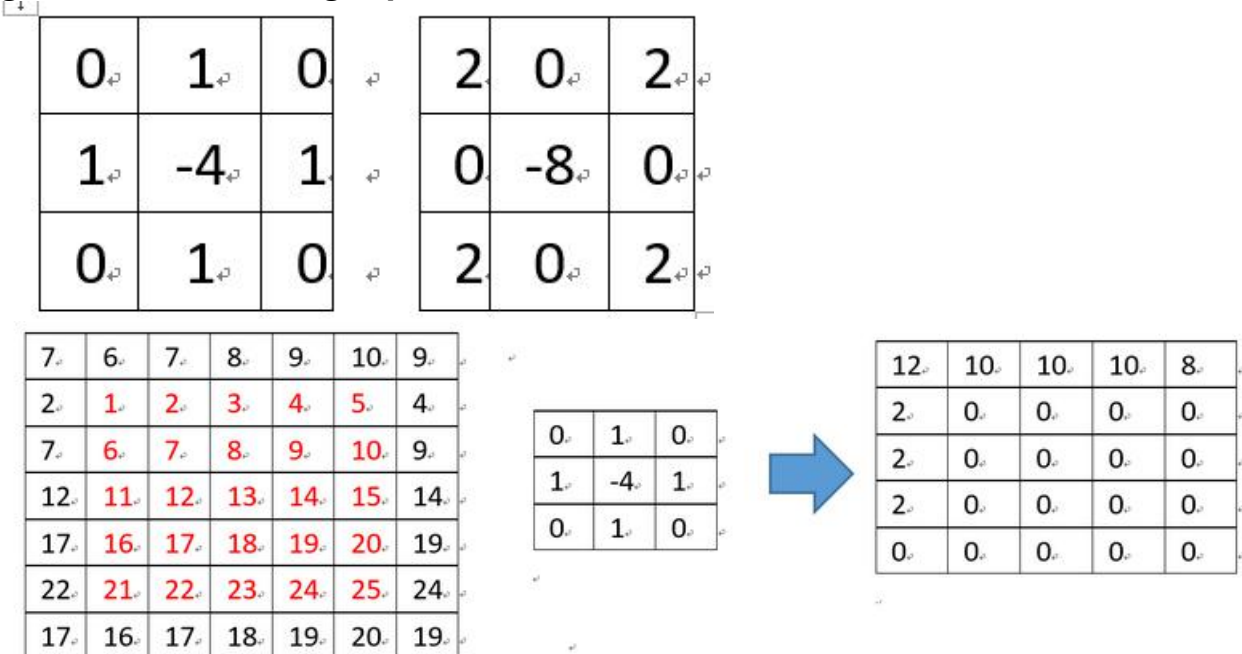
$$G = \sqrt{G_x^2 + G_y^2}$$

Note: To reduce the complexity of above square root calculation, gradient magnitude is some times approximated by absolute summation i.e  $G = \text{abs}(G_x) + \text{abs}(G_y)$ .

## Direction

$$\Theta = \text{atan}\left(\frac{G_y}{G_x}\right)$$

### g) Phát hiện biên bằng Laplace:



## LAB4 (PHÁT HIỆN BIÊN CẠNH BẰNG CANNY):

SV thực hiện: 1712919\_ Lê Văn Vũ:

Nguồn tham khảo: <https://www.itread01.com/content/1543455430.html>

### 1. Giải thuật phát hiện cạnh Canny - Canny Edge Detection:

Trong hình ảnh, thường tồn tại các thành phần như: vùng trơn, góc / cạnh và nhiễu. Cạnh trong ảnh mang đặc trưng quan trọng, thường là thuộc đối tượng trong ảnh (object). Do đó, để phát hiện cạnh trong ảnh, giải thuật Canny là một trong những giải thuật phổ biến / nổi tiếng nhất trong Xử lý ảnh.

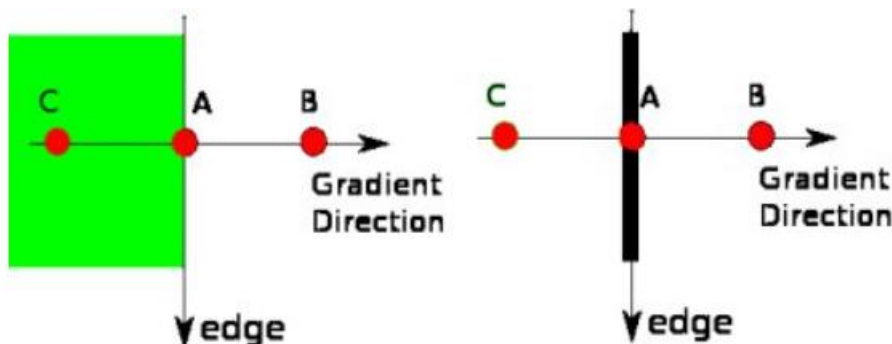
Giải thuật phát hiện cạnh Canny gồm 4 bước chính sau:

- **Giảm nhiễu:** Làm mờ ảnh, giảm nhiễu dùng bộ lọc Gaussian kích thước 5x5. Kích thước 5x5 thường hoạt động tốt cho giải thuật Canny. Dĩ nhiên bạn cũng có thể thay đổi kích thước của bộ lọc làm mờ cho phù hợp.
- **Tính Gradient và hướng gradient:** ta dùng bộ lọc Sobel X và Sobel Y (3x3) để tính được ảnh đạo hàm  $G_x$  và  $G_y$ . Sau đó, ta tiếp tục tính ảnh Gradient và góc của Gradient theo công thức. Ảnh đạo hàm  $G_x$  và  $G_y$  là ma trận (ví dụ: 640x640), thì kết quả tính ảnh đạo hàm Edge Gradient cũng là một ma trận (640x640), mỗi pixel trên ma trận này thể hiện độ lớn của biến đổi mức sáng ở vị trí tương ứng trên ảnh gốc. Tương tự, ma trận Angle cũng có cùng kích thước (640x640), mỗi pixel trên Angle thể hiện góc, hay nói cách khác là hướng của cạnh. Ví dụ dễ hiểu, nếu góc gradient là 0 độ, thì cạnh của ta trên ảnh sẽ là một đường thẳng đứng (tức tạo góc 90 độ so với trục hoành) (**vuông góc hướng gradient**). Khi tính toán, giá trị hướng gradient sẽ nằm trong đoạn  $[-180, 180]$  độ, ta không giữ nguyên các góc này mà gom các giá trị này về 4 bin đại diện cho 4 hướng: hướng ngang (0 độ), hướng chéo bên phải (45 độ), hướng dọc (90 độ) và hướng chéo trái (135 độ).

$$Edge\_Gradient (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle (\theta) = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

- **Non-maximum Suppression** (viết tắt NMS): loại bỏ các pixel ở vị trí không phải cực đại toàn cục. Ở bước này, ta dùng một filter 3x3 lần lượt chạy qua các pixel trên ảnh gradient. Trong quá trình lọc, ta xem xét xem độ lớn gradient của pixel trung tâm có phải là cực đại (lớn nhất trong cục bộ - local maximum) so với các gradient ở các pixel xung quanh. Nếu là cực đại, ta sẽ ghi nhận sẽ giữ pixel đó lại. Còn nếu pixel tại đó không phải là cực đại lân cận, ta sẽ set độ lớn gradient của nó về zero. Ta chỉ so sánh pixel trung tâm với 2 pixel lân cận theo **hướng gradient**. Ví dụ: nếu hướng gradient đang là 0 độ, ta sẽ so pixel trung tâm với pixel liền trái và liền phải nó. Trường hợp khác nếu hướng gradient là 45 độ, ta sẽ so sánh với 2 pixel hàng xóm là góc trên bên phải và góc dưới bên trái của pixel trung tâm. Tương tự cho 2 trường hợp hướng gradient còn lại. Kết thúc bước này ta được một mặt nạ nhị phân. Tham khảo hình dưới:



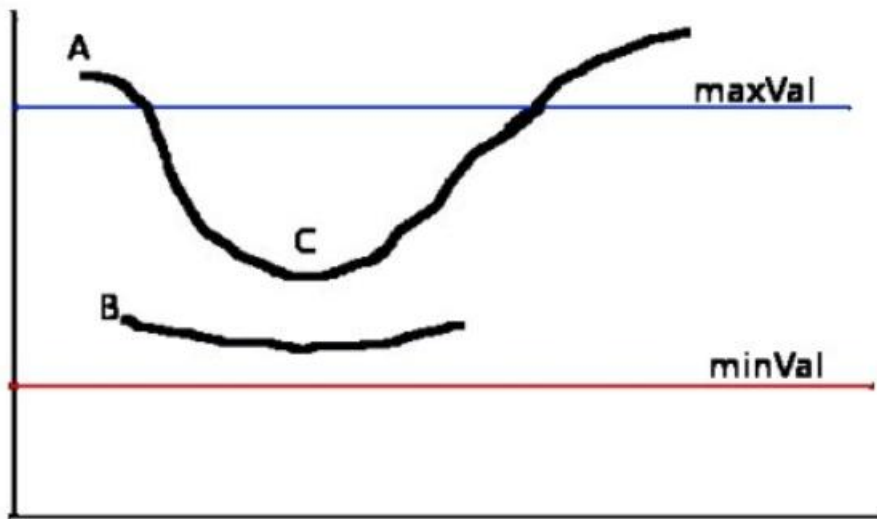
=> Bước này chỉ giữ lại những pixel thuộc cạnh mà có độ lớn gradient lớn nhất

Xem xét 3 pixel trong vùng 3 x 3 xung quanh pixel (x,y):

- Nếu  $\theta(x, y) = 00$  thì (x+1, y), (x, y) và (x-1, y) được xem xét.
- Nếu  $\theta(x, y) = 90$  thì (x, y+1), (x, y) và (x, y-1).
- Nếu  $\theta(x, y) = 45$  thì (x+1, y+1), (x, y) và (x-1, y-1).
- Nếu  $\theta(x, y) = 135$  thì (x-1, y+1), (x, y) và (x+1, y-1).

Nếu pixel (x, y) có gradient lớn nhất của 3 pixel xem xét thì pixel đó là cạnh.

- **Lọc ngưỡng:** ta sẽ xét các pixel dương trên mặt nạ nhị phân kết quả của bước trước. Nếu giá trị gradient vượt ngưỡng **max\_val** thì pixel đó **chắc chắn là cạnh**. Các pixel có độ lớn gradient nhỏ hơn ngưỡng **min\_val** sẽ bị loại bỏ. Còn các pixel nằm trong khoảng 2 ngưỡng trên sẽ được xem xét rằng nó có nằm liên kề với những pixel được cho là "chắc chắn là cạnh" hay không. Nếu liền kề thì ta giữ, còn không liền kề bất cứ pixel cạnh nào thì ta loại. Sau bước này ta có thể áp dụng thêm bước hậu xử lý loại bỏ nhiễu (tức những pixel cạnh rời rạc hay cạnh ngắn) nếu muốn. Ảnh minh họa về ngưỡng lọc:



Lý thuyết giải thuật Canny trong bài viết này tham khảo chính tại:

[https://docs.opencv.org/master/da/d22/tutorial\\_py\\_canny.html](https://docs.opencv.org/master/da/d22/tutorial_py_canny.html)

[https://en.wikipedia.org/wiki/Canny\\_edge\\_detector](https://en.wikipedia.org/wiki/Canny_edge_detector)

**2. Chương trình cài đặt (ngôn ngữ C/C++):** gồm các hàm sau:

```
/Gaussian kernel
int Gaussian_kernel(int kernel_size, int sigma, Mat& kernel);
```

```
/*
Tính giá trị độ dốc và hướng
imageSource: hình ảnh thang độ thô
hình ảnh độ dốc hướng X của imageX
```

```

hình ảnh gradient hướng hình ảnh Y
gradXY độ lớn của độ dốc tại điểm này
góc hướng theta độ dốc theta = arctan (imageY / imageX)
*/
int GradDirection(const Mat imageSource, Mat& imageX, Mat& imageY, Mat& gradXY, Mat& theta);

 //Đọc theo hướng của độ dốc của điểm này, so sánh biên độ của hai điểm trước và sau
 điểm.Nếu điểm lớn hơn hai điểm trước và sau, nó được giữ lại.
 //Nếu điểm nhỏ hơn bất kỳ điểm nào trong hai điểm trước và sau, điểm đó được đặt thành
 0;

 //đầu vào imageInput để có được hình ảnh gradient
 //đầu ra hình ảnh không bị triệt tiêu tối đa bởi imageOutput
 //góc hướng theta của từng pixel
 //độ dốc hướng X của imageX
 //độ dốc hình ảnh hướng Y
int NonLocalMaxValue(const Mat imageInput, Mat& imageOutput, const Mat& theta, const Mat&
imageX, const Mat& imageY);

/*Cơ chế của ngưỡng kép là:
Chỉ định giá trị ngưỡng low A và giá trị ngưỡng high B và chọn 70% tổng biểu đồ là B và B gấp
1,5 đến 2 lần kích thước của A;
Nếu giá trị màu xám nhỏ hơn A, đặt giá trị này thành 0; nếu giá trị màu xám lớn hơn B, hãy
đặt giá trị này thành 255;*/

int DoubleThreshold(Mat& imageInput);

/*
Nếu giá trị màu xám nằm giữa A và B, kiểm tra xem 8 pixel liền kề với pixel có giá trị xám là
255 hay không.
Nếu không có 255, nó chỉ ra rằng đây là điểm tối đa cục bộ bị cô lập, được loại trừ và được
đặt thành 0;
Nếu có 255, điều đó có nghĩa đây là vật liệu có thể xây dựng có "đường viền" với các cạnh
khác. Đặt nó thành 255.
Bước này được lặp lại cho đến khi pixel cuối cùng được kiểm tra.

==> Thuật toán theo dõi vùng lân cận bắt đầu từ các pixel có giá trị 255 để tìm các điểm xung
quanh đáp ứng các yêu cầu
và đặt các điểm đáp ứng các yêu cầu thành 255.
=> Sau đó sửa đổi các giá trị tọa độ của i, j và các giá trị i, j được khôi phục.
Trên cơ sở thay đổi i, j, tiếp tục tìm các điểm xung quanh 255 đáp ứng yêu cầu.
=>Sau khi tất cả các điểm kết nối 255 được sửa đổi, hãy đặt tất cả các điểm tối đa cục bộ
được đề cập ở trên thành 0;
(thuật toán có thể tiếp tục được tối ưu hóa).

Tham số 1, imageInput: hình ảnh độ dốc của đầu vào và đầu ra
Tham số 2, lowTh: ngưỡng thấp
Tham số 3, highTh: ngưỡng cao
*/
int Apply(Mat& imageInput, Mat& imageOutput);

```

## LAB5 (PHÂN NGƯỠNG, PHÂN ĐOẠN):

**SV thực hiện: 1712298\_ Nguyễn Văn Tam Bình:**

***Nguồn tham khảo:***

<https://github.com/ImageMagick/ImageMagick/blob/master/MagickCore/threshold.c>

<https://www.codeproject.com/Articles/37850/XMLFoundation-2>

<https://perso.liris.cnrs.fr/christian.wolf/software/binarize/>

<https://github.com/aditya1601/kmeans-clustering-cpp>



## I. Đánh giá:

| STT | Tên công việc          | Hoàn thành |
|-----|------------------------|------------|
| 1   | Phân ngưỡng tĩnh       | 100%       |
| 2   | Phân ngưỡng trung bình | 100%       |
| 3   | Phân ngưỡng trung vị   | 100%       |
| 4   | Phân ngưỡng Sauvola    | 100%       |
| 5   | Phân đoạn K_mean       | 90%        |

## III. Nội dung chương trình

Object: 255

Background: 0

### 1. Phân ngưỡng tĩnh

Có hai ngưỡng: low và up. Các pixel được coi là object nếu giá trị màu nằm trong khoảng giữa low và up.

`pixel = (pixel > low && pixel < up)? object: background`

### 2. Phân ngưỡng dựa trên trung bình

Là phương pháp phân ngưỡng cục bộ. Ngưỡng được tính dựa vào giá trị trung bình trừ đi hằng số C trong lân cận của pixel đang xét

`pixel = (pixel > mean - C)? object: background`  
Tham số C mặc định bằng 0, có thể thay đổi

### 3. Phân ngưỡng dựa trên trung vị

Là phương pháp phân ngưỡng cục bộ. Ngưỡng được tính dựa vào giá trị trung vị trừ đi hằng số C trong lân cận của pixel đang xét

`index_median = ceilf (GetPixel.size() / 2.0);  
median = GetPixel[index_median];  
pixel = (pixel > median - C)? object: background`  
Tham số C mặc định bằng 0, có thể thay đổi

### 4. Phân ngưỡng Sauvola

Ngưỡng được tính dựa vào trung bình và độ lệch chuẩn trong lân cận của pixel đang xét

`pixel = ( pixel > mean * ( 1 + k *  
( standard_deviation / r - 1 ) ) ) ? object :  
background`  
Tham số k trong khoảng [0..1], giá trị mặc định là 0.5  
Tham số r trong khoảng [0..255], giá trị mặc định là 120

- Độ lệch chuẩn cho biết độ phân tán của các giá trị trong bộ số liệu
- Phương sai là một giá trị đại diện cho độ phân tán của các số liệu so với giá trị trung bình

## 5. Phân đoạn K\_means

1. Tìm K pixel ngẫu nhiên trong ảnh
2. Tiến hành tính khoảng cách (khoảng cách giá trị màu), xếp các điểm pixel vào cụm gần nhất
3. Tính trung bình tất cả pixel trong mỗi cụm, cập nhật lại tâm cụm mới nếu khác cũ
4. Lặp lại bước 2 cho đến khi không còn cập nhật được tâm mới
5. Gán các giá trị pixel thuộc k tâm cuối cùng bằng với giá trị pixel của cụm đó

## IV. Hướng dẫn sử dụng

1. Chương trình chạy bằng cmd
2. Chọn đến đường dẫn labX.exe
3. Nhập vào đường dẫn file hình ảnh
4. Nhập vào tùy chọn và các giá trị tham số cần thiết

VD: Lab5.exe D:\hinhcho.jpg kmeans và nhập k = 3

## LAB6 (NÉN ẢNH):

**SV thực hiện: 1712898\_ Trần Việt Văn:**

***Nguồn tham khảo:***

[https://github.com/hmhuan/HCMUS\\_HK5\\_XLAV](https://github.com/hmhuan/HCMUS_HK5_XLAV)

### 1.Đánh giá

| Câu | Yêu cầu          | Tên câu lệnh | Tham số câu lệnh              | Mức độ hoàn thành |
|-----|------------------|--------------|-------------------------------|-------------------|
| 1   | Nén ảnh bằng SVD | --compress   | K: số thành phần được giữ lại | 100%              |
| 2   | Giải nén         | --decompress |                               | 100%              |

### 2.Các hàm

Hàm viết các ma trận tính được bằng SVD vào file nhị phân

int writeMatUSV(fstream& outputFile, const Mat& outputMat)

Hàm đọc các ma trận từ file nhị phân

int readMatUSV(fstream& inputFile, Mat& inputMat)

### 3.Cách thức hoạt động

Ảnh nhận vào được chia làm 3 kênh màu, tính SVD cho từng kênh màu, sau đó nén theo từng kênh màu và ghép lại thành ảnh hoành chính, các ma trận tính được lưu dưới dạng file nhị phân, khi phục hồi ảnh sẽ lấy các ma trận đó để suy ngược lại ảnh gốc