

Computer Vision

Đặng Văn Ngọc

Chapter 1 – Computer Vision

1.1 Khái niệm Computer Vision

1.2 OpenCV

1.3 Tổng kết

1.4 Bài tập

1.1 Khái Niệm

❖ Computer Vision là gì?

Thị giác máy tính là một lĩnh vực trong Artificial Intelligence ([Trí tuệ nhân tạo](#)) và Computer Science (Khoa học máy tính) nhằm giúp máy tính có được khả năng nhìn và hiểu giống như con người.

1.1 Khái Niệm

❖ Computer Vision là gì?

Thị giác máy tính (computer vision) được định nghĩa là một lĩnh vực bao gồm các phương pháp thu nhận, xử lý ảnh kỹ thuật số, phân tích và nhận dạng các hình ảnh và, nói chung là dữ liệu đa chiều từ thế giới thực để cho ra các thông tin số hoặc biểu tượng. *Theo Wikipedia*

Thị giác máy tính cũng được mô tả là sự tổng thể của một dải rộng các quá trình tự động, tích hợp và các thể hiện cho các nhận thức thị giác.

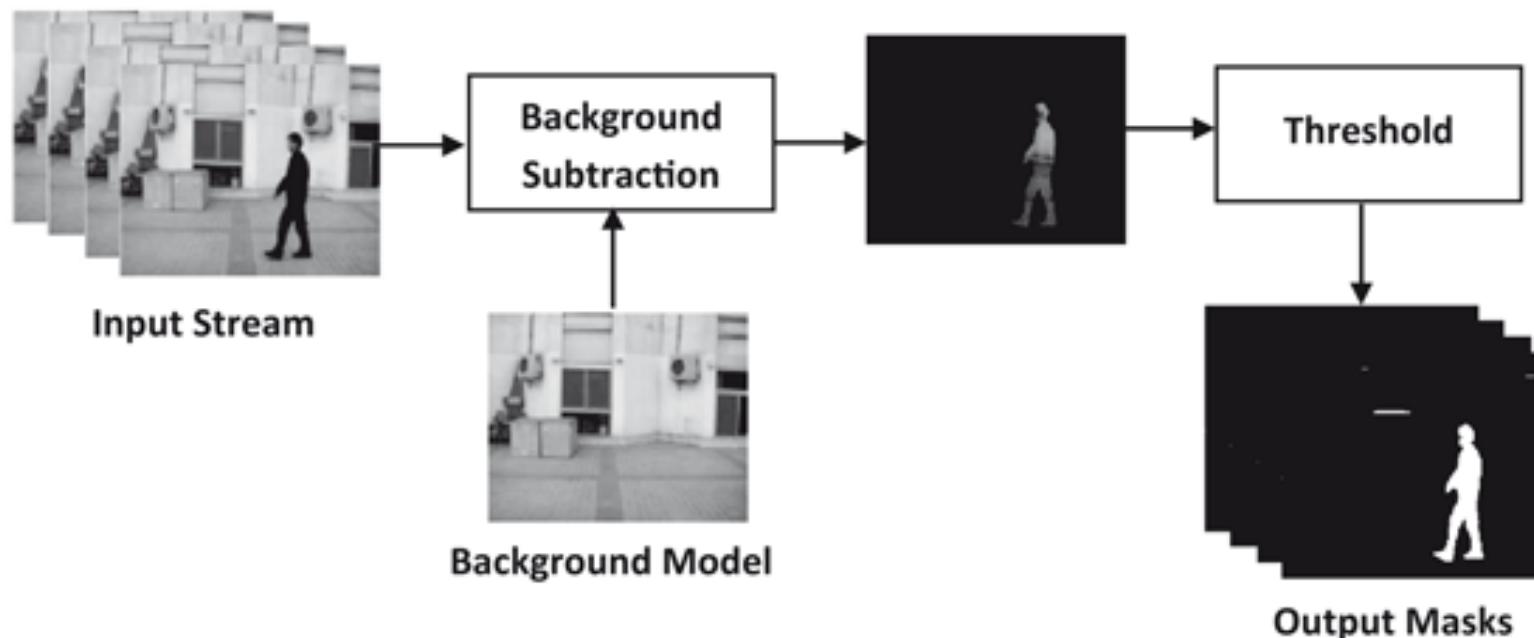
1.1 Khái Niệm

❖ Computer Vision hoạt động như thế nào?

- + **Xử lý hình ảnh:** Nhiệm vụ cơ bản của xử lý ảnh như tăng/giảm chất lượng ảnh, lọc nhiễu...
- + **Nhận diện mẫu:** Cung cấp cho nó hàng nghìn, hàng triệu hình ảnh nếu có thể đã được “đánh dấu” và sau đó áp dụng các kỹ thuật phần mềm hoặc thuật toán khác nhau cho phép máy tính tìm kiếm trong tất cả các phần tử có liên quan đến các mẫu được đánh dấu đó.

1.1 Khái Niệm

- ❖ Computer Vision hoạt động như thế nào?



1.1 Khái Niệm

❖ **Computer Vision có liên quan gì tới AI ?**

CV và AI sẽ luôn đi chung với nhau, phát triển cùng nhau.



1.1 Khái Niệm

❖ Úng dụng của Computer Vision

- + Xác định vật thể (Identify): xe chạy lấn làn hoặc quá tốc độ.
- + Theo dõi vật thể (Track): Theo dõi chiếc xe khả nghi trên đường
- + Đo đạc (Measure): Đo kích thước vật thể
- + Phát hiện vật thể (Detect): biển số xe, vết nứt kim loại, lỗi sơn, bản in xấu, đối tượng xâm nhập, vv
- + Phân loại vật thể (Classify): có cây, có người, có nhà cao tầng, hay đang có xe, có đồ ăn, có bãi biển, vv

Chapter 1 – Computer Vision

1.1 Khái niệm Computer Vision

1.2 OpenCV

1.3 Tổng kết

1.4 Bài tập

1.2 OpenCV

❖ OpenCV là gì?

Project OpenCV được bắt đầu từ Intel năm 1999 bởi Gary Bradsky.

OpenCV (Open Source Computer Vision Library): là thư viện nguồn mở hàng đầu cho Computer Vision và Machine Learning, và hiện có thêm tính năng tăng tốc GPU cho các hoạt động theo real-time.

1.2 OpenCV

❖ OpenCV là gì?

OpenCV được phát hành theo *giấy phép BSD* (*), do đó nó miễn phí cho cả học tập và sử dụng với mục đích thương mại. Nó có trên các giao diện C++, C, Python và Java và hỗ trợ Windows, Linux, Mac OS, iOS và Android. OpenCV được thiết kế để hỗ trợ hiệu quả về tính toán và chuyên dùng cho các ứng dụng real-time (thời gian thực).

* *Giấy phép BSD*: dành riêng cho các loại mã nguồn mở nhằm cho phép sử dụng miễn phí và hạn chế tối đa các rào cản luật lệ thông thường.

1.2 OpenCV

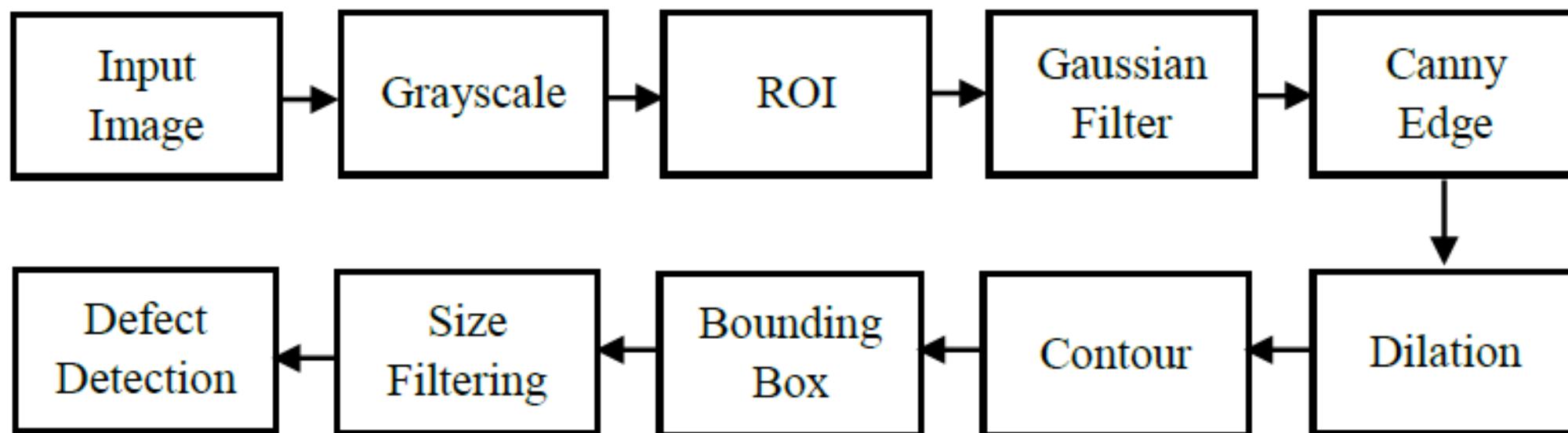
❖ **Ứng dụng của OpenCV**

- + Kiểm tra và giám sát tự động
- + Robot và xe hơi tự lái
- + Phân tích hình ảnh y học
- + Phục hồi hình ảnh/video
- + Thực tế ảo

1.2 OpenCV

❖ Úng dụng của OpenCV

- + Ví dụ: Thuật toán phát hiện khiếm khuyết



1.2 OpenCV

- ❖ Cài đặt OpenCV

`pip install opencv-python`

- ❖ Truy cập OpenCV

`import cv2`

`import cv2 as cv`

1.2 OpenCV

❖ link tham khảo: https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html

1.2 OpenCV

❖ Đọc hình ảnh trong OpenCV bằng Python

Các loại tệp sau được hỗ trợ trong thư viện OpenCV:

- + Ảnh bitmap của Windows - * .bmp, * .dib
- + Tệp JPEG - * .jpeg, * .jpg
- + Portable Network Graphics - * .png
- + WebP - * .webp
- + Sun rasters - * .sr, * .ras
- + Tệp TIFF - * .tiff, * .tif

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Các bước để đọc và hiển thị một hình ảnh trong OpenCV là:

1. Đọc ảnh bằng hàm imread().
2. Tạo cửa sổ GUI và hiển thị hình ảnh bằng hàm imshow().
3. Sử dụng waitkey(0) để giữ cửa sổ hình ảnh trên màn hình theo số giây được chỉ định, giá trị 0 có nghĩa là cho đến khi người dùng đóng nó, nó sẽ giữ cửa sổ GUI trên màn hình.
4. Xóa cửa sổ hình ảnh khỏi bộ nhớ sau khi hiển thị bằng cách sử dụng hàm destroyAllWindows().

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Hàm cv2.imread(): Sử dụng để đọc ảnh

Cú pháp:

cv2.imread(path, flag)

Trong đó:

path: Tên ảnh trong thư mục source, hoặc đường dẫn ở ngoài thư mục source

flag: Nó chỉ định cách mà hình ảnh sẽ được đọc. Giá trị mặc định của nó là cv2.IMREAD_COLOR

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Hàm cv2.imread(): Sử dụng để đọc ảnh

Tất cả ba loại cờ được mô tả dưới đây:

cv2.IMREAD_COLOR: Chuyển đổi hình ảnh sang hình ảnh màu BGR 3 kênh. Mọi sự trong suốt của hình ảnh sẽ bị bỏ qua. Nó là cờ mặc định. Ngoài ra, có thể chuyển giá trị số nguyên 1 cho cờ này.

cv2.IMREAD_GRAYSCALE: Chỉ định tải hình ảnh ở chế độ thang độ xám. Ngoài ra, có thể chuyển giá trị số nguyên 0 cho cờ này.

cv2.IMREAD_UNCHANGED: Trả lại hình ảnh đã tải như hiện tại. Ngoài ra, có thể chuyển giá trị số nguyên -1 cho cờ này.

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Hàm cv2.imshow(): hiển thị một hình ảnh

Cú pháp:

cv2.imshow(window_name, image)

Trong đó:

window_name: tên cửa sổ sẽ được hiển thị trên cửa sổ.

image: hình ảnh muốn hiển thị.

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Hàm `cv2.waitKey()`: chức năng liên kết bàn phím

Cú pháp:

`cv2.waitKey(time)`

Trong đó:

time: là thời gian mà cửa sổ sẽ hiển thị (mini giây)

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Hàm cv2.destroyAllWindows(): xóa cửa sổ hoặc hình ảnh khỏi bộ nhớ chính của hệ thống

Cú pháp:

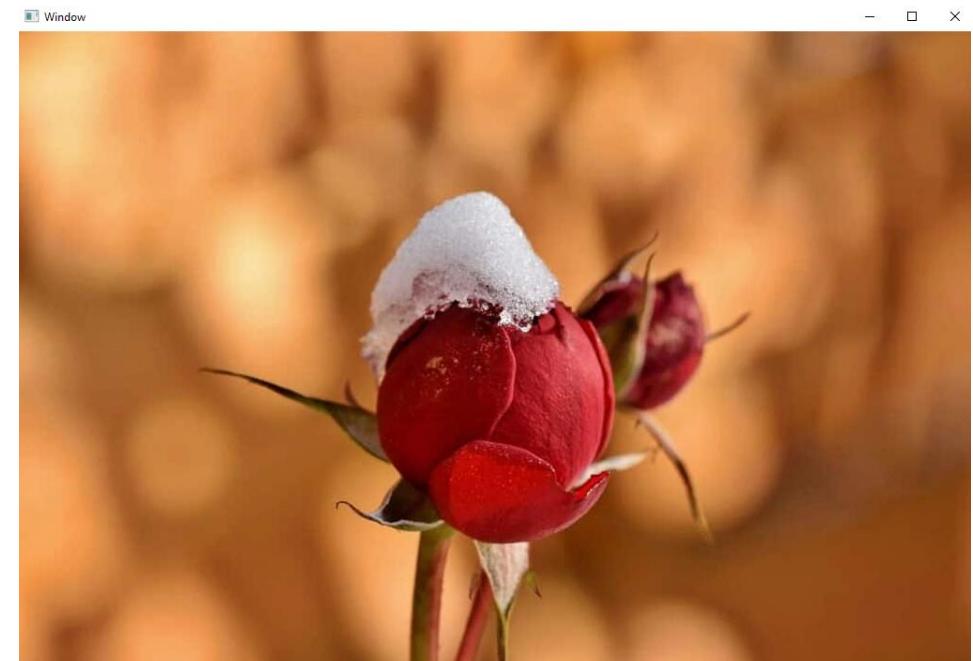
`cv2.destroyAllWindows()`

1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Ví dụ:

```
1 import cv2  
2  
3 # Doc anh  
4 img = cv2.imread('input_image.jpg', cv2.IMREAD_COLOR)  
5 # Tạo cửa sổ để hiển thị ảnh  
6 cv2.imshow('Window', img)  
7 # Giữ cửa sổ hình ảnh  
8 cv2.waitKey(0)  
9 # tắt cửa sổ ảnh  
10 cv2.destroyAllWindows()  
11 |  
12 |
```

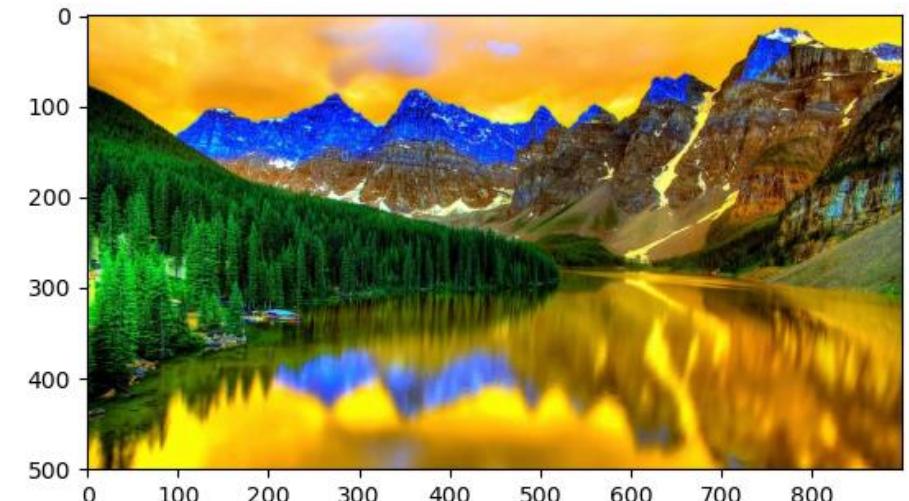


1.2 OpenCV

- ❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Ví dụ sử dụng thư viện matplotlib:

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 # đọc ảnh
5 img=cv2.imread("img_2.png")
6 # Hiển thị ảnh
7 plt.figure()
8 plt.imshow(img)
9 plt.show()
10 |
```

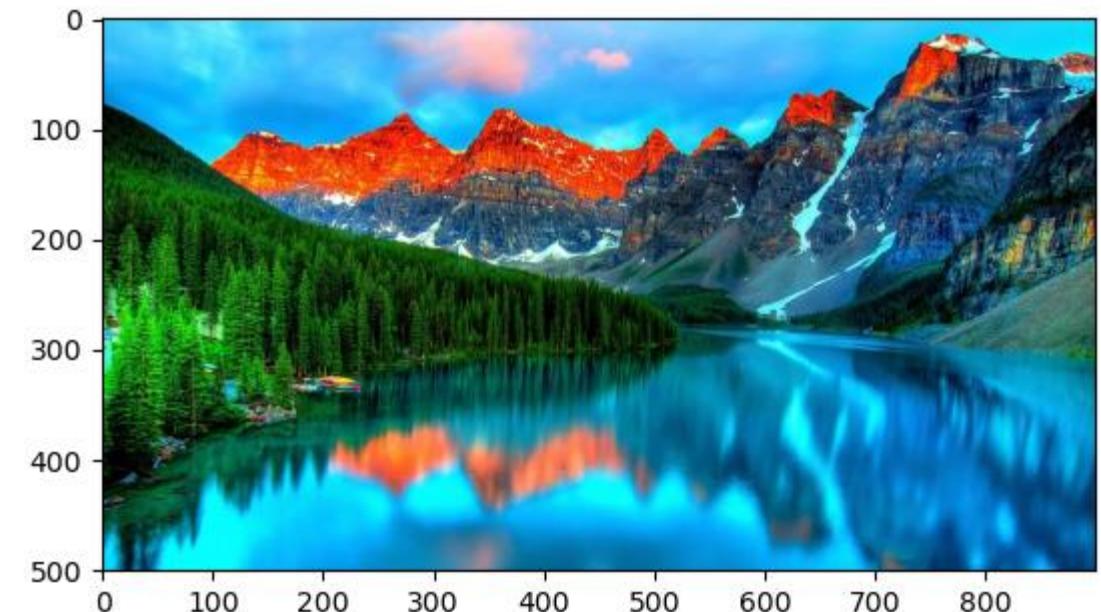


1.2 OpenCV

❖ Đọc, hiển thị hình ảnh trong OpenCV bằng Python

Ví dụ sử dụng thư viện matplotlib:

```
1 import cv2
2 import matplotlib.pyplot as plt
3
4 # đọc ảnh
5 img=cv2.imread("img_2.png")
6 # chuyển BGR sang RGB
7 RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
8 # Hiển thị ảnh
9 plt.figure()
10 plt.imshow(RGB_img)
11 plt.show()
```



1.2 OpenCV

❖ Lưu hình ảnh trong OpenCV bằng Python

Hàm cv2.imwrite(): ghi/lưu một hình ảnh vào thư mục tệp

Cú pháp:

`cv2.imwrite(filename, image)`

Trong đó:

filename: là tên tệp, phải bao gồm phần mở rộng tên tệp (ví dụ: .png, .jpg, v.v.)

Image: hình ảnh muốn lưu

1.2 OpenCV

❖ Lưu hình ảnh trong OpenCV bằng Python

Ví dụ:

```
\ 1 import cv2
 2
 3 # đọc ảnh
 4 img=cv2.imread("img_2.png")
 5 # chuyển BGR sang RGB
 6 cv2.imwrite('img_1.jpg',img)
 7 |
```

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh sử dụng OpenCV

Các phép toán số học như phép cộng, phép trừ và phép toán bit (AND, OR, NOT, XOR) có thể được áp dụng cho các hình ảnh đầu vào. Các thao tác này có thể hữu ích trong việc nâng cao các thuộc tính của hình ảnh đầu vào, được áp dụng để làm rõ, tạo ngưỡng, giãn nở, v.v. của hình ảnh.

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh sử dụng OpenCV

- **Phép cộng:** sử dụng hàm cv2.addweighted() cộng các pixel hình ảnh trong hai hình ảnh.

Cú pháp:

cv2.addWeighted (img1, wt1, img2, wt2, gammaValue)

Trong đó:

img1: hình ảnh đầu vào đầu tiên, img2: Hình ảnh Đầu vào Thứ hai

wt1: Trọng số của phần tử hình ảnh đầu vào đầu tiên được áp dụng hình ảnh cuối cùng

wt2: Trọng số của các phần tử hình ảnh đầu vào thứ hai được áp dụng cho hình ảnh cuối cùng

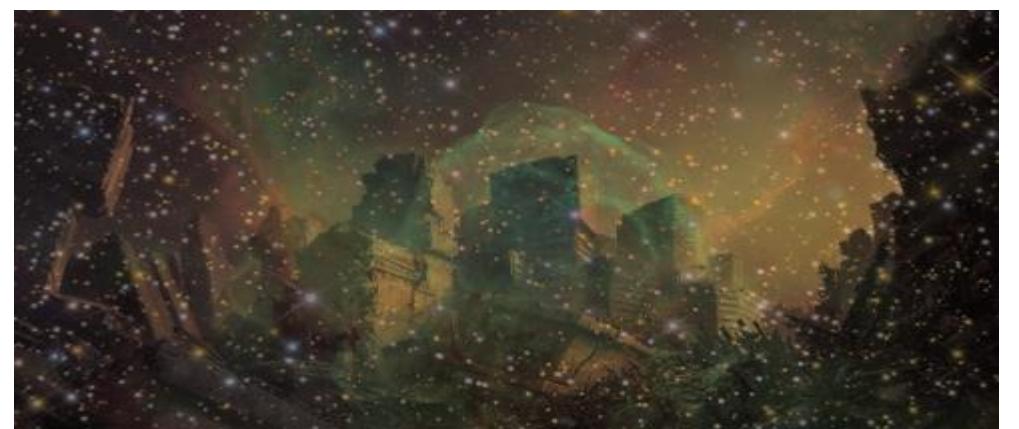
gammaValue : Độ ánh sáng

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ:

```
1 import cv2  
2  
3 image1 = cv2.imread('img_1.png')  
4 image2 = cv2.imread('img_3.png')  
5 addimage = cv2.addWeighted(image1, 0.5, image2, 0.3, 0)  
6 cv2.imshow('Image', addimage)  
7  
8 if cv2.waitKey(0) & 0xff == 27:  
9     cv2.destroyAllWindows()  
10 |
```



1.2 OpenCV

- ❖ **Các phép toán số học trên hình ảnh sử dụng OpenCV**
 - Phép trừ: sử dụng hàm cv2.subtract(). Các hình ảnh phải có kích thước và độ sâu bằng nhau.

Cú pháp:

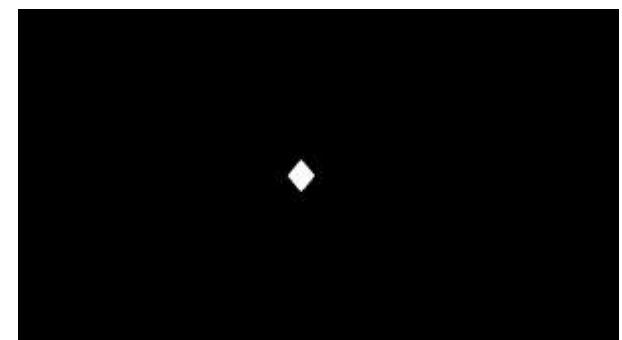
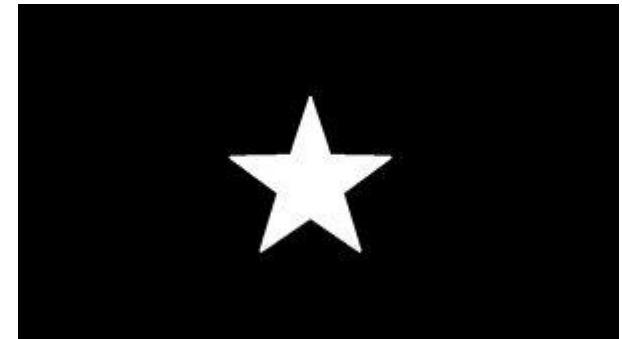
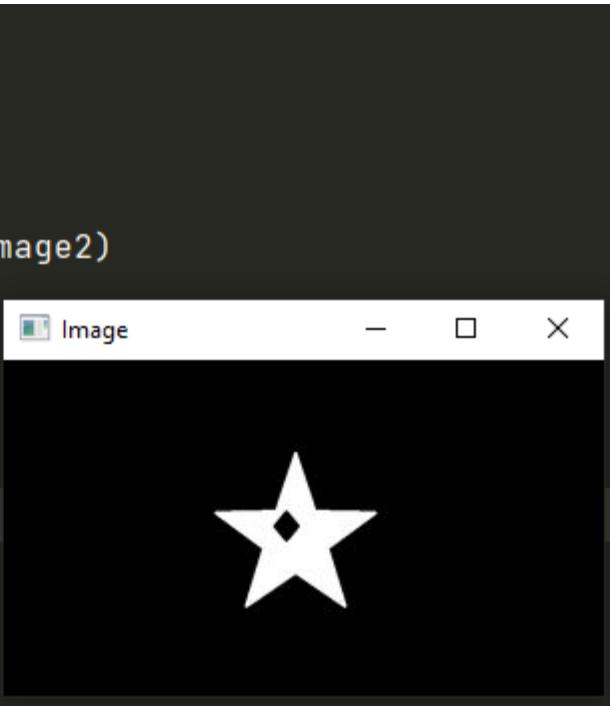
`cv2.subtract(src1, src2)`

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ:

```
1 import cv2
2
3 image1 = cv2.imread('img_4.png')
4 image2 = cv2.imread('img_5.png')
5 addimage = cv2.subtract(image1, image2)
6 cv2.imshow('Image', addimage)
7
8 if cv2.waitKey(0) & 0xff == 27:
9     cv2.destroyAllWindows()
10
```



1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Phép toán Bitwise được sử dụng trong thao tác hình ảnh và được sử dụng để trích xuất các phần thiết yếu trong hình ảnh. Trong bài viết này, các phép toán Bitwise được sử dụng là:

- + **AND**
- + **OR**
- + **XOR**
- + **NOT**

LƯU Ý: Các hoạt động Bitwise nên được áp dụng trên hình ảnh đầu vào có cùng kích thước

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

▪ BitwiseAND

Cú pháp:

`cv2.bitwise_and(source1, source2, mask)`

Trong đó:

source1: hình ảnh đầu vào đầu tiên

source2: mảng hình ảnh đầu vào thứ hai

mask: None

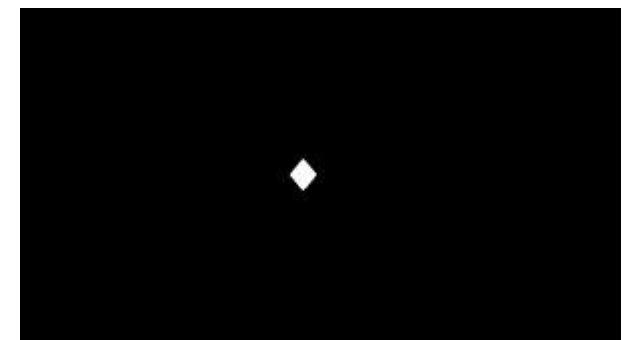
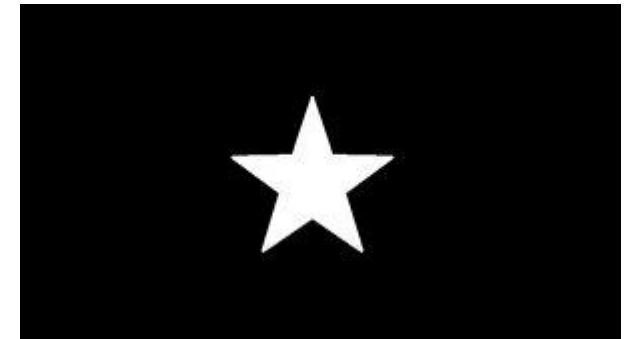
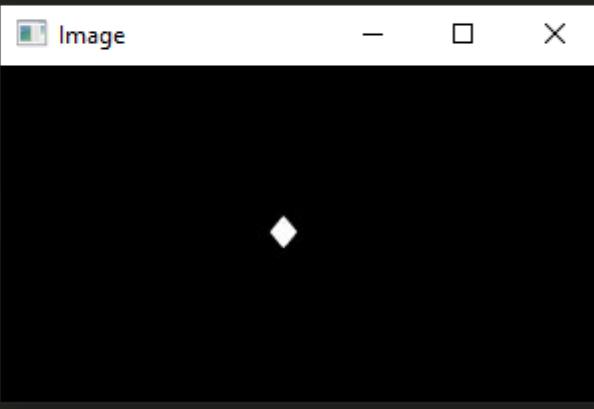
Number 1	1	0	1	0	1
Number 2	1	1	1	0	0
<hr/>					
AND	1	0	1	0	0
OR	1	1	1	0	1
XOR	0	1	0	0	1

1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ: **bitwise_and**

```
1 import cv2  
2  
3 image1 = cv2.imread('img_4.png')  
4 image2 = cv2.imread('img_5.png')  
5 addimage = cv2.bitwise_and(image1, image2, mask=None)  
6 cv2.imshow('Image', addimage)  
7  
8 if cv2.waitKey(0) & 0xff == 27:  
9     cv2.destroyAllWindows()  
10
```

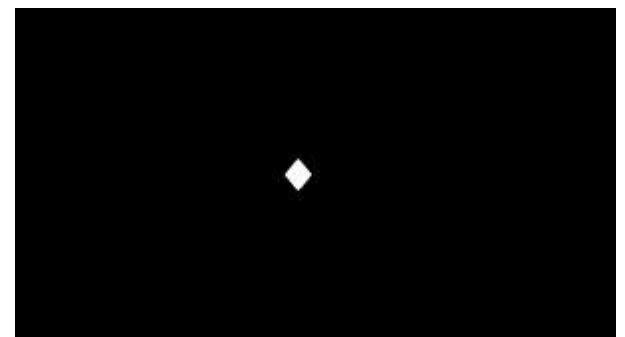
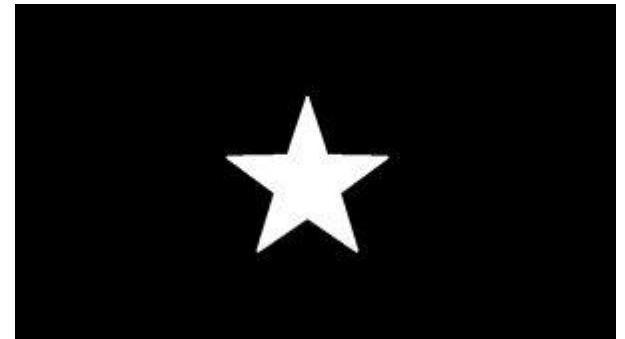
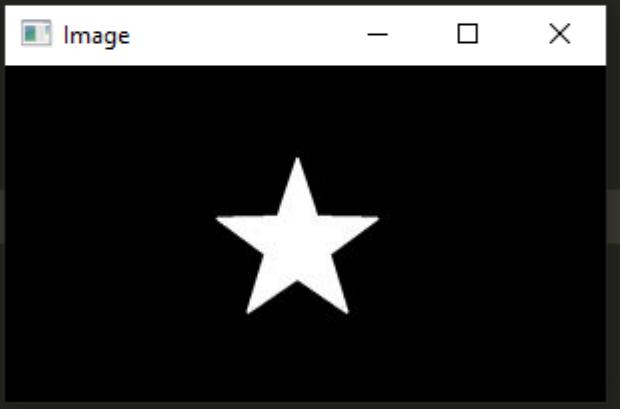


1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ: **bitwise_or**

```
1 import cv2
2
3 image1 = cv2.imread('img_4.png')
4 image2 = cv2.imread('img_5.png')
5 addimage = cv2.bitwise_or(image1, image2, mask=None)
6 cv2.imshow('Image', addimage)
7
8 if cv2.waitKey(0) & 0xff == 27:
9     cv2.destroyAllWindows()
10
```

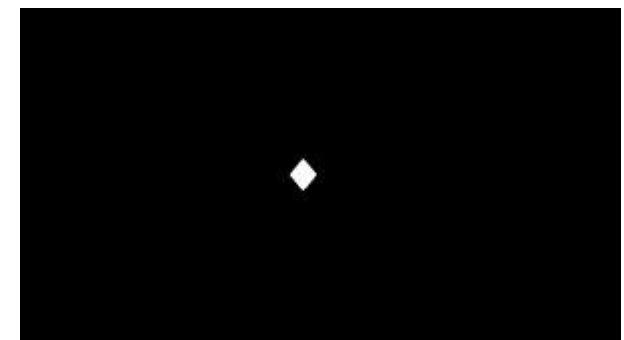
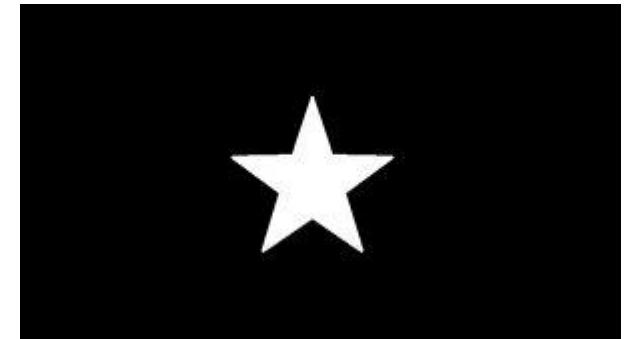
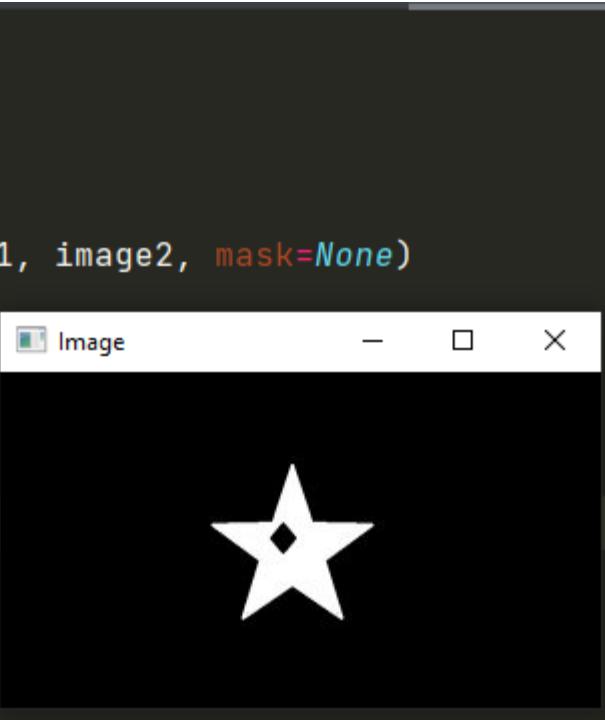


1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ: `bitwise_xor`

```
1 import cv2
2
3 image1 = cv2.imread('img_4.png')
4 image2 = cv2.imread('img_5.png')
5 addimage = cv2.bitwise_xor(image1, image2, mask=None)
6 cv2.imshow('Image', addimage)
7
8 if cv2.waitKey(0) & 0xff == 27:
9     cv2.destroyAllWindows()
10
```

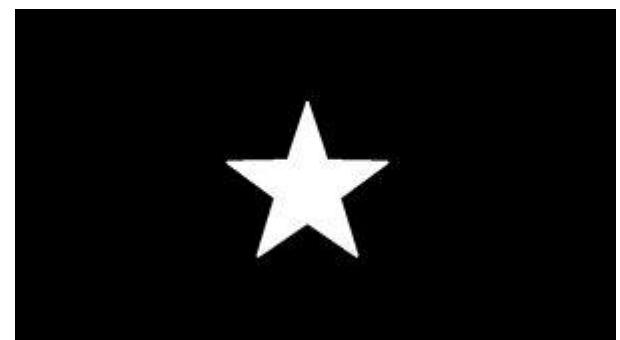
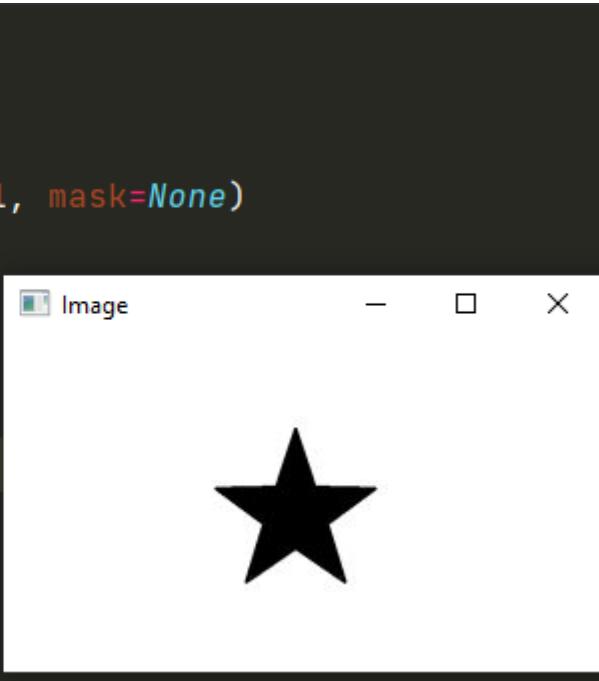


1.2 OpenCV

❖ Các phép toán số học trên hình ảnh

Ví dụ: `bitwise_not`

```
1 import cv2
2
3 image1 = cv2.imread('img_4.png')
4 addimage = cv2.bitwise_not(image1, mask=None)
5 cv2.imshow('Image', addimage)
6
7 if cv2.waitKey(0) & 0xff == 27:
8     cv2.destroyAllWindows()
9
```



1.2 OpenCV

❖ **Tiền xử lý ảnh OpenCV**

Vai trò của tiền xử lý ảnh: Khi phát triển một thuật toán phân loại ảnh có thể gặp phải một số trường hợp không mong đợi như: Kết quả huấn luyện có độ chính xác rất cao trên cả tập huấn luyện (train dataset) và tập phát triển (dev dataset). Nhưng khi áp dụng vào thực tiễn lại cho độ chính xác thấp. Nguyên nhân đó là:

+ Các bức ảnh được huấn luyện khác xa so với những bức ảnh được người dùng upload về các khía cạnh: độ phân giải, cường độ màu sắc, chất lượng ảnh, độ to nhỏ của vật thể, chiều, hướng và tư thế của vật thể bên trong ảnh.

1.2 OpenCV

❖ **Tiền xử lý ảnh OpenCV**

Vai trò của tiền xử lý ảnh: Nguyên nhân đó là:

+ Có thể các bức ảnh được người dùng upload lên mặc dù cùng nhãn nhưng khác về tính chất so với các bức ảnh đã huấn luyện.

Ví dụ trong một thuật toán phân loại **dog and cat**, tập huấn luyện chỉ bao gồm những con mèo trưởng thành nhưng thực tế người dùng lại upload lên rất nhiều hình ảnh của mèo con có thể dẫn tới thuật toán bị nhầm lẫn.

1.2 OpenCV

❖ **Tiền xử lý ảnh OpenCV**

Vai trò của tiền xử lý ảnh: Nguyên nhân đó là:

+ Đối với một số tác vụ phân loại ảnh khó, đòi hỏi chuyên gia gán nhãn, rất dễ mắc sai lầm như chuẩn đoán bệnh nhãn cầu. Một số ít các ảnh trong tập huấn luyện có thể bị gán sai nhãn. Do đó ảnh hưởng đến khả năng dự báo của thuật toán.

1.2 OpenCV

❖ **Tiền xử lý ảnh OpenCV**

Vai trò của tiền xử lý ảnh: Nguyên nhân đó là:

- + Bộ dữ liệu huấn luyện có kích thước quá nhỏ và không đại diện cho toàn bộ các class được huấn luyện.
- + Phân phối của tập huấn luyện khác xa so với thực tế. Chẳng hạn tập huấn luyện chứa ảnh chó và mèo theo tỷ lệ 50:50 nhưng số lượng bức ảnh người dùng upload lên ảnh chó chiếm đa số theo tỷ lệ 90:10.
- + Và rất nhiều các nguyên nhân khác dẫn tới thuật toán hoạt động không được như kì vọng.

1.2 OpenCV

❖ **Tiền xử lý ảnh OpenCV**

Vai trò của tiền xử lý ảnh: Nguyên nhân đó là:

- + Trong trường hợp dữ liệu không đủ lớn, dữ liệu gán nhãn với chi phí cao (như chuẩn đoán bệnh qua hình ảnh, phải tìm được bệnh nhân gấp đúng bệnh đó và bác sĩ chuyên khoa để chuẩn đoán), việc thay đổi tập dữ liệu là khá tốn chi phí.
- + Có một phương pháp sẽ giúp gia tăng số lượng ảnh đầu vào. Đó chính là học tăng cường (data augmentation) sử dụng các biến đổi **tiền xử lý hình ảnh** đầu vào. Đây là một phương pháp hiệu quả nhằm thay đổi tập dữ liệu huấn luyện và từ đó giúp cải thiện hiệu quả dự báo.

1.2 OpenCV

❖ Tiền xử lý ảnh OpenCV

▪ Các biến đổi hình học.

- + Đây là tập hợp các phép biến đổi hình ảnh từ một hình dạng này sang một hình dạng khác thông qua việc làm thay đổi phương, chiều, góc, cạnh mà không làm thay đổi nội dung chính của bức ảnh.

1.2 OpenCV

❖ Tiề́n xử lý ảnh OpenCV

▪ Các biến đổi hình học.

+ Mỗi một phép biến đổi hình học sẽ được xác định bởi một ma trận dịch chuyển (translation matrix) M. Khi đó bất kì 1 điểm có tọa độ (x,y) trên ảnh gốc thông qua phép biến đổi T sẽ có tọa độ trong không gian mới sau dịch chuyển là T(x,y) theo công thức:

$$T(x, y) = M \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y \\ a_{21}x + a_{22}y \end{bmatrix}$$

1.2 OpenCV

❖ Tiề̉n xử lý ảnh OpenCV

▪ Các biến đổi hình học.

+ Để xác định một phép biến đổi hình học ta sẽ cần phải xác định được ma trận dịch chuyển của nó là gì? Các dạng biến đổi sẽ sẽ được đặc trưng bởi các dạng ma trận dịch chuyển khác nhau.

1.2 OpenCV

❖ **Thay đổi kích thước hình ảnh với OpenCV (Scale ảnh)**

Thay đổi kích thước hình ảnh tức là thu nhỏ, phóng to hoặc mở rộng quy mô để đáp ứng các yêu cầu về kích thước dài, rộng của ảnh mà không làm thay đổi tính chất song song của các đoạn thẳng trên ảnh gốc so với các trục tọa độ X và Y.

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

Theo định nghĩa về phép biến đổi hình học thì một biến đổi phóng đại các chiều (x, y) theo hệ số ($a_{\{1\}}$, $a_{\{2\}}$) sẽ có ma trận dịch chuyển M là ma trận đường chéo. Tức là ma trận vuông có đường chéo chính là [a1,a2] và các phần tử còn lại bằng 0. Khi đó phép dịch chuyển sẽ là:

$$T(x, y) = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 & 0 \\ 0 & a_2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_1 x \\ a_2 y \end{bmatrix}$$

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

▪ Hàm hay đổi kích thước resize():

Cú pháp:

resize(src, dsize[, dst[, fx[, fy[, interpolation]]]])

- **src**: Đây là hình ảnh đầu vào bắt buộc, nó có thể là một chuỗi với đường dẫn hình ảnh đầu vào
- **dsize**: Đó là kích thước mong muốn của hình ảnh đầu ra,
- **fx**: Hệ số tỷ lệ dọc theo trực hoành.
- : Hệ số tỷ lệ theo trực tung.
- **interpolation**: Tùy chọn các phương pháp thay đổi kích thước hình ảnh khác nhau.

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

▪ Ví dụ

```
1 import cv2
2
3 # đọc ảnh
4 image = cv2.imread('img_2.png')
5 cv2.imshow('Image', image)
6 # thu nhỏ ảnh
7 down_width = 300
8 down_height = 200
9 down_points = (down_width, down_height)
10 resized_down = cv2.resize(image, down_points, interpolation= cv2.INTER_LINEAR)
11 # phóng to ảnh
12 up_width = 600
13 up_height = 400
14 up_points = (up_width, up_height)
15 resized_up = cv2.resize(image, up_points, interpolation= cv2.INTER_LINEAR)
16 # hiển thị
17 cv2.imshow('Ảnh thu nhỏ', resized_down)
18 cv2.waitKey()
19 cv2.imshow('Ảnh phóng to', resized_up)
20 cv2.waitKey()
21 # đóng cửa sổ
22 cv2.destroyAllWindows()
```

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

▪ Thay đổi kích thước với một hệ số tỷ lệ

Hệ số tỷ lệ là một số chia tỷ lệ hoặc nhân một số đại lượng, trong trường hợp là chiều rộng và chiều cao của hình ảnh. Hệ số tỷ lệ giúp giữ nguyên tỷ lệ khung hình và bảo toàn chất lượng hiển thị. Vì vậy, hình ảnh không bị biến dạng, trong khi tăng hoặc giảm tỷ lệ của ảnh.

1.2 OpenCV

- ❖ Thay đổi kích thước hình ảnh với OpenCV
- Thay đổi kích thước với một hệ số tỷ lệ

```
1 import cv2
2
3 # đọc ảnh
4 image = cv2.imread('img_2.png')
5 cv2.imshow('Image', image)
6 # tăng tỉ lệ
7 scale_up_x = 1.2
8 scale_up_y = 1.2
9 # giảm tỉ lệ
10 scale_down = 0.6
11
12 scaled_f_down = cv2.resize(image, None, fx= scale_down, fy= scale_down, interpolation= cv2.INTER_LINEAR)
13 scaled_f_up = cv2.resize(image, None, fx= scale_up_x, fy= scale_up_y, interpolation= cv2.INTER_LINEAR)
14 cv2.imshow('Ảnh thu nhỏ', scaled_f_down)
15 cv2.waitKey()
16 cv2.imshow('Ảnh phóng to', scaled_f_up)
17 cv2.waitKey()
18 # đóng cửa sổ
19 cv2.destroyAllWindows()
20 |
```

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

▪ Thay đổi kích thước bằng các phương pháp nội suy khác nhau

Các phương pháp nội suy khác nhau được sử dụng để thay đổi kích thước khác nhau.

- **INTER_AREA:** INTER_AREA sử dụng quan hệ vùng pixel để lấy mẫu lại. Điều này phù hợp nhất để **giảm kích thước của hình ảnh (thu nhỏ)**.
- **INTER_CUBIC:** Điều này sử dụng phép nội suy hai chiều để thay đổi kích thước hình ảnh. Trong khi thay đổi kích thước và nội suy các pixel mới, phương pháp này hoạt động trên các pixel lân cận 4×4 của hình ảnh. Sau đó, lấy trung bình trọng số của 16 pixel để tạo pixel nội suy mới.

1.2 OpenCV

❖ Thay đổi kích thước hình ảnh với OpenCV

▪ Thay đổi kích thước bằng các phương pháp nội suy khác nhau

Các phương pháp nội suy khác nhau được sử dụng để thay đổi kích thước khác nhau.

•**INTER_LINEAR:** Phương pháp này tương tự như phương pháp INTER_CUBIC. Nhưng không giống như INTER_CUBIC, là sử dụng 2×2 pixel lân cận để lấy giá trị trung bình có trọng số cho pixel được nội suy.

•**INTER_NEAREST:** Phương pháp sử dụng khái niệm láng giềng gần nhất để nội suy. Đây là một trong những phương pháp đơn giản nhất, chỉ sử dụng một pixel lân cận từ hình ảnh để nội suy (Phóng to ảnh).

1.2 OpenCV

- ❖ Thay đổi kích thước hình ảnh với OpenCV
- Thay đổi kích thước bằng các phương pháp nội suy khác nhau

```
1 import cv2
2
3 # đọc ảnh
4 image = cv2.imread('img_2.png')
5 # cv2.imshow('Image', image)
6 # giảm tỉ lệ
7 scale_down = 0.6
8
9 # Scaling Down the image 0.6 times using different Interpolation Method
10 res_inter_nearest = cv2.resize(image, None, fx= scale_down, fy= scale_down, interpolation= cv2.INTER_NEAREST)
11 res_inter_linear = cv2.resize(image, None, fx= scale_down, fy= scale_down, interpolation= cv2.INTER_LINEAR)
12 res_inter_area = cv2.resize(image, None, fx= scale_down, fy= scale_down, interpolation= cv2.INTER_AREA)
13 cv2.imshow('nearest', res_inter_nearest)
14 cv2.imshow('linear', res_inter_linear)
15 cv2.imshow('area', res_inter_area)
16 cv2.waitKey()
17 #đóng cửa sổ
18 cv2.destroyAllWindows()
```

1.2 OpenCV

- ❖ **Thay đổi kích thước hình ảnh với OpenCV**
- **Tính năng thay đổi kích thước đang hoạt động trong Ứng dụng web**

<https://opencv-image-resize.herokuapp.com/>

1.2 OpenCV

❖ Hiển thị kích thước của ảnh

+ Kích thước không chỉ bao gồm chiều rộng và chiều cao của ma trận 2-D mà còn bao gồm cả số kênh (ví dụ: hình ảnh RGB có 3 kênh - Đỏ, Xanh lục và Xanh lam).

```
1 import cv2
2
3 # đọc ảnh
4 img = cv2.imread('img_2.png')
5 print(img.shape)
6
7 # Display the image
8 cv2.imshow("Image", img)
9 cv2.waitKey(0)
10 cv2.destroyAllWindows()
11 |
```

```
C:\Users\NGOC_DANG\anaconda3\envs\Code_
(501, 900, 3)
```

1.2 OpenCV

❖ Cắt hình ảnh bằng OpenCV

▪ Cắt ảnh bằng OpenCV hoạt động như thế nào?

Cắt được thực hiện để loại bỏ tất cả các đối tượng hoặc khu vực không mong muốn khỏi hình ảnh. Hoặc thậm chí để làm nổi bật một tính năng cụ thể của hình ảnh.



1.2 OpenCV

❖ Cắt hình ảnh bằng OpenCV

▪ Cắt ảnh bằng OpenCV hoạt động như thế nào?

- + Công việc cắt mảng NumPy. Mọi hình ảnh được đọc vào sẽ được lưu trữ trong một mảng 2D (cho mỗi kênh màu). Chỉ cần chỉ định chiều cao và chiều rộng (tính bằng pixel) của khu vực sẽ được cắt.

1.2 OpenCV

❖ Cắt hình ảnh bằng OpenCV

+ Ví dụ:

```
1 import cv2
2
3 # đọc ảnh
4 img = cv2.imread('img_2.png')
5 # cắt ảnh
6 cropped_image = img[80:280, 150:330]
7
8 # Hiển thị
9 cv2.imshow("cropped", cropped_image)
10 cv2.waitKey(0)
11 cv2.destroyAllWindows()
12
```



1.2 OpenCV

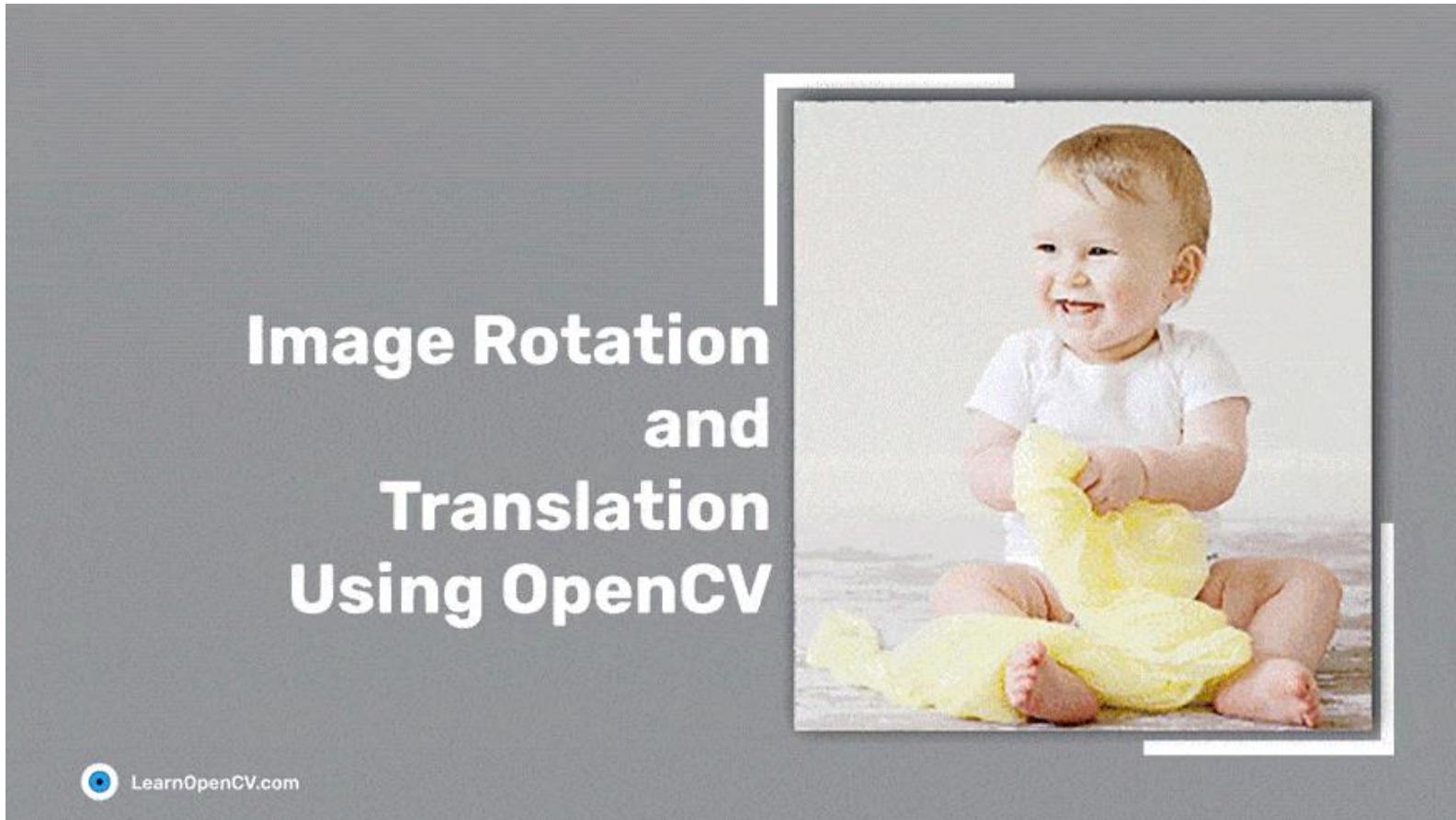
❖ Cắt hình ảnh bằng OpenCV

Ứng dụng Crop bằng Streamlit:

<https://opencv-image-crop.herokuapp.com/>

1.2 OpenCV

- ❖ Xoay và dịch hình ảnh bằng OpenCV
- Xoay hình ảnh bằng OpenCV



1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

+ Xoay ảnh được hiểu là quay một bức ảnh theo một góc xác định quanh một điểm nào đó. Phép xoay sẽ không đảm bảo tính chất song song với các trục X hoặc Y như phép dịch chuyển nhưng nó sẽ bảo toàn độ lớn góc.

Xoay hình ảnh theo một góc nhất định θ bằng cách xác định một ma trận chuyển đổi M. Ma trận này thường có dạng:

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

+ Ngoài ra OpenCV hỗ trợ một phép xoay phóng đại (scaled rotation) với khả năng vừa biến đổi ảnh theo phép xoay theo tâm xác định và điều chỉnh lại kích thước ảnh sau xoay. Như vậy có thể xoay theo bất kỳ vùng nào. Phép dịch chuyển ma trận được đưa ra như sau:

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha)c_x - \beta c_y \\ -\beta & \alpha & \beta c_x + (1 - \alpha)c_y \end{bmatrix}$$

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

+ Ngoài ra OpenCV hỗ trợ một phép xoay phóng đại (scaled rotation) với khả năng vừa biến đổi ảnh theo phép xoay theo tâm xác định và điều chỉnh lại kích thước ảnh sau xoay. Như vậy có thể xoay theo bất kỳ vùng nào. Phép dịch chuyển ma trận được đưa ra như sau:

$$\begin{bmatrix} \alpha & \beta & (1 - \alpha)c_x - \beta c_y \\ -\beta & \alpha & \beta c_x + (1 - \alpha)c_y \end{bmatrix}$$

trong đó:

$$\alpha = scale.\cos(\theta) \quad \beta = scale.\sin(\theta)$$

(cx,cy) là tọa độ tâm của phép xoay và scale là độ phóng đại.

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

+ OpenCV cung cấp `getRotationMatrix2D()` chức năng tạo ma trận biến đổi trên.

Cú pháp:

`getRotationMatrix2D(center, angle, scale)`

Trong đó:

center: tâm quay cho hình ảnh đầu vào

angle: góc quay theo độ (+ hướng ngược chiều kim đồng hồ)

scale: hệ số tỷ lệ đằng hướng giúp chia tỷ lệ hình ảnh lên hoặc xuống theo giá trị được cung cấp

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

Xoay vòng là một hoạt động gồm ba bước:

- + Đầu tiên cần lấy tâm của vòng quay. Đây thường là tâm của hình ảnh mà đang cố xoay.
- + Tạo ma trận quay 2D. OpenCV cung cấp `getRotationMatrix2D()`
- + Áp dụng phép biến đổi affine cho hình ảnh, sử dụng ma trận xoay đã tạo ở bước 2. Hàm `warpAffine()` trong OpenCV thực hiện công việc.

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

Hàm warpAffine() áp dụng một phép biến đổi affine cho hình ảnh. Sau khi áp dụng phép biến đổi affine, tất cả các đường thẳng song song trong hình ảnh ban đầu cũng sẽ vẫn song song trong hình ảnh đầu ra.

Cú pháp:

warpAffine(src, M, dsize)

- src: mage nguồn
- M: ma trận biến đổi
- dsize: kích thước của hình ảnh đầu ra

1.2 OpenCV

❖ Xoay và dịch hình ảnh bằng OpenCV

▪ Xoay hình ảnh bằng OpenCV

Ví dụ:

```
1 import cv2
2
3 # đọc ảnh
4 image = cv2.imread('img_2.png')
5
6 # chia chiều cao và chiều rộng cho 2 để lấy tâm của hình ảnh
7 height, width = image.shape[:2]
8 # lấy tọa độ tâm của hình ảnh để tạo ma trận quay 2D
9 center = (width/2, height/2)
10
11 # sử dụng cv2.getRotationMatrix2D () để lấy ma trận xoay
12 rotate_matrix = cv2.getRotationMatrix2D(center=center, angle=45, scale=1)
13
14 # xoay hình ảnh bằng cv2.warpAffine
15 rotated_image = cv2.warpAffine(src=image, M=rotate_matrix, dsize=(width, height))
16
17 cv2.imshow('Original image', image)
18 cv2.imshow('Rotated image', rotated_image)
19 cv2.waitKey(0)
20 # lưu ảnh
21 cv2.imwrite('abc.jpg', rotated_image)
```

1.2 OpenCV

- ❖ **Xoay và dịch hình ảnh bằng OpenCV**
- ❖ **Dịch hình ảnh bằng OpenCV**

Dịch chuyển ảnh thường được thực hiện trong trường hợp muốn dịch chuyển ảnh đến các vị trí khác nhau. Ví dụ tới các góc trái, phải, ở giữa, bên trên, bên dưới. Phép dịch chuyển sẽ giữ nguyên tính chất song song của các đoạn thẳng sau dịch chuyển đối với các trục x hoặc y nếu trước dịch chuyển chúng cũng song song với một trong hai trục này. Để dịch chuyển hình ảnh phải xác định được (tx, ty) là các giá trị di chuyển ảnh theo trục x và y. Ma trận dịch chuyển M sẽ có dạng như bên dưới:

$$M = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix}$$

1.2 OpenCV

- ❖ Xoay và dịch hình ảnh bằng OpenCV
- ❖ Dịch hình ảnh bằng OpenCV

Giả sử mọi điểm ảnh đều nằm trên không gian 2 chiều. Khi đó ta coi chiều thứ 3 là một hằng số.

Phép biến đổi theo ma trận dịch chuyển là:

$$T(x, y) = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$

1.2 OpenCV

- ❖ **Xoay và dịch hình ảnh bằng OpenCV**
- ❖ **Dịch hình ảnh bằng OpenCV**

Một số điểm cần lưu ý khi thay đổi hình ảnh theo txvà ty các giá trị:

- + Cung cấp các giá trị dương cho tx sẽ chuyển hình ảnh sang phải và các giá trị âm sẽ chuyển hình ảnh sang trái.
- + Tương tự, các giá trị dương của ty sẽ chuyển hình ảnh xuống trong khi các giá trị âm sẽ chuyển hình ảnh lên.

1.2 OpenCV

- ❖ **Xoay và dịch hình ảnh bằng OpenCV**
- ❖ **Dịch hình ảnh bằng OpenCV**

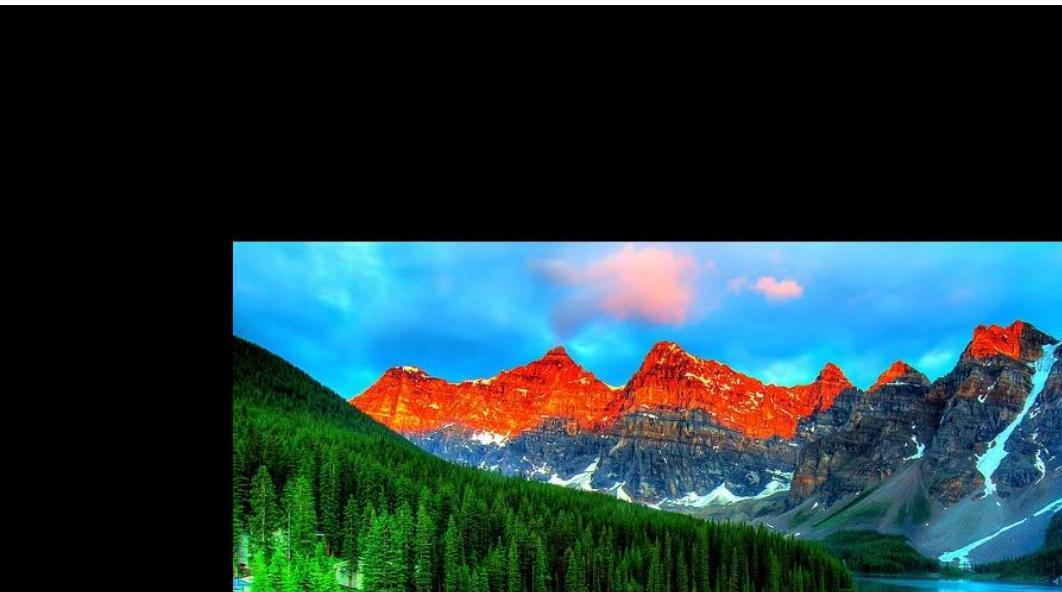
Các bước sau để dịch một hình ảnh, sử dụng OpenCV:

1. Đầu tiên, đọc hình ảnh và lấy chiều rộng và chiều cao của nó.
2. Tiếp theo, tạo một ma trận chuyển đổi, là một mảng 2D. Ma trận này chứa thông tin cần thiết để dịch chuyển hình ảnh, dọc theo trục x và y.
3. Sử dụng hàm warpAffine() để áp dụng phép biến đổi affine.

1.2 OpenCV

- ❖ Xoay và dịch hình ảnh bằng OpenCV
- ❖ Dịch hình ảnh bằng OpenCV

Ví dụ:



```
1 import cv2
2 import numpy as np
3
4 # đọc ảnh
5 image = cv2.imread('img_2.png')
6
7 # lấy chiều cao và rộng của ảnh
8 height = image.shape[0]
9 width = image.shape[1]
10 print(height)
11 print(width)
12 tx, ty = (200, 200)
13 M1 = np.array([[1, 0, tx],
14                 [0, 1, ty]], dtype=np.float32)
15 center = (width/2, height/2)
16
17 # xoay hình ảnh bằng cv2.warpAffine
18 rotated_image = cv2.warpAffine(src=image, M=M1, dsize=(width, height))
19
20 cv2.imshow('Original image', image)
21 cv2.imshow('Rotated image', rotated_image)
22 cv2.waitKey(0)
```

1.2 OpenCV

❖ Biến đổi Affine

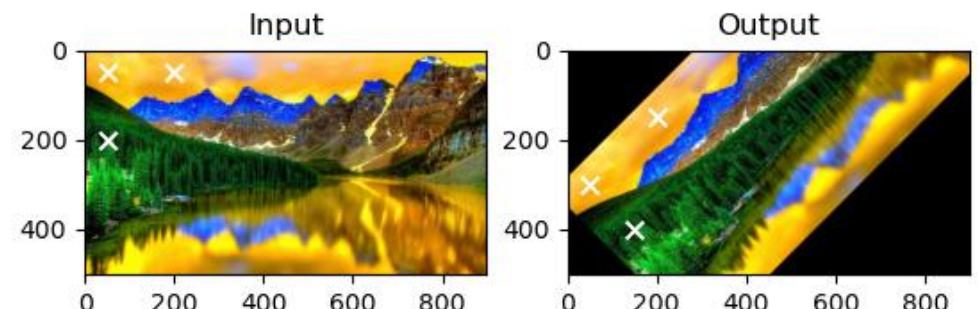
- + Trong biến đổi affine, toàn bộ các đường thẳng song song trong bức ảnh gốc giữ nguyên tính chất song song ở ảnh đầu ra.
- + Để tìm ma trận chuyển vị, cần xác định ra 3 điểm từ ảnh đầu vào và tọa độ tương ứng của chúng trong hình ảnh đầu ra. Hàm `cv2.getAffineTransform` sẽ tạo ra được một ma trận **2x3** được truyền vào hàm `cv2.warpAffine`.

1.2 OpenCV

❖ Biến đổi Affine

+ Ví dụ:

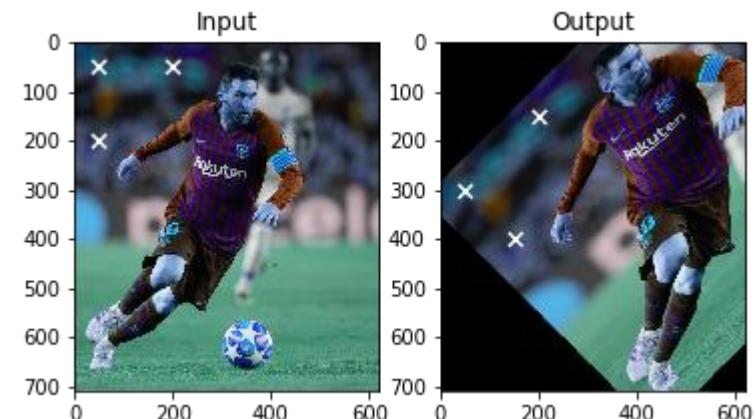
```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # đọc ảnh
6 image = cv2.imread('img_2.png')
7 rows,cols,ch = image.shape
8
9 pts1 = np.float32([[50,50], [200,50], [50,200]])
10 # pts2 = np.float32([[50,100], [200,50], [50,200]])
11 pts2 = np.float32([[50,300], [200,150], [150, 400]])
12
13 M = cv2.getAffineTransform(pts1,pts2)
14 imageAffine = cv2.warpAffine(image,M,(cols,rows))
15
16 # Hiển thị hình ảnh gốc và 3 điểm ban đầu trên ảnh
17 plt.subplot(1,2,1),plt.imshow(image),plt.title('Input')
18 for (x, y) in pts1:
19     plt.scatter(x, y, s=50, c='white', marker='x')
20
21 # Hiển thị hình ảnh sau dịch chuyển và 3 điểm mục tiêu của phép dịch chuyển.
22 plt.subplot(1,2,2),plt.imshow(imageAffine),plt.title('Output')
23 for (x, y) in pts2:
24     plt.scatter(x, y, s=50, c='white', marker='x')
25 plt.show()
```



1.2 OpenCV

❖ Biến đổi Affine

- + Sử dụng phép biến đổi Affine có thể giúp tạo thành nhiều biến thể, tư thế khác nhau cho cùng một vật thể. Thường được áp dụng để làm giàu dữ liệu trong trường hợp số lượng ảnh không nhiều.
- + Ví dụ: ảnh mặt người đang chụp nghiêng ta có thể xoay theo các chiều sao cho từ mặt nghiêng trở thành mặt chính diện hoặc như hình ảnh messi đang chạy dáng nghiêng đã được chuyển sang chạy dáng thẳng đứng và nghiêng với các độ lớn góc khác nhau.



1.2 OpenCV

❖ Biến đổi phối cảnh (Perspective Transform)

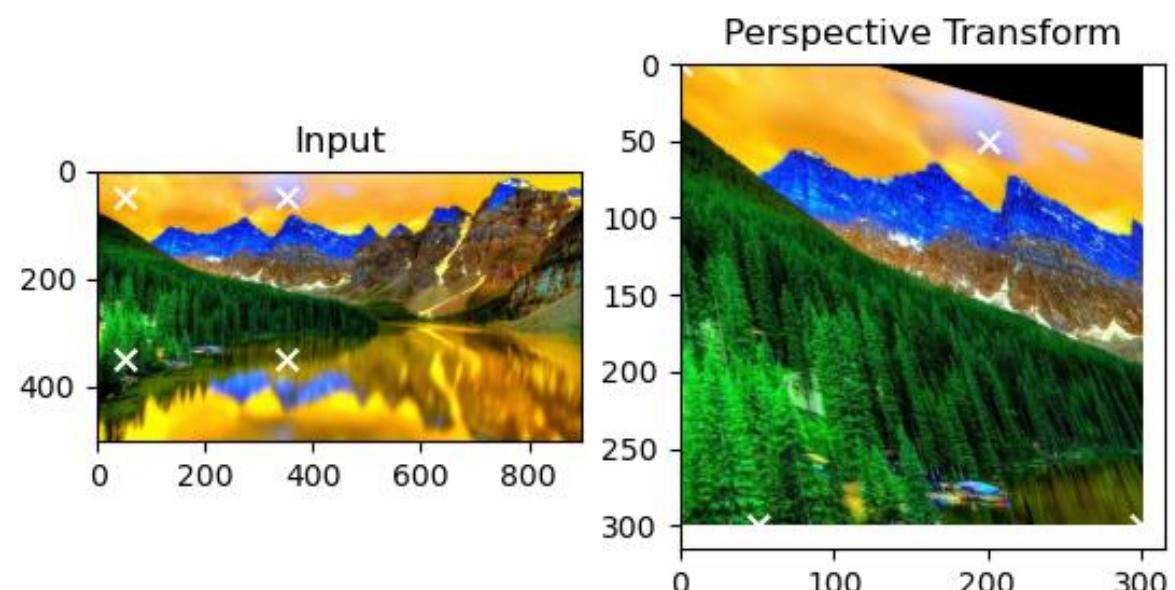
- + Sử dụng biến đổi phối cảnh thì cần một ma trận biến đổi 3×3 . Đường thẳng sẽ giữ nguyên là đường thẳng sau biến đổi.
- + Để tìm ra ma trận biến đổi này, cần tìm ra 4 điểm trong ảnh đầu vào tương ứng với các điểm trong ảnh đầu ra. Trong số 4 điểm này, không có bất kỳ 3 điểm nào thẳng hàng. Sau đó ma trận biến đổi có thể được thiết lập thông qua hàm số `cv2.getPerspectiveTransform`. Và áp dụng `cv2.warpPerspective` với ma trận biến đổi 3×3 .

1.2 OpenCV

❖ Biến đổi phối cảnh (Perspective Transform)

+ Ví dụ:

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # đọc ảnh
6 image = cv2.imread('img_2.png')
7 rows,cols,ch = image.shape
8 pts1 = np.float32([[50,50],[350,50],[50,350],[350,350]])
9 pts2 = np.float32([[0,0],[200,50],[50,300],[300,300]])
10
11 M = cv2.getPerspectiveTransform(pts1,pts2)
12
13 dst = cv2.warpPerspective(image,M,(300,300))
14
15 plt.subplot(1,2,1),plt.imshow(image),plt.title('Input')
16 for (x, y) in pts1:
17     plt.scatter(x, y, s=50, c='white', marker='x')
18 plt.subplot(1,2,2),plt.imshow(dst),plt.title('Perspective Transform')
19 for (x, y) in pts2:
20     plt.scatter(x, y, s=50, c='white', marker='x')
21 plt.show()
22 |
```



1.2 OpenCV

❖ Biến đổi phối cảnh (Perspective Transform)

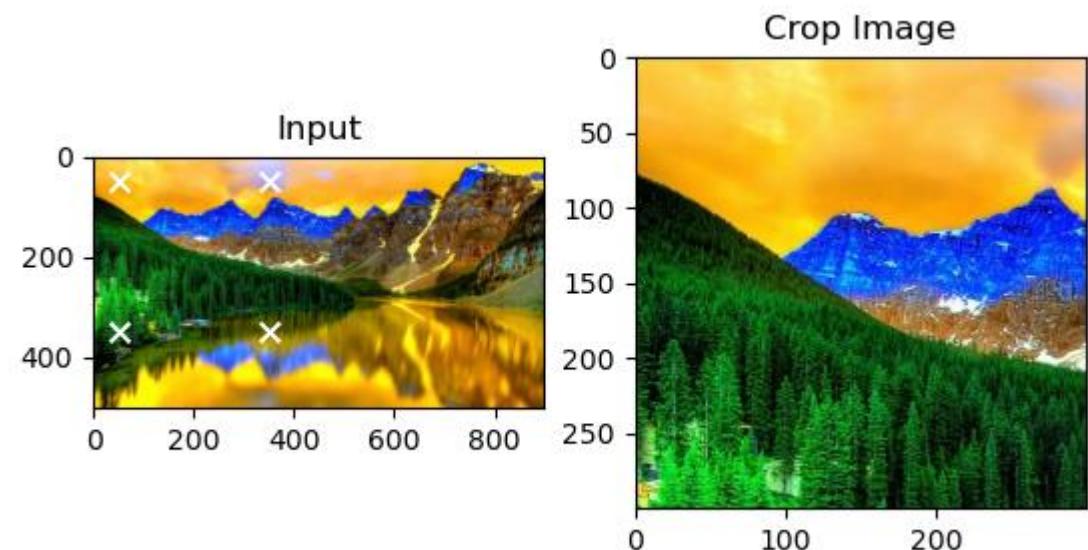
- + Phép biến đổi này cũng gần giống như phép biến đổi Affine. Khác biệt đó là nó chỉ trả ra bức ảnh là biến đổi trên vùng ảnh bị giới hạn trong tọa độ của 4 điểm gốc thay vì biến đổi trên toàn bộ bức ảnh ban đầu như phép biến đổi Affine.
- + Trong trường hợp muốn crop ảnh ta sẽ xác định trước tọa độ của 4 góc và sử dụng phép biến đổi phối cảnh giữa 4 điểm với chính các điểm đó.

1.2 OpenCV

❖ Biến đổi phối cảnh (Perspective Transform)

+ Ví dụ:

```
1 import cv2
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # đọc ảnh
6 image = cv2.imread('img_2.png')
7 rows,cols,ch = image.shape
8
9 pts1 = np.float32([[50,50],[350,50],[50,350],[350,350]])
10
11 M = cv2.getPerspectiveTransform(pts1,pts1)
12
13 dst = cv2.warpPerspective(image,M,(300,300))
14
15 plt.subplot(121),plt.imshow(image),plt.title('Input')
16 for (x, y) in pts1:
17     plt.scatter(x, y, s=50, c='white', marker='x')
18 plt.subplot(122),plt.imshow(dst),plt.title('Crop Image')
19 plt.show()
```



1.2 OpenCV

- ❖ Chú thích hình ảnh bằng OpenCV



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

Chú thích hình ảnh và video phục vụ nhiều mục đích và OpenCV làm cho quá trình trở nên đơn giản và dễ dàng. Có thể sử dụng:

- + Thêm thông tin vào ảnh
- + Vẽ các hộp giới hạn xung quanh các đối tượng trong trường hợp phát hiện đối tượng
- + Đánh dấu các điểm ảnh với các màu khác nhau để phân đoạn hình ảnh

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ đường thẳng

Hàm copy() sẽ đảm bảo rằng bất kỳ thay đổi nào bạn thực hiện đối với hình ảnh sẽ không ảnh hưởng đến hình ảnh gốc.

```
10      # Copy ảnh gốc  
11      imageLine = img.copy()
```

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ đường thẳng

Chú thích hình ảnh bằng một đường màu, sử dụng line():

Cú pháp:

line(image, start_point, end_point, color, thickness)

Trong đó:

image là hình ảnh.

start_point, end_point là điểm bắt đầu và điểm kết thúc cho dòng.

color: màu của đường thẳng

thickness: Là độ dày của đường

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ đường thẳng

Vẽ một đường thẳng từ điểm A (x_1, y_1) đến điểm B (x_2, y_2), trong đó A và B là hai điểm bất kỳ trong hình. Nhìn vào góc trên cùng bên trái của hình ảnh, sẽ tìm thấy ở đó điểm gốc của hệ tọa độ xy.

Trục x thể hiện hướng nằm ngang hoặc các cột của hình ảnh.

Trục y thể hiện hướng dọc hoặc các hàng của hình ảnh.

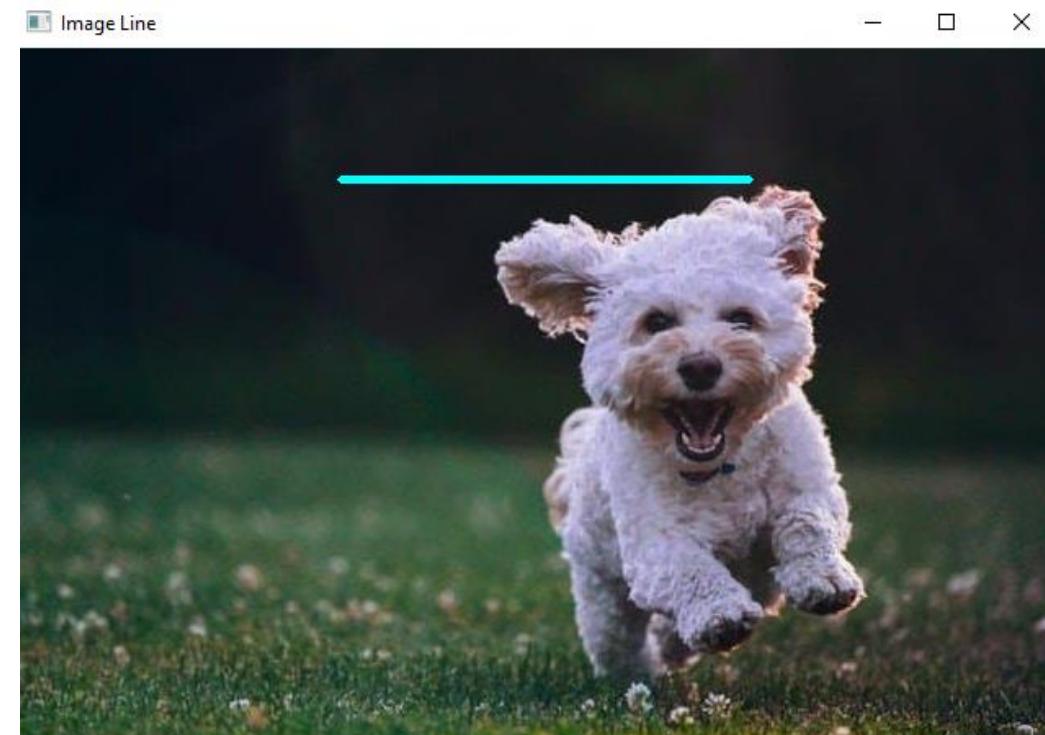
1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ đường thẳng

Ví dụ:

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 imageLine = img.copy()
6 # vẽ đường thẳng từ điểm A đến điểm B vào ảnh
7 pointA = (200,80)
8 pointB = (450,80)
9 cv2.line(imageLine, pointA, pointB, (255, 255, 0), thickness=3)
10 cv2.imshow('Image Line', imageLine)
11 cv2.waitKey(0)
12
```



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ một vòng tròn

Chú thích hình ảnh bằng một vòng tròn, sử dụng hàm circle()

Cú pháp:

circle(image, center_coordinates, radius, color, thickness)

Trong đó:

image là hình ảnh.

center_coordinates, radius là cho tâm của hình tròn và bán kính của nó.

color: màu của đường thẳng

thickness: Là độ dày của đường

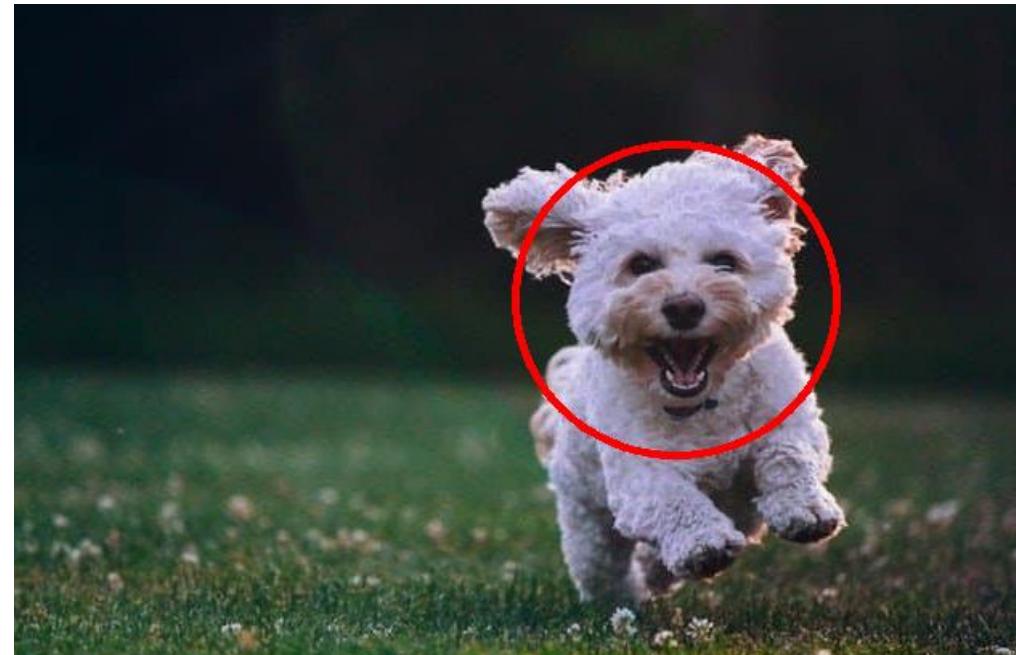
1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ một vòng tròn

Ví dụ:

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 image = img.copy()
6 # vẽ đường tròn
7 circle_center = (415,190)
8 # đường kính
9 radius =100
10 cv2.circle(image, circle_center, radius, (0, 0, 255), thickness=3)
11 cv2.imshow('Image', image)
12 cv2.waitKey(0)
```



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ một vòng tròn được lấp đầy

Chú thích hình ảnh bằng một vòng tròn, sử dụng hàm circle()

Cú pháp:

circle(image, center_coordinates, radius, color, thickness)

Trong đó:

image là hình ảnh.

center_coordinates, radius là cho tâm của hình tròn và bán kính của nó.

color: màu của đường thẳng

thickness: thay đổi đối số độ dày thành -1

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ một vòng tròn được lấp đầy

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 image = img.copy()
6 # vẽ đường tròn
7 circle_center = (415,190)
8 # đường kính
9 radius =100
10 cv2.circle(image, circle_center, radius, (0, 0, 255), thickness=-1)
11 cv2.imshow('Image', image)
12 cv2.waitKey(0)
```



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ hình chữ nhật

Vẽ một hình chữ nhật trên hình ảnh sử dụng hàm rectangle()

Cú pháp:

rectangle(image, start_point, end_point, color, thickness)

Trong đó:

image là hình ảnh.

start_point, end_point là điểm bắt đầu và điểm kết thúc cho dòng.

color: màu của đường thẳng

thickness: Là độ dày của đường

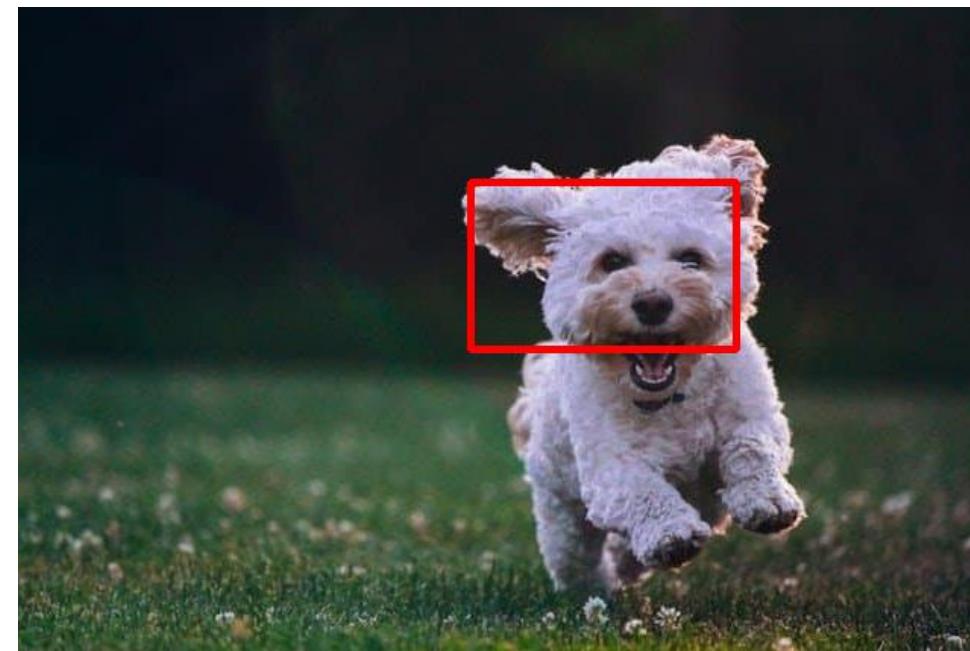
1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ hình chữ nhật

Ví dụ:

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 image = img.copy()
6 # Điểm A và B
7 start_point = (300,115)
8 end_point = (475,225)
9 # Vẽ hình chữ nhật
10 cv2.rectangle(image, start_point, end_point, (0, 0, 255), thickness= 3)
11 cv2.imshow('Image', image)
12 cv2.waitKey(0)
```



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ hình elip

Vẽ một hình elip trên hình ảnh bằng cách sử dụng hàm ellipse()

Cú pháp:

ellipse(image, centerCoordinates, axesLength, angle, startAngle, endAngle, color, thickness)

Trong đó:

centerCoordinates, axesLength: độ dài trục chính và trục nhỏ của hình elip

angle: góc quay

startAngle, endAngle: góc bắt đầu và góc kết thúc của hình elip

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Vẽ hình elip

Ví dụ:

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 image = img.copy()
6 # Điểm a và B
7 center = (415, 190)
8 axis1 = (100, 50)
9 # Vẽ hình chữ nhật
10 cv2.ellipse(image, center, axis1, 0, 0, 360, (255, 0, 0), thickness=3)
11 cv2.imshow('Image', image)
12 cv2.waitKey(0)
```



1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Thêm văn bản

Chú thích hình ảnh bằng văn bản sử dụng hàm putText()

Cú pháp:

putText(image, text, org, font, fontScale, color)

Trong đó:

image: hình ảnh

text: chuỗi văn bản thực tế mà muốn chú thích cho hình ảnh

org: chỉ định vị trí bắt đầu cho góc trên cùng bên trái của chuỗi văn bản

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Thêm văn bản

Trong đó:

font: một số kiểu phông chữ từ bộ sưu tập phông chữ Hershey và cả phông chữ nghiêng.

FONT_HERSHEY_SIMPLEX = 0,
FONT_HERSHEY_PLAIN = 1,
FONT_HERSHEY_DUPLEX = 2,
FONT_HERSHEY_COMPLEX = 3,
FONT_HERSHEY_TRIPLEX = 4,
FONT_HERSHEY_COMPLEX_SMALL = 5,
FONT_HERSHEY_SCRIPT_SIMPLEX = 6,
FONT_HERSHEY_SCRIPT_COMPLEX = 7,
FONT_ITALIC = 16

fontScale: Tỷ lệ phông chữ là một giá trị dấu phẩy động, được sử dụng để chia tỷ lệ kích thước cơ bản của phông chữ lên hoặc xuống

1.2 OpenCV

❖ Chú thích hình ảnh bằng OpenCV

▪ Thêm văn bản

Ví dụ:

```
1 import cv2
2 # đọc ảnh
3 img = cv2.imread('img_6.png')
4 # Copy ảnh gốc
5 image = img.copy()
6 text = 'Dog'
7 org = (50,350)
8 cv2.putText(image, text, org,
9             fontFace = cv2.FONT_HERSHEY_COMPLEX,
10            fontScale = 1.5, color = (0,0,250))
11 cv2.imshow('Image', image)
12 cv2.waitKey(0)
13 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Không gian màu trong OpenCV

Không gian màu (Colour Space) được hiểu là các mô hình toán để miêu tả màu sắc. Mỗi không gian màu đều có một tác dụng và ứng dụng trong các bài toán khác nhau.

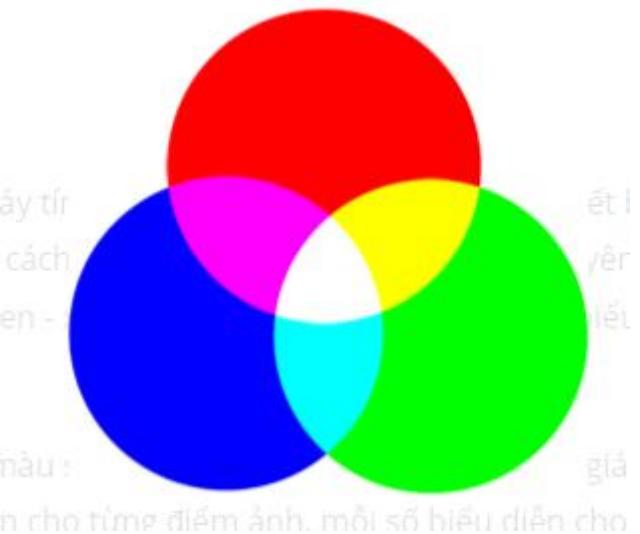


1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Hệ màu RGB

RGB là không gian màu phổ biến dùng trong máy tính, máy ảnh, điện thoại và nhiều thiết bị kĩ thuật số khác nhau. Không gian màu này khá gần với cách mắt người tổng hợp màu sắc. Nguyên lý cơ bản là sử dụng 3 màu sắc cơ bản R (red - đỏ), G (green - xanh lục) và B (blue - xanh lam) để biểu diễn tất cả các màu sắc.



1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Hệ màu RGB

- + Trong mô hình 24 bit mỗi kênh màu sẽ sử dụng 8bit để biểu diễn, tức là giá trị R, G, B nằm trong khoảng 0 - 255. Bộ 3 số này biểu diễn cho từng điểm ảnh, mỗi số biểu diễn cho cường độ của một màu.
- + Với mô hình màu 24bit thì số màu tối đa có thể tạo ra là $255 \times 255 \times 255 = 16581375$ màu.
- + Một điểm cần lưu ý là với các thư viện đọc ảnh và hiển thị ảnh như matplotlib thì các ảnh được đọc theo RGB tuy nhiên **Opencv đọc ảnh theo các kênh BGR**.

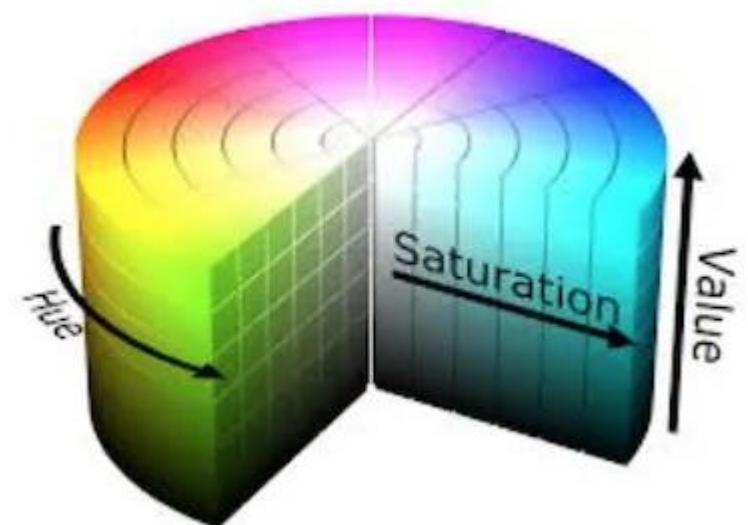
1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Hệ màu HSV

Không gian màu HSV (còn gọi là HSB) là một cách tự nhiên hơn để mô tả màu sắc, dựa trên 3 số liệu:

- + H: (Hue) Vùng màu (0-359)
- + S: (Saturation) Độ bão hòa màu (là lượng màu xám trong màu)
- + B (hay V): (Bright hay Value) Độ sáng (0-100)



1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Ảnh xám (GrayScale)

Chuyển đổi từ ảnh màu sang ảnh xám, giá trị mỗi pixel được tính bằng công thức:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Ảnh xám (GrayScale)

Ảnh xám cung cấp ít thông tin hơn cho mỗi pixel. Với ảnh thông thường thì mỗi pixel thường được cung cấp 3 (RGB – $8 \times 8 \times 8 = 24$ bit) trường thông tin trong khi với ảnh xám chỉ có 1 (gray – 8bit [0-255]) trường thông tin, việc giảm khối lượng thông tin giúp tăng tốc độ xử lý nhưng vẫn đảm bảo các tác vụ cần thiết.



Color



Grayscale

1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Ảnh nhị phân (binary)

Ảnh nhị phân (binary) là 1 bước giảm thông tin nữa so với ảnh xám. Với ảnh xám thì ngưỡng xám của mỗi pixel được miêu tả trong khoảng [0;255] nhưng với ảnh nhị phân thì mỗi pixel chỉ có giá trị bằng 0 hoặc 255.



Grayscale



Black-and-white

1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Chuyển đổi giữa các không gian màu

OpenCV hỗ trợ phép chuyển đổi ảnh giữa một số cặp không gian màu bằng hàm cvtColor():

Cú pháp:

cv2.cvtColor(src, code)

Trong đó:

src: ảnh đầu vào.

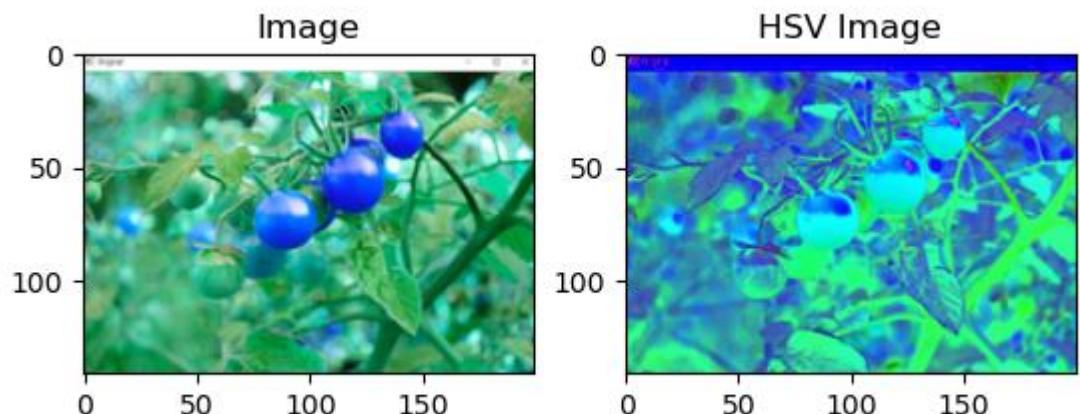
code: mã chuyển không gian màu.

1.2 OpenCV

- ❖ Không gian màu trong OpenCV
 - Chuyển đổi giữa các không gian màu

Ví dụ

```
1 import cv2
2 import matplotlib.pyplot as plt
3 img=cv2.imread("img.png")
4 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
5 plt.figure()
6 plt.subplot(1,2,1)
7 plt.imshow(img)
8 plt.title('Image')
9 plt.subplot(1,2,2)
10 plt.imshow(hsv)
11 plt.title('HSV Image')
12 plt.show()
```



1.2 OpenCV

- ❖ **Không gian màu trong OpenCV**
- **Giới hạn vùng mã màu của đỏ, hay những giá trị trong không gian màu HSV “gần”.**

Sử dụng mảng numpy cho vùng giới hạn màu

Ví dụ:

```
lower_green = np.array([0, 220, 0])
```

```
upper_green = np.array([50, 255, 50])
```

1.2 OpenCV

❖ Không gian màu trong OpenCV

▪ Lọc ra những pixel có giá trị nằm trong giới hạn mã màu tìm được

OpenCV cung cấp hàm `inRange()` để lọc ra những pixel có giá trị nằm trong vùng chọn. Tạo ra một mask thể hiện đúng vùng màu mong muốn.

Cú pháp:

`cv.inRange (src, lowerb, upperb)`

Trong đó:

src: ảnh đầu vào

lowb: mảng biên dưới

upperb: mảng biên trên

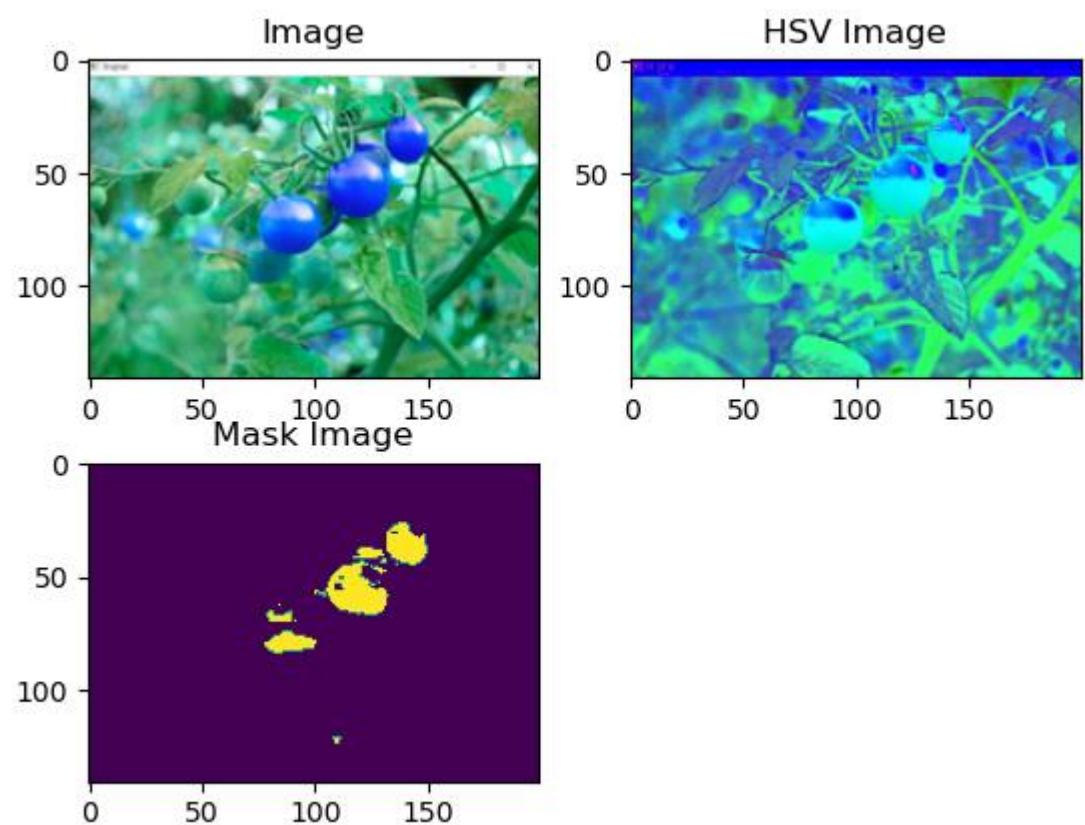
1.2 OpenCV

❖ Không gian màu trong OpenCV

- Lọc ra những pixel có giá trị nằm trong giới hạn mã màu tìm được

Ví dụ:

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4 img=cv2.imread("img.png")
5 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
6 lower_green = np.array([0, 108, 200])
7 upper_green = np.array([10, 255, 255])
8 mask = cv2.inRange(hsv, lower_green, upper_green)
9 plt.figure()
10 plt.subplot(2,2,1)
11 plt.imshow(img)
12 plt.title('Image')
13 plt.subplot(2,2,2)
14 plt.imshow(hsv)
15 plt.title('HSV Image')
16 plt.subplot(2,2,3)
17 plt.imshow(mask)
18 plt.title('Mask Image')
19 plt.show()
```



1.2 OpenCV

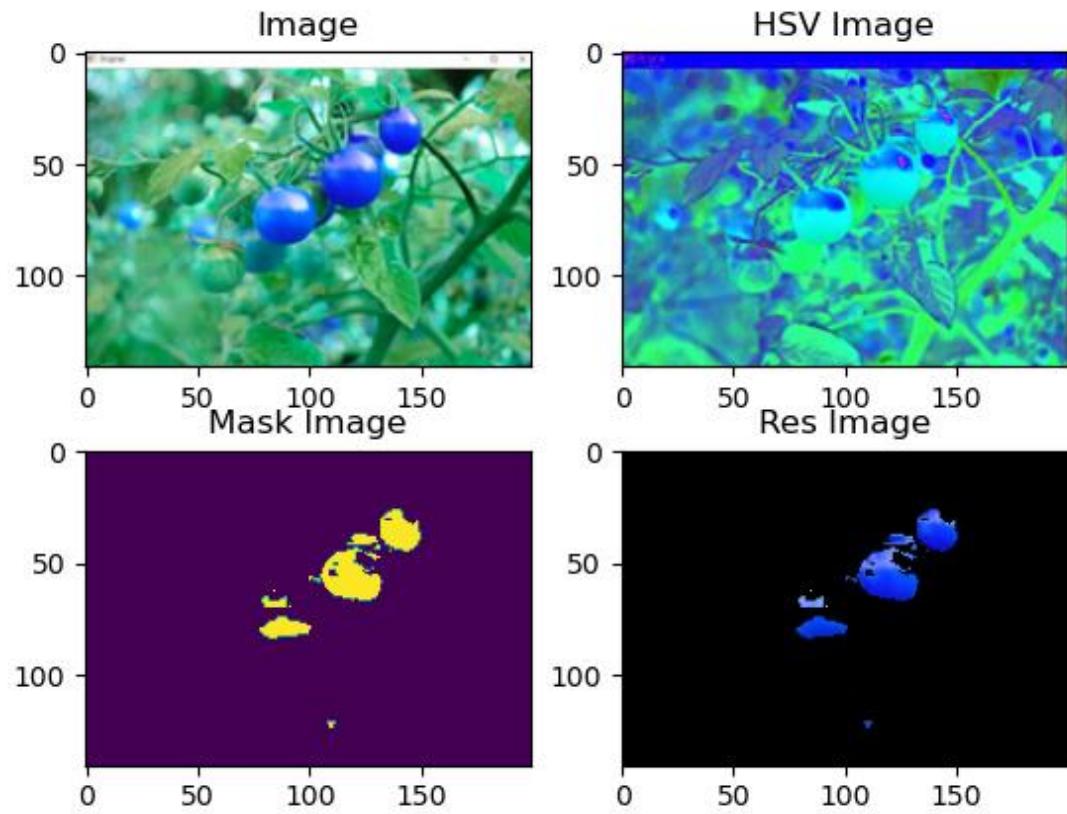
- ❖ **Không gian màu trong OpenCV**
- Sử dụng phép and thông qua hàm `bitwise_and()` trên ảnh gốc ban đầu và mask là ảnh nhị phân, thu được ảnh màu mới bao gồm các pixel “gần” xanh, những pixel khác trên ảnh gốc đều trở thành giá trị 0 (màu đen) trên ảnh.

1.2 OpenCV

❖ Không gian màu trong OpenCV

Ví dụ:

```
1 import cv2
2 import matplotlib.pyplot as plt
3 import numpy as np
4 img=cv2.imread("img.png")
5 hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
6 lower_green = np.array([0, 108, 200])
7 upper_green = np.array([10, 255, 255])
8 mask = cv2.inRange(hsv, lower_green, upper_green)
9 res = cv2.bitwise_and(img, img, mask= mask)
10 plt.figure()
11 plt.subplot(2,2,1)
12 plt.imshow(img)
13 plt.title('Image')
14 plt.subplot(2,2,2)
15 plt.imshow(hsv)
16 plt.title('HSV Image')
17 plt.subplot(2,2,3)
18 plt.imshow(mask)
19 plt.title('Mask Image')
20 plt.subplot(2,2,4)
21 plt.imshow(res)
22 plt.title('Res Image')
23 plt.show()
```



1.2 OpenCV

- ❖ Lọc hình ảnh bằng tích chập trong OpenCV



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

Hình ảnh được lọc với đa dạng các bộ lọc truyền dẫn thấp (low-pass filters LPF), bộ lọc truyền dẫn cao (high-pass filters HPF). Một HPF sẽ giúp tìm ra các cạnh trong một hình ảnh, còn LPF sẽ lọc nhiễu cho ảnh và làm mờ ảnh.

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Bộ lọc tích chập 2D (2D convolution)

Trong xử lý ảnh, nhân chập là một ma trận 2D được sử dụng để lọc ảnh. Còn được gọi là ma trận tích chập, nhân tích chập thường là ma trận vuông, $M \times N$, trong đó cả M và N đều là các số nguyên lẻ (ví dụ: 3×3 , 5×5 , 7×7 , v.v.). Xem ma trận ví dụ 3×3 2D được đưa ra bên dưới.

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

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Bộ lọc tích chập 2D (2D convolution)

Được sử dụng để thực hiện các phép toán trên mỗi pixel của hình ảnh nhằm đạt được hiệu ứng mong muốn (như làm mờ hoặc làm sắc nét hình ảnh). Nhưng tại sao lại muốn làm mờ một hình ảnh? Đây là hai lý do quan trọng:

- + Bởi vì nó làm giảm một số loại nhiễu nhất định trong hình ảnh. Vì lý do này, làm mờ thường được gọi là làm mịn.
- + Để xóa phông nền gây mất tập trung, có thể cố ý làm mờ các phần của hình ảnh, như được thực hiện ở chế độ 'Chân dung', trên máy ảnh thiết bị di động.

1.2 OpenCV

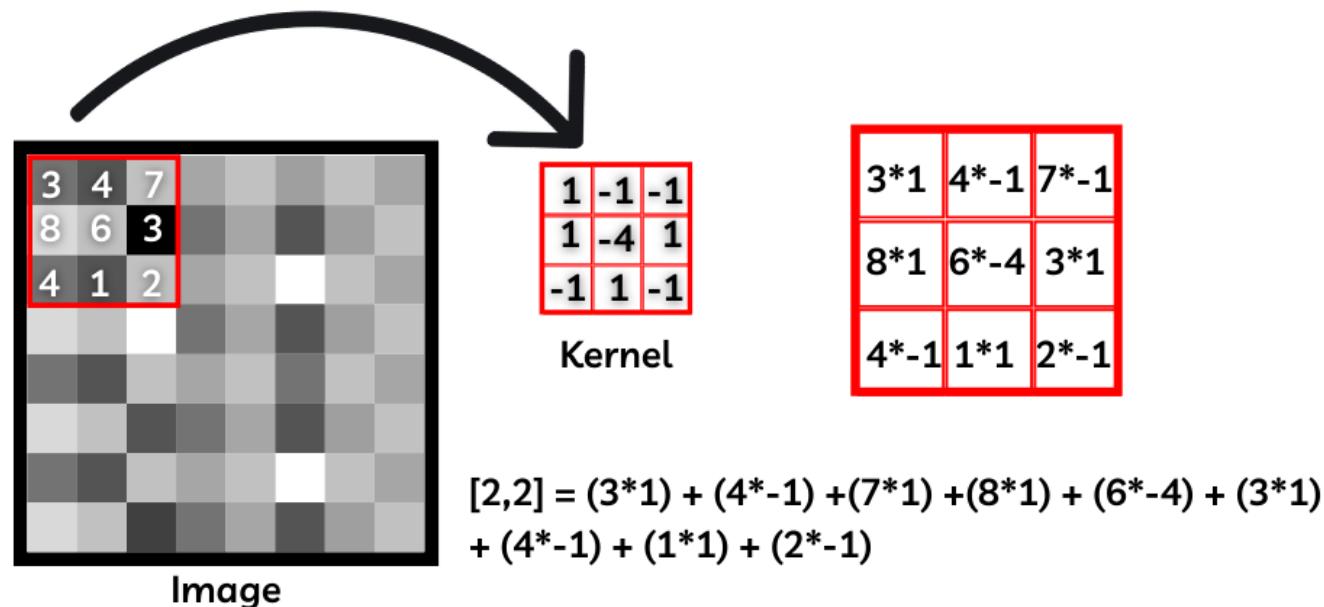
❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Cách sử dụng Kernels để làm sắc nét hoặc làm mờ hình ảnh

Việc lọc ảnh nguồn được thực hiện bằng cách kết hợp hạt nhân (ma trận) với ảnh. Nói một cách dễ hiểu, tích chập của một hình ảnh với một nhân (ma trận) đại diện cho một phép toán đơn giản, giữa nhân và các phần tử tương ứng của nó trong hình ảnh.

1.2 OpenCV

- ❖ Lọc hình ảnh bằng tích chập trong OpenCV
 - Cách sử dụng Kernels để làm sắc nét hoặc làm mờ hình ảnh



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Cách sử dụng Kernels để làm sắc nét hoặc làm mờ hình ảnh

Một số hạt nhân phổ biến là

+ Nhân định danh

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

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

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

+ Kernel phát hiện cạnh

Identity kernel

Edge detection

Sharpen kernel

+ Làm sắc nét nhân

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

Box blur

$$\frac{1}{256} \begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

Gaussian blurr kernel

+ Kernel làm mờ hộp

+ Kernel làm mờ Gaussian

1.2 OpenCV

- ❖ **Lọc hình ảnh bằng tích chập trong OpenCV**

- **Áp dụng nhân định danh cho hình ảnh trong OpenCV**

Nhân định danh là một ma trận vuông, trong đó phần tử ở giữa là 1 và tất cả các phần tử khác bằng 0.

Ví dụ: Một hạt nhân định danh 3×3

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

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Các bước sau được thực hiện

- + Đọc hình ảnh thử nghiệm
- + Xác định hạt nhân nhận dạng, sử dụng mảng NumPy 3×3
- + Sử dụng filter2D() hàm trong OpenCV để thực hiện hoạt động lọc tuyến tính
- + Hiển thị hình ảnh gốc và hình ảnh đã lọc, sử dụng hàm imshow()
- + Lưu hình ảnh đã lọc vào đĩa, sử dụng imwrite()

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Hàm filter2D()

+ Sử dụng filter2D() hàm trong OpenCV để thực hiện hoạt động lọc tuyến tính

Cú pháp:

filter2D(src, dst, ddepth, kernel)

Trong đó:

src: là hình ảnh nguồn

dst: Tên của hình ảnh đầu ra sau khi áp dụng bộ lọc

ddepth: Độ sâu của hình ảnh đầu ra [-1 sẽ cung cấp độ sâu hình ảnh đầu ra giống như hình ảnh đầu vào]

kernel: Ma trận 2D mà chúng ta muốn hình ảnh biến đổi theo

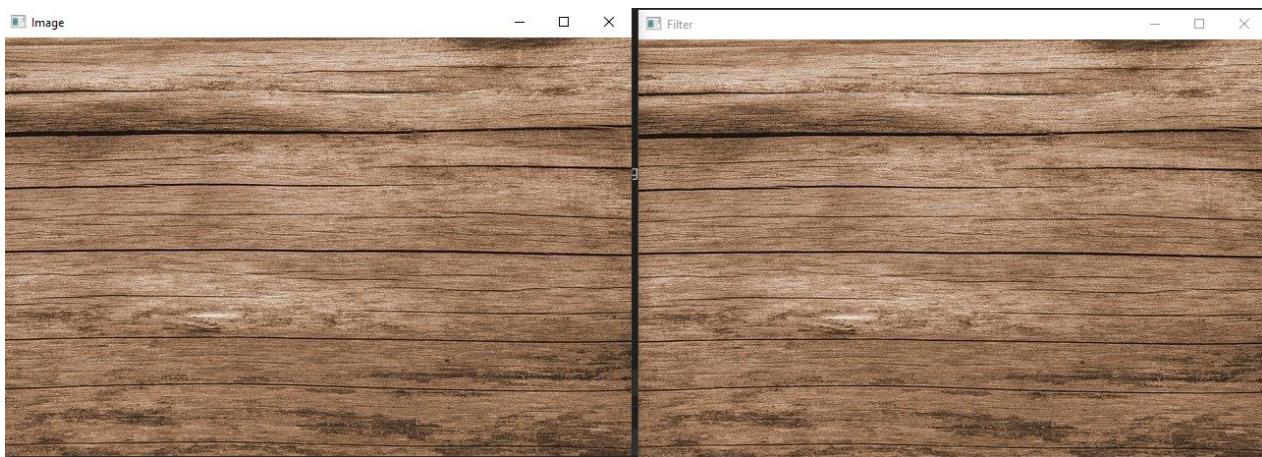
1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Hàm filter2D()

+ Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_7.png')
4
5 kernel1 = np.array([[0, 0, 0],
6                     [0, 1, 0],
7                     [0, 0, 0]])
8
9 identity = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)
10
11 # We should get the same image
12 cv2.imshow('Image', image)
13 cv2.imshow('Filter', identity)
14
15 cv2.waitKey()
16 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ hình ảnh bằng ma trận tích chập 2d

Sử dụng ma trận tích chập 2D, cần đảm bảo rằng tất cả các giá trị đều được chuẩn hóa. Điều này được thực hiện bằng cách chia mỗi phần tử của nhân cho số phần tử trong nhân, trong trường hợp này là 25. Điều này đảm bảo tất cả các giá trị nằm trong phạm vi [0,1].

$$\frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ hình ảnh bằng ma trận tích chập 2d

Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_7.png')
4
5 kernel1 = np.ones((5, 5), np.float32) / 25
6
7 identity = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)
8
9 # We should get the same image
10 cv2.imshow('Image', image)
11 cv2.imshow('Filter 2D', identity)
12
13 cv2.waitKey()
14 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ hình ảnh bằng Trung bình (Average)

Sử dụng hàm cv2.blur()

Cú pháp:

cv2.blur (src, ksize)

Trong đó:

rc: Là hình ảnh được làm mờ.

ksize: Một bộ giá trị đại diện cho kích thước hạt nhân làm mờ.

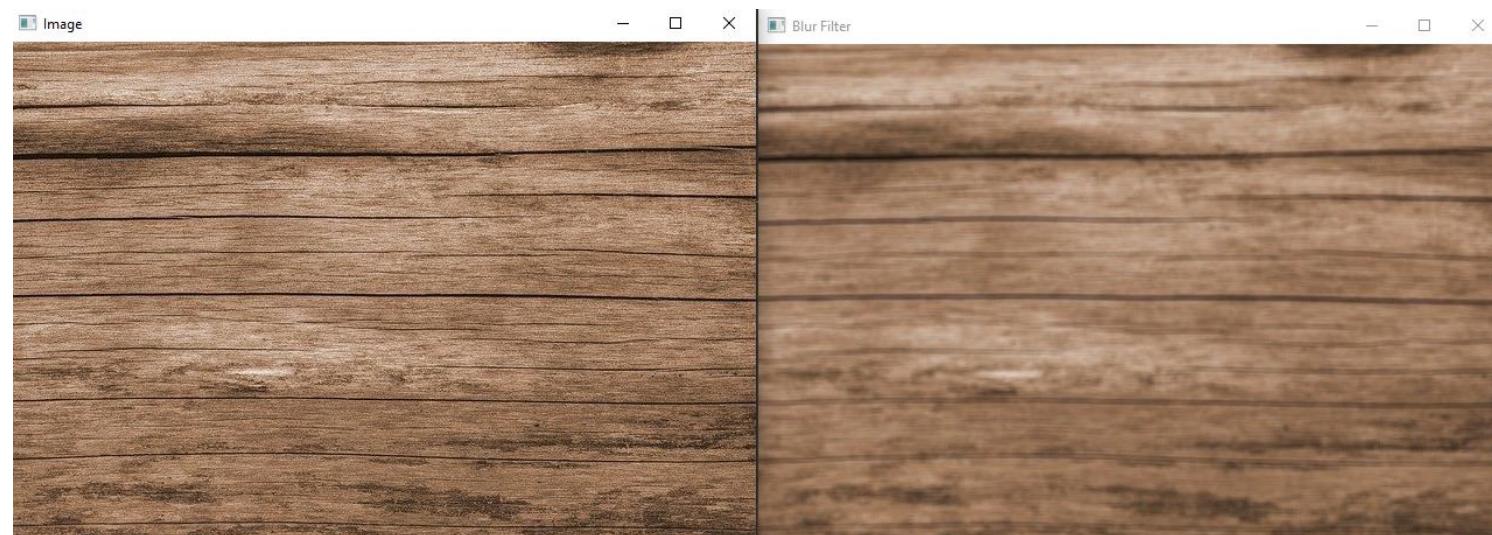
1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ hình ảnh bằng Trung bình (Average)

Ví dụ:

```
1 import cv2  
2  
3 image = cv2.imread('img_7.png')  
4  
5 img_blur = cv2.blur(src=image, ksize=(5,5))  
6  
# We should get the same image  
8 cv2.imshow('Image', image)  
9 cv2.imshow('Blur Filter', img_blur)  
10  
11 cv2.waitKey()  
12 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ Gaussian cho hình ảnh trong OpenCV

- + Áp dụng hiệu ứng mờ Gaussian cho một hình ảnh. Kỹ thuật này sử dụng một bộ lọc Gaussian, thực hiện một trung bình có trọng số, Trong trường hợp này, Gaussian làm mờ trọng số các giá trị pixel, dựa trên khoảng cách của chúng từ tâm của hạt nhân. Các điểm ảnh xa trung tâm hơn có ít ảnh hưởng hơn đến mức trung bình có trọng số.
- + Bộ lọc Gaussian rất hiệu quả trong việc xóa bỏ noise khỏi hình ảnh.

1.2 OpenCV

- ❖ Lọc hình ảnh bằng tích chập trong OpenCV
 - Làm mờ Gaussian cho hình ảnh trong OpenCV

Bộ lọc Gaussian sử dụng hàm cv2.GaussianBlur()

Cú pháp:

GaussianBlur(src, ksize, sigmaX, sigmaY)

Trong đó:

- + src là hình ảnh nguồn
- + ksize: xác định kích thước của hạt nhân Gaussian
- + sigmaX và sigmaY: đặt thành 0. Đây là độ lệch chuẩn của hạt nhân Gaussian, theo hướng X (ngang) và Y (dọc)

1.2 OpenCV

- ❖ Lọc hình ảnh bằng tích chập trong OpenCV
 - Làm mờ Gaussian cho hình ảnh trong OpenCV

Ví dụ:

```
1 import cv2
2
3 image = cv2.imread('img_7.png')
4
5 gaussian_blur = cv2.GaussianBlur(src=image, ksize=(5,5),
6                                 sigmaX=0, sigmaY=0)
7
8 cv2.imshow('Image', image)
9 cv2.imshow('Gauss Filter', gaussian_blur)
10
11 cv2.waitKey()
12 cv2.destroyAllWindows()
```



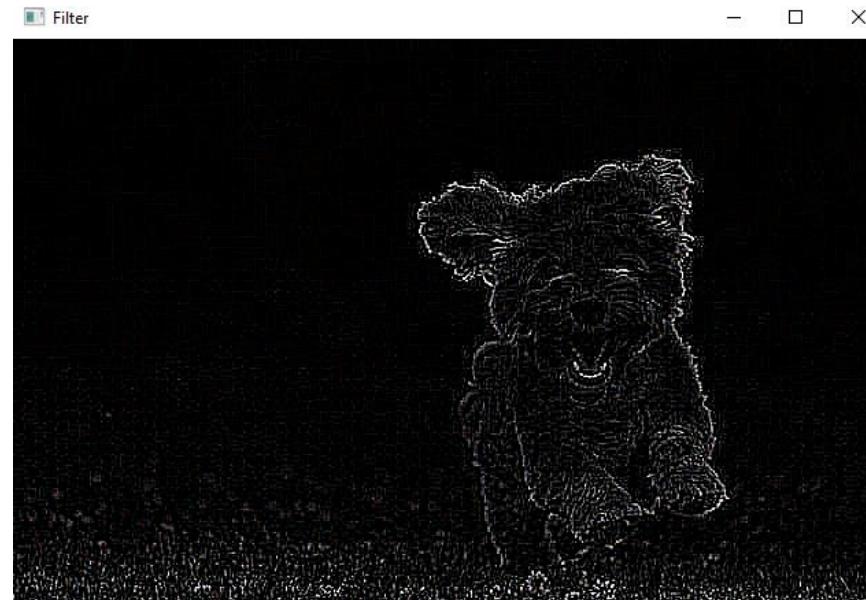
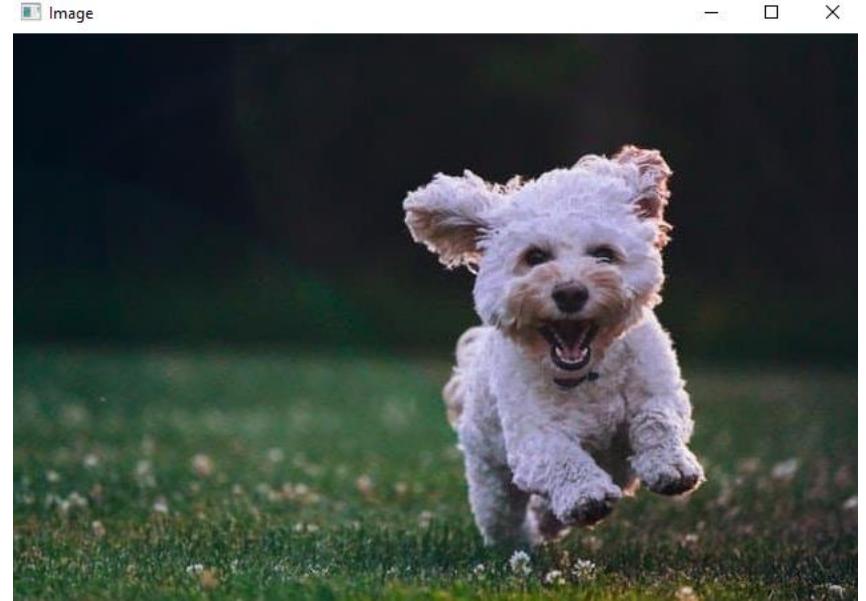
1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Phát hiện cạnh một hình ảnh có ma trận tích chập 2d

Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_6.png')
4
5 kernel2 = np.array([[-1, -1, -1],
6                     [-1, 8, -1],
7                     [-1, -1, -1]])
8
9 img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)
10
11 cv2.imshow('Image', image)
12 cv2.imshow('Filter', img)
13
14 cv2.waitKey()
15 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ Trung vị cho hình ảnh

Tính toán các median của toàn bộ các pixels trên một vùng cục bộ và thay thế các điểm ảnh trên vùng cục bộ bằng median. Sử dụng hàm cv2.medianBlur()

Cú pháp:

cv2.medianBlur(src, ksize)

Trong đó:

src là hình ảnh nguồn.

ksize là kích thước hạt nhân, phải là một số nguyên dương, lẻ.

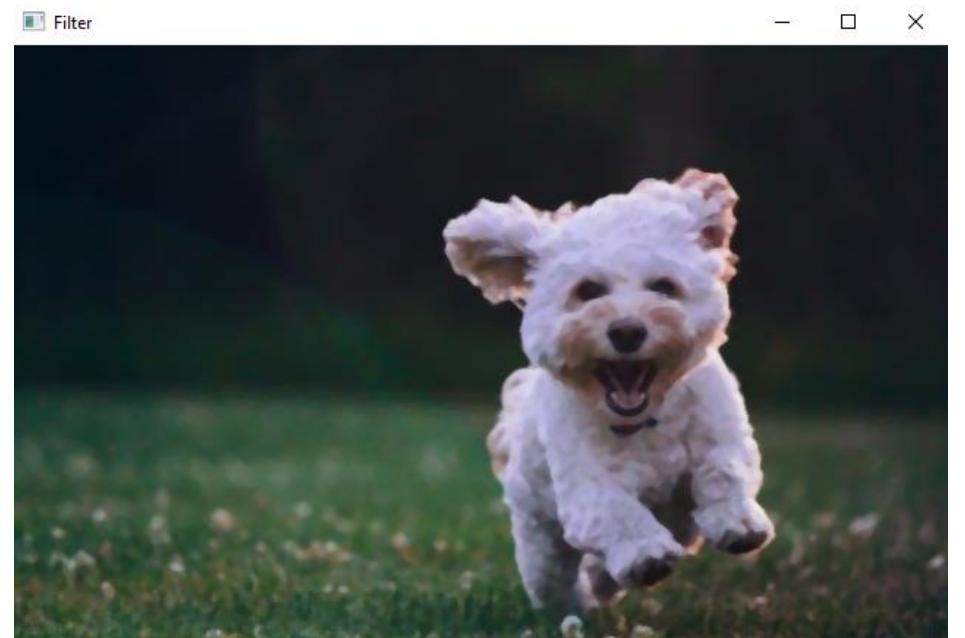
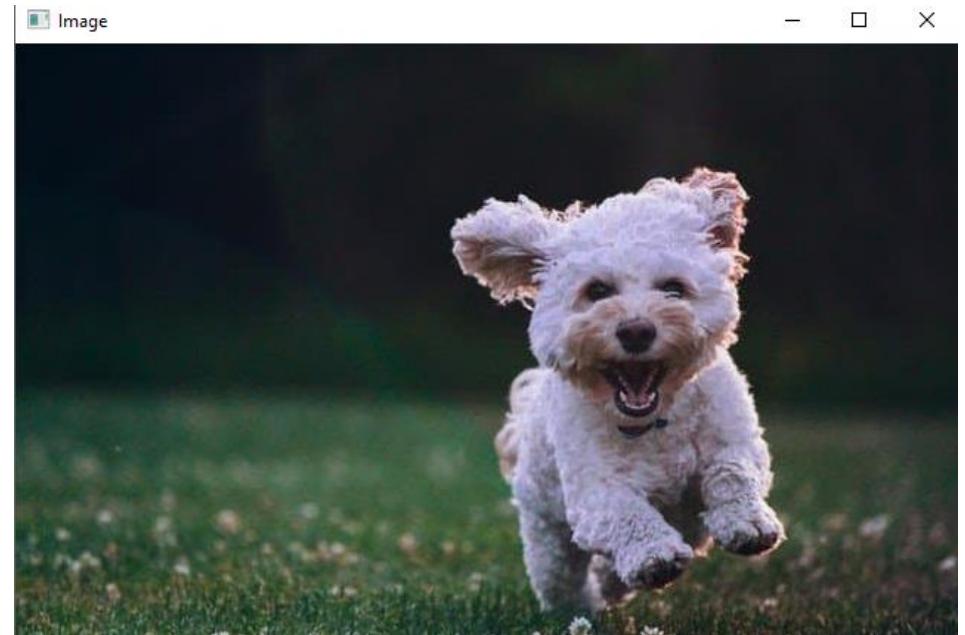
1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Làm mờ Trung vị cho hình ảnh

Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_6.png')
4
5 median = cv2.medianBlur(src=image, ksize=5)
6
7 cv2.imshow('Image', image)
8 cv2.imshow('Filter', median)
9
10 cv2.waitKey()
11 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Lọc song phương cho hình ảnh

Mặc dù làm mờ có thể là một cách hiệu quả để giảm nhiễu trong hình ảnh, nhưng thường không nên làm mờ toàn bộ hình ảnh, vì các chi tiết quan trọng và các cạnh sắc nét có thể bị mất.

+ Kỹ thuật này áp dụng bộ lọc một cách chọn lọc để làm mờ các pixel có cường độ tương tự trong một vùng lân cận. Các cạnh sắc nét được giữ nguyên, ở bất cứ đâu có thể.

+ Nó cho phép kiểm soát không chỉ kích thước không gian của bộ lọc mà còn cả mức độ bao gồm các pixel lân cận trong đầu ra được lọc. Điều này được thực hiện, dựa trên sự thay đổi về cường độ màu và cũng như khoảng cách từ pixel được lọc.

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Lọc song phương cho hình ảnh

Lọc song phương về cơ bản áp dụng hiệu ứng mờ Gaussian 2D (có trọng số) cho hình ảnh, đồng thời xem xét sự thay đổi về cường độ của các pixel lân cận để giảm thiểu hiện tượng mờ gần các cạnh. Điều này có nghĩa là hình dạng của hạt nhân thực sự phụ thuộc vào nội dung hình ảnh cục bộ, tại mọi vị trí pixel.

1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Lọc song phương cho hình ảnh

Sử dụng bilateralFilter()

Cú pháp:

cv2.bilateralFilter(src, d, sigmaColor, sigmaSpace)

Trong đó:

Đối số tiếp theo d, xác định đường kính của vùng lân cận pixel được sử dụng để lọc.

sigmaColor và sigmaSpace xác định độ lệch chuẩn của phân bố cường độ màu (1D) và phân bố không gian (2D) tương ứng.

Tham sigmaSpaces xác định phạm vi không gian của hạt nhân, theo cả hướng x và y Tham sigmaColors xác định phân phối Gaussian một chiều, chỉ định mức độ mà sự khác biệt về cường độ pixel có thể được chấp nhận.

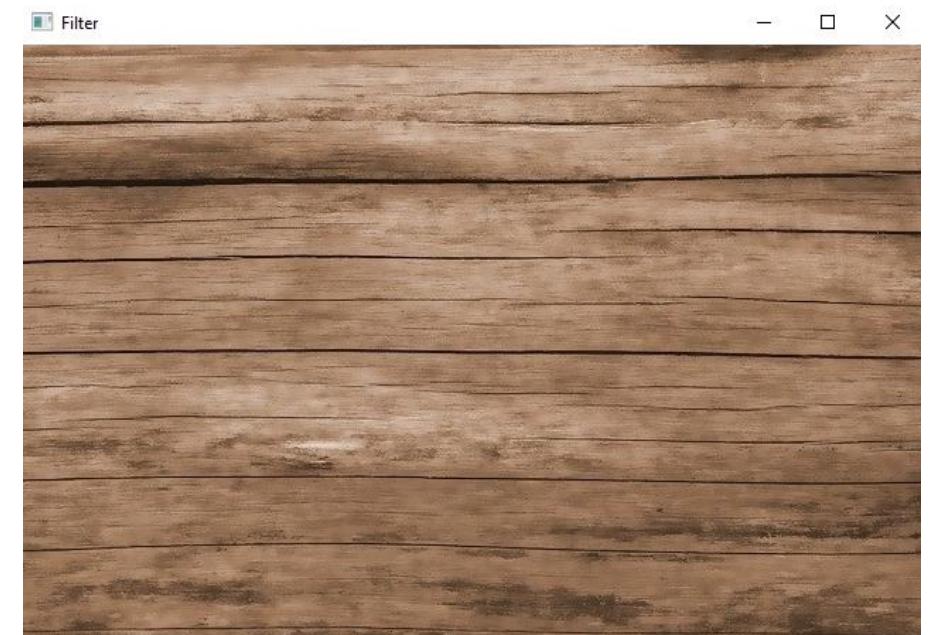
1.2 OpenCV

❖ Lọc hình ảnh bằng tích chập trong OpenCV

▪ Lọc song phương cho hình ảnh

Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_7.png')
4
5 bilateral_filter = cv2.bilateralFilter(src=image, d=9,
6                                         sigmaColor=75, sigmaSpace=75)
7
8 cv2.imshow('Image', image)
9 cv2.imshow('Filter', bilateral_filter)
10
11 cv2.waitKey()
12 cv2.destroyAllWindows()
```

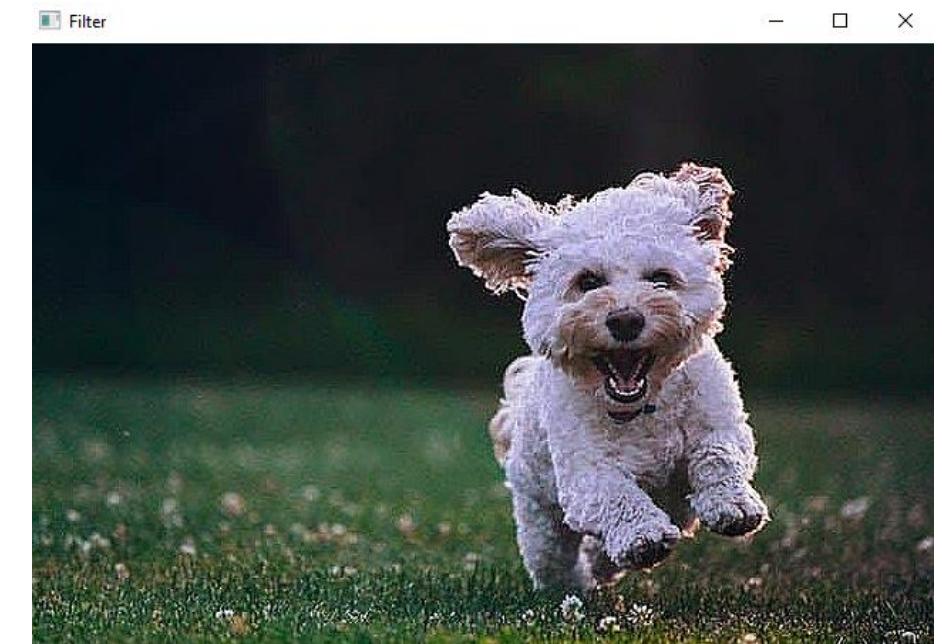
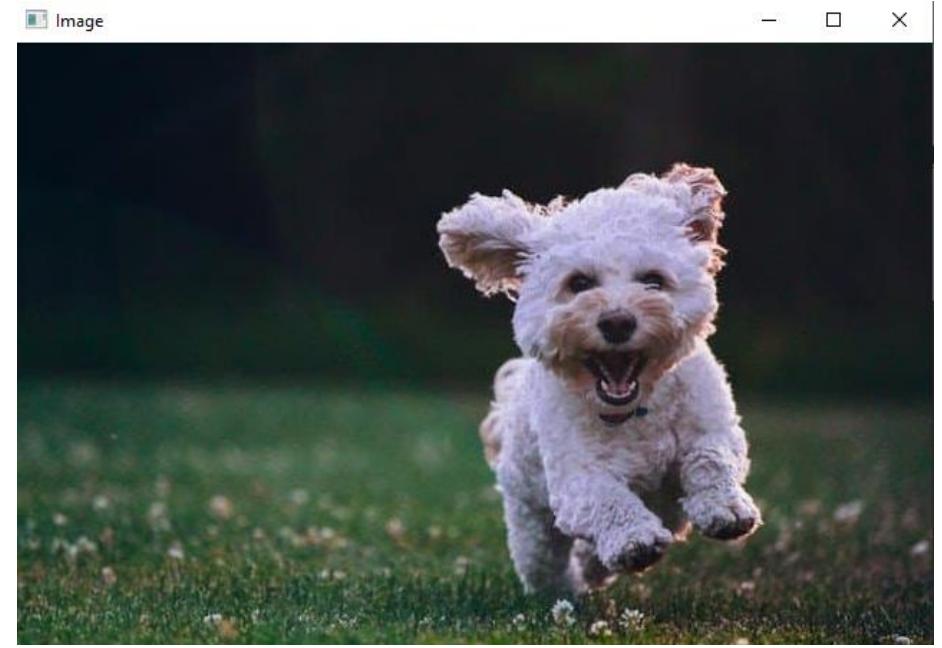


1.2 OpenCV

- ❖ Lọc hình ảnh bằng tích chập trong OpenCV
 - Làm sắc nét hình ảnh sử dụng nhân tích chập 2D

Ví dụ:

```
1 import cv2
2 import numpy as np
3 image = cv2.imread('img_6.png')
4
5 kernel2 = np.array([[0, -1, 0],
6                     [-1, 5, -1],
7                     [0, -1, 0]])
8
9 img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)
10
11 cv2.imshow('Image', image)
12 cv2.imshow('Filter', img)
13
14 cv2.waitKey()
15 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

- + Thresholding là một kỹ thuật trong OpenCV, là việc gán các giá trị pixel liên quan đến giá trị ngưỡng được cung cấp. Trong tạo ngưỡng, mỗi giá trị pixel được so sánh với giá trị ngưỡng.
- + Trong Computer Vision, kỹ thuật xác định ngưỡng này được thực hiện trên hình ảnh thang độ xám. Vì vậy, ban đầu, hình ảnh phải được chuyển đổi trong không gian màu thang độ xám.

1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

Thuật toán tạo Thresholding lấy hình ảnh nguồn (src) và giá trị ngưỡng (thresh) làm đầu vào và tạo ra hình ảnh đầu ra (dst), bằng cách so sánh cường độ pixel tại vị trí pixel nguồn (x,y) với ngưỡng. Nếu $\text{src}(x,y) > \text{thresh}$, thì $\text{dst}(x,y)$ được gán một số giá trị. Nếu không, $\text{dst}(x,y)$ được gán một số giá trị khác.

1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

Ví dụ: Hình ảnh đầu vào chứa các số được viết với cường độ (giá trị thang độ xám) bằng chính số đó. cường độ pixel của số '200' là 200, cường độ của số '32' là 32. Và '32' xuất hiện tối hơn nhiều so với '200'.



1.2 OpenCV

❖ **Nguõng hình ảnh trong OpenCV**

Hàm cv2.threshold được sử dụng để tạo nguong.

Cú pháp:

cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)

Trong đó:

thresholdValue: Giá trị của Nguõng thấp hơn và cao hơn mà giá trị pixel sẽ thay đổi tương ứng.

maxVal : Giá trị tối đa có thể được gán cho một pixel.

nguongTechnique : Loại nguong được áp dụng.

1.2 OpenCV

❖ Nguõng hình ảnh trong OpenCV

Các Kỹ thuật Nguõng gồm:

+ cv2.THRESH_BINARY

+ cv2.THRESH_BINARY_INV

+ cv.THRESH_TRUNC

+ cv.THRESH_TOZERO

+ cv.THRESH_TOZERO_INV

Binary

$$dst(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

Inverted Binary

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$$

Truncated

$$dst(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

To Zero

$$dst(x, y) = \begin{cases} \text{src}(x, y) & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

To Zero Inverted

$$dst(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng nhị phân (THRESH_BINARY)**

Nếu cường độ pixel lớn hơn ngưỡng đã đặt, giá trị được đặt thành 255, giá trị khác được đặt thành 0 (màu đen).

if $\text{src}(x,y) > \text{thresh}$:

$\text{dst}(x,y) = \text{maxValue}$

else:

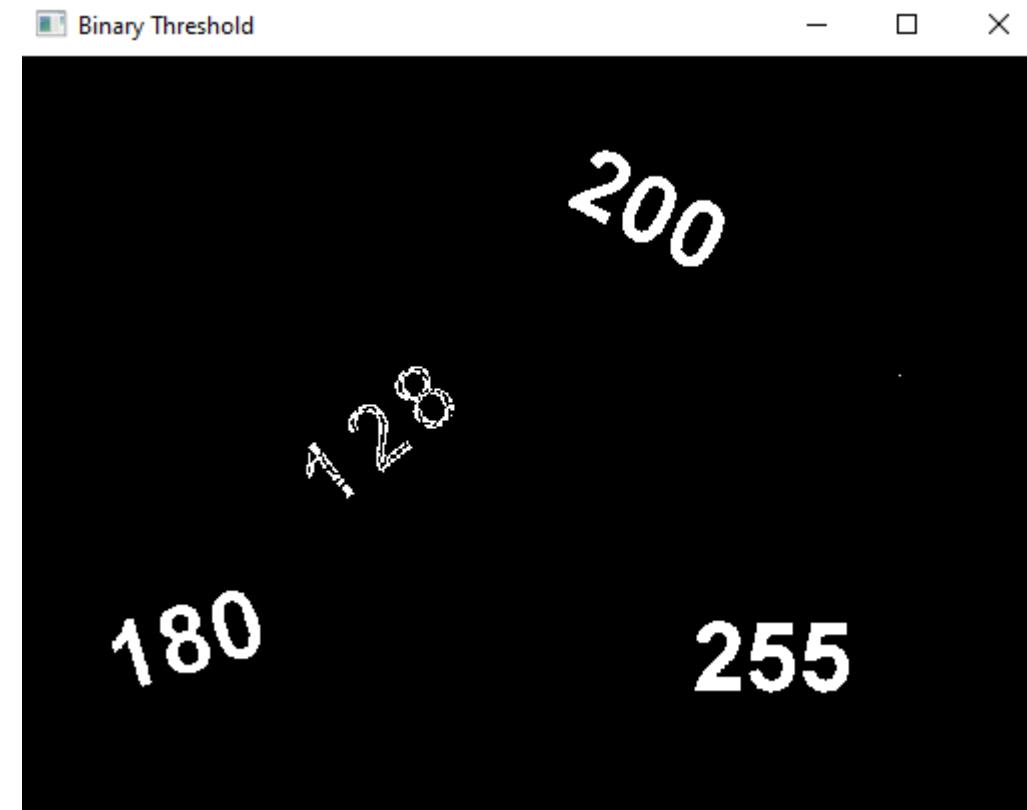
$\text{dst}(x,y) = 0$

1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng nhị phân (THRESH_BINARY)**

Ví dụ:

```
1 import cv2
2
3 src = cv2.imread("img_8.png")
4 img = cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
5
6 thresh = 0
7 maxValue = 255
8
9 th, dst = cv2.threshold(img, thresh, maxValue, cv2.THRESH_BINARY)
10 cv2.imshow('THRESH_BINARY Threshold', dst)
11 if cv2.waitKey(0) & 0xff == 27:
12     cv2.destroyAllWindows()
```



1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng nhị phân nghịch đảo (THRESH_BINARY_INV)**

Ngưỡng nhị phân nghịch đảo ngược lại với Ngưỡng nhị phân. Pixel đích được đặt thành:

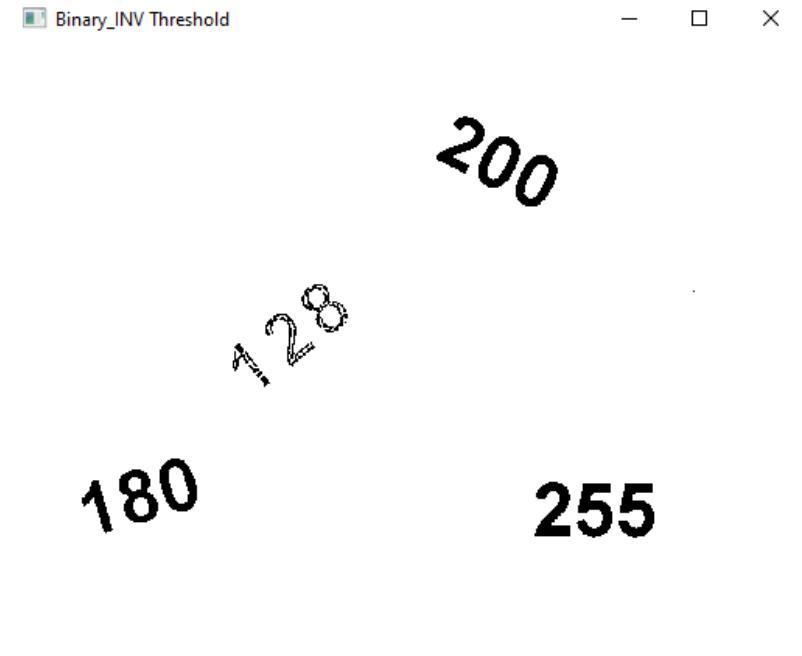
- + 0, nếu pixel nguồn tương ứng lớn hơn ngưỡng
- + maxValue, nếu pixel nguồn nhỏ hơn ngưỡng

1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng nhị phân nghịch đảo (THRESH_BINARY_INV)**

Ví dụ:

```
1 import cv2
2
3 src = cv2.imread("img_8.png")
4 img = cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
5
6 thresh = 128
7 maxValue = 255
8
9 th, dst = cv2.threshold(img, thresh, maxValue, cv2.THRESH_BINARY_INV)
10 cv2.imshow('THRESH_BINARY_INV Threshold', dst)
11 if cv2.waitKey(0) & 0xff == 27:
12     cv2.destroyAllWindows()
```



1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

▪ **Ngưỡng cắt bớt (THRESH_TRUNC)**

- + Pixel đích được đặt thành ngưỡng (thresh), nếu giá trị pixel nguồn lớn hơn ngưỡng.
- + Nếu không, nó được đặt thành giá trị pixel nguồn (không thay đổi)
- + Bỏ qua maxValue.

if $\text{src}(x,y) > \text{thresh}$:

$\text{dst}(x,y) = \text{thresh}$

else:

$\text{dst}(x,y) = \text{src}(x,y)$

1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng cắt bớt (THRESH_TRUNC)**

Ví dụ:

```
1 import cv2
2
3 src = cv2.imread("img_8.png")
4 img = cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
5
6 thresh = 128
7 maxValue = 255
8
9 th, dst = cv2.threshold(img, thresh, maxValue, cv2.THRESH_TRUNC)
10 cv2.imshow('THRESH_TRUNC Threshold', dst)
11 if cv2.waitKey(0) & 0xff == 27:
12     cv2.destroyAllWindows()
```



1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

▪ **Ngưỡng về 0 (THRESH_TOZERO)**

+ Giá trị pixel đích được đặt thành giá trị pixel của nguồn tương ứng, nếu giá trị pixel nguồn lớn hơn ngưỡng.

+ Nếu không, nó được đặt thành 0

+ Bỏ qua maxValue

if $\text{src}(x,y) > \text{thresh}$

$\text{dst}(x,y) = \text{src}(x,y)$

else

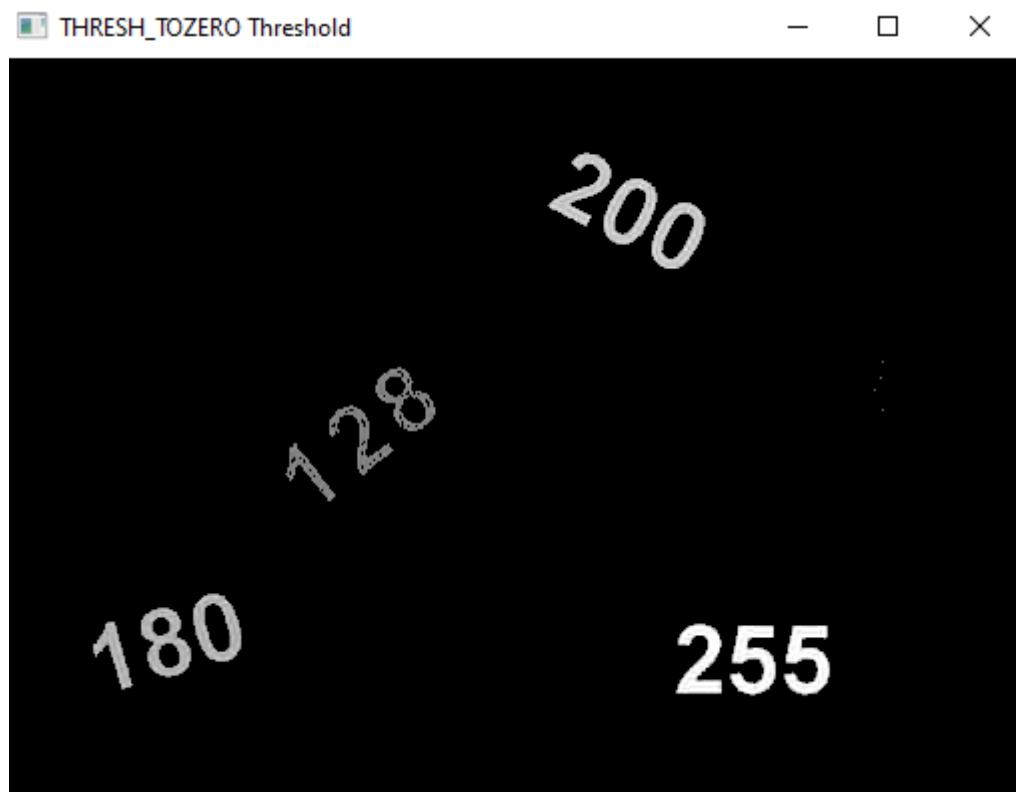
$\text{dst}(x,y) = 0$

1.2 OpenCV

- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng về 0 (THRESH_TOZERO)**

Ví dụ

```
1 import cv2
2
3 src = cv2.imread("img_8.png")
4 img = cv2.cvtColor(src, cv2.COLOR_BGR2GRAY)
5
6 thresh = 120
7 maxValue = 255
8
9 th, dst = cv2.threshold(img, thresh, maxValue, cv2.THRESH_TOZERO)
10 cv2.imshow('THRESH_TOZERO Threshold', dst)
11 if cv2.waitKey(0) & 0xff == 27:
12     cv2.destroyAllWindows()
```



1.2 OpenCV

❖ **Ngưỡng hình ảnh trong OpenCV**

▪ **Ngưỡng đảo ngược thành 0 (THRESH_TOZERO_INV)**

- + Giá trị pixel đích được đặt thành 0, nếu giá trị pixel nguồn lớn hơn ngưỡng.
- + Nếu không, nó được đặt thành giá trị pixel nguồn

+ Bỏ qua maxValue

if $\text{src}(x,y) > \text{thresh}$

$\text{dst}(x,y) = 0$

else

$\text{dst}(x,y) = \text{src}(x,y)$

1.2 OpenCV

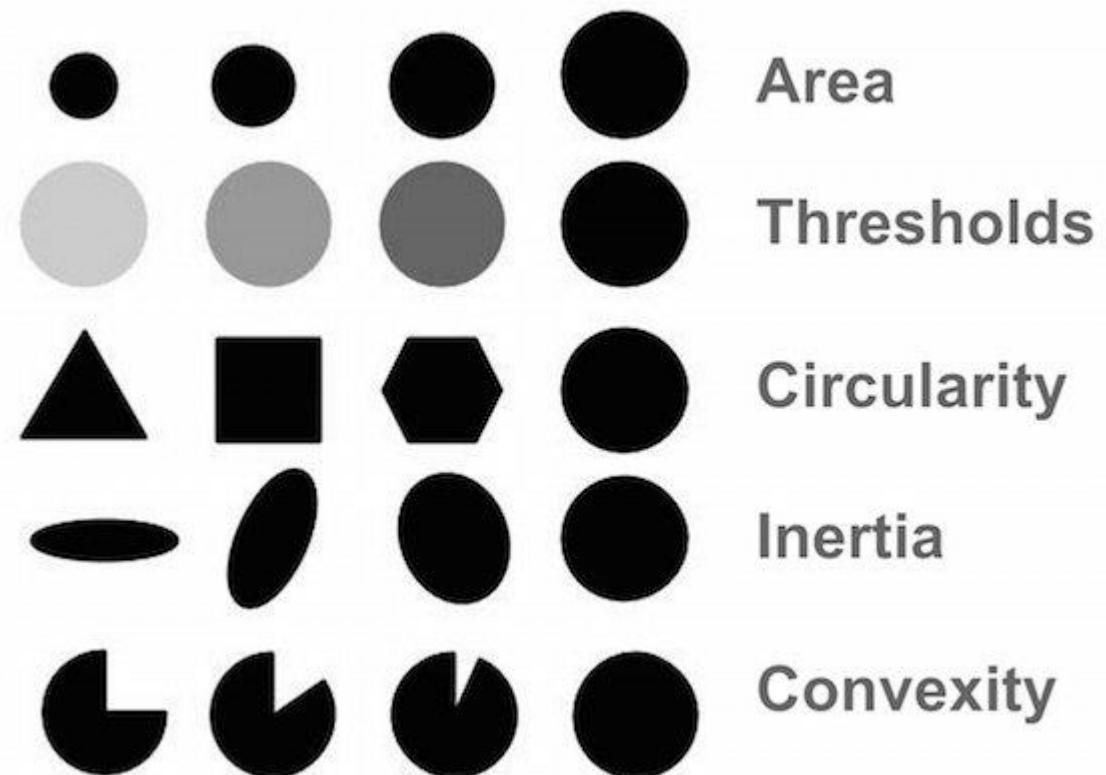
- ❖ **Ngưỡng hình ảnh trong OpenCV**
- **Ngưỡng đảo ngược thành 0 (THRESH_TOZERO_INV)**

```
1 import cv2
2
3 src = cv2.imread("img_8.png")
4 img = cv2.cvtColor(src, cv2.COLOR_RGB2GRAY)
5
6 thresh = 120
7 maxValue = 255
8
9 th, dst = cv2.threshold(img, thresh, maxValue, cv2.THRESH_TOZERO_INV)
10 cv2.imshow('THRESH_TOZERO_INV Threshold', dst)
11 if cv2.waitKey(0) & 0xff == 27:
12     cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV



1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Blob là gì?

Blob là một nhóm các pixel được kết nối trong một hình ảnh có chung một số thuộc tính (Ví dụ: giá trị thang độ xám, vv). Mục tiêu của việc phát hiện đốm màu là xác định và đánh dấu các vùng này.

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

- + Phát hiện các đốm màu hoặc vòng tròn trong một hình ảnh bằng cách sử dụng SimpleBlobDetector.
- + Nếu không phát hiện ra các đốm màu nhỏ hơn hoặc lớn hơn khu vực được chỉ định. Có thể sử dụng phương thức Params() của lớp SimpleBlobDetector để thay đổi hình dạng muốn phát hiện.

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

+ Muốn phát hiện các đốm màu hoặc vòng tròn rơi bên trong một khu vực cụ thể, phải đặt đối số filterByArea là true. Sau đó, sử dụng minArea để đặt diện tích tối thiểu của đốm màu và maxArea để đặt diện tích tối đa của đốm màu.

```
1 import cv2
2 import numpy as np
3
4 image = cv2.imread('img_9.png', cv2.IMREAD_GRAYSCALE)
5 params = cv2.SimpleBlobDetector_Params()
6
7 params.filterByArea = True
8 params.minArea = 10
9 params.maxArea = 1500
10
```

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

+ Muốn phát hiện một đốm màu bằng cách sử dụng tính tuần hoàn. Độ tuần hoàn là 1 đối với hình tròn và 0,78 đối với hình hộp.

+ Đặt đối số filterByCircularity là True. Sử dụng minCircularity và maxCircularity để đặt giá trị tuần hoàn tối thiểu và tối đa cho hình dạng.

```
11     params.filterByCircularity = True  
12     params.minCircularity = 0.001
```

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

+ Muốn phát hiện các đốm màu bằng cách sử dụng độ lồi. Đầu tiên, đặt filterByConvexity thành true, sau đó sử dụng minConvexity và maxConvexity để đặt giá trị nhỏ nhất và lớn nhất của độ lồi.

```
14     params.filterByConvexity = True  
15     params.minConvexity = 0.01
```

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

- + Muốn phát hiện các đốm màu bằng cách sử dụng quán tính. Đầu tiên, đặt filterInertia thành true, sau đó sử dụng minInertiaRatio và maxInertiaRatio để đặt giá trị nhỏ nhất và lớn nhất của độ lồi.
- + Đối với một đường tròn hoàn hảo là 1 và đối với hình elip nằm trong khoảng từ 0 đến 1.

```
17 | params.filterByInertia = True  
18 | params.minInertiaRatio = 0.01  
19 |
```

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

+ Lọc các đốm màu theo màu sắc bằng cách đặt filterByColor với số là true. Sau đó, sử dụng minThreshold và maxThreshold để đặt các ngưỡng tối thiểu và tối đa của màu muốn phát hiện.

1.2 OpenCV

❖ Phát hiện Blob bằng OpenCV

▪ Lọc các đốm màu theo diện tích, hình dạng, tỷ lệ quán tính

+ Ví dụ

```
1 import cv2
2 import numpy as np
3
4 image = cv2.imread('img_9.png', cv2.IMREAD_GRAYSCALE)
5 params = cv2.SimpleBlobDetector_Params()
6
7 params.filterByArea = True
8 params.minArea = 10
9 params.maxArea = 1500
10
11 params.filterByCircularity = True
12 params.minCircularity = 0.001
13
14 params.filterByConvexity = True
15 params.minConvexity = 0.01
16
17 params.filterByInertia = True
18 params.minInertiaRatio = 0.01
19 #tạo đối tượng để phát hiện đốm màu
```

```
19 #tạo đối tượng để phát hiện đốm màu
20 detector = cv2.SimpleBlobDetector_create(params)
21 #tim các điểm chính để phát hiện đốm màu
22 keypoints = detector.detect(image)
23 #tao mask
24 blank = np.zeros((1, 1))
25 # vẽ các hình tròn màu đỏ trên các điểm chính được phát hiện
26 blobs = cv2.drawKeypoints(image, keypoints, blank, (0, 0, 255),cv2.DRAW_MATCHES_FLAGS_DEFAULT)
27
28 number_of_blobs = len(keypoints)
29 text = "Circular Blobs: " + str(len(keypoints))
30 cv2.putText(blobs, text, (10, 100),cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2)
31 |
32 cv2.imshow("Original Image",image)
33 cv2.imshow("Circular Blobs Only", blobs)
34 cv2.waitKey(0)
35 cv2.destroyAllWindows()
```

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

Phát hiện cạnh là một kỹ thuật xử lý hình ảnh, được sử dụng để xác định ranh giới (cạnh) của các đối tượng hoặc các vùng trong một hình ảnh. Các cạnh là một trong những tính năng quan trọng nhất liên quan đến hình ảnh.



1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Các cạnh được phát hiện như thế nào?

Các cạnh được đặc trưng bởi sự thay đổi đột ngột của cường độ pixel. Để phát hiện các cạnh, cần phải tìm kiếm những thay đổi như vậy trong các pixel lân cận. Sử dụng hai thuật toán phát hiện cạnh quan trọng có sẵn trong OpenCV: Sobel Edge Detection và Canny Edge Detection.

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Sobel

Hạt nhân được sử dụng cho Sobel Edge Detection:

$$\begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}$$

Hạt nhân X-Direction

$$\begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Y-Direction Kernel

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Sobel

- + Sử dụng nhân dọc, tích chập cho ra hình ảnh Sobel, với các cạnh được nâng cao theo hướng X
- + Sử dụng nhân ngang tạo ra hình ảnh Sobel, với các cạnh được nâng cao theo hướng Y.

Cú pháp:

Sobel(src, ddepth, dx, dy)

+ ddepth chỉ định độ chính xác của hình ảnh đầu ra

+ dx và dy chỉ định thứ tự của đạo hàm theo mỗi hướng. Ví dụ:

Nếu $dx=1$ và $dy=0$, chúng tôi tính hình ảnh Sobel đạo hàm cấp 1 theo hướng x.

Nếu cả hai $dx=1$ và $dy=1$, chúng tôi tính toán hình ảnh Sobel đạo hàm 1 theo cả hai hướng

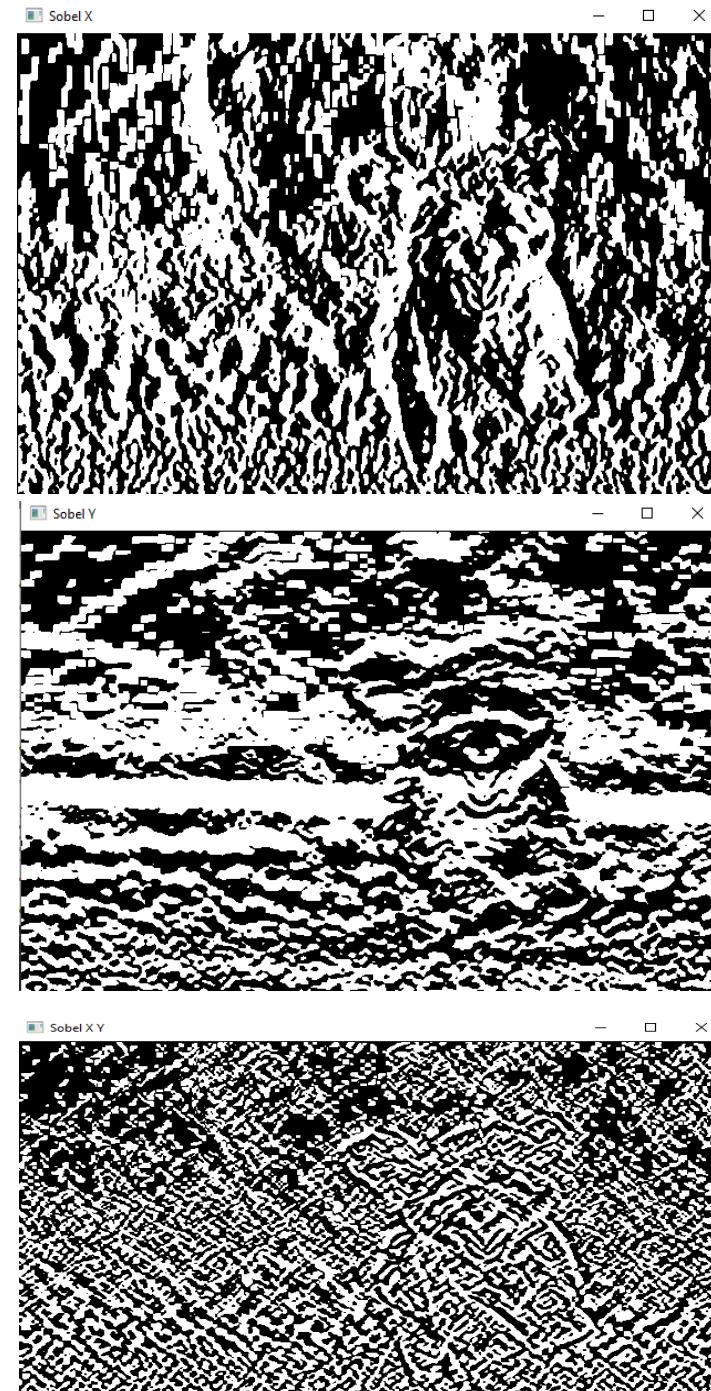
1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Sobel

+ Ví dụ

```
1 import cv2
2
3 img = cv2.imread('img_6.png')
4 cv2.imshow('Image', img)
5 cv2.waitKey(0)
6
7 # chuyển sang ảnh xám
8 img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
9 # lọc nhiễu
10 img.blur = cv2.GaussianBlur(img_gray, (7,7), 0)
11
12 # Phát hiện cạnh theo Sobel
13 sobelx = cv2.Sobel(src=img.blur, ddepth=cv2.CV_64F, dx=1, dy=0, ksize=7)
14 sobely = cv2.Sobel(src=img.blur, ddepth=cv2.CV_64F, dx=0, dy=1, ksize=7)
15 sobelxy = cv2.Sobel(src=img.blur, ddepth=cv2.CV_64F, dx=1, dy=1, ksize=7)
16 # hiển thị
17 cv2.imshow('Sobel X', sobelx)
18 cv2.waitKey(0)
19 cv2.imshow('Sobel Y', sobely)
20 cv2.waitKey(0)
21 cv2.imshow('Sobel X Y ', sobelxy)
22 cv2.waitKey(0)
```



1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Canny Edge Detection là một trong những phương pháp phát hiện cạnh phổ biến nhất được sử dụng hiện nay vì nó rất mạnh mẽ và linh hoạt.Thêm vào đó là tính năng làm mờ hình ảnh, một bước xử lý trước cần thiết để giảm nhiễu.

+ Quá trình gồm bốn giai đoạn, bao gồm:

- Giảm nhiễu
- Tính toán cường độ gradient của bức ảnh
- Ngăn chặn các cạnh sai
- Ngưỡng độ trễ

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Canny Edge Detection là một trong những phương pháp phát hiện cạnh phổ biến nhất được sử dụng hiện nay vì nó rất mạnh mẽ và linh hoạt.Thêm vào đó là tính năng làm mờ hình ảnh, một bước xử lý trước cần thiết để giảm nhiễu.

+ Quá trình gồm bốn giai đoạn, bao gồm:

- Giảm nhiễu
- Tính toán cường độ gradient của bức ảnh
- Ngăn chặn các cạnh sai
- Ngưỡng độ trễ

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Giảm nhiễu: Các pixel ảnh thô thường có thể dẫn đến các cạnh bị nhiễu, vì vậy điều quan trọng là phải giảm nhiễu trước khi tính toán các cạnh Trong Canny Edge Detection, bộ lọc làm mờ Gaussian được sử dụng để loại bỏ hoặc giảm thiểu các chi tiết không cần thiết có thể dẫn đến các cạnh không mong muốn.

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Tính toán cường độ gradient của bức ảnh: Ảnh được làm mịn sau đó được lọc qua bộ lọc Sobel theo cả 2 chiều dọc và ngang để thu được đạo hàm bậc 1 theo 2 phương x và y lần lượt là G_x và G_y . Từ những hình ảnh này có thể tìm được độ dài cạnh gradient và phương cho mỗi pixel. Phương gradient luôn luôn vuông góc với các cạnh, được làm tròn thành một trong bốn góc biểu diễn trực dọc, ngang và 2 phương của đường chéo.

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

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

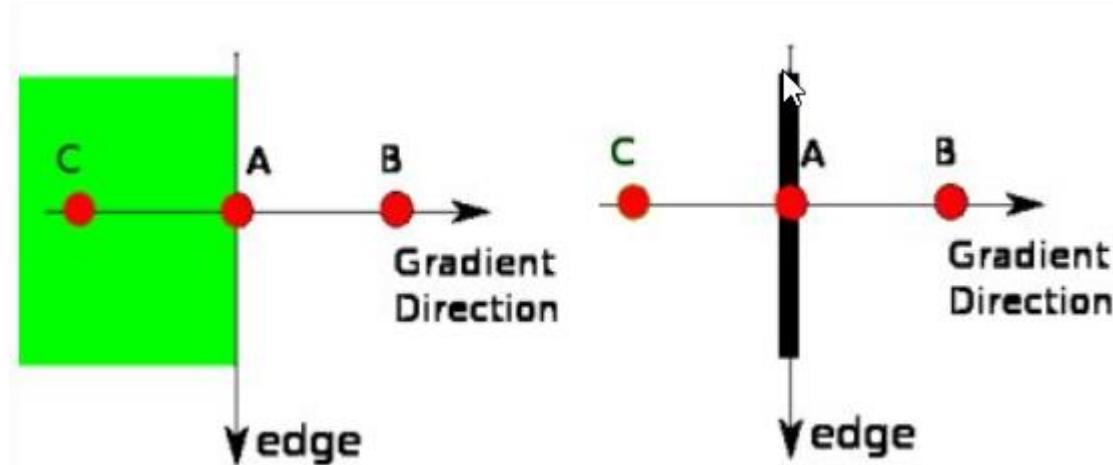
- + Ngăn chặn các cạnh sai: là một thuật toán loại bỏ nếu như không phải là giá trị lớn nhất.
- + Sau khi nhận được độ lớn của gradient và phương, một lượt quét được thực hiện trên các bức ảnh để loại bỏ bất kì pixels nào không tạo thành cạnh. Để thực hiện điều đó, tại mỗi pixel, pixel được kiểm tra xem liệu nó có là một cực đại cục bộ trong số các lân cận của nó theo phương của gradient.

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Ngăn chặn các cạnh sai: Điểm A nằm trên cạnh (theo chiều dọc). Phương gradient là norm chuẩn của cạnh. Điểm B và C là điểm nằm trên phương gradient. Nếu điểm A được kiểm tra với điểm B và C để xem liệu nó có là một cực đại cục bộ. Như vậy, được xem xét giữ lại cho bước tiếp theo, trái lại thì nó sẽ bị triệt tiêu (bằng cách thiết lập bằng 0).



1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

- + Ngưỡng độ trễ: Bước này sẽ quyết định xem toàn bộ các cạnh có những cạnh nào là thực sự và những cạnh nào không thông qua việc thiết lập ngưỡng gồm 2 giá trị, minVal và maxVal.
- + Một cạnh bất kì có cường độ gradient lớn hơn maxVal thì chắc chắn là các cạnh và những cạnh có cường độ gradient nhỏ hơn sẽ không được coi là cạnh. Nếu một cạnh mà có những điểm ảnh nằm trong ngưỡng này thì có thể được xem xét thuộc cùng một cạnh hoặc không thuộc dựa trên sự kết nối của các điểm với nhau.

1.2 OpenCV

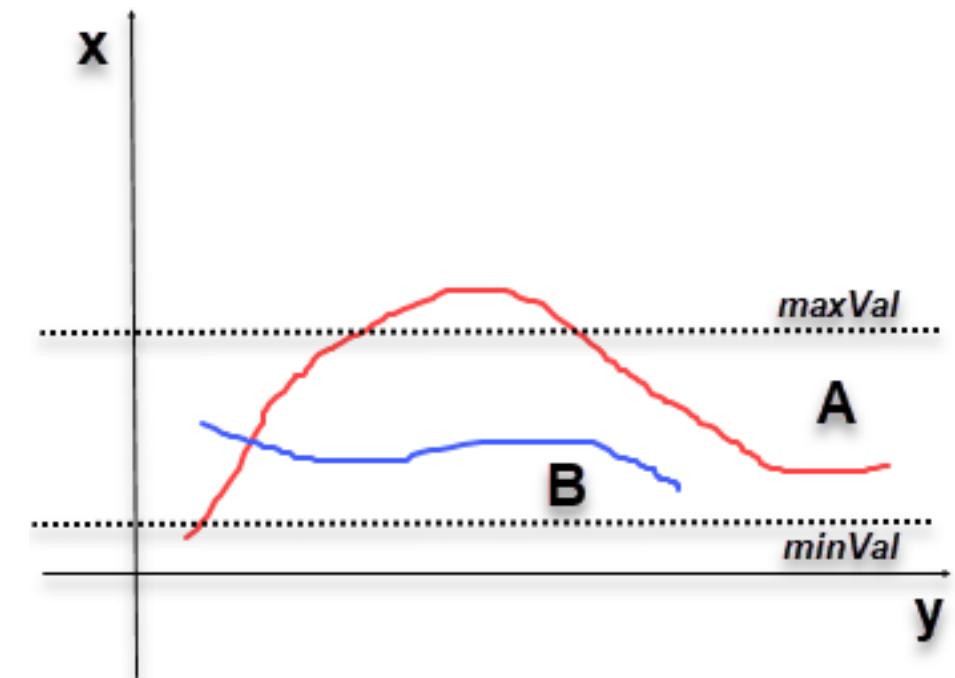
❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Nguồn độ trễ

Cạnh A màu đỏ và B màu xanh. Cạnh A do có các điểm có cường độ gradient nằm trên maxVal nên được xem là những cạnh đạt tiêu chuẩn. Mặc dù trên cạnh A có những điểm nằm trong ngưỡng từ minVal tới maxVal . Cạnh B được xem là không đạt tiêu chuẩn vì toàn bộ các điểm nằm trên cạnh B đều nằm trong ngưỡng minVal và maxVal .

Như vậy cạnh A sẽ được giữ lại và cạnh B sẽ được xóa bỏ.



1.2 OpenCV

- ❖ **Phát hiện cạnh bằng OpenCV**

- **Phát hiện cạnh Canny**

- + Cú pháp để áp dụng tính năng phát hiện cạnh Canny bằng OpenCV:

`cv2.canny(image, threshold1, threshold2)`

Trong đó:

Image: là hình ảnh đầu vào

threshold1, threshold2: lần lượt là ngưỡng minVal và maxVal.

1.2 OpenCV

❖ Phát hiện cạnh bằng OpenCV

▪ Phát hiện cạnh Canny

+ Ví dụ:

```
1 import cv2
2
3 img = cv2.imread('img_6.png')
4
5 cv2.imshow('Original', img)
6 cv2.waitKey(0)
7
8 img_gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
9
10 img_blur = cv2.GaussianBlur(img_gray, (5,5), 0)
11 # Canny Edge Detection
12 edges = cv2.Canny(image=img_blur, threshold1=100, threshold2=200)
13
14 cv2.imshow('Canny Edge Detection', edges)
15 cv2.waitKey(0)
16 cv2.destroyAllWindows()
```



1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

Contour là một đường cong liên kết toàn bộ các điểm liên tục (đọc theo đường biên) mà có cùng màu sắc hoặc giá trị cường độ. Contour rất hữu ích trong phân tích hình dạng, phát hiện vật thể và nhận diện vật thể. Một số lưu ý khi sử dụng contour.

- + Để độ chính xác cao hơn thì nên sử dụng hình ảnh nhị phân (chỉ gồm 2 màu đen và trắng). Do đó trước khi phát hiện contour thì nên áp dụng threshold hoặc thuật toán canny để chuyển sang ảnh nhị phân.
- + Trong openCV, tìm các contours như là tìm các vật thể màu trắng từ nền màu đen. Do đó object cần tìm nên là màu trắng và background nên là màu đen.

1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Ứng dụng đường viền

+ Phát hiện chuyển động

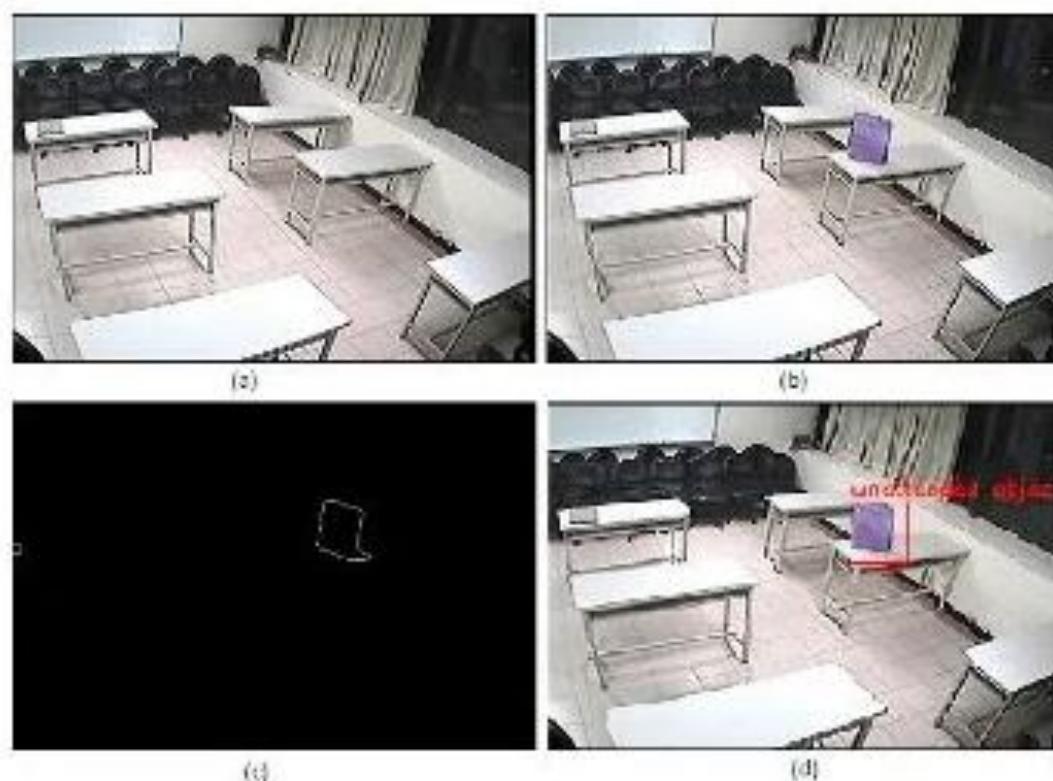


1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Ứng dụng đường viền

+ Phát hiện đối tượng



1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Ứng dụng đường viền

+ Phân đoạn nền/tiền cảnh



1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Đường viền là gì?

- + Khi nối tất cả các điểm trên ranh giới của một đối tượng sẽ thu được một đường bao. Thông thường, một đường bao cụ thể đề cập đến các pixel ranh giới có cùng màu sắc và cường độ.

OpenCV cung cấp hai hàm đơn giản:

+ findContours()

+ drawContours()

Ngoài ra, nó có hai thuật toán khác nhau để phát hiện đường viền:

CHAIN_APPROX_SIMPLE

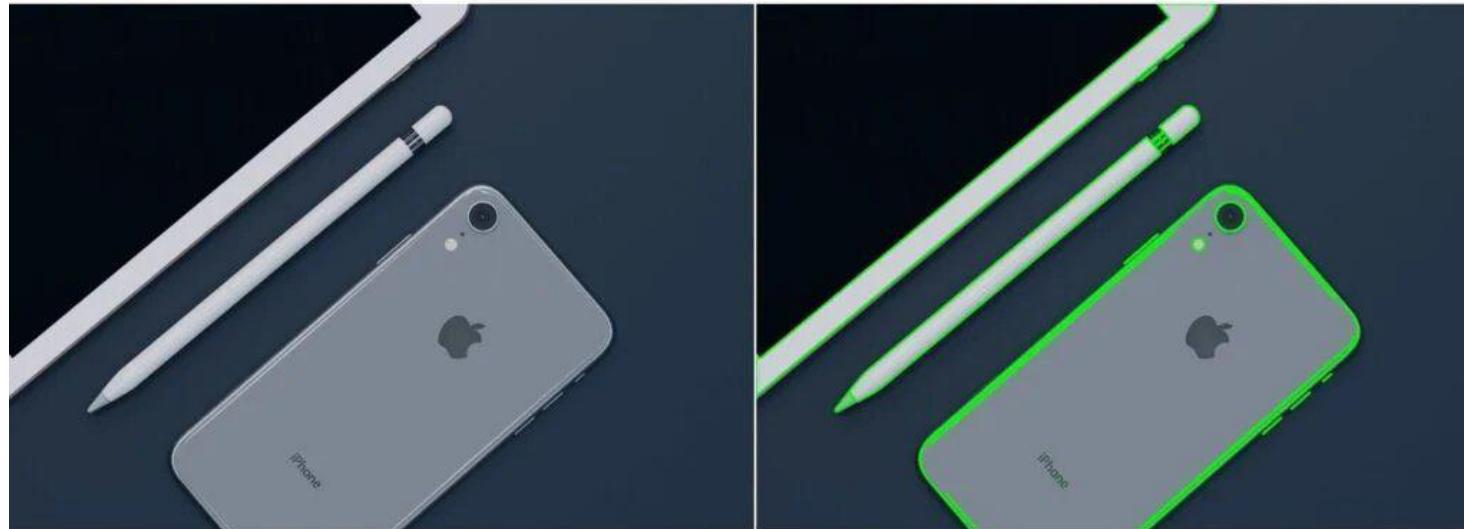
CHAIN_APPROX_NONE

1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Đường viền là gì?

+ Ví dụ



1.2 OpenCV

- ❖ **Phát hiện đường viền (Contour) bằng OpenCV**
 - **Các bước phát hiện và vẽ đường viền trong OpenCV**

1. Đọc hình ảnh và chuyển đổi sang định dạng thang độ xám

Chuyển đổi hình ảnh sang thang độ xám là rất quan trọng vì nó chuẩn bị hình ảnh cho bước tiếp theo. Việc chuyển đổi hình ảnh sang hình ảnh thang độ xám kênh đơn là rất quan trọng đối với việc tạo ngưỡng

1.2 OpenCV

❖ Phát hiện đường viền (Contour) bằng OpenCV

▪ Các bước phát hiện và vẽ đường viền trong OpenCV

2. Áp dụng ngưỡng nhị phân

+ Trước tiên luôn áp dụng ngưỡng nhị phân hoặc phát hiện cạnh Canny cho hình ảnh thang độ xám. Điều này sẽ chuyển đổi hình ảnh sang màu đen và trắng, làm nổi bật các đối tượng quan tâm để làm cho mọi thứ trở nên dễ dàng cho thuật toán phát hiện đường viền.

+ Thresholding sẽ biến đường viền của đối tượng trong ảnh thành màu trắng hoàn toàn, với tất cả các pixel có cùng cường độ. Thuật toán hiện có thể phát hiện đường viền của các đối tượng từ các pixel màu trắng này.

Lưu ý : Các pixel màu đen, có giá trị 0, được coi là pixel nền và bị bỏ qua.

1.2 OpenCV

- ❖ **Phát hiện đường viền (Contour) bằng OpenCV**
 - **Các bước phát hiện và vẽ đường viền trong OpenCV**

3. Tìm các đường viền

Sử dụng hàm `findContours()` để phát hiện các đường viền trong hình ảnh.

Cú pháp:

`cv2.findContours(image, mode, method)`

Trong đó:

`image`: Hình ảnh đầu vào nhị phân thu được ở bước trước.

1.2 OpenCV

- ❖ Phát hiện đường viền (Contour) bằng OpenCV
 - Các bước phát hiện và vẽ đường viền trong OpenCV

3. Tìm các đường viền

Trong đó:

mode: Đây là chế độ truy xuất đường viền. RETR_TREE truy xuất tất cả các đường bao có thể có từ hình ảnh nhị phân. cv2.RETR_LIST truy xuất tất cả các đường bao mà không cần thiết lập bất kỳ mối quan hệ phân cấp nào. RETR_EXTERNAL, RETR_CCOMP.

method: CHAIN_APPROX_NONE xác định phương pháp xấp xỉ đường viền. CHAIN_APPROX_SIMPLE, phương pháp lưu trữ TẤT CẢ các điểm đường viền.

mode đề cập đến loại đường bao sẽ được truy xuất, đồng thời method đề cập đến điểm nào trong đường bao được lưu trữ.

1.2 OpenCV

- ❖ Phát hiện đường viền (Contour) bằng OpenCV
 - Các bước phát hiện và vẽ đường viền trong OpenCV

4. Vẽ đường viền trên hình ảnh gốc.

Sử dụng hàm drawContours() để phủ các đường viền lên hình ảnh ban đầu.

Cú pháp:

cv.drawContours(image, contours, contourIdx, color, thickness, lineType)

Trong đó:

image: Hình ảnh đầu vào

1.2 OpenCV

- ❖ Phát hiện đường viền (Contour) bằng OpenCV
 - Các bước phát hiện và vẽ đường viền trong OpenCV

4. Vẽ đường viền trên hình ảnh gốc.

Trong đó:

contours: Cho biết giá trị **contours** thu được từ hàm **findContours()**

contourIdx: Tham số chỉ đường bao cần vẽ. Nếu nó là âm, tất cả các đường viền sẽ được vẽ.

color: Màu sắc của các đường viền.

thickness: độ dày của các điểm đường viền.

lineType: Loại đường (cv2.LINE_4, cv2.LINE_8, cv.LINE_AA)

1.2 OpenCV

- ❖ Phát hiện đường viền (Contour) bằng OpenCV
 - Các bước phát hiện và vẽ đường viền trong OpenCV

Ví dụ:

```
1 import cv2
2 # Bước 1
3 image = cv2.imread('img_12.png')
4 img_gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
5 # bước 2
6 ret, thresh = cv2.threshold(img_gray, 150, 255, cv2.THRESH_BINARY)
7 # bước 3
8 contours, hierarchy= cv2.findContours(image=thresh, mode=cv2.RETR_TREE,
9                                     method=cv2.CHAIN_APPROX_NONE)
10 # Bước 4
11 image_copy = image.copy()
12 cv2.drawContours(image=image_copy, contours=contours, contourIdx=-1,
13                  color=(0, 255, 0), thickness=2,
14                  lineType=cv2.LINE_4)
15 # hiển thị
16 cv2.imshow('Image Contour', image_copy)
17 cv2.waitKey(0)
18 cv2.destroyAllWindows()
```



1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
- **Tại sao chọn Mô-đun OpenCV DNN?**

Mô-đun OpenCV DNN chỉ hỗ trợ suy luận học sâu trên hình ảnh và video. **Nó không hỗ trợ tinh chỉnh và đào tạo.**

1.2 OpenCV

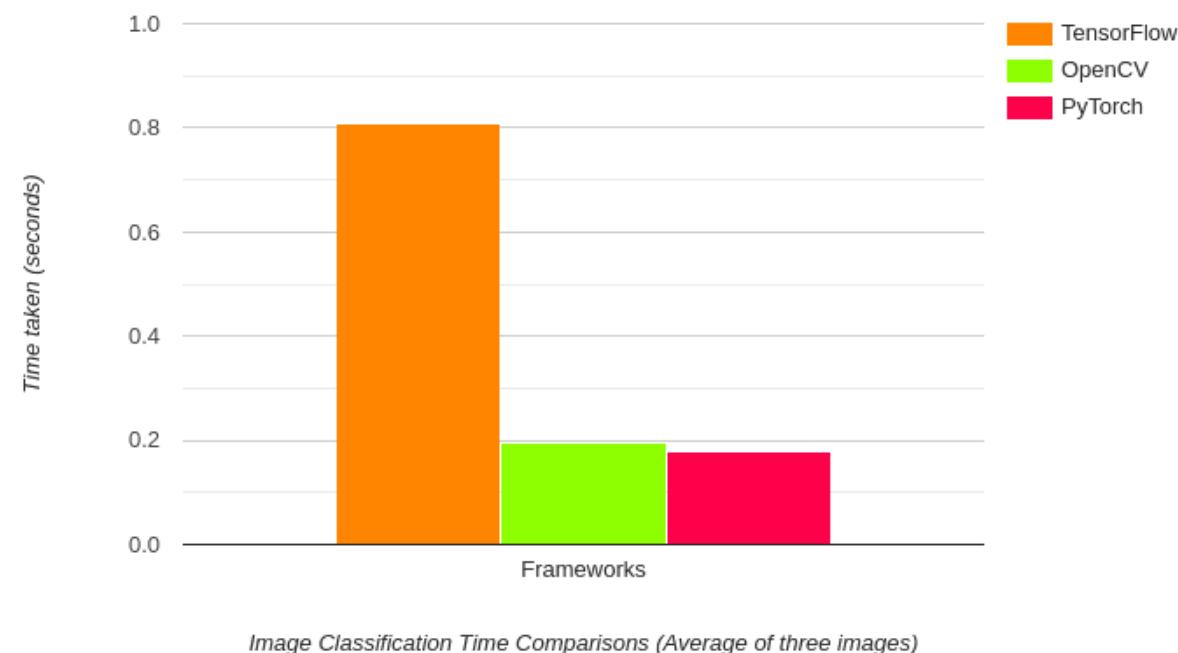
- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
- **Tại sao chọn Mô-đun OpenCV DNN?**

Một trong những điểm tốt nhất của mô-đun OpenCV DNN là nó được tối ưu hóa cao cho bộ vi xử lý Intel. Chúng có thể đạt FPS (số khung hình trên giây) tốt khi chạy suy luận trên video thời gian thực cho các ứng dụng phát hiện đối tượng và phân đoạn hình ảnh.

1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
- **Tại sao chọn Mô-đun OpenCV DNN?**

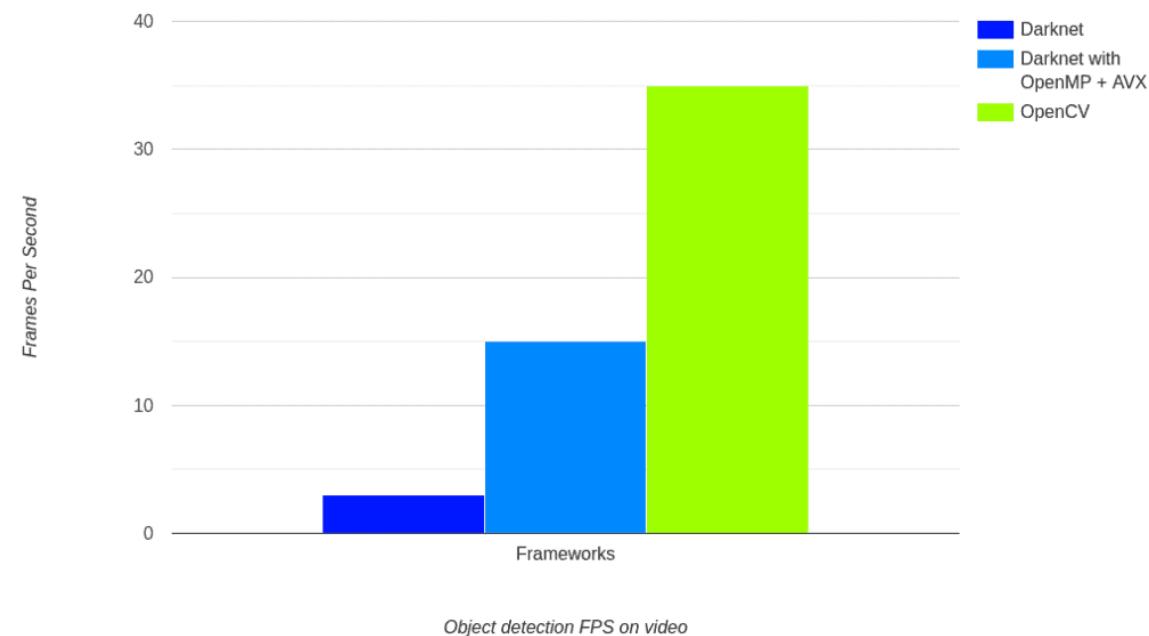
So sánh tốc độ suy luận phân loại hình ảnh trên CPU cho các framework khác nhau.



1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
- **Tại sao chọn Mô-đun OpenCV DNN?**

Trường hợp phát hiện đối tượng.



1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
 - **Các chức năng học sâu khác nhau mà mô-đun DNN hỗ trợ**
 - + Phân loại hình ảnh.
 - + Phát hiện đối tượng.
 - + Phân đoạn hình ảnh.
 - + Phát hiện và nhận dạng văn bản.
 - + Uớc tính tư thế.
 - + Uớc tính độ sâu.
 - + Xác minh và phát hiện người và khuôn mặt.

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**

▪ **Các mô hình khác nhau mô-đun DNN hỗ trợ**

Phân loại hình ảnh	Phát hiện đối tượng	Phân đoạn hình ảnh	Phát hiện và nhận dạng văn bản	Ước tính tư thế người	Phát hiện người và khuôn mặt
Alexnet	MobileNet SSD	DeepLab	Easy OCR	Open Pose	Open Face
GoogLeNet	VGG SSD	UNet	CRNN	Alpha Pose	Torchreid
VGG	Faster R-CNN	FCN			Mobile FaceNet
ResNet	EfficientDet				OpenCV FaceDetector
SqueezeNet					
DenseNet					
ShuffleNet					
EfficientNet					

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**

▪ **Các Framework khác nhau hỗ trợ mô-đun DNN**

+ Caffe: Để sử dụng mô hình Caffe được đào tạo trước với DNN, cần chuẩn bị. Một là tệp model.caffemodel chứa các trọng số được đào tạo trước. Hai là tệp kiến trúc mô hình có phần mở rộng .prototxt.

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**

▪ **Các Framework khác nhau hỗ trợ mô-đun DNN**

+ TensorFlow: Tệp trọng số mô hình và tệp văn bản protobuf chứa cấu hình mô hình. Tệp trọng lượng có phần mở rộng .pb là tệp protobuf chứa tất cả các trọng lượng được đào tạo trước. Tệp .pb là điểm kiểm tra mô hình nhận được sau khi lưu mô hình và đóng bằng các trọng số. Cấu hình mô hình được giữ trong tệp văn bản protobuf, có phần mở rộng là tệp .pbtxt.

1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
 - **Các Framework khác nhau hỗ trợ mô-đun DNN**
 - + PyTorch: Cần tệp chứa các trọng số được đào tạo trước, tệp này có phần mở rộng .t7 hoặc .net.
.pth

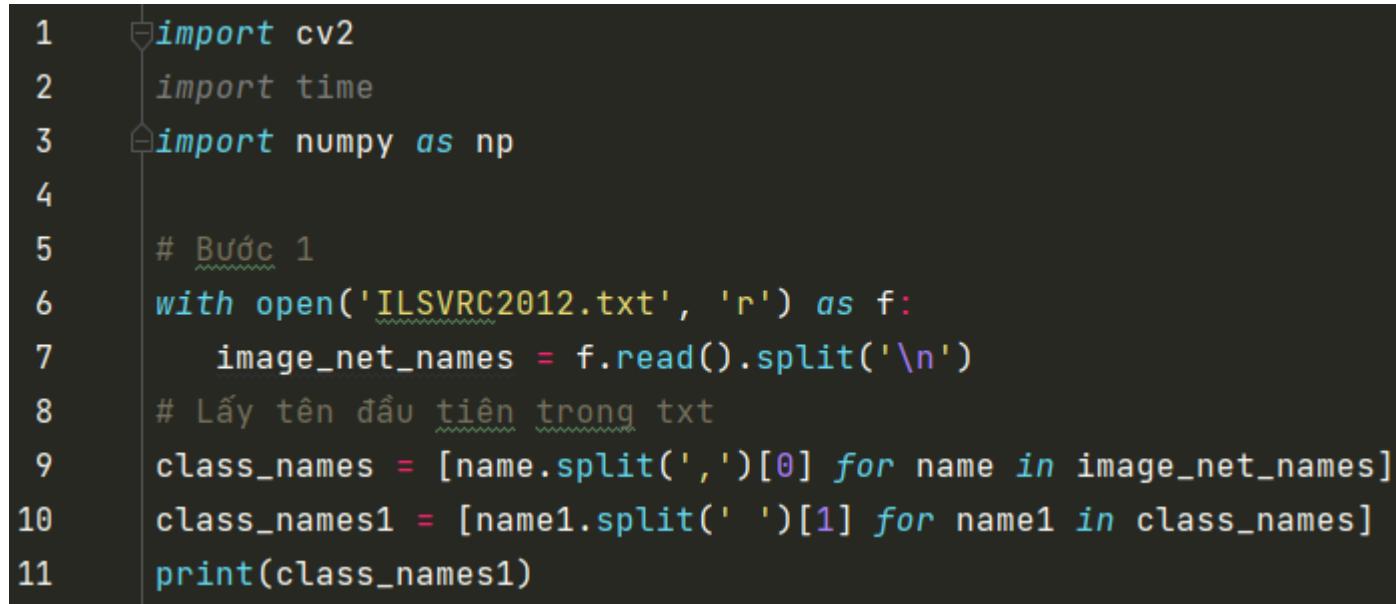
1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
 - **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**
 - + Các bước khi phân loại hình ảnh.
 - Tải tệp văn bản tên lớp và trích xuất các nhãn cần thiết.
 - Tải mô hình mạng nơ-ron được đào tạo
 - Tải hình ảnh và chuẩn bị hình ảnh ở định dạng đầu vào chính xác cho mô hình học sâu.
 - Chuyển tiếp truyền hình ảnh đầu vào qua mô hình và thu được kết quả đầu ra.

1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks) của OpenCV**
- **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**
 - Tải tệp văn bản tên lớp và trích xuất các nhãn cần thiết.

```
1 import cv2
2
3 import time
4
5 import numpy as np
6
7
8
9
10
11
```



The screenshot shows a dark-themed code editor with white text. It displays a Python script with line numbers 1 through 11 on the left. The script imports cv2, time, and numpy. It then reads a file named 'ILSVRC2012.txt' and splits its contents into a list of names. These names are then split by commas and spaces to extract the class names. Finally, it prints the resulting list. The code is annotated with comments in Vietnamese explaining the steps: '# Bước 1' (Step 1), '# Lấy tên đầu tiên trong txt' (Extract the first name in the txt), and 'print(class_names1)' (Print the class names1).

```
# Bước 1
with open('ILSVRC2012.txt', 'r') as f:
    image_net_names = f.read().split('\n')
# Lấy tên đầu tiên trong txt
class_names = [name.split(',')[0] for name in image_net_names]
class_names1 = [name1.split(' ')[1] for name1 in class_names]
print(class_names1)
```

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

- Tải mô hình mạng nơ-ron được đào tạo

Sử dụng hàm readNet()

Cú pháp:

`cv2.dnn.readNet(model, config, framework)`

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

- Tải mô hình mạng nơ-ron được đào tạo

Trong đó:

model: Tệp chứa các trọng số đã được huấn luyện

*.caffemodel (Caffe, <http://caffe.berkeleyvision.org/>)

*.pb (TensorFlow, <https://www.tensorflow.org/>)

*.t7 or *.net (Torch, <http://torch.ch/>)

*.weights (Darknet, <https://pjreddie.com/darknet/>)

*.bin (DLDT, <https://software.intel.com/openvino-toolkit>)

*.onnx (ONNX, <https://onnx.ai/>)

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

- Tải mô hình mạng nơ-ron được đào tạo

Trong đó:

config: Tệp văn bản chứa cấu hình

*.prototxt (Caffe, <http://caffe.berkeleyvision.org/>)

*.pbtxt (TensorFlow, <https://www.tensorflow.org/>)

*.cfg (Darknet, <https://pjreddie.com/darknet/>)

*.xml (DLDT, <https://software.intel.com/openvino-toolkit>)

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DenseNet121 cho nhiệm vụ phân loại**

- Tải mô hình mạng nơ-ron được đào tạo

Trong đó:

framework: cung cấp các framework

Ví dụ

```
12      # bước 2
13      model = cv2.dnn.readNet(model='DenseNet_121.caffemodel',
14                                config='DenseNet_121.prototxt', framework='Caffe')
15
```

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

- Tải hình ảnh và chuẩn bị hình ảnh ở định dạng đầu vào chính xác cho mô hình học sâu.

Sử dụng hàm imread() và hàm blobFromImage()

Cú pháp:

`cv2.dnn.blobFromImage(image, scalefactor, size, mean)`

image: là hình ảnh đầu vào

scalefactor: hệ số nhân cho các giá trị hình ảnh

size: kích thước mà hình ảnh sẽ được thay đổi kích thước

mean: giá trị trung bình được trừ khỏi các kênh màu RGB của hình ảnh

1.2 OpenCV

❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**

▪ **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

- Tải hình ảnh và chuẩn bị hình ảnh ở định dạng đầu vào chính xác cho mô hình học sâu.

Ví dụ:

```
16     img = cv2.imread('img_15.png')
17     blob = cv2.dnn.blobFromImage(image=img, scalefactor=0.01,
18                                 size=(224, 224), mean=(104, 117, 123))
```

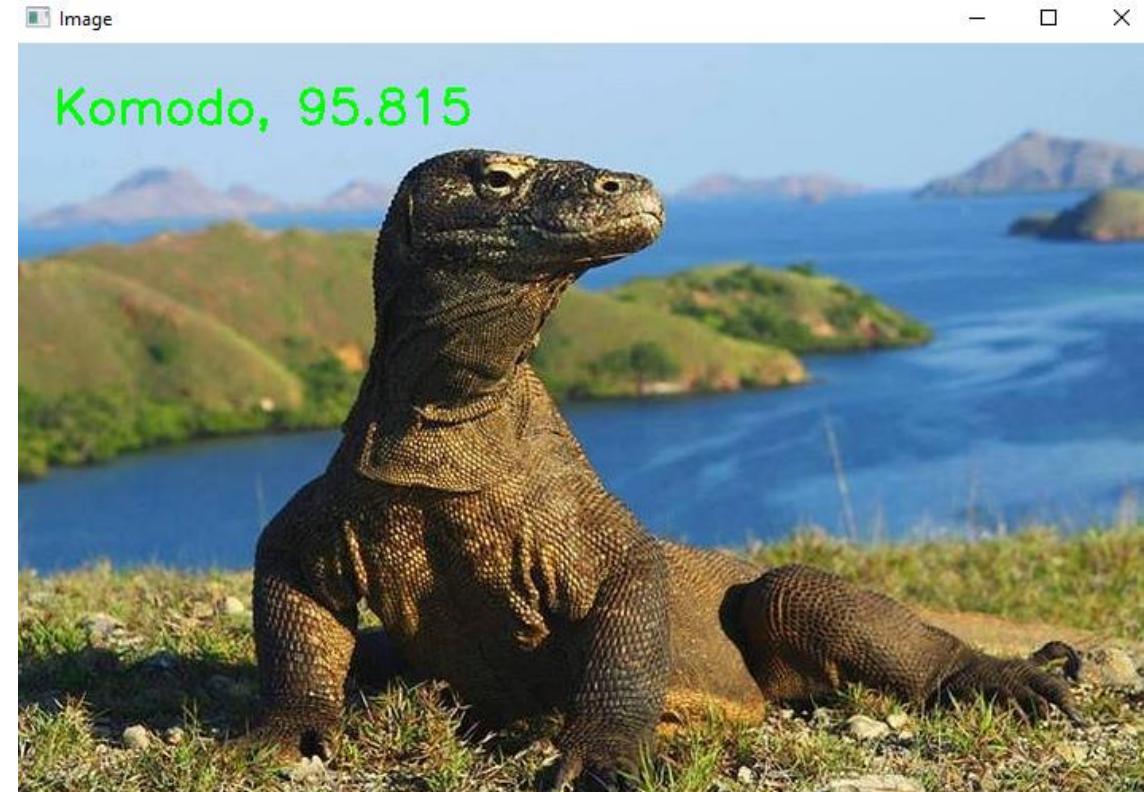
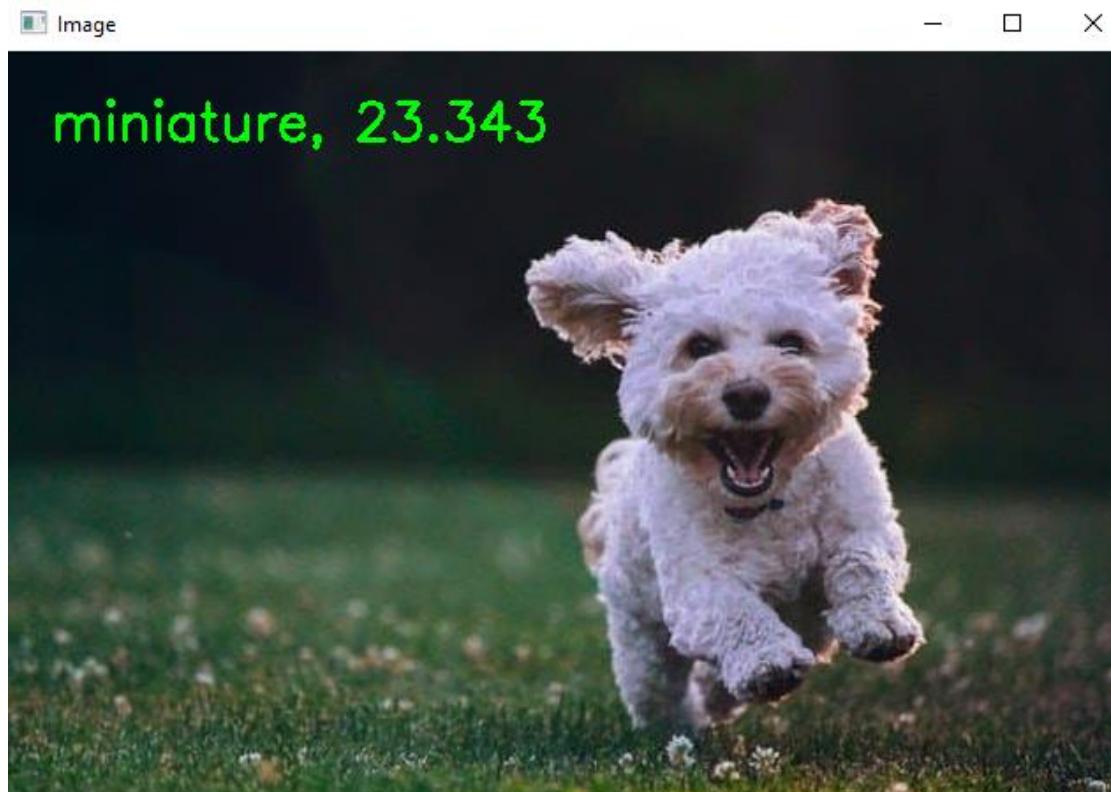
1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**
- **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**

```
25 # chuyển đầu ra về mảng 1D
26 final_outputs = final_outputs.reshape(1000, 1)
27 # Lấy nhãn lớp
28 label_id = np.argmax(final_outputs) # lấy index có giá trị lớp nhất
29 print(label_id)
30 # xác suất mà mô hình dự đoán
31 probs = np.exp(final_outputs) / np.sum(np.exp(final_outputs))
32 # nhân điểm xác suất cao nhất với 100 để có được tỷ lệ phần trăm dự đoán
33 final_prob = np.max(probs) * 100.
34
35 # ánh xạ độ tin cậy tối đa cho các tên nhãn lớp
36 out_name = class_names1[label_id]
37 out_text = f'{out_name}, {final_prob:.3f}'
38 # đặt văn bản tên lớp lên hình ảnh
39 cv2.putText(img, out_text, (25, 50), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
40 cv2.imshow('Image', img)
41 cv2.waitKey(0)
42 |
```

1.2 OpenCV

- ❖ **Ứng dụng Phân loại Hình ảnh sử dụng Mô-đun DNN (Deep Neural Networks)**
 - **Sử dụng mô hình mạng nơ-ron sâu DensNet121 cho nhiệm vụ phân loại**



1.2 OpenCV

❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**

Các bước khi phân loại hình ảnh.

- Tải tệp văn bản tên lớp và trích xuất các nhãn cần thiết.
- Tải mô hình mạng nơ-ron được đào tạo
- Tải hình ảnh và chuẩn bị hình ảnh ở định dạng đầu vào chính xác cho mô hình học sâu.
- Chuyển tiếp truyền hình ảnh đầu vào qua mô hình và thu được kết quả đầu ra.

1.2 OpenCV

❖ Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)

- Tải tệp văn bản tên lớp và trích xuất các nhãn cần thiết.

```
1 import cv2
2 import numpy as np
3
4 # Bước 1
5 with open('object_detection_classes_coco.txt', 'r') as f:
6     class_names = f.read().split('\n')
7
8 # COLORS chứa các bộ giá trị của ba số nguyên.
9 # Đây là những màu ngẫu nhiên áp dụng trong khi vẽ hộp giới hạn cho mỗi lớp.
10 COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))
11
```

1.2 OpenCV

❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**

- Tải mô hình mạng nơ-ron được đào tạo

```
12     # Bước 2
13     model = cv2.dnn.readNet(model='frozen_inference_graph.pb',
14                               config='ssd_mobilenet_v2_coco_2018_03_29.pbtxt',
15                               framework='TensorFlow')
```

1.2 OpenCV

❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**

- Tải hình ảnh và chuẩn bị hình ảnh ở định dạng đầu vào chính xác cho mô hình học sâu.

```
16 # Bước 3
17 image = cv2.imread('img_6.png')
18 image_height, image_width, _ = image.shape
19
20 blob = cv2.dnn.blobFromImage(image=image, size=(300, 300),
21                               mean=(104, 117, 123), swapRB=True)
22 # Bước 4
```

1.2 OpenCV

❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**

- Chuyển tiếp truyền hình ảnh đầu vào qua mô hình và thu được kết quả đầu ra.

```
22     # Bước 4  
23     model.setInput(blob)  
24     output = model.forward()  
25     print(output)  
26     |
```

1.2 OpenCV

❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**

Output có dạng như sau:

```
[ 0.0000000e+00  8.8000000e+01  1.37889879e-02  6.08750463e-01
 3.50991845e-01  7.90041685e-01  6.80126786e-01]
```

- + vị trí chỉ mục 1 chứa lớp nhãn
- + Vị trí chỉ số 2 chứa điểm tin cậy. Đây không phải là điểm xác suất mà là độ tin cậy của mô hình đối với đối tượng thuộc lớp mà nó đã phát hiện.
- + Bốn giá trị cuối cùng, hai giá trị đầu tiên là tọa độ hộp giới hạn x, y và hai giá trị cuối cùng là chiều rộng và chiều cao của hộp giới hạn.

1.2 OpenCV

❖ Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)

```
26     # Lắp lại tống phát hiện
27     for detection in output[0, 0, :, :]:
28         # lấy độ tin cậy
29         confidence = detection[2]
30         if confidence > .4:
31             # lấy lớp nhãn
32             class_id = detection[1]
33             # ánh xạ nó với tên lớp
34             class_name = class_names[int(class_id) - 1]
35             color = COLORS[int(class_id)]
36             # lấy tọa độ hộp giới hạn
37             box_x = detection[3] * image_width
38             box_y = detection[4] * image_height
39             # lấy chiều rộng và chiều cao của hộp giới hạn
40             box_width = detection[5] * image_width
41             box_height = detection[6] * image_height
42             # Vẽ hình chữ nhật quanh đối tượng
43             cv2.rectangle(image, (int(box_x), int(box_y)),
44                           (int(box_width), int(box_height)), color, thickness=1)
45             # đặt văn bản
46             cv2.putText(image, class_name, (int(box_x), int(box_y) - 5),
47                         cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
48             cv2.imshow('image', image)
49             cv2.waitKey(0)
50             cv2.destroyAllWindows()
```

1.2 OpenCV

- ❖ **Ứng dụng Phát hiện đối tượng sử dụng MobileNet SSD (Single Shot Detector)**



Chapter 1 – Computer Vision

1.1 Khái niệm Computer Vision

1.2 OpenCV

1.3 Tổng kết

1.4 Bài tập

1.3 Tổng kết

+ Hệ thống lại các bước tiền xử lý ảnh

Chapter 1 – Computer Vision

1.1 Khái niệm Computer Vision

1.2 OpenCV

1.3 Tổng kết

1.4 Bài tập

1.4 BÀI TẬP:

Bài tập 1

THANKS!

