

8/30/2024

# COMPUTER VISION

Lê Việt Hùng  
HUST

## Table of Contents

<b>Giới thiệu .....</b>	3
<b>Nội dung .....</b>	5
<b>1. Cách đọc và lưu hình ảnh trong python .....</b>	5
<b>2. Cách đọc nhiều file ảnh trong thư mục .....</b>	6
<b>3. Thay đổi kích thước ảnh .....</b>	10
<b>4. Sao chép và cắt dán vùng ảnh .....</b>	10
<b>5. Xoay ảnh.....</b>	12
<b>6. Ảnh màu và ảnh xám, cách chuyển đổi màu ảnh .....</b>	13
<b>7. Thư viện matplotlib .....</b>	15
<b>8. Vẽ các loại biểu đồ sử dụng thư viện matplotlib .....</b>	20
<b>9. Hiển thị lược đồ histogram của ảnh xám bằng matplotlib .....</b>	32
<b>10. Vẽ đường biên ảnh bằng matplotlib .....</b>	35
<b>11. Tương tác với ảnh .....</b>	36
<b>12. Thư viện Numpy và các phép tính trên mảng .....</b>	38
<b>13. Lấy thông tin cơ bản của ảnh bằng numpy .....</b>	46
<b>14. Biến đổi ảnh xám .....</b>	47
<b>15. Cân bằng lược đồ ảnh xám .....</b>	49
<b>16. Tạo ảnh mới bằng cách tính trung bình từ nhiều ảnh .....</b>	54
<b>17. Giới thiệu về thư viện Scipy .....</b>	58
<b>18. Bộ lọc Gaussian làm mờ ảnh .....</b>	60
<b>19. Các bộ lọc đạo hàm .....</b>	63
<b>20. Xử lý ảnh hình thái và đếm số lượng đối tượng .....</b>	68
<b>21. Thư viện OpenCv và các thao tác cơ bản .....</b>	76
<b>22. Sử dụng openCV để vẽ các hình cơ bản .....</b>	83
<b>23. OpenCV và các thao tác cơ bản trên video .....</b>	85
<b>24. Sử dụng openCV để lấy hình ảnh từ webcam .....</b>	90
<b>25. Sử dụng openCV phát hiện khuôn mặt .....</b>	96
<b>26. Giới thiệu DNN và openCV DNN .....</b>	106
<b>27. Nhận diện khuôn mặt bằng openCV DNN .....</b>	112
<b>28. Giới thiệu về Google colab .....</b>	133
<b>29. Trích xuất đặc trưng khuôn mặt .....</b>	135
<b>30. Tiền xử lí ảnh cho bài toán nhận diện khuôn mặt .....</b>	143

## Giới thiệu

Computer Vision (Thị giác máy tính) là một lĩnh vực của trí tuệ nhân tạo (AI) và khoa học máy tính, tập trung vào việc giúp máy tính "nhìn" và hiểu được hình ảnh hoặc video từ thế giới thực. Thị giác máy tính kết hợp các kỹ thuật xử lý hình ảnh, học máy và deep learning để phân tích, nhận dạng và đưa ra quyết định dựa trên dữ liệu hình ảnh.

Các ứng dụng phổ biến của thị giác máy tính bao gồm nhận dạng khuôn mặt, xử lý hình ảnh y tế, phân loại đối tượng, giám sát an ninh, xe tự lái, và thực tế ảo. Trong đó, hệ thống sử dụng các mô hình học sâu (deep learning) để trích xuất các đặc trưng từ hình ảnh và dự đoán kết quả. Thị giác máy tính đóng vai trò quan trọng trong việc tự động hóa các nhiệm vụ liên quan đến thị giác của con người, giúp cải thiện hiệu suất và độ chính xác trong nhiều lĩnh vực khác nhau như y tế, giao thông, sản xuất, và giải trí.

Python là một trong những ngôn ngữ lập trình phổ biến nhất trong lĩnh vực **Computer Vision** (Thị giác máy tính) nhờ vào cú pháp dễ hiểu và thư viện phong phú hỗ trợ xử lý hình ảnh và học máy. Với sự kết hợp của Python và các thư viện mạnh mẽ như **OpenCV**, **TensorFlow**, **Keras**, **PyTorch**, và **Scikit-image**, các nhà phát triển và nhà nghiên cứu có thể dễ dàng xây dựng và triển khai các ứng dụng thị giác máy tính.

### Các thư viện Python tiêu biểu cho Computer Vision:

- **OpenCV (Open Source Computer Vision Library)**: Là thư viện mã nguồn mở phổ biến nhất, hỗ trợ nhiều chức năng xử lý ảnh và video như lọc, phát hiện cạnh, theo dõi đối tượng, và nhận dạng đối tượng.

- **TensorFlow và PyTorch:** Hai thư viện học sâu (deep learning) mạnh mẽ giúp xây dựng các mô hình thị giác máy tính, đặc biệt trong việc nhận dạng ảnh, phân loại đối tượng, phát hiện và phân vùng đối tượng trong hình ảnh.
- **Keras:** Một API cao cấp chạy trên TensorFlow, giúp dễ dàng xây dựng các mô hình học máy, đặc biệt là các mô hình mạng nơ-ron phức tạp trong thị giác máy tính.
- **Scikit-image:** Thư viện chuyên dùng để xử lý ảnh, hỗ trợ các thao tác như biến đổi ảnh, phân tích cấu trúc hình học và phát hiện đặc trưng.

### **Ứng dụng của Python trong thị giác máy tính:**

- **Nhận diện khuôn mặt:** Sử dụng Python và OpenCV để phát hiện và nhận diện khuôn mặt từ hình ảnh hoặc video.
- **Phân loại hình ảnh:** Dùng các mô hình học sâu để phân loại các đối tượng trong hình ảnh thành các nhóm khác nhau.
- **Phân đoạn ảnh (Image Segmentation):** Xác định vùng chứa đối tượng cụ thể trong ảnh, được sử dụng trong y tế và xe tự lái.
- **Phát hiện và theo dõi đối tượng:** Theo dõi chuyển động của đối tượng trong video hoặc các hệ thống giám sát.

Python đã trở thành công cụ mạnh mẽ và dễ sử dụng trong việc giải quyết các vấn đề phức tạp của thị giác máy tính, từ nghiên cứu đến triển khai ứng dụng thực tế.

# Nội dung

## 1. Cách đọc và lưu hình ảnh trong python

Pillow là 1 thư viện mạnh mẽ xử lí ảnh, thực hiện nhiều tác vụ liên quan đến xử lí và chỉnh sửa hình ảnh, bao gồm tạo, chỉnh sửa, cắt, xoay, thay đổi kích thước, lọc ảnh và nhiều thao tác khác.

Sử dụng pillow : from PIL import Image

```
from PIL import Image
```

```
#thư mục chứa hình ảnh
```

```
dir = 'C:/computer_vision/img'
```

```
#đường dẫn đến hình ảnh
```

```
image_path = dir + '/hust.JPG'
```

```
print(image_path)
```

```
#sử dụng Image.open() để đọc ảnh
```

```
img01 = Image.open(image_path)
```

```
#hiển thị hình ảnh
```

```
img01.show()
```

```
#in 1 số thông tin
```

```
#in định dạng ảnh
```

```
print("định dạng ảnh: " , img01.format)
```

```
#in ra size
```

```
print("kích thước ảnh: " , img01.size)
```

```
#lưu ảnh bằng tên mới và chuyển đổi định dạng ảnh
new_img_01_path = dir + '/new.JPG' #đổi tên
new_img_png_01_path= dir + '/new_01.PNG' #đổi tên và định
dạng
#lưu 2 ảnh mới
img01.save(new_img_01_path)
img01.save(new_img_png_01_path)

#đóng tệp tin sau khi đọc
img01.close()
```

## 2. Cách đọc nhiều file ảnh trong thư mục

Tạo imgtools.py để code những hàm cần thiết và tái sử dụng cho các bài học sau

Để đọc nhiều thư mục, ta dùng thư viện os: import os

```
import os
from PIL import Image
```

#hàm load file, ở py ko dùng {} mà dùng :

```
def load_image(image_path):
```

```
    """
```

*đọc ảnh từ đường dẫn cho trước và trả về đối tượng ảnh*

*Args: image\_path:*

*:return: đối tượng hình ảnh*

```
    """
```

try:

```
    img = Image.open(image_path)
```

```
    return img
```

```
except Exception as e:
```

```
    # nếu lỗi thì in ra
```

```
    print("Lỗi khi đọc hình ảnh từ: ", image_path, " ",e)
    return None #lỗi thì phải về none(ko phải về null)
# try except thì dù đoạn code bị lỗi thì vẫn sẽ được thực thi
chứ không crash
```

```
def is_image_file(file_path):
    """
    return true nếu là ảnh, false nếu ko phải ảnh
    """
    extensions = ("jpg","jpeg","png","gif","bmp") #quy định đuôi
    ntn là ảnh
    return file_path.lower().endswith(extensions)
    #lower() để chuyển về viết thường thì có kthuc như kia thì
    true

def get_image_list(folder_path):
    #hàm lấy hết ảnh
    image_list = []
    #ta kiểm tra xem thư mục có tồn tại hay ko và có phải thư
    mục.isdir) hay ko
    if os.path.exists(folder_path) and os.path.isdir(folder_path):
        filenames = os.listdir(folder_path) #lấy toàn bộ file
        for filename in filenames: #duyệt toàn bộ file
            file_path = os.path.join(folder_path, filename) #kết nối
            folder_path vs filename
            if os.path.isfile(file_path) and is_image_file(file_path):
                img = load_image(file_path)
                image_list.append(img)
    return image_list
```

Ta tiến hành tạo file `read_multiple_file_image.py` có nội dung như sau:

```
from PIL import Image

from IPython.display import display #thư viện IPython để dùng
display

#import imgtools.py vừa code và lấy mọi phương thức = *
from imgtools import *

#thư mục chứa hình ảnh
my_dir = 'C:/computer_vision/img'

#đọc nhiều ảnh
imgs = get_image_list(my_dir)

"""

#đường dẫn đến hình ảnh
#image_path = dir + '/hust.JPG' #nếu ảnh ko tồn tại thì sẽ lỗi
nhg do try except lên vẫn thực thi dù có lỗi
#vậy nếu đường dẫn có tồn tại nhưng lại dẫn đến 1 file không
phải ảnh thì sao
#gia sử có file a.txt
#image_path= dir + '/a.txt' #thì bị lỗi khi đọc hình ảnh cannot
identify image file file ko đúng loại
#viết thêm hàm kiểm tra xem có đúng là hình ảnh hay ko ở
imgtools.py

print(image_path)
```

```
#đọc đường dẫn từ phương thức vừa viết ra thì phải import vào  
#đọc ảnh từ function
```

```
img01 = load_image(image_path)
```

```
#hiển thị hình ảnh  
img01.show()
```

```
#in 1 số thông tin  
#in định dạng ảnh  
print("định dạng ảnh: " , img01.format)  
#in ra size  
print("kích thước ảnh: ", img01.size)
```

```
#lưu ảnh bằng tên mới và chuyển đổi định dạng ảnh  
new_img_01_path = dir + '/new.JPG' #đổi tên  
new_img_png_01_path= dir + '/new_01.PNG' #đổi tên và định  
dạng  
#lưu 2 ảnh mới  
img01.save(new_img_01_path)  
img01.save(new_img_png_01_path)
```

```
#đóng tệp tin sau khi đọc  
img01.close()
```

```
.....
```

```
#hiển thị toàn bộ hình ảnh = display  
for img in imgs:  
    display(img)
```

tuy nhiên display ko hiển thị ảnh ở vscode hay pycharm mà chỉ hiển thị trên jupiter notebook

### 3. Thay đổi kích thước ảnh

```
4. from PIL import Image
5.
6. from IPython.display import display #thư viện IPython để dùng display
7.
8. #import imgtools.py vừa code và lấy mọi phương thức =
9. from imgtools import *
10.
11.my_path = 'C:/computer_vision/img/hust.jpg'
12.
13.img = load_image(my_path)
14.
15.display(img)
16.
17.#in kích thước ban đầu:
18.print(img.size)
19.
20.#thay đổi kích thước
21.new_image = img.resize((100, 100))
22.display(new_image)
23.new_image.show()      #show ra kích thước đã thay đổi
24.
25.new_image_2= img.thumbnail(50,50)      #thumbnail cũng có tác dụng như
   resize
26.#tuy nhiên có sự khác biệt giữa resize (thay đổi kích thước và tạo ra ảnh
   mới, ảnh cũ vẫn nguyên) còn thumbnail thì thay đổi kích thước ảnh gốc luôn
27.new_image_2.show()
```

### 4. Sao chép và cắt dán vùng ảnh

Ví dụ ảnh 1 bó hoa thì ta sẽ cắt ra ảnh 1 bông hoa, ở đây là cắt ngôi sao trên logo hust và dính lại vào ảnh cũ

```
from PIL import Image
```

```
from IPython.display import display #thư viện IPython để dùng  
display
```

```
#import imgtools.py vừa code và lấy mọi phương thức = *  
from imgtools import *
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

```
img = load_image(my_path)
```

```
#img.show()
```

```
#định vị khu vực cắt
```

```
region = (80,65, 210, 200) #80, 65 lần lượt là tọa độ xy, 210 200 là  
kích thước vùng cắt xy
```

```
# cắt vùng ảnh thành 1 ảnh mới
```

```
cropped_image= img.crop(region)
```

```
cropped_image.show()
```

```
#dán hình ảnh vừa cut
```

```
#xác định vị trí cần dán
```

```
paste_position= (250,250)
```

```
    img.paste(cropped_image,paste_position) #dán ảnh đã  
cắt(cropped_image) vào vị trí dán(paste_position)
```

```
    img.show()
```

## 5. Xoay ảnh

```
from PIL import Image
```

```
from IPython.display import display #thư viện IPython để dùng  
display
```

```
#import imgtools.py vừa code và lấy mọi phương thức = *
```

```
from imgtools import *
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

```
img = load_image(my_path)
```

```
#img.show()
```

```
#xoay ảnh dựa trên 1 trục ảnh
```

```
rotate_image = img.rotate(90) #xoay ảnh 90 độ, lưu ý ngược  
chiều kim đồng hồ
```

```
#lưu ý do kích thước ko đổi nên khi xoay ảnh có thể bị mất đi 1  
vài phần
```

```
#như vậy trước khi xoay ảnh nên resize ảnh về ảnh vuông
```

```
rotate_image.show()
```

```
#đảo chiều ảnh
```

```

transposed_image =
img.transpose(Image.Transpose.ROTATE_90)

transposed_image.show()

#khi transpose thì ảnh sẽ ko bị mất pixel nào mà chỉ đơn giản là
đảo chiều

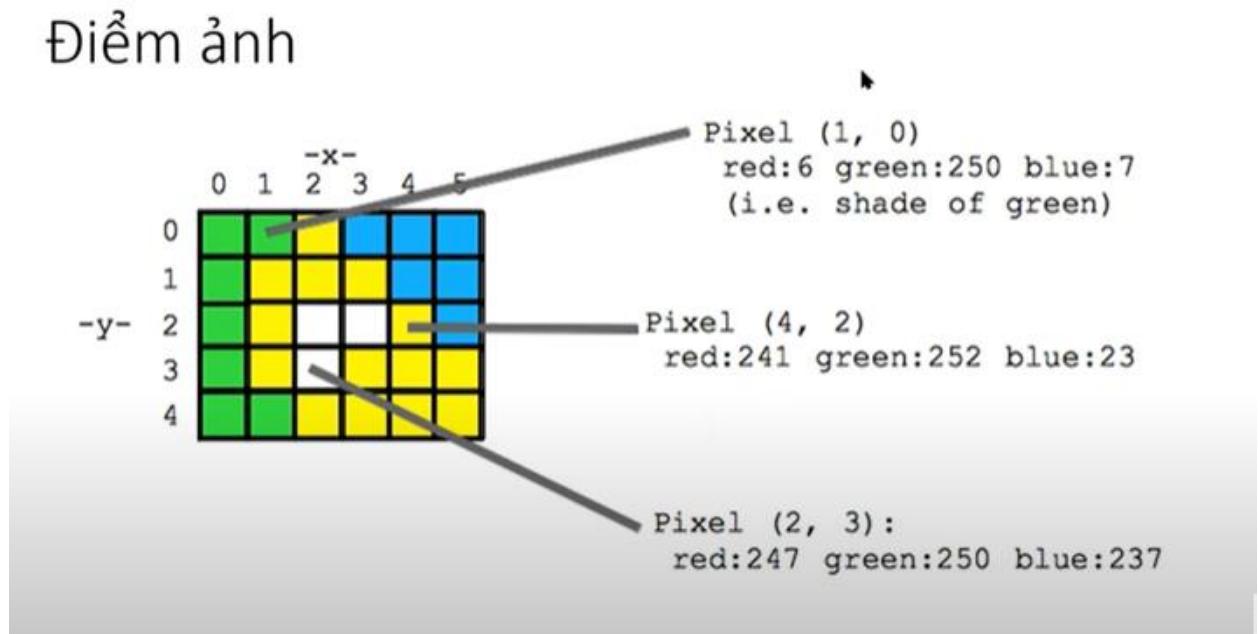
#nôm na là chiều rộng thành chiều cao và ngược lại

```

## 6. Ảnh màu và ảnh xám, cách chuyển đổi màu ảnh

- ảnh màu RGB là một loại hình ảnh được biểu diễn bằng 3 kênh màu chính: Đỏ(Red), xanh lục (Green) và xanh lam (Blue)

- điểm ảnh:



Giá trị red green blue sẽ từ 0 -> 255, ví dụ:



- ảnh xám: lưu ý chỉ chuyển được từ màu sang xám chứ rất khó để từ xám sang màu được.

Ưu: giảm chi phí tính toán, tập trung vào độ sáng, tiết kiệm dung lượng lưu trữ, giảm nhiễu, dễ dàng hiểu và xử lý

Tuy nhiên có thể gây mất mát thông tin về màu sắc

```
from PIL import Image
```

```
from IPython.display import display #thư viện IPython để dùng
display
```

```
#import imgtools.py vừa code và lấy mọi phương thức = *
```

```
from imgtools import *
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

```
img = load_image(my_path)
```

```
#chuyển từ ảnh RGB sang ảnh xám
gray_image= img.convert("L") #L là dạng ảnh xám
gray_image.show()

#phân RGB thành 3 tầng màu khác nhau
#lưu ý ko hiểu nhầm tách màu sẽ ra đỏ, xanh hoàn toàn mà chỉ là
màu tầng khác nhau(độ đậm khác nhau)
red_band, green_band, blue_band = img.split()
red_band.show()
green_band.show()
blue_band.show()

#trộn ảnh
#khi trộn 3 ảnh trên vào sẽ ra ảnh ban đầu( ảnh gốc có màu)
merge_image = Image.merge("RGB", (red_band, green_band,
blue_band))
merge_image.show()
```

## 7. Thư viện matplotlib

-Có thể minh họa và xử lý ảnh cơ bản:

Vẽ được nhiều kiểu đồ thị khác nhau( dùng nhiều trong ai và  
khoa học dữ liệu, data)

Có hàm để làm việc với ảnh

## Tạo ra biểu đồ và đồ thị chất lượng cao

- giúp trực quan hóa dữ liệu 1 cách dễ dàng và hiệu quả
- cung cấp 1 loạt các biểu đồ như đường, cột, tròn và nhiều biểu đồ phức tạp khác
- cho phép tùy chỉnh nhiều khía cạnh biểu đồ như màu sắc, kích thước, tiêu đề và hình dạng khác

Sử dụng matplotlib như sau:

```
import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
```

tiến hành code:

```
import matplotlib as mpl
```

```
import matplotlib.pyplot as plt
```

```
from PIL import Image
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

#đọc ảnh và hiển thị

```
img = Image.open(my_path)
```

```
plt.imshow(img) #hiển thị bằng imshow thì sẽ hiển thị ảnh kèm theo cả tọa độ
```

```
#plt.axis('off') # Tắt hiển thị trực tọa độ (nếu bạn không muốn thấy trực)
```

```
plt.show() #hiển thị ảnh kèm đồ thị
```

Figure 1

- □ ×



#vẽ đồ thị cơ bản

x=[1,2,3,4,5]

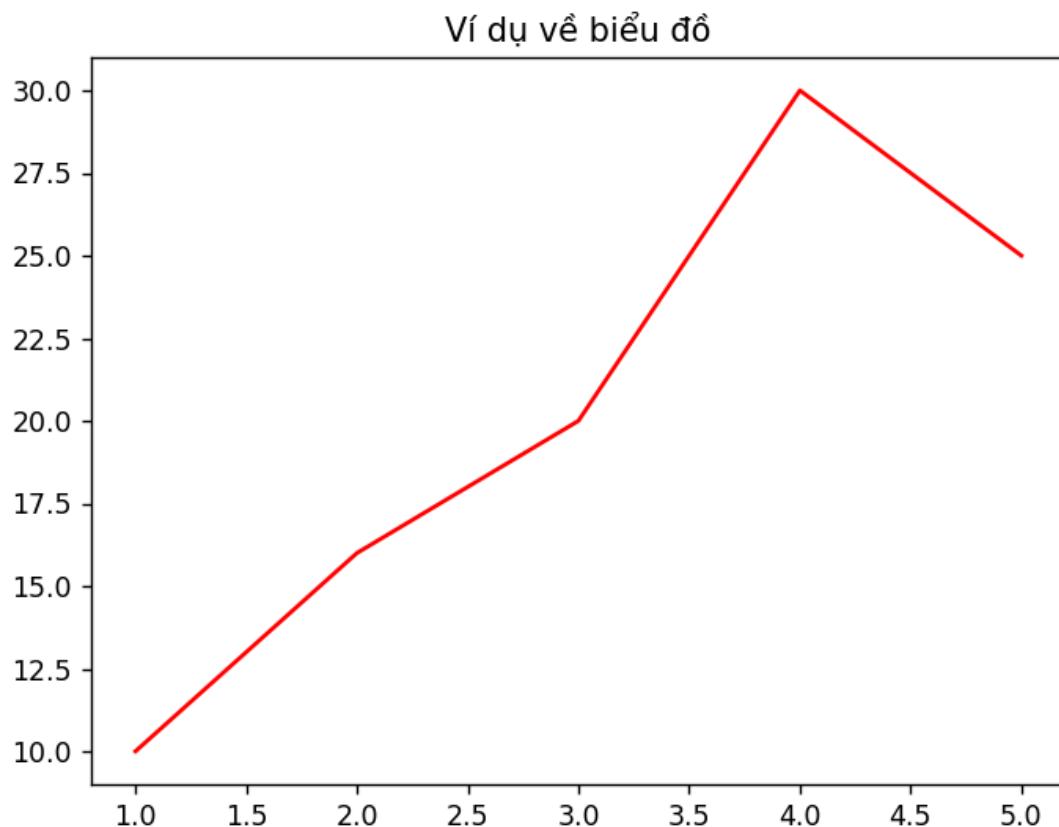
y=[10,16,20,30,25]

plt.plot(x,y, color='red') #vẽ từng cặp x,y và đường kẻ bằng màu đỏ

plt.title('Ví dụ về biểu đồ') #tiêu đề

plt.show()

Figure 1



```
#vẽ đồ thị trên ảnh  
im = Image.open(my_path)  
x=[100, 100, 400, 400]  
y=[200, 300, 200,300]  
plt.imshow(im) #hiển thị hình ảnh  
#plt.plot(x,y, 'r*') #đánh dấu các điểm bằng dấu sao màu đỏ  
#plt.plot(x,y, 'ks:') #đánh dấu các điểm bằng ô vuông và nối với  
nhau bằng nét đứt (đều màu đen)
```

```
plt.plot(x,y, color='red') #đánh dấu các điểm và nối chúng với  
nhau bằng màu đỏ
```

```
plt.show()
```



1 số quy tắc vẽ:

## Một số định dạng khi vẽ

color	
'b'	blue
'g'	green
'r'	red
'c'	cyan
'm'	magenta
'y'	yellow
'k'	black
'w'	white

marker	
'.'	point
'o'	circle
's'	square
'*'	star
'+'	plus
'x'	x

line style	
'-	solid
'--'	dashed
':'	dotted

Ví dụ : plt.plot(x,y, 'bo--')

Thì hiển thị chấm blue tròn nối với nhau bằng nét đứt --

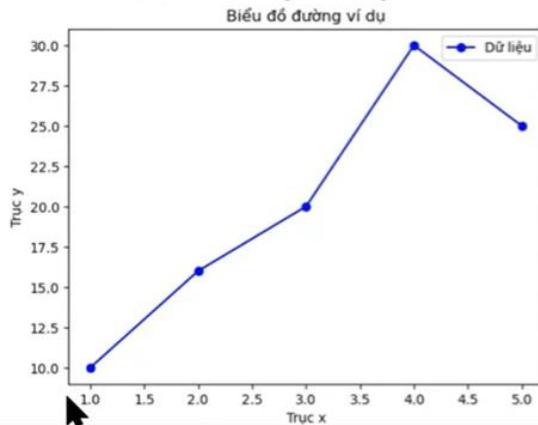
## 8. Vẽ các loại biểu đồ sử dụng thư viện matplotlib

Các loại biểu đồ:

- biểu đồ đường (line chart)
- biểu đồ cột (bar chart)
- biểu đồ hình tròn (pie chart)
- biểu đồ phân tán (scatter plot)
- biểu đồ hộp (box plot)
- các biểu đồ khác

## Biểu đồ đường (Line Chart)

- Thường được sử dụng để biểu diễn dữ liệu theo một loạt các điểm dữ liệu được nối với nhau bằng các đoạn thẳng.



### Ứng dụng:

- Biểu diễn xu hướng và biến đổi
- So sánh dữ liệu
- Phát hiện biên độ và biên giới
- Phân tích chuỗi thời gian

Ta thử code như sau:

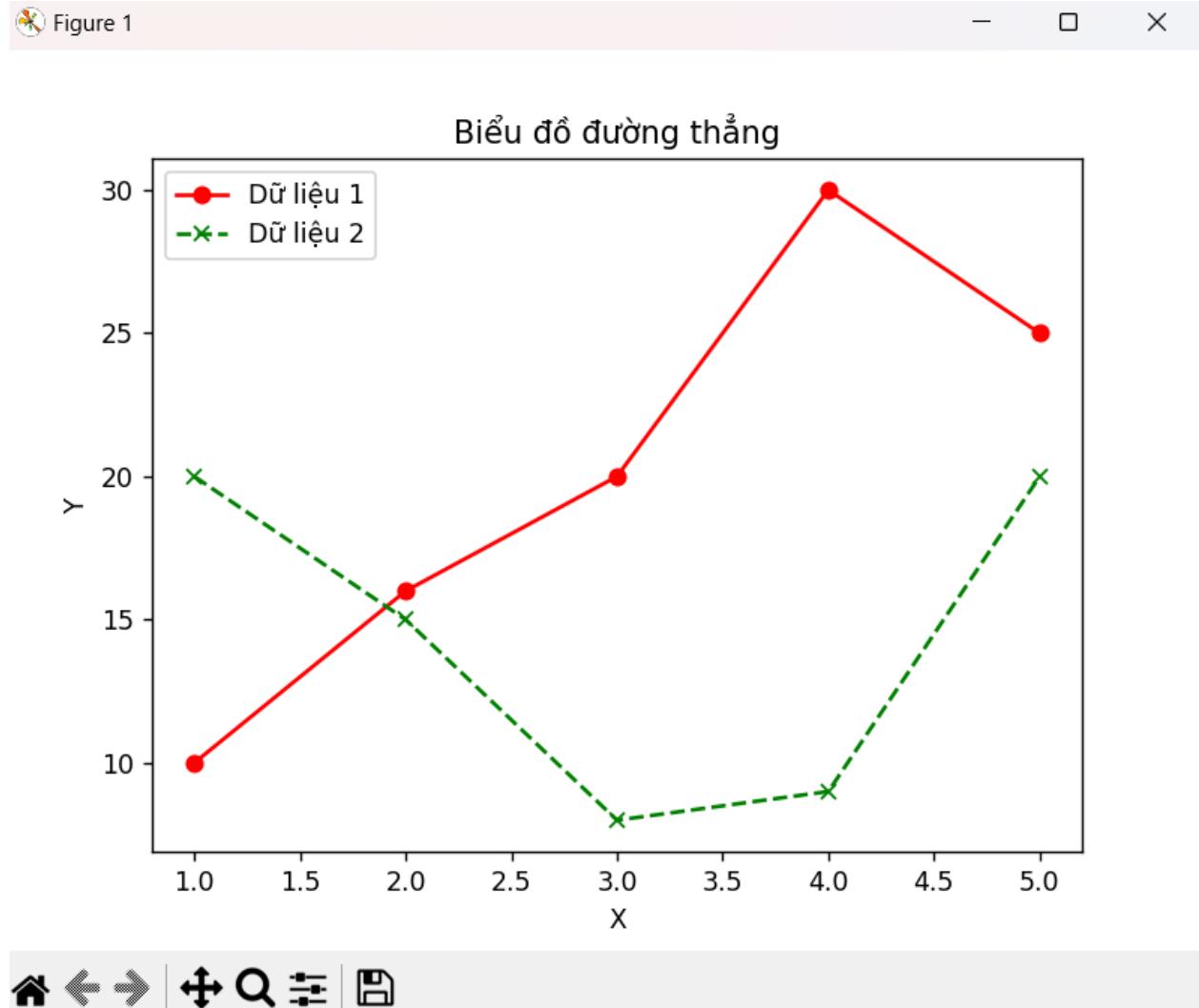
```
import matplotlib.pyplot as plt
```

```
#line chart
```

```
x = [1,2,3,4,5]  
y = [10,16 ,20, 30, 25]  
x1=[1,2,3,4,5]  
y1=[20,15,8,9,20]
```

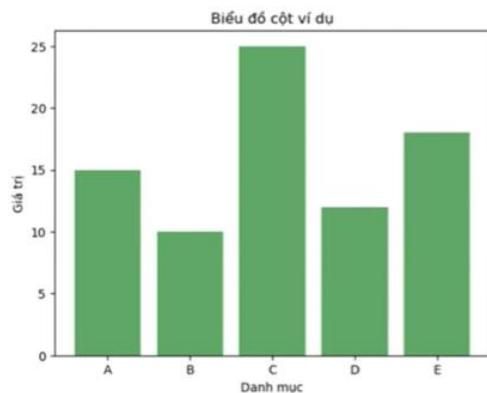
```
plt.plot(x,y,'ro-',label='Dữ liệu 1') #label dán nhãn chú thích  
cho đường  
plt.plot(x1,y1,'gx--',label='Dữ liệu 2')  
plt.legend() #cho phép hiển thị label  
plt.xlabel('X') #label cho trục x  
plt.ylabel('Y') #label cho trục y  
plt.title('Biểu đồ đường thẳng') #tên của biểu đồ  
plt.show()
```

Kết quả:



## Biểu đồ cột (Bar Chart)

- Là một loại biểu đồ phổ biến được sử dụng để hiển thị dữ liệu theo các cột dọc, với chiều cao của mỗi cột biểu thị giá trị tương ứng



### Ứng dụng:

- So sánh dữ liệu
- Biểu diễn dữ liệu rời rạc
- Phân phối dữ liệu
- Thời gian và chuỗi thời gian
- Sự biến đổi qua các biến số chính

Ta sẽ thử code như sau:

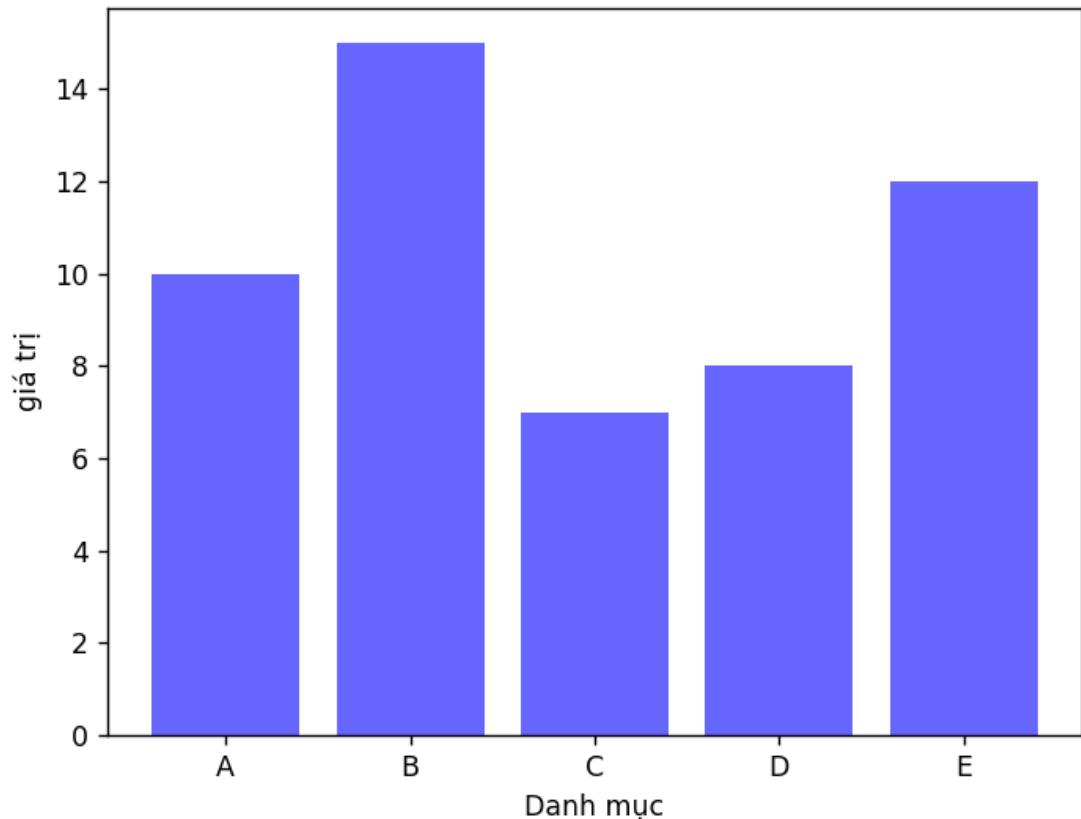
```
# bar chart  
categories = ['A','B', 'C', 'D', 'E']  
values = [10, 15, 7, 8, 12]
```

```
plt.bar(categories, values, color='b', alpha =0.6) #hiển thị  
biểu đồ đường với độ trong suốt alpha là 0.6  
#alpha =0 là trong suốt(ko nhìn thấy đường) và =1 là màu đầy  
đủ  
plt.xlabel('Danh mục')  
plt.ylabel('giá trị')  
plt.title('Biểu đồ cột')  
plt.show()
```

Kết quả:

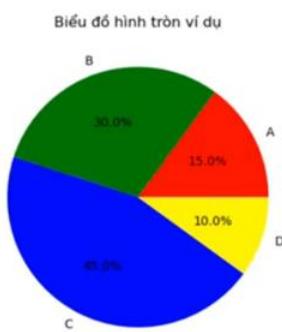
Figure 1

Biểu đồ cột



## Biểu đồ hình tròn (Pie Chart)

- Là một loại biểu đồ hình hình tròn được chia thành các phần tương ứng với các phần trăm của tổng giá trị. Biểu đồ hình tròn thường được sử dụng để biểu diễn cơ cấu, phân chia hoặc tỷ lệ của các phần trong một tập hợp dữ liệu.



### Ứng dụng:

- Biểu diễn cơ cấu
- Hiển thị phần trăm
- Phân tích cơ cấu thị trường
- Biểu diễn phần trăm trong khảo sát

Ta sẽ thử code như sau:

```
# pie chart
```

```
my_labels = ['A', 'B', 'C', 'D']
```

```
sizes = [15, 30, 45, 10]
```

```
my_color = ['red', 'green', 'blue', 'yellow']
```

```
plt.pie(sizes, labels = my_labels, colors= my_color, autopct ='%1.1f%%') #vẽ biểu đồ tròn
```

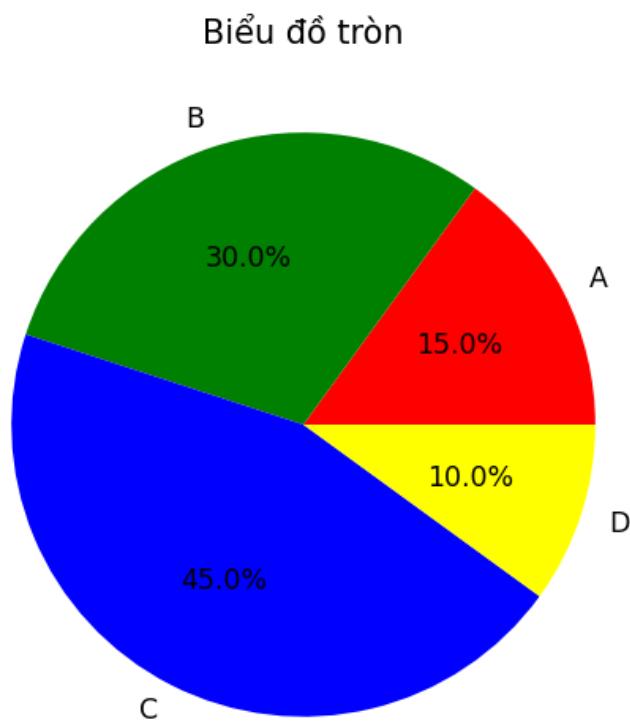
```
# autopct ='%1.1f%%' để hiển thị số liệu với 1 chữ số sau dấu phẩy
```

```
plt.title('Biểu đồ tròn')
```

```
plt.show()
```

Kết quả:

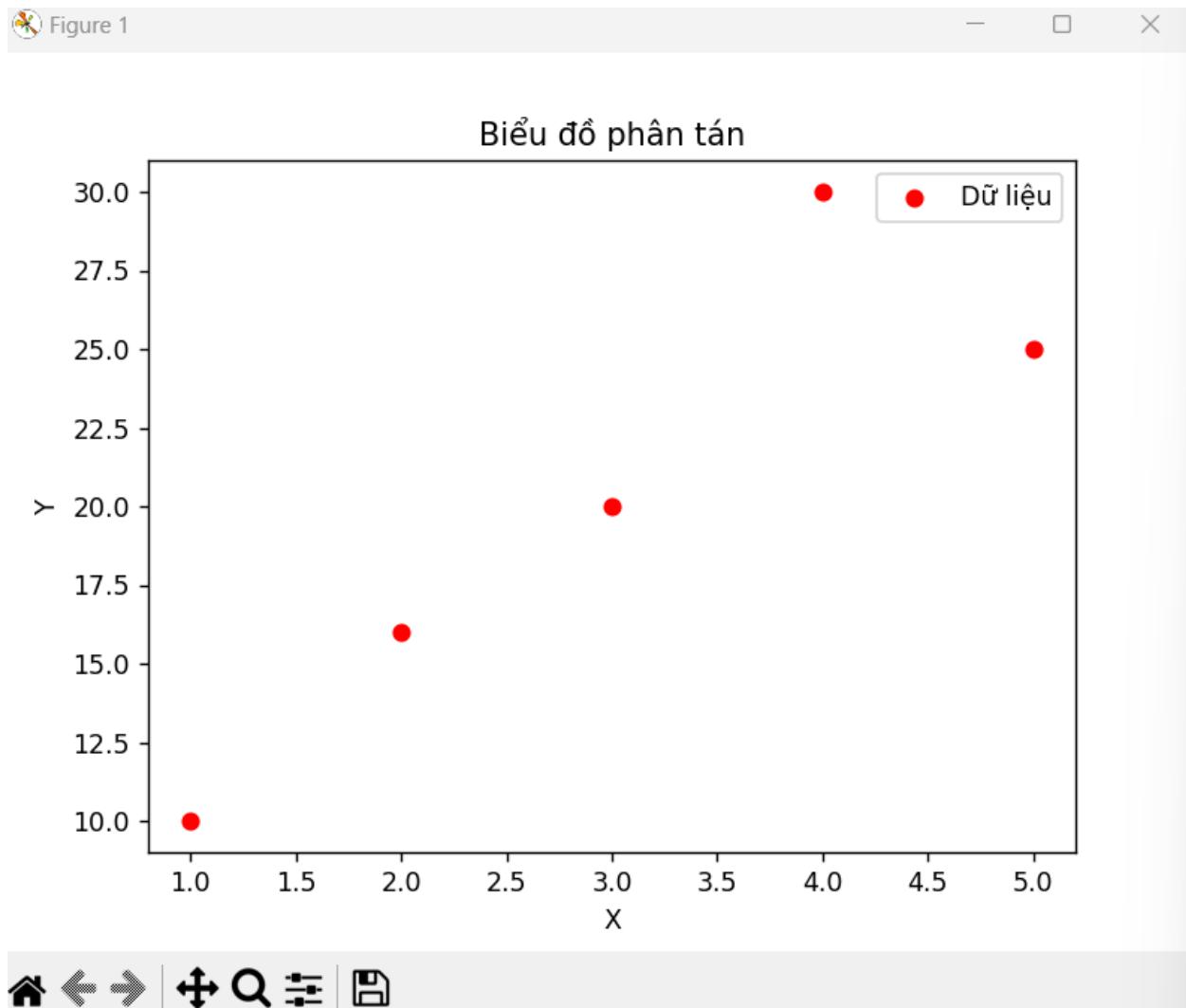
Figure 1



- Đối với biểu đồ phân tán, ta code như sau:

```
• #scatter plot
• x = [1,2,3,4,5]
• y = [10,16 ,20, 30, 25]
•
• plt.scatter(x,y,c='r',marker='o', Label='Dữ liệu')    #vẽ biểu đồ phân
  tán
• plt.legend()    #cho phép hiển thị label
• plt.xlabel('X')    #label cho trục x
• plt.ylabel('Y')    #label cho trục y
• plt.title('Biểu đồ phân tán')    #tên của biểu đồ
• plt.show()
```

Kết quả:



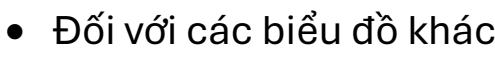
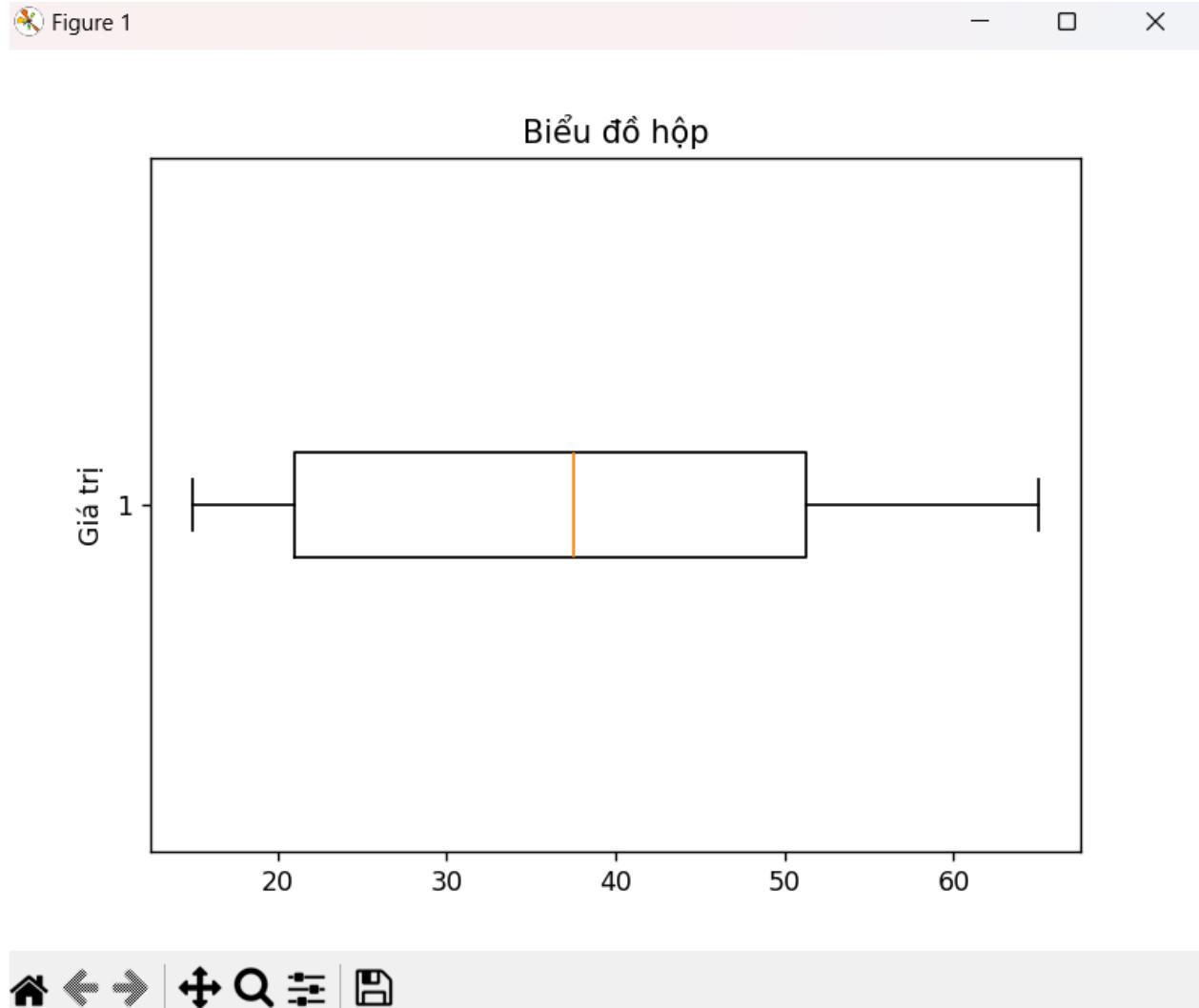
- Đối với biểu đồ hộp để xem trọng tâm khu vực nằm ở đâu  
Ta code như sau:

```
#box plot  
data =[15,18,22,30,45,50,55,65]
```

```
plt.boxplot(data,vert=False) #vẽ biểu đồ hộp  
#vert=False để biểu đồ hộp nằm ngang, nếu ko có vert mà  
chỉ plt.boxplot(data) thì sẽ mặc định nằm dọc  
plt.ylabel('Giá trị')  
plt.title('Biểu đồ hộp')
```

```
plt.show()
```

Kết quả:



- Đối với các biểu đồ khác

- Violin plot:

Code như sau:

```
#violin plot
```

```
data = [15,18,22,30,35,45,50,55,65]
```

```
plt.violinplot(data,showmeans=True) #vẽ biểu đồ hình violin
```

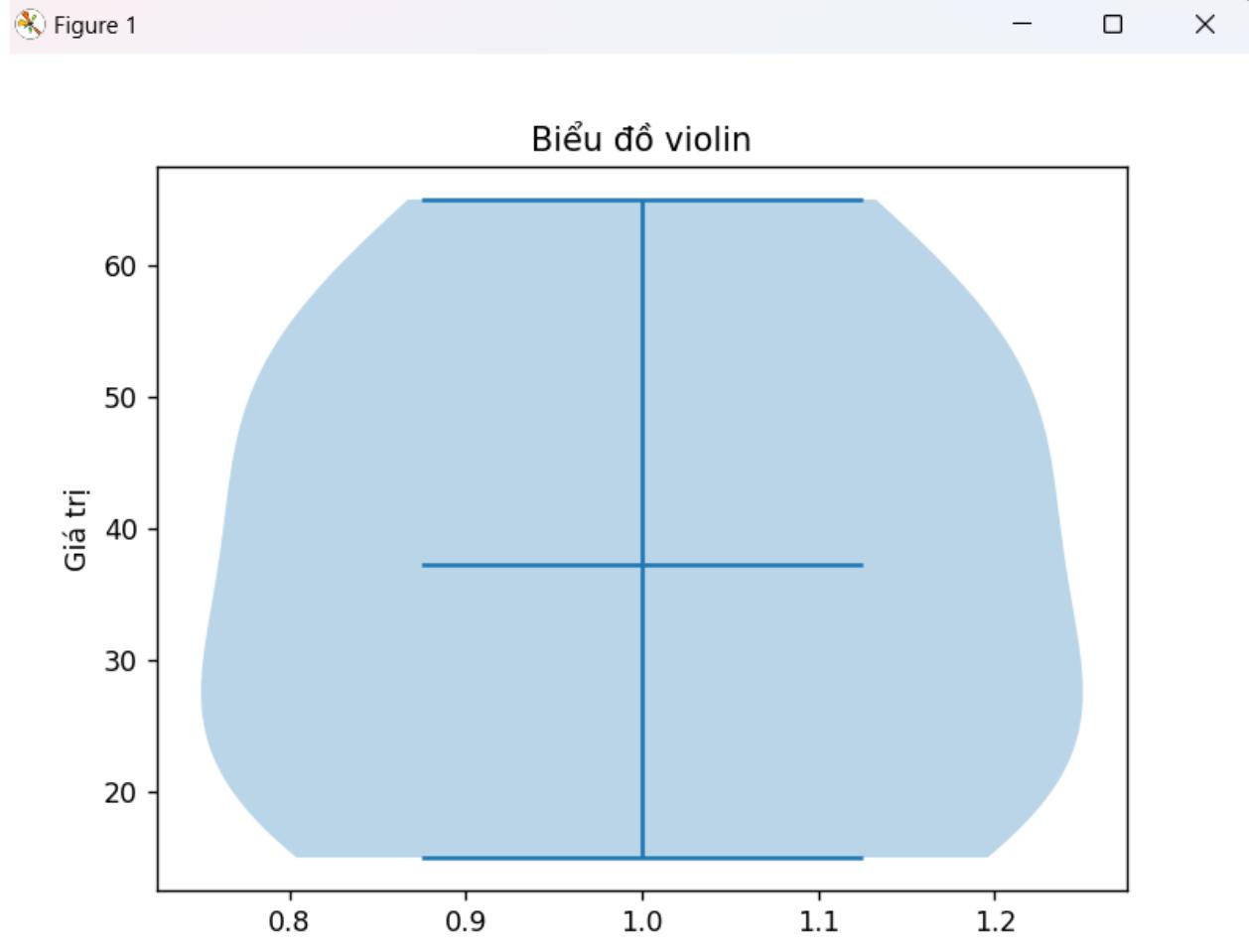
```
#showmeans=True để hiển thị giá trị trung bình(mean)
```

```
plt.title ('Biểu đồ violin')
```

```
plt.ylabel('Giá trị')
```

```
plt.show()
```

Kết quả:



x=1.1623 y=61.0

- Word cloud (biểu đồ đám mây)  
Thống kê từ khóa nào xuất hiện nhiều, từ khóa nào càng quan trọng thì chữ càng to  
Để dùng được wordcloud thì cần install như sau: pip install wordcloud  
Sau khi cài thành công ta tiến hành code như sau:  
#wordcloud  
from wordcloud import WordCloud

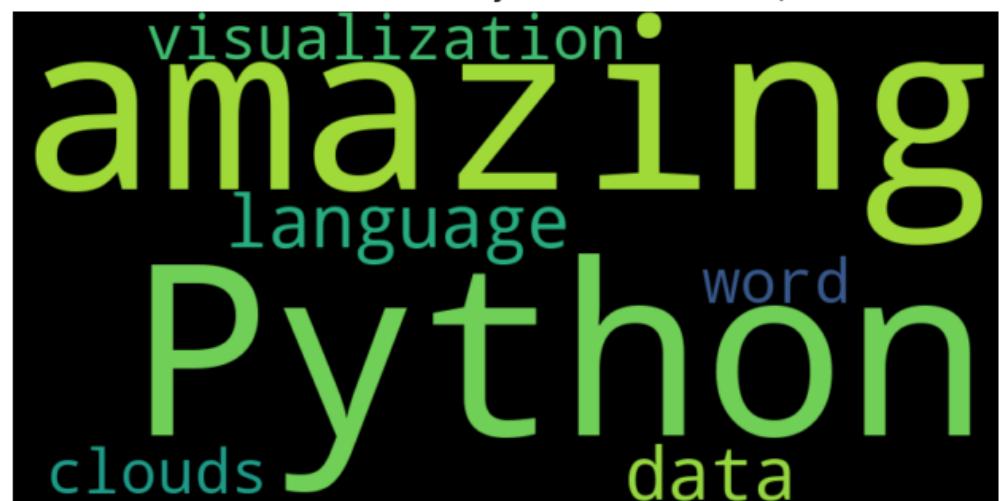
```
text = "Python is an amazing language for data visualization  
and word clouds."
```

```
wordcloud = WordCloud(width= 800,  
height=400).generate(text)  
plt.imshow(wordcloud, interpolation='bilinear')  
plt.axis('off')  
plt.title('Biểu đồ đám mây từ văn bản ví dụ')  
plt.show()
```

Kết quả:



Biểu đồ đám mây từ văn bản ví dụ



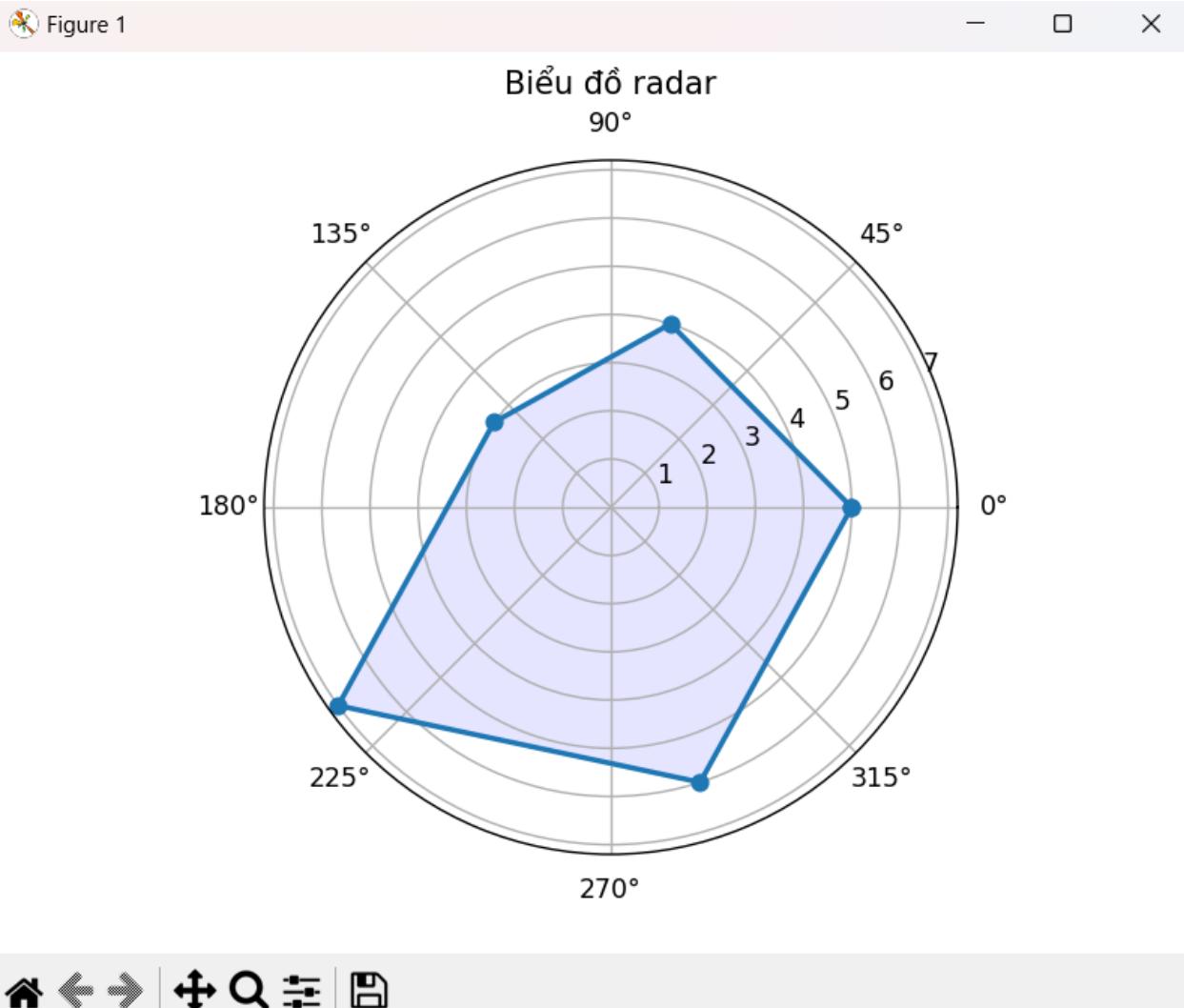
- Radar chart

```

- #radar chart
- import numpy as np # Thư viện numpy được import để thực hiện các phép
  toán số học, đặc biệt là với mảng (arrays).
- # Khởi tạo danh sách các danh mục (categories) và giá trị (values)
  tương ứng.
- categories = ['A', 'B', 'C', 'D', 'E']
- values = [5, 4, 3, 7, 6]
- N = len(categories) # Xác định số lượng danh mục (categories), lưu vào
  biến N.
- # Thêm giá trị đầu tiên vào cuối danh sách values để tạo thành một chu
  kỲ khép kín cho biểu đồ radar.
- values += values[:1]
- # Tính toán các góc (angles) tương ứng cho mỗi danh mục trên biểu đồ
  radar.
- # Angles được tính toán dựa trên tỷ lệ của số danh mục, với mỗi danh
  mục cách đều nhau trên vòng tròn 360 độ ( $2\pi$  radians).
- angles = [n / float(N) * 2 * np.pi for n in range(N)]
- angles += angles[:1] # Thêm góc của danh mục đầu tiên vào cuối để đảm
  bảo rằng biểu đồ radar khép kín.
- # Vẽ biểu đồ radar với các giá trị và góc tương ứng.
- # 'o-' chỉ định kiểu đường nối giữa các điểm là đường liền với các điểm
  tròn, linewidth=2 chỉ định độ dày của đường.
- plt.polar(angles, values, 'o-', linewidth=2)
- # Tô màu vùng bên trong biểu đồ radar để làm nổi bật diện tích dưới
  đường biểu diễn.
- # 'b' chỉ định màu xanh dương, alpha=0.1 chỉ định độ trong suốt của màu
  (10% trong suốt).
- plt.fill(angles, values, 'b', alpha=0.1)
- plt.title('Biểu đồ radar')
- plt.show()

```

Kết quả:



## 9. Hiển thị lược đồ histogram của ảnh xám bằng matplotlib

Lược đồ ảnh (image histogram) là một biểu đồ thống kê dùng để thể hiện phân bố của các giá trị pixel trong một hình ảnh.

Mỗi giá trị pixel thường tương ứng với một mức xám hoặc màu sắc cụ thể.

Lược đồ ảnh giúp biết tần suất xuất hiện của các giá trị này trong hình ảnh, qua đó cung cấp thông tin về cường độ sáng hoặc màu sắc trong hình ảnh.

```
from PIL import Image
```

```
import matplotlib.pyplot as plt
import numpy as np

#đọc ảnh gốc
my_path = 'C:/computer_vision/img/hust.jpg'
im= Image.open(my_path)
plt.imshow(im)
#im.show()

#chuyển sang ảnh xám
im_gray = im.convert('L')
plt.imshow(im_gray, cmap='gray')
#im_gray.show()

#vẽ đồ thị histogram của ảnh xám
#biến toàn bộ ảnh thành mảng
im_array = np.array(im_gray)
print(im_array)

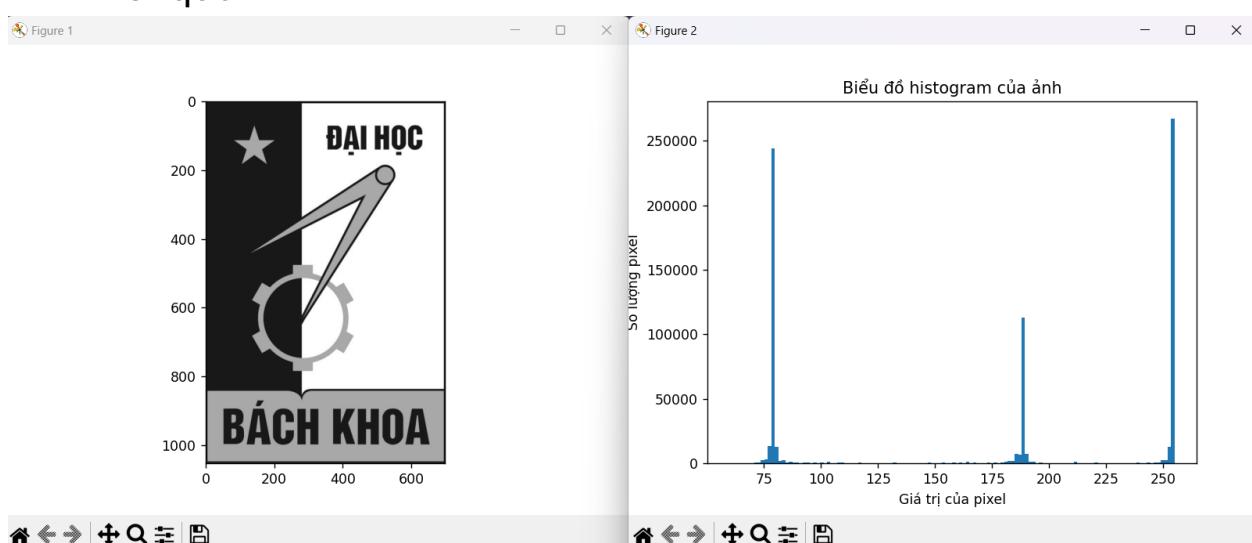
#từ mảng có thể hiển thị ra histogram
plt.figure() #hiển thị dưới dạng mới
plt.hist(im_array.flatten(), bins = 128)
#flatten() biến mảng 2 chiều -> 1 chiều , bins là số lượng ngăn
```

```

plt.title('Biểu đồ histogram của ảnh')
plt.xlabel('Giá trị của pixel')
plt.ylabel('Số lượng pixel')
plt.show()

```

kết quả:



Thực tế, ta chuyển ảnh về ảnh xám rồi sau đó thành mảng 2 chiều kiểu như thế này:

`[[ 79 79 79 ... 79 79 79]`

`[ 79 79 79 ... 79 79 79]`

`[ 79 79 79 ... 79 79 79]`

`...`

`[ 79 79 79 ... 79 79 79]`

`[ 79 79 79 ... 79 79 79]`

`[117 117 117 ... 117 117 117]]` rồi chuyển thành mảng 1 chiều và cuối cùng là thành đồ thị histogram.

## 10. Vẽ đường biên ảnh bằng matplotlib

Giả sử trong ảnh có rất nhiều vật thể thì mỗi vật thể sẽ tạo thành 1 đường biên.

Đường biên của 1 ảnh (ảnh được hiểu là một ma trận hoặc một mảng số) là tập hợp các điểm biên giới của các đối tượng hoặc các khu vực trong ảnh.

Đường biên xác định ranh giới giữa các đối tượng và nền trong ảnh.

Điều này thường quan trọng trong xử lý ảnh và phân đoạn hình ảnh, khi muốn tách các đối tượng trong ảnh hoặc phát hiện biên của chúng.

Thư viện matplotlib chưa phải tối ưu nhất khi xử lý vấn đề này, thường thì dùng openCV sẽ đem lại hiệu quả cao hơn. Tuy nhiên mới làm quen thì ta sẽ sử dụng matplotlib.

```
from PIL import Image
```

```
import matplotlib.pyplot as plt
```

```
#đọc ảnh gốc
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

```
im= Image.open(my_path).convert("L")
```

```
plt.imshow(im) #hiển thị ảnh dưới dạng trực tọa độ
```

```
plt.figure() #tạo hình vẽ mới
```

```
plt.gray() #đặt màu sắc là xám
```

```

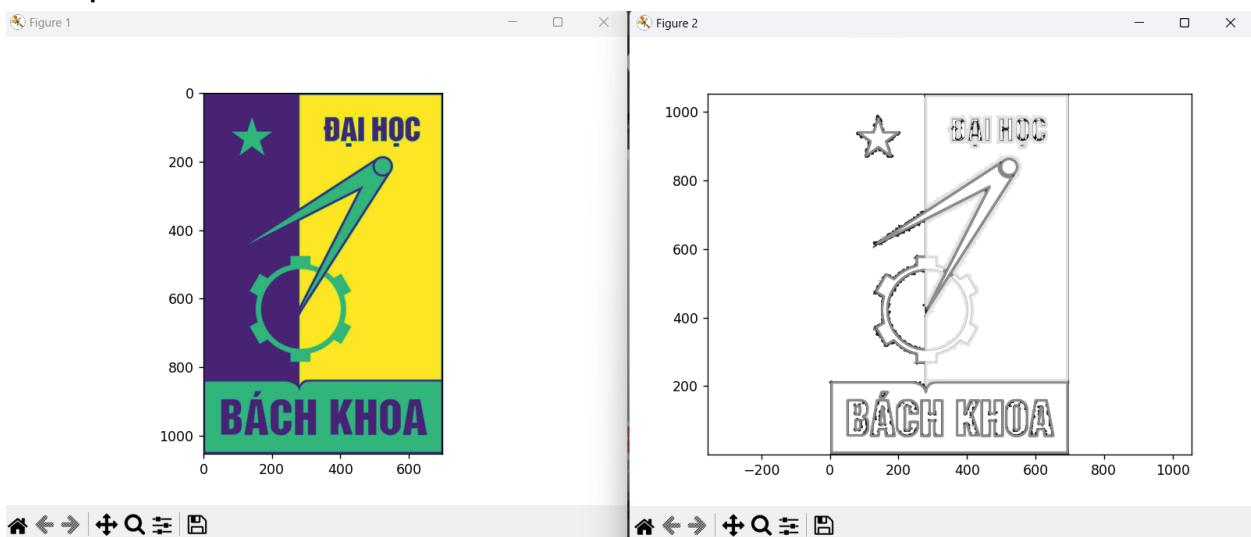
plt.contour(im, origin="image") #vẽ biểu đồ đường biên sử
dụng contour

plt.axis('equal') #đảm bảo tỉ lệ trực x và y trên biểu đồ là bằng
nhau

plt.show()

```

Kết quả:



ở figure 2, ảnh đã được viền các đối tượng lại thành công, các đường biên đã được làm rõ.

## 11. Tương tác với ảnh

Ta cần đánh dấu điểm ảnh, chú thích dữ liệu huấn luyện AI,... thì cần tương tác người dùng với điểm ảnh.

```

from PIL import Image
import matplotlib.pyplot as plt

```

```

my_path = 'C:/computer_vision/img/hust.jpg'
im = Image.open(my_path)

```

```
#plt.switch_backend('tkagg') chuyển đổi môi trường nếu đang sử  
dụng jupiter notebook, còn VSC và pycharm thì ko cần  
plt.imshow(im)  
plt.title('Click on the image to select points')
```

```
#sử dụng hàm ginput để chọn điểm trên ảnh  
points = plt.ginput(5) #được chọn 5 điểm  
print(points)      #in ra tọa độ 5 điểm mình đã chọn
```

```
#hiển thị  
plt.show()
```

```
#vẽ lại các điểm đã chọn bằng dấu * màu đỏ  
plt.close()  
plt.imshow(im)  
for point in points:  
    x, y = point  
    plt.plot(x, y, 'r*')  
plt.show()
```

Kết quả: sau khi hiện ảnh thì ta sẽ click lên ảnh 5 điểm bất kì và sẽ hiện ảnh kèm 5 điểm đã click và tọa độ của chúng.

Figure 1

- □ ×



Tọa độ 5 điểm như sau: [(146.1493506493507, 129.41558441558436), (522.2207792207793, 211.46753246753246), (271.5064935064936, 626.2857142857142), (563.2467532467532, 99.78571428571422), (339.88311688311694, 943.0974025974025)]

## 12. Thư viện Numpy và các phép tính trên mảng

Numpy là một thư viện Python mạnh mẽ cho tính toán khoa học và số học, viết tắt của Numerical Python.

Cho phép xử lý dữ liệu nhiều chiều (mảng đa chiều) và các phép toán trên chúng một cách hiệu quả.

Hiệu suất: Numpy được viết bằng C nên nhanh hơn việc sử dụng list python thông thường.

Hỗ trợ đa chiều: Numpy giúp xử lí mảng đa chiều dễ dàng.

Hỗ trợ đa dạng loại dữ liệu số học: Số nguyên, số thực, số phức và nhiều loại dữ liệu khác.

Để sử dụng : import numpy as np

#tạo mảng 1 chiều

a = np.array([1,2,3])

#in mảng

print(a)

#in ra phần tử bất kì

element = a[1]

print(element) #vị trí số 1 của mảng là 2

Ta thu được kết quả:

[1 2 3]

2

#tạo mảng 2 chiều

matrix = np.array([[1,2,3], [4,5,6], [7,8,9]])

#in mảng

print(matrix)

#in phần tử bất kì

element = matrix[1,2] #vị trí hàng thứ 1 và cột thứ 2(tính từ 0)

print(element) #có giá trị là 6

Kết quả thu được:

[[1 2 3]

[4 5 6]

[7 8 9]]

6

#tạo mảng toàn số 0

a = np.zeros((3,5)) #3 hàng và 5 cột

print(a)

Kết quả thu được:

[[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0.]]

Tương tự:

#tạo mảng toàn số 1

a = np.ones((3,5))

print(a)

#tạo mảng rỗng

a = np.empty((3,5)) #lưu ý mảng rỗng nó sẽ mặc định hiển thị

là 1

print(a)

```
#tạo mảng từ 1 đến 100, bước nhảy 1
```

```
a = np.arange(1,101,1)
```

```
print(a)
```

```
#tạo mảng gồm các phần tử với khoảng cách đều nhau
```

```
a = np.linspace(0,10,5) #mảng start 0 end là 10 và có 5 phần tử cách đều nhau
```

```
print(a) #[ 0.  2.5  5.  7.5 10. ]
```

```
#xác định kiểu dữ liệu
```

```
a = np.ones(5, dtype= np.int64)
```

```
print(a.dtype) #int64
```

```
#thêm, xóa, sắp xếp các bảng
```

```
#tạo mảng ban đầu
```

```
arr = np.array([3,1,2,4,5]) #3,1,2,4,5
```

```
print(arr)
```

```
#sắp xếp
```

```
arr = np.sort(arr)
```

```
print(arr) #[1 2 3 4 5]
```

```
#sắp xếp ngược (sắp xếp nhưng in theo chiều ngược lại)
```

```
arr = np.sort(arr)[::-1]
```

```
print(arr) #[5 4 3 2 1]
```

```
# thêm phần tử vào mảng
```

```
arr = np.append(arr, 6) #thêm 6 vào mảng
```

```
print(arr) #[5 4 3 2 1 6]
```

```
#xóa 1 vị trí nào đó trong mảng
```

```
arr = np.delete(arr, 3) #xóa vị trí thứ 3
```

```
print(arr) #[5 4 3 1 6] mất số 2
```

```
#tạo mảng 2 chiều
```

```
arr = np.array([[3,1,2],[4,6,8],[9,7,5]])
```

```
#sắp xếp tăng dần
```

```
sap_xep_theo_hang = np.sort(arr, axis=1) #theo hàng thì axis  
=1
```

```
print(sap_xep_theo_hang) #[[1 2 3] [4 6 8] [5 7 9]]
```

```
sap_xep_theo_cot = np.sort(arr, axis=0) #theo cột thì axis =0
```

```
print(sap_xep_theo_cot) #[[3 1 2] [4 6 5] [9 7 8]]
```

```
#sắp xếp giảm dần(thêm dấu -)
```

```
sap_xep_theo_hang = -np.sort(-arr, axis=1) #theo hàng thì axis  
=1
```

```
print(sap_xep_theo_hang) #[[3 2 1] [8 6 4] [9 7 5]]
```

```
sap_xep_theo_cot = -np.sort(-arr, axis =0) #theo cột thì axis =0  
print(sap_xep_theo_cot) #[[9 7 8] [4 6 5] [3 1 2]]
```

```
#đưa ra các thông số về mảng  
#tạo mảng 2 chiều  
arr = np.array([[1,2,3], [4,5,6]])  
#sử dụng các thuộc tính để lấy thông tin về chúng  
so_chieu = arr.ndim #số chiều  
kich_thuoc = arr.size #kích thước = tổng số phần tử  
hinh_dang = arr.shape #hình dạng (số hàng x số cột)  
#in ra các thông số  
print("Số chiều: ", so_chieu) #Số chiều: 2  
print("Kích thước: ", kich_thuoc) #Kích thước: 6  
print("Hình dạng: ", hinh_dang) #Hình dạng: (2,3)
```

```
#chuyển đổi kiểu dữ liệu  
arr = np.array([1,2,3,4,5])  
#chuyển sang kiểu float  
arr_float = arr.astype(float)  
print(arr_float) #[1. 2. 3. 4. 5.]
```

```

#thay đổi hình dạng của mảng
#từ 1 chiều thành nhiều chiều
arr = np.array([1,2 ,3 ,5 ,0,6])
reshaped_arr = arr.reshape(2,3)
print(reshaped_arr) #[[1 2 3] [5 0 6]]
reshaped_arr = arr.reshape(3,2)
print(reshaped_arr) #[[1 2] [3 5] [0 6]]
#từ nhiều chiều thành 1 chiều
reshaped_arr = reshaped_arr.flatten()
print(reshaped_arr) #[1 2 3 5 0 6]

#cắt lát mảng
arr = np.array([1,2,3,4,5,6,7,8])
#cắt từ vị trí 3 đến 5
arr_cut = arr[3:5]
print(arr_cut) #[4 5]
#cắt từ phần tử đầu tiên đến -1 (python cho phép duyệt lùi)
arr_cut = arr[:-1] #từ ptu 0 (là 1) đếm lùi lại thì ptu -1 là 7
print(arr_cut) #[1 2 3 4 5 6 7]
#cắt từ phần -3 đến hết
arr_cut = arr[-3:] #từ phần tử 0 (là 1)đếm lùi lại thì ptu -3 là 6
print(arr_cut) #[6 7 8]

```

```
#chuyển vị (hàng thành cột, cột thành hàng) -> lật mảng lại  
arr = np.array([[1,2,3],[4,5,6]])  
print(arr) #[[1,2,3],[4,5,6]]  
transposed_arr = arr.T #T là chuyển vị  
print(transposed_arr) #[[1 4] [2 5] [3 6]]
```

```
#nối mảng  
arr1 = np.array([1,2,3])  
arr2 = np.array([4,5,6])  
arr= np.concatenate((arr1, arr2)) #nối arr1 và arr2  
print(arr) #[1 2 3 4 5 6]
```

```
# tính toán  
arr = np.array([1,2,3,4,5])  
#tính tổng  
sum_arr = np.sum(arr)  
print(sum_arr) #15  
#tính trung bình  
avg_arr = np.mean(arr)  
print(avg_arr) #3.0  
#tính max  
max_arr = np.max(arr)
```

```
print(max_arr) #5  
#tính min  
min_arr = np.min(arr)  
print(min_arr) #1  
#tính độ lệch chuẩn, đo lường mức độ phân tán của dữ liệu  
std_arr = np.std(arr)  
print(std_arr) #1.4142135623730951  
#tính phương sai, đo lường mức độ biến thiên của dữ liệu  
var_arr = np.var(arr)  
print(var_arr) #2.0  
#tính tổng tích chập  
arr1 = np.array([1,2,3])  
arr2 = np.array([4,5,6])  
dot_product = np.dot(arr1, arr2)  
print(dot_product) #32
```

### 13. Lấy thông tin cơ bản của ảnh bằng numpy

Trong Numpy, hàm `numpy.array()` để tạo 1 mảng đại diện cho hình ảnh. Hình ảnh thường được biểu diễn dưới dạng mảng 3 chiều (cho hình ảnh màu) hoặc 2 chiều (cho hình ảnh xám)

```
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
im = Image.open(my_path)

#chuyển ảnh về array
im_array = np.array(im)
#kiểm tra kích thước ảnh
print(im_array.shape)      #(1053, 699, 3) ảnh màu nên kq là mảng
                            3 chiều
print(im_array.dtype)      #uint8

#chuyển đổi sang ảnh xám
im_array_2 = np.array(im.convert('L'))
print(im_array_2.shape)    #(1053, 699) ảnh xám nên kq là mảng
                            2 chiều
print(im_array_2.dtype)    #uint8
```

## 14. Biến đổi ảnh xám

Ở bài này ta sẽ thấy các phép tính biến đổi mảng có thể làm biến đổi cả ảnh

Thử code như sau:

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
im = Image.open(my_path)
#chuyển đổi sang ảnh xám
image_array = np.array(im.convert('L'))

#thử biến đổi ảnh xám
inverted_image = 255 - image_array #đảo chiều màu (đen -> trắng,
trắng -> đen)
squared_image = image_array **2 #bình phương giá trị của pixel
clamped_image = np.clip(image_array, 100, 200) #giới hạn giá trị
pixel trong khoảng 100-200, < 100 thì thành 100, >200 thì thành 200

#hiển thị ảnh gốc và hình ảnh đã biến đổi với khoảng trắng (4 ảnh)
fig, axs = plt.subplots(2,2, figsize =(10,10)) # tạo khung ảnh chiều
ngang là 2, dọc là 2 và kích thước là 10x10
fig.subplots_adjust(wspace =0.2, hspace= 0.2)

axs[0,0].imshow(image_array,cmap="gray")
axs[0,0].set_title('Ảnh gốc')

axs[0,1].imshow(inverted_image,cmap ="gray")
axs[0,1].set_title('Ảnh đã biến đổi')
```

```
axs[1,0].imshow(squared_image,cmap ="gray")
```

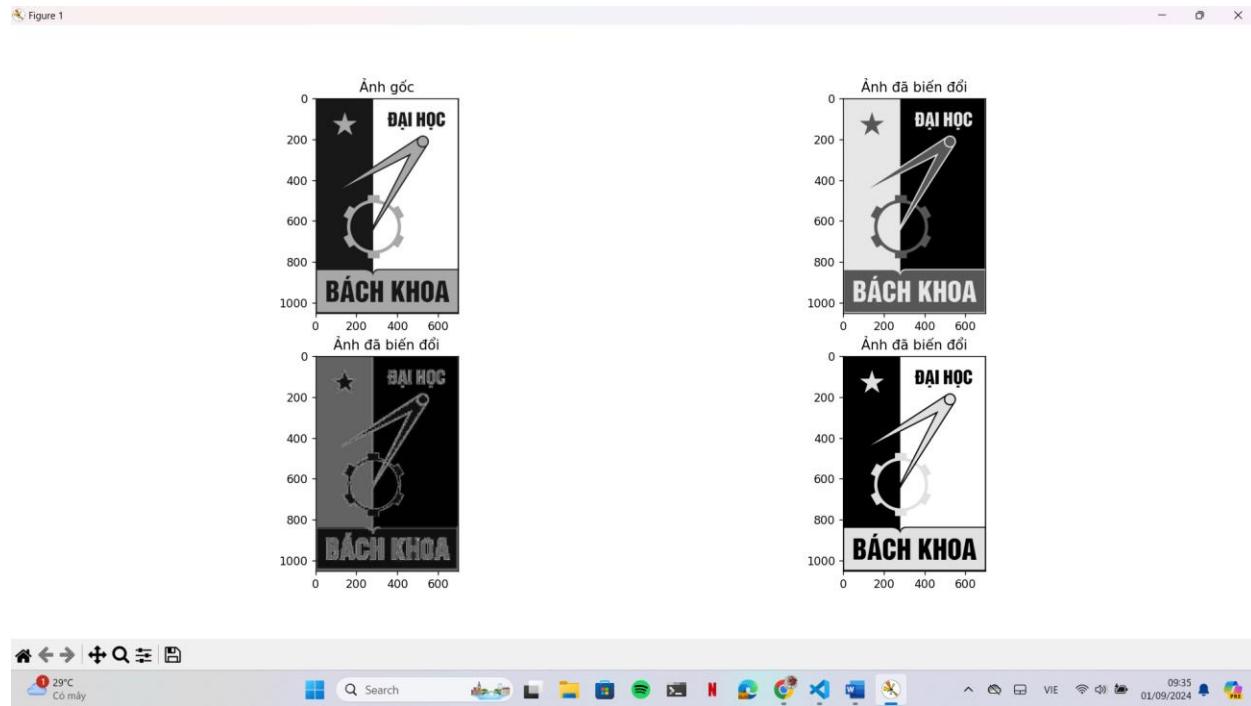
```
axs[1,0].set_title('Ảnh đã biến đổi')
```

```
axs[1,1].imshow(clampes_image,cmap ="gray")
```

```
axs[1,1].set_title('Ảnh đã biến đổi')
```

```
plt.show()
```

Ta thu được kết quả như sau:



## 15. Cân bằng lược đồ ảnh xám

Đặt vấn đề: Trong 1 ảnh có khu vực rất sáng và khu vực rất tối.

Như vậy khi cân bằng lược đồ ảnh xám thì sẽ tăng cường độ tương phản. Một trong những lợi ích quan trọng của cân bằng lược đồ là nó có thể tăng cường tương phản hình ảnh. Bằng cách tái phân phối giá

trị pixel trên toàn phạm vi, cân bằng lược đồ có thể làm cho các chi tiết trong hình ảnh trở nên dễ nhận biết hơn.

Ta sẽ thử code như sau:

```
import numpy as np  
from PIL import Image  
import matplotlib.pyplot as plt
```

```
def histogram_equalization(image, nbr_bins=256):
```

# Đảm bảo rằng ảnh đầu vào là ảnh xám (grayscale).

# Nếu ảnh không phải là ảnh xám (chế độ màu 'L'), thì chuyển đổi nó thành ảnh xám.

# Ảnh xám có các giá trị pixel từ 0 (màu đen) đến 255 (màu trắng).

```
if image.mode != 'L':
```

```
    image = image.convert('L')
```

# Chuyển đổi hình ảnh từ định dạng của thư viện PIL thành một mảng numpy.

# Mảng này chứa các giá trị pixel của ảnh, mỗi giá trị là một số nguyên từ 0 đến 255.

```
image_array = np.array(image)
```

# Tính toán histogram của ảnh.

# Histogram là một biểu đồ biểu diễn sự phân bố tần suất của các giá trị pixel trong ảnh.

# Hàm np.histogram() trả về hai giá trị:

# - 'histogram': chứa số lượng pixel ở mỗi mức độ xám (trong khoảng từ 0 đến 255).

# - 'bins': là các khoảng giá trị tương ứng với mỗi bin trong histogram (ở đây chia làm 'nbr\_bins' bin).

```
histogram, bins = np.histogram(image_array, bins=nbr_bins,  
range=(0, 256), density=True)
```

# Tính toán hàm phân phối tích lũy (CDF) từ histogram.

# CDF cho biết xác suất tích lũy rằng một pixel sẽ có giá trị nhỏ hơn hoặc bằng một giá trị nhất định.

# Hàm cumsum() tính tổng tích lũy của histogram, tức là cộng dồn các giá trị lại với nhau.

```
cdf = histogram.cumsum()
```

# Chuẩn hóa CDF để các giá trị nằm trong khoảng từ 0 đến 255 (tương ứng với mức độ xám trong ảnh).

# Việc chuẩn hóa này đảm bảo rằng khi áp dụng CDF lên ảnh, giá trị pixel mới cũng sẽ nằm trong khoảng từ 0 đến 255.

```
cdf = 255 * cdf / cdf[-1] # cdf[-1] là giá trị cuối cùng của CDF, tương  
đương với tổng số pixel trong ảnh.
```

```
# Ánh xạ các giá trị pixel ban đầu trong 'image_array' sang các giá trị mới dựa trên CDF.
```

```
# Hàm np.interp() thực hiện nội suy (interpolation) để tìm giá trị mới cho từng pixel dựa trên bins và CDF.
```

```
# 'bins[:-1]' là các giá trị bin trừ đi giá trị cuối cùng (vì bins có nhiều hơn một phần tử so với histogram).
```

```
image_equalized = np.interp(image_array, bins[:-1], cdf)
```

```
# Chuyển đổi mảng numpy sau khi cân bằng histogram thành một hình ảnh.
```

```
# Hàm astype('uint8') chuyển đổi các giá trị pixel về kiểu số nguyên 8 bit (giá trị từ 0 đến 255).
```

```
image_equalized =  
Image.fromarray(image_equalized.astype('uint8'))
```

```
# Trả về hình ảnh đã được cân bằng histogram.
```

```
return image_equalized
```

```
my_path = 'C:/computer_vision/img/sang.jpg'
```

```
im = Image.open(my_path)
```

```
# áp dụng cân bằng lược đồ ảnh xám
```

```
equalized_image = histogram_equalization(im)
```

```
# tạo một lưới 2x2 để hiển thị hình ảnh và biểu đồ
```

```
plt.figure(figsize=(15,10))
```

```
#hiển thị ảnh gốc
plt.subplot(2,2,1)
plt.imshow(im.convert('L'), cmap='gray')
plt.title('Ảnh gốc')

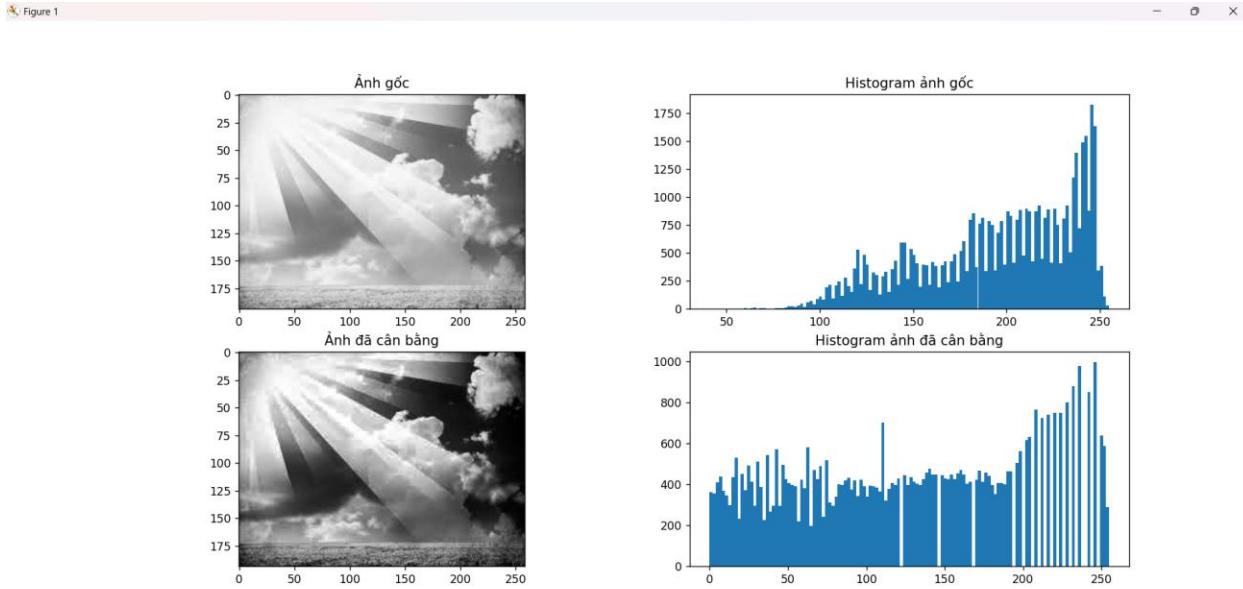
# vẽ biểu đồ histogram của ảnh gốc
plt.subplot(2,2,2)
plt.hist(np.array(im.convert('L')).flatten(), bins= 128)
plt.title('Histogram ảnh gốc')

#hiển thị ảnh sau khi cân bằng
plt.subplot(2,2,3)
plt.imshow(equalized_image, cmap='gray')
plt.title('Ảnh đã cân bằng' )

#vẽ biểu đồ histogram của hình ảnh đã cân bằng
plt.subplot(2,2,4)
plt.hist(np.array(equalized_image).flatten(), bins= 128)
plt.title('Histogram ảnh đã cân bằng')

plt.show()
```

Kết quả thu được:



Quan sát ảnh gốc, ta thấy các điểm sáng quá nhiều dẫn đến ảnh bị mất cân bằng, lược đồ histogram cũng lệch hẳn về bên sáng.

Sau khi cân bằng thì ảnh đã có độ tương phản rõ ràng hơn, biểu đồ histogram cũng cân bằng sáng tối hơn.

## 16. Tạo ảnh mới bằng cách tính trung bình từ nhiều ảnh

$$\forall_{i,j} \text{OutImage}_{i,j} = \frac{\text{InImage1}_{i,j} + \text{InImage2}_{i,j}}{2}$$

Cộng ảnh là cộng từng giá trị của các điểm ảnh, như vậy để cộng thì hai ảnh phải có cùng kích cỡ. Ví dụ:



Hình tạo ra (dưới cùng) chứa cả 2 chấm từ 2 ảnh đã tính trung bình. Vật thể vẫn rõ nhưng 2 điểm nhiễu(2 dấu chấm) đã mờ đi.

**Ứng dụng vào thực tế:** Khi chụp ảnh, tay ta sẽ luôn có độ rung nhất định làm ảnh bị nhiễu. Nhưng tại sao ảnh chụp ra vẫn nét như vậy ? Đó là do khi nhấn 1 lần nút chụp máy đã chụp rất nhiều tấm ảnh và trung bình nó để ra 1 ảnh cuối cùng.

Tính trung bình từ tập ảnh:

- + Tính trung bình từ rất nhiều hình ảnh có cùng kích thước để làm giảm nhiễu hình ảnh là một phương pháp gọi là “kết hợp ảnh”. Quá trình này kết hợp nhiều hình ảnh vào 1 hình ảnh duy nhất bằng cách tính trung bình các giá trị pixel tương ứng từ nhiều ảnh.

- + Giảm nhiễu: khi nhiễu ngẫu nhiên xuất hiện trong cách hình ảnh, tính trung bình giúp giảm nhiễu và cải thiện chất lượng hình ảnh bằng cách làm cho ngẫu nhiên trở nên ít quan trọng hơn trong hình ảnh kết quả.

+ Nâng cao chất lượng hình ảnh: Quá trình kết hợp ảnh có thể được sử dụng để tạo ra hình ảnh có chất lượng tốt hơn trong các ứng dụng thiên văn học, hình ảnh y tế và nhiếp ảnh.

```
import numpy as np
from PIL import Image

#xây dựng hàm tính trung bình ảnh
def average_images(image_list):      #hàm nhập vào 1 danh sách ảnh
    total_array = np.array(Image.open(image_list[0]), 'f')  #ban đầu total_array là ảnh đầu tiên
    count= 1  #biến chỉ số lượng ảnh
    for image_path in image_list[1:]:
        try:
            image_array = np.array(Image.open(image_path),'f')
            #'f' là tham số chỉ định kiểu dữ liệu của các phần tử trong mảng NumPy.
            #'f' là viết tắt của float32, tức là mỗi giá trị pixel trong mảng sẽ được lưu trữ dưới dạng số thực (32-bit floating-point).
            #Kiểu float32 thường được dùng khi cần thực hiện các phép tính toán có độ chính xác cao trên các giá trị pixel (ví dụ như xử lý ảnh phức tạp hoặc áp dụng các bộ lọc ảnh).
            total_array += image_array  #cộng hết phần tử trên mảng lại với nhau
        count += 1
```

```
except:
```

```
    print("Skip ", image_path)
```

```
average_array= total_array/count #tính trung bình  
average_image =  
Image.fromarray(average_array.astype('uint8')) #chuyển đổi kết  
quả trung bình thành hình ảnh  
return average_image
```

```
#sử dụng hàm
```

```
#lấy 2 ảnh cùng kích cỡ
```

```
my_path_1 = 'C:/computer_vision/img/1.png'
```

```
my_path_2 = 'C:/computer_vision/img/2.png'
```

```
image_list = [my_path_1,my_path_2] #danh sách ảnh tính  
trung bình
```

```
result_image = average_images(image_list)
```

```
result_image.show()
```

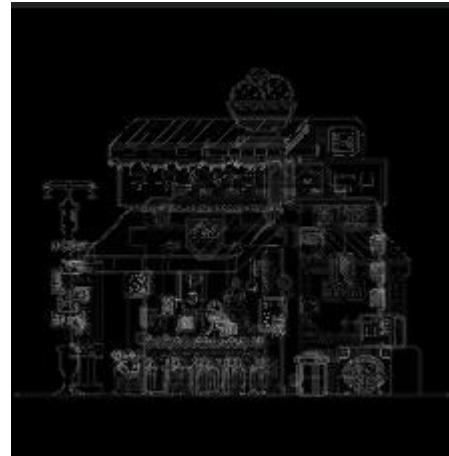


Kết quả:



và

Thành



## 17. Giới thiệu về thư viện Scipy

Scipy là thư viện mã nguồn mở (cho phép cộng đồng cùng tham gia đóng góp) mà ngôn ngữ lập trình Python dùng cho tính toán khoa học và kĩ thuật, được xây dựng từ Numpy.

Nó cung cấp một tập hợp rộng rãi các chức năng và công cụ để giải quyết các vấn đề phức tạp trong nhiều lĩnh vực, từ toán học cho đến khoa học dữ liệu.

```
import numpy as np
```

```
#giải hệ phương trình tuyến tính
from scipy import linalg
#định nghĩa hệ phương trình tuyến tính
a = np.array([[2,1],[3,2]])
b = np.array([5,7])
# như vậy ta đã có hệ  $2x+y=5$ ,  $3x+2y=7$  và giải = pp ma trận
#giải hệ pp tuyến tính
kq_hpt= linalg.solve(a,b) #dùng solve
```

```

print("kết quả:", kq_hpt)  #kết quả: [ 3. -1.] -> x=3, y=-1

#tính tích phân của 1 hàm số
from scipy import integrate
#định nghĩa hàm f(x) = x^2
def my_func(x):
    return x**2

#tính tích phân của f(x) từ 0 đến 1
kq_tp= integrate.quad(my_func,0,1)  #dùng quad
print("kết quả:", kq_tp)  #kết quả: (0.3333333333333337,
3.700743415417189e-15) -> 0.333...7(tính đến 10^-15)

#tính giá trị riêng và vector riêng của ma trận
a = np.array([[2,1],[3,2]])
#vẫn sử dụng linalg
evals, evecs= linalg.eig(a)  #dùng eig
print("giá trị riêng:", evals) #giá trị riêng: [3.73205081+0.j
0.26794919+0.j]
print("vector riêng:", evecs)  #vector riêng: [[ 0.5      -0.5      ]
#[ 0.8660254  0.8660254]]

```

## 18. Bộ lọc Gaussian làm mờ ảnh

Bộ lọc(mặt nạ) Gaussian (gọi tắt là Gauss) hay kernel Gaussian, là một loại bộ lọc được sử dụng trong xử lí ảnh để làm mờ hoặc làm trơn hình ảnh.

Nó dựa trên phân phối Gaussian (phân phối chuẩn) và được sử dụng để tạo ra hiệu ứng làm mờ trên ảnh bằng cách giảm độ biến đổi giữa các điểm ảnh gần nhau.

Mặt nạ Gaussian thường được áp dụng để xử lí ảnh kĩ thuật số và chúng thường được sử dụng trong các ứng dụng như giảm nhiễu hoặc làm mịn ảnh.

Công thức:

- Công thức của mặt nạ Gaussian được tính dựa trên hàm Gaussian, và công thức chung cho một mặt nạ Gaussian 2D là:
  - $G(x,y)$  là giá trị tại điểm  $(x,y)$  của mặt nạ Gaussian.
  - $\sigma$  là tham số đặc trưng cho độ rộng của phân phối Gaussian (được gọi là độ lệch chuẩn).
  - $x$  và  $y$  là tọa độ của điểm ảnh trong mặt nạ.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Công thức này cho ta thấy giá trị của bộ lọc tại từng điểm ảnh trong mặt nạ. Đối với 1 Gauss 2D, giá trị tại các điểm xa tâm của mặt nạ sẽ nhỏ hơn, trong khi giá trị tại các điểm gần tâm sẽ lớn hơn.

Khi bộ lọc Gauss áp dụng lên ảnh, nó sẽ tạo ra hiệu ứng làm mờ bằng cách tính trung bình trọng số của các điểm ảnh trong cửa sổ của bộ lọc, với các điểm ảnh gần tâm thì trọng số lớn hơn. Điều này tạo ra hiệu ứng làm mờ trên ảnh một cách mượt mà và tạo sự mịn màng cho hình ảnh. Công thức và tham số  $\sigma$  có thể được điều chỉnh để kiểm soát mức độ làm mờ trong ảnh xử lý.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from scipy import ndimage

image_path = 'C:/computer_vision/img/hust.jpg'
image = Image.open(image_path)
gray_image = image.convert('L') # Chuyển ảnh sang ảnh xám
image_a = np.array(gray_image) # Chuyển ảnh xám thành mảng

# Làm mờ ảnh với bộ lọc Gauss với các giá trị sigma khác nhau
blurred_image_a = ndimage.gaussian_filter(image_a, sigma=1,
mode="constant")
blurred_image_b = ndimage.gaussian_filter(image_a, sigma=3,
mode="constant")
blurred_image_c = ndimage.gaussian_filter(image_a, sigma=5,
mode="constant")

# Hiển thị ảnh bằng cách sử dụng subplots
fig, axs = plt.subplots(2, 2, figsize=(10, 10))

# Hiển thị ảnh gốc xám
axs[0, 0].imshow(gray_image, cmap='gray')
```

```
axs[0, 0].set_title('Ảnh xám gốc')

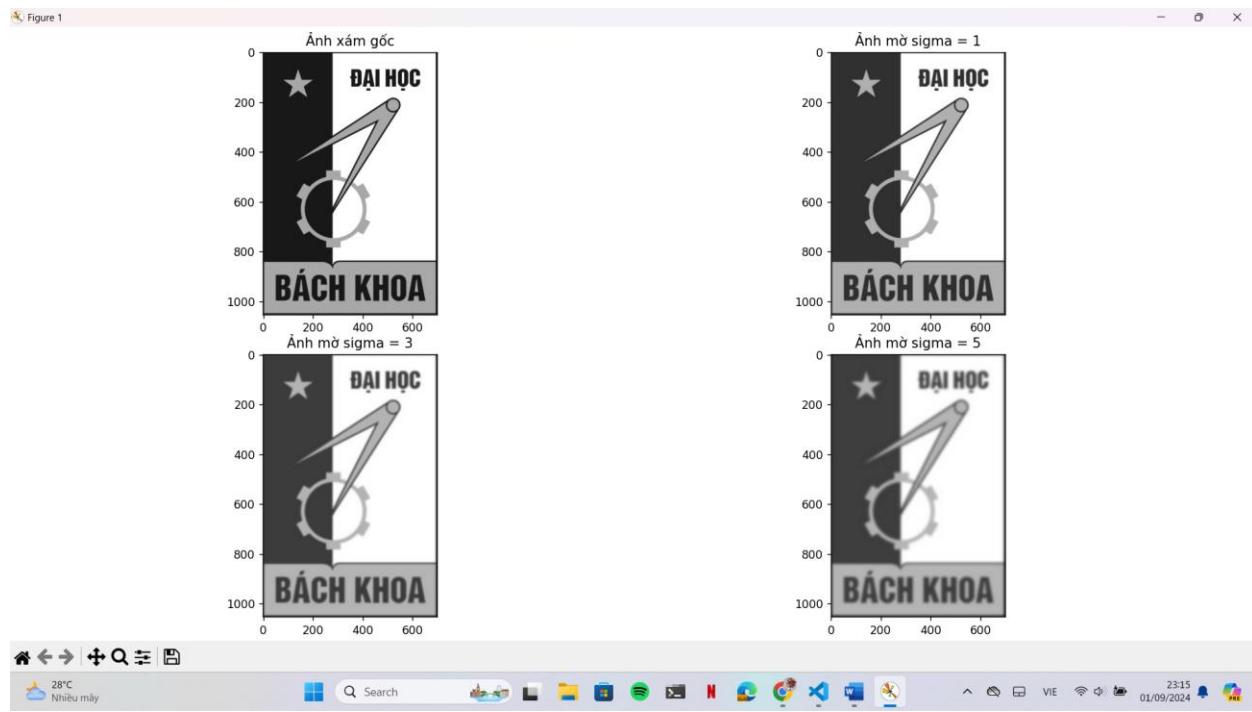
# Hiển thị các ảnh mờ với các giá trị sigma khác nhau
axs[0, 1].imshow(blurred_image_a, cmap='gray')
axs[0, 1].set_title('Ảnh mờ sigma = 1')

axs[1, 0].imshow(blurred_image_b, cmap='gray')
axs[1, 0].set_title('Ảnh mờ sigma = 3')

axs[1, 1].imshow(blurred_image_c, cmap='gray')
axs[1, 1].set_title('Ảnh mờ sigma = 5')

# Điều chỉnh bố cục và hiển thị hình ảnh
plt.tight_layout() #đảm bảo các subplots sẽ không bị chồng chéo và
# khớp vừa vặn trong cửa sổ figure
plt.show()
```

Kết quả:



Sigma càng tăng thì ảnh càng mờ.

## 19. Các bộ lọc đạo hàm

Trong lĩnh vực xử lý ảnh và thị giác máy tính, tính đạo hàm của ảnh (image derivatives) là một phần quan trọng của nhiều ứng dụng, bao gồm phát hiện biên, xác định cạnh và nhiều nhiệm vụ khác.

Các bộ lọc đạo hàm (derivative filters) là các mặt nạ được sử dụng trong xử lý ảnh để tính đạo hàm của ảnh. Chúng thường được áp dụng để tìm sự biến đổi của cường độ màu sắc tại mỗi điểm ảnh và thường được sử dụng cho các ứng dụng như phát hiện biên, xác định cạnh và nhiều tác vụ khác trong lĩnh vực thị giác máy tính và xử lý ảnh.

Trong thực tế có rất nhiều bộ lọc, nhưng ở bài này ta chỉ xét 2 bộ lọc sau

## Bộ lọc Sobel

Sobel

- Bộ lọc Sobel sử dụng hai mặt nạ (mặt nạ Sobel x và mặt nạ Sobel y) để tính đạo hàm riêng theo hướng ngang và dọc. Chúng thường được sử dụng để phát hiện cạnh trong ảnh.

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

Để sử dụng Sobel, ta code như sau:

```
from PIL import Image  
import numpy as np  
import matplotlib.pyplot as plt  
from scipy import ndimage
```

```
image_path = 'C:/computer_vision/img/hust.jpg'  
image = Image.open(image_path)  
gray_image = image.convert('L') # Chuyển ảnh sang ảnh xám  
để dễ quan sát độ mờ  
image_a = np.array(gray_image) # Chuyển ảnh xám thành  
mảng  
  
#Bộ lọc sobel
```

```
#tính gradient theo hướng x
gradient_x = ndimage.sobel(image_a, axis= 0, mode
='constant')

#tính gradient theo hướng y
gradient_y = ndimage.sobel(image_a, axis= 1, mode
='constant')

#hiển thị ảnh
fig, axs = plt.subplots(2,2,figsize=(10,8))

axs[0,0].imshow(image, cmap='gray')
axs[0,0].set_title('Ảnh gốc')

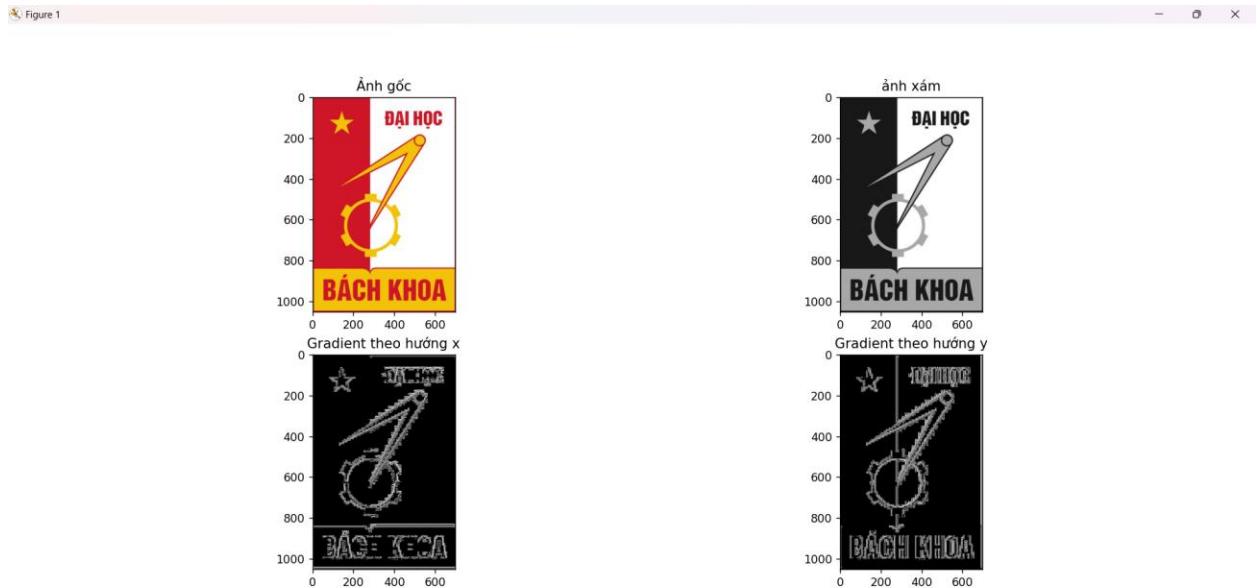
axs[0,1].imshow(gray_image, cmap='gray')
axs[0,1].set_title('Ảnh xám')

axs[1,0].imshow(gradient_x, cmap='gray')
axs[1,0].set_title('Gradient theo hướng x')

axs[1,1].imshow(gradient_y, cmap='gray')
axs[1,1].set_title('Gradient theo hướng y')

plt.show()
```

Kết quả quan sát được:



Như vậy, bộ lọc sẽ làm rõ đường biên và viền theo ox hoặc oy.

Ta sẽ xét thêm 1 bộ lọc nữa như sau:

## Bộ lọc Prewitt

Prewitt

- Bộ lọc Prewitt là một trong những bộ lọc đạo hàm được sử dụng rộng rãi trong xử lý ảnh và thị giác máy tính để tính đạo hàm của ảnh.
- Nó là một phần quan trọng của việc phát hiện biên và cạnh trong ảnh. Bộ lọc Prewitt được thiết kế để tìm sự biến đổi của cường độ màu sắc tại mỗi điểm ảnh trong hình ảnh.

1	0	-1
1	0	-1
1	0	-1

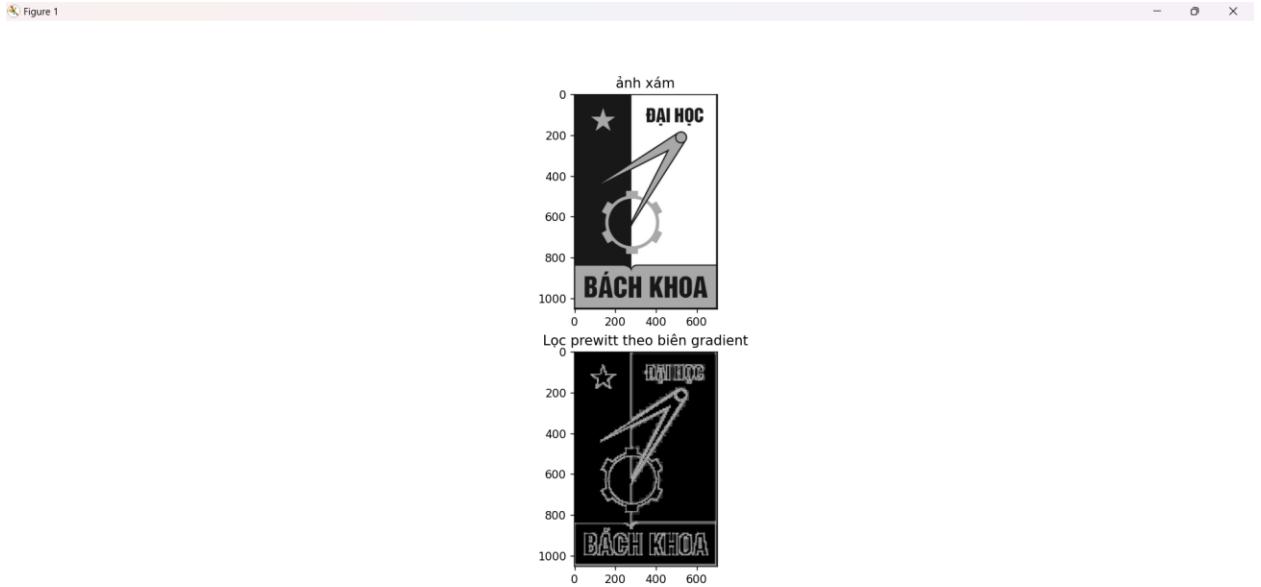
1	1	1
0	0	0
-1	-1	-1

Để sử dụng Prewitt ta code như sau:

#bộ lọc prewitt

```
#các ma trận này là công thức của bộ lọc prewitt  
#tính gradient theo hướng x(đạo hàm riêng theo x)  
gradient_x = ndimage.convolve(image_a, np.array([[-1,0,1],[-1,  
0,1],[-1,0,1]]))  
#tính gradient theo hướng y(đạo hàm riêng theo y)  
gradient_y = ndimage.convolve(image_a, np.array([[-1,-1,-  
1],[0,0,0],[1,1,1]]))  
#tính biên độ gradient  
gradient_magnitude = np.sqrt(gradient_x**2 + gradient_y**2)  
#hiển thị ảnh  
fig, axs = plt.subplots(2,1,figsize=(10,8))  
axs[0].imshow(gray_image, cmap='gray')  
axs[0].set_title('ảnh xám')  
axs[1].imshow(gradient_magnitude, cmap='gray')  
axs[1].set_title('Lọc prewitt theo biên gradient')  
plt.show()
```

Kết quả thu được như sau:



Qua đó, ảnh sẽ có điểm sáng và điểm tối, điểm sáng sẽ đóng vai trò như đường viền của ảnh.

Ta có thể tìm hiểu thêm nhiều bộ lọc khác như: Edge, Box Blur, Square, Gauss, Sharpen, Laplacian,....

## 20. Xử lý ảnh hình thái và đếm số lượng đối tượng

Hình thái học (morphology) là một phần quan trọng của xử lý ảnh, tập trung vào việc xử lý hình dạng và cấu trúc của đối tượng trong ảnh.

Trong hình thái học, ta làm việc với ảnh nhị phân, trong đó các điểm ảnh có giá trị 0 (đen tuyệt đối) hoặc 1 (trắng tuyệt đối).

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from scipy import ndimage
```

```
from PIL import Image
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
im= Image.open(my_path)
gray_image = im.convert('L') #chuyển về ảnh xám
image_array = np.array(gray_image) #chuyển sang mảng

#chuyển sang ảnh nhị phân
binary_image = 1*(image_array>200) #điểm nào lớn hơn 200 thì
quay về 1, ngược lại thì về 0 -> vật thể thành đen, nền trắng
#ngược lại, nếu 1*(image_array<200) thì vật thể trắng nền đen
#mốc 200 có thể thay đổi theo ý mình
#đánh nhãn
labeled_image, num_features = ndimage.label(binary_image)
#tạo lưới subplot với 3 dòng và 1 cột
fig, axs = plt.subplots(2, 2, figsize=(50, 50))
# ảnh gốc
axs[0, 0].imshow(im)
axs[0, 0].set_title('Hình ảnh gốc')
# ảnh xám
axs[0, 1].imshow(gray_image, cmap='gray')
axs[0, 1].set_title('Ảnh xám')
#hình ảnh nhị phân
```

```

axs[1, 0].imshow(binary_image, cmap='gray')
axs[1, 0].set_title('Hình ảnh nhị phân')

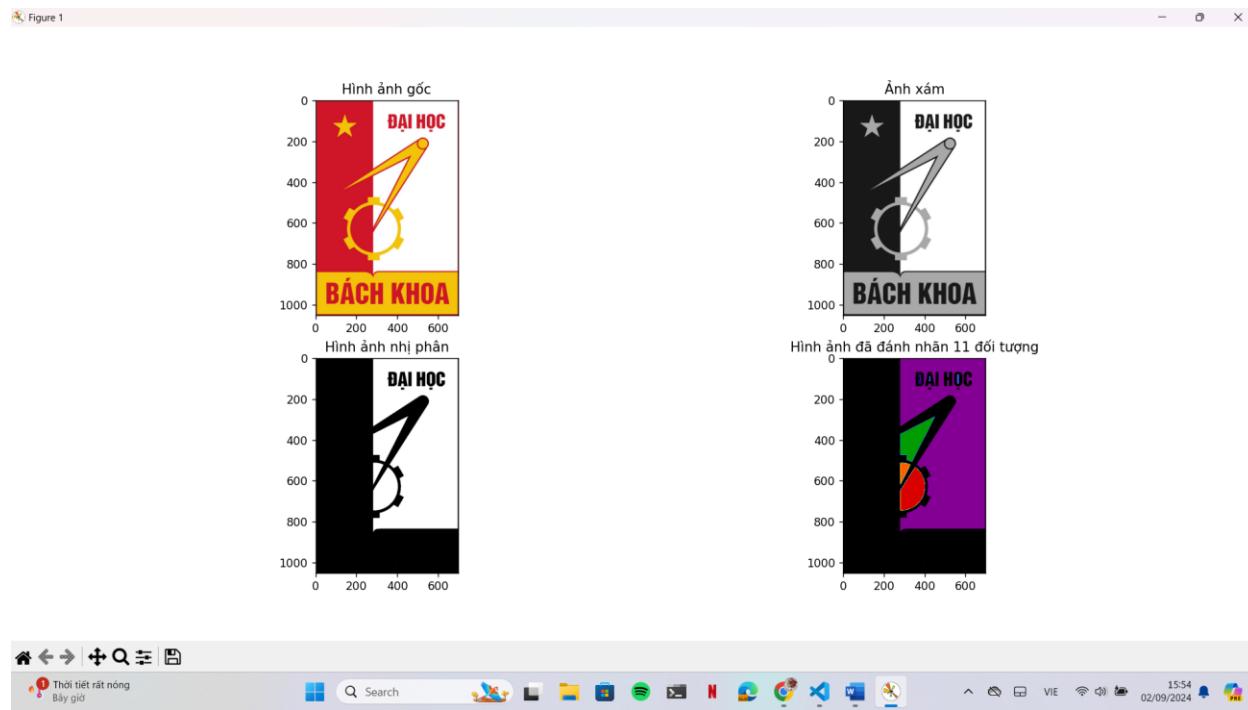
#hình ảnh đã đánh nhãn

axs[1, 1].imshow(labeled_image,
cmap='nipy_spectral') #nipy_spectral tô mỗi đối tượng 1 màu
axs[1, 1].set_title(f'Hình ảnh đã đánh nhãn {num_features} đối
tượng')

# f'Hình ảnh... thể hiện num_features là kiểu float
plt.show()

```

Kết quả thu được như sau:



Có 11 đối tượng được đánh nhãn (ứng với màu đen trong ảnh nhị phân). Kết quả này không được chính xác lắm.

Các phép toán hình thái học cơ bản:

- Erosion (co ngắn): Thu nhỏ kích thước của đối tượng trong ảnh.
- Dilation (co giãn) : Mở rộng kích thước của đối tượng trong ảnh.
- Opening (Mở) : thực hiện erosion sau đó dilation, giúp loại bỏ nhiễu và nối các đối tượng.
- Closing (Đóng): thực hiện dilation sau đó erosion, giúp lấp đầy lỗ và loại bỏ nhiễu.

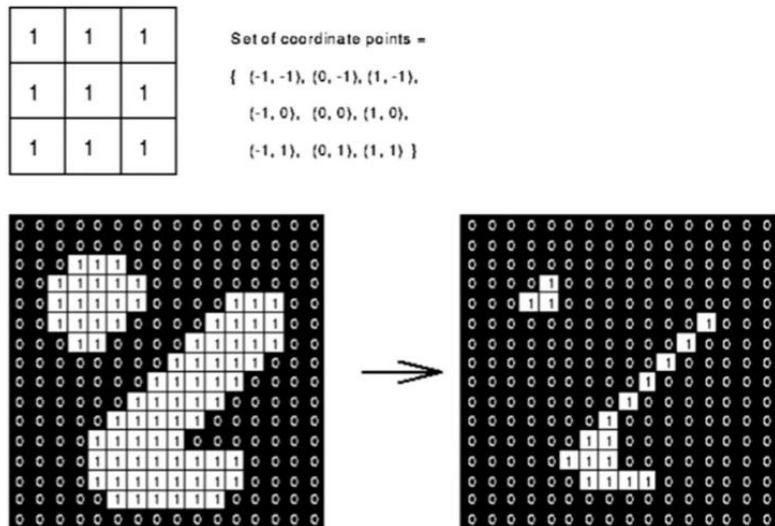
Tham khảo thêm :

<https://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm>

## Erosion

Phép toán Erosion thường được thực hiện bằng cách trượt một cấu trúc kernel (hay còn gọi là mask) trên mảng và kiểm tra xem tất cả các điểm trong kernel có giá trị 1 không.

Nếu ít nhất một điểm là 0, thì điểm tương ứng trên ảnh kết quả sẽ được đặt thành 0. Nếu tất cả điểm trong kernel đều là 1, thì điểm tương ứng trên ảnh kết quả sẽ được đặt thành 1.



Hiểu nôm na là nếu *toàn bộ cấu trúc* (ô vuông 3x3) đều là 1 thì ô trung tâm sẽ là 1, nếu có *bất kì* một số 0 nào tồn tại trên cấu trúc thì ô trung tâm sẽ bằng 0. Ta sẽ trượt dần cấu trúc cho đến hết ảnh (ví dụ mẫu) để biến đổi thành như trên.

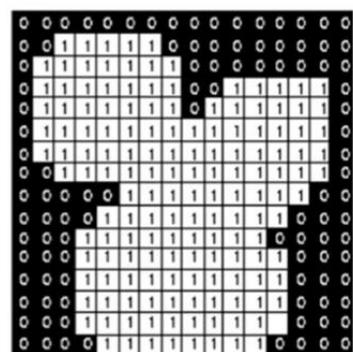
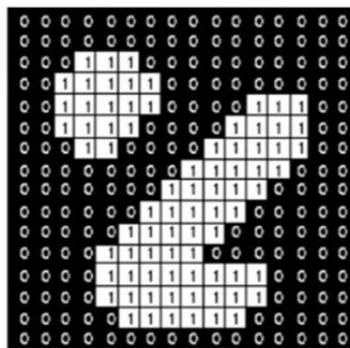
## Dilation

Phép toán Dilation cũng trượt kernel trên mảng, nhưng điểm tương ứng trên ảnh kết quả sẽ được đặt thành 1 nếu có ít nhất một điểm 1 trong kernel. Nếu tất cả điểm trong kernel đều là 0, thì điểm tương ứng trên ảnh kết quả sẽ được đặt thành 0.

1	1	1
1	1	1
1	1	1

Set of coordinate points =

{ (-1, -1), (0, -1), (1, -1),  
(-1, 0), (0, 0), (1, 0),  
(-1, 1), (0, 1), (1, 1) }



Phép toán này ngược lại, chỉ cần có số 1 thì điểm trung tâm cấu trúc sẽ là 1, nếu toàn bộ cấu trúc là 0 thì điểm trung tâm cấu trúc là 0.

## Opening

Opening = (Erosion → Dilation)

Opening thường được sử dụng để loại bỏ nhiễu và thu nhỏ đối tượng trong ảnh, chẳng hạn như việc loại bỏ đặc trưng nhỏ không cần thiết từ ảnh X-quang.

# Closing

## Opening = (Dilation → Erosion)

Closing thường được sử dụng để lấp đầy lỗ trong đối tượng và loại bỏ các phần nhiễu trong ảnh, chẳng hạn như việc xử lý ảnh y khoa để phát hiện các khối u.

Ta sẽ thử dùng kĩ thuật opening để quan sát sự khác nhau:

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
from scipy import ndimage
```

```
from PIL import Image
```

```
my_path = 'C:/computer_vision/img/hust.jpg'
```

```
im= Image.open(my_path)
```

```
gray_image = im.convert('L') #chuyển về ảnh xám
```

```
image_array = np.array(gray_image) #chuyển sang mảng
```

```
#chuyển sang ảnh nhị phân
```

```
binary_image = 1*(image_array>200) #điểm nào lớn hơn 200 thì  
quay về 1, ngược lại thì về 0 -> vật thể thành đen, nền trắng
```

```
# ngược lại, nếu 1*(image_array<200) thì vật thể trắng nền đen  
#mốc 200 có thể thay đổi theo ý mình
```

```
# thực hiện opening để loại bỏ nhiễu và nối các đối tượng  
# thực hiện closing thì là binary_closing  
opened_image = ndimage.binary_opening(binary_image, structure =  
np.ones((3,3)))  
# sử dụng cấu trúc 3x3 để thực hiện, tất nhiên cấu trúc này là tùy  
mình quy định, ví dụ 9,5  
# ta sẽ tùy chỉnh cấu trúc sao cho chính xác nhất
```

```
# đánh nhãn  
labeled_image, num_features = ndimage.label(binary_image)  
# đánh nhãn ảnh sau khi thực hiện opening  
labeled_opening_image, num_opening_features =  
ndimage.label(opened_image)
```

```
# tạo lưới subplot với 3 dòng và 1 cột  
fig, axs = plt.subplots(2, 2, figsize=(50, 50))  
# ảnh xám  
axs[0, 0].imshow(gray_image, cmap='gray')  
axs[0, 0].set_title('Ảnh xám')  
# hình ảnh nhị phân
```

```
axs[0, 1].imshow(binary_image, cmap='gray')
axs[0, 1].set_title('Hình ảnh nhị phân')

#hình ảnh đã đánh nhãn

axs[1, 0].imshow(labeled_image,
cmap='nipy_spectral') #nipy_spectral tô mỗi đối tượng 1 màu
axs[1, 0].set_title(f'Hình ảnh đã đánh nhãn {num_features} đối
tượng')

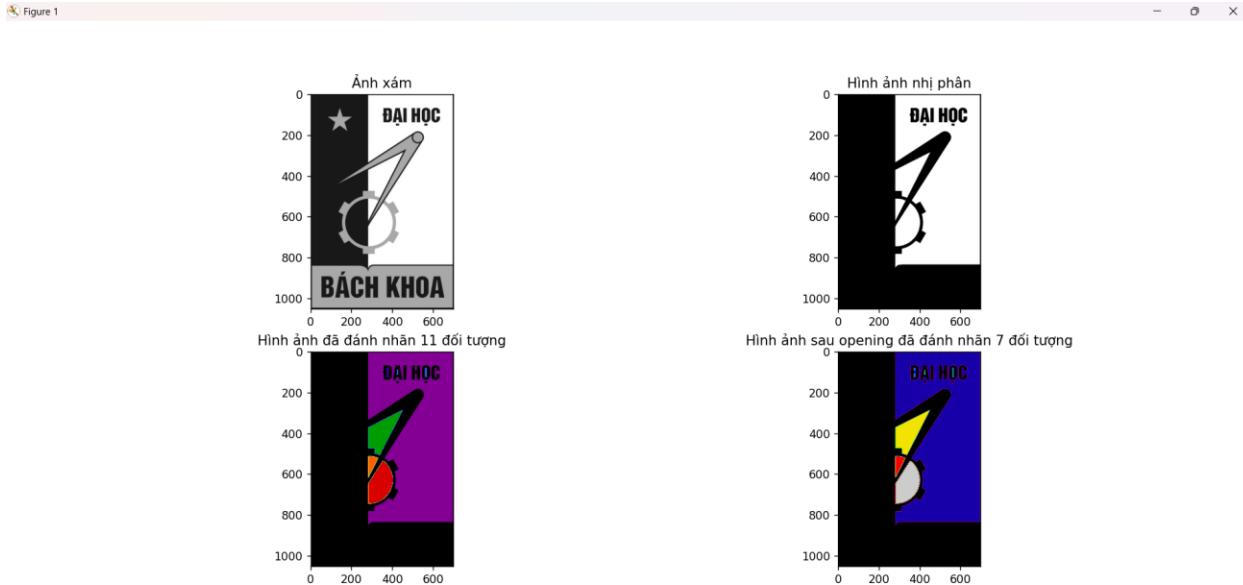
# f'Hình ảnh... thể hiện num_features là kiểu float

#hình ảnh sau khi thực hiện opening đã đánh nhãn

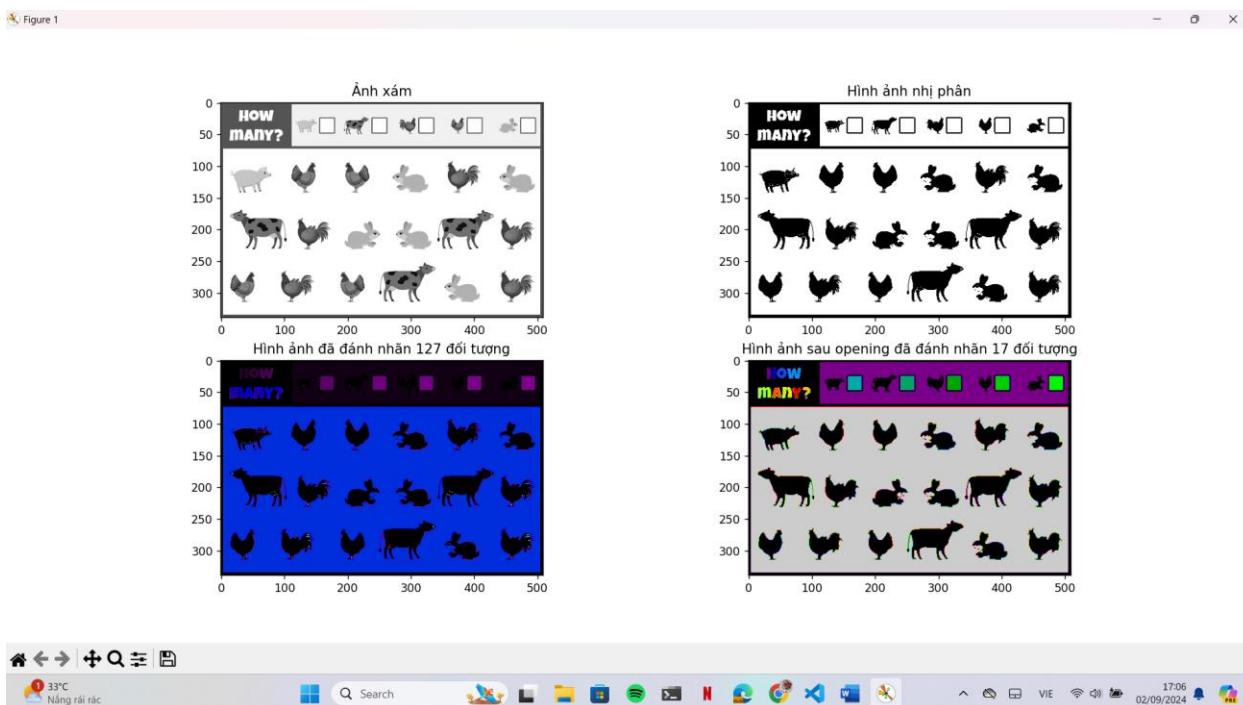
axs[1, 1].imshow(labeled_opening_image,
cmap='nipy_spectral') #nipy_spectral tô mỗi đối tượng 1 màu
axs[1, 1].set_title(f'Hình ảnh sau opening đã đánh nhãn
{num_opening_features} đối tượng')

plt.show()
```

Kết quả thu được:



Số lượng đối tượng được nhận diện đã sát hơn. Hãy thử với hình ảnh khác để thấy rõ sự ưu việt :



## 21. Thư viện OpenCv và các thao tác cơ bản

Ta tiến hành download bằng pip install opencv-python.

Một số thao tác cơ bản :

- Đọc ảnh: img= cv2.imread('path')
- In thông tin: print(img)
- Kích thước ảnh: img.shape
- Hiển thị hình ảnh: cv2.imshow('title', img)
- Tạm dừng và chờ: cv2.waitKey(time – milliseconds) -> chương trình tạm dừng 1 khoảng thời gian để chờ 1 phím được nhấn
- Hủy khung ảnh: cv2.destroyAllWindows()
- Lưu hình ảnh: cv2.imwrite('path',img)

Import cv2 để sử dụng thư viện

```
import cv2 #import open CV
```

```
#đọc một ảnh  
path='C:/computer_vision/img/hust.jpg'  
img=cv2.imread(path)  
  
#in thông tin ảnh  
print(img)      #kết quả là bảng 3 chiều  
  
#lấy kích thước và in ra  
print(img.shape)    #(1053, 699, 3)  
  
x, y,z = img.shape  
  
print(x)    #1053  
print(y)    #699  
print(z)    #3
```

```
#hiển thị ảnh
```

```
cv2.imshow('hust',img) #title là hust
```

```
# chờ 1 khoảng thời gian
```

```
cv2.waitKey(10000) # ảnh hiển thị trong 10s
```

```
# đóng cửa sổ sau 10s
```

```
cv2.destroyAllWindows() #đóng cửa sổ tên hust
```

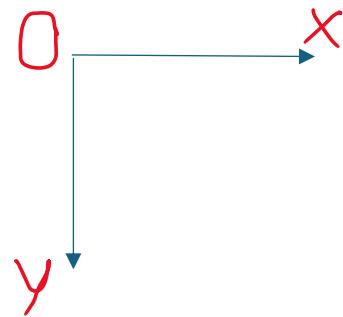
Hãy ôn tập lại 1 chút về pixel:

- Pixel (viết tắt của picture element) trong ảnh là đơn vị cơ bản nhỏ nhất của một hình ảnh kỹ thuật số.
- Mỗi pixel đại diện cho một màu sắc hoặc độ sáng cụ thể tại một vị trí cụ thể trên một hình ảnh.
- Các pixel kết hợp lại tạo nên hình ảnh hoàn chỉnh.



Độ phân giải của một hình ảnh được đo bằng số lượng pixel theo chiều ngang và chiều dọc.

Ví dụ, một hình ảnh có độ phân giải 1920x1080 có tổng cộng 1.920 pixel theo chiều ngang và 1.080 pixel theo chiều dọc.

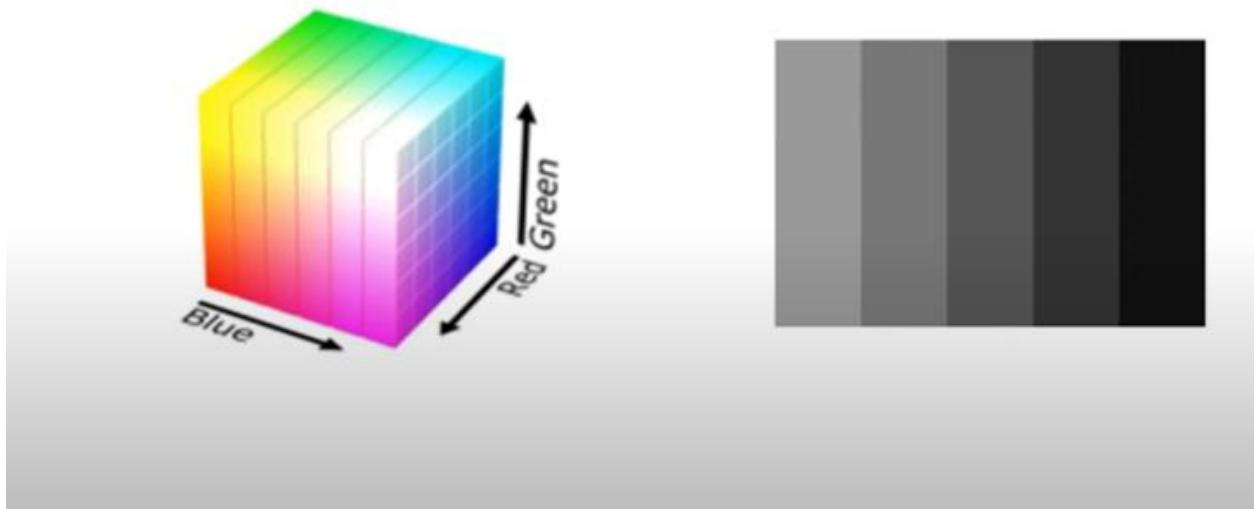


Mỗi pixel thường được biểu thị bằng các giá trị số, đại diện cho màu sắc hoặc độ sáng của nó.

Trong hình ảnh RGB (Red-Green-Blue), mỗi pixel có thể được biểu thị bằng ba giá trị số, tương ứng với màu đỏ, xanh lá cây và xanh dương.

Đối với hình ảnh đen trắng, mỗi pixel chỉ có giá trị độ sáng (từ 0->255)

## Color



1 số mã màu:

- RGB (255, 0, 0) - Màu đỏ đậm
- RGB (0, 255, 0) - Màu xanh lá cây đậm
- RGB (0, 0, 255) - Màu xanh dương đậm
- RGB (255, 255, 0) - Màu vàng
- RGB (255, 0, 255) - Màu magenta
- RGB (0, 255, 255) - Màu cyan
- RGB (128, 128, 128) - Màu xám (màu trung tính)
- RGB (255, 255, 255) - Màu trắng
- RGB (0, 0, 0) - Màu đen
- RGB (128, 0, 128) - Màu tím

⇒ Như vậy ta có thể biểu diễn tới  $256^3$  mã màu khác nhau.

Tách và chuyển đổi hệ màu của ảnh:

#tách màu

```
b,g,r = cv2.split(img)  
cv2.imshow('base',img)  
cv2.imshow('blue',b)  
cv2.imshow('green',g)  
cv2.imshow('red',r)
```

# chờ 1 khoảng thời gian

```
cv2.waitKey(15000) # ảnh hiển thị trong 15s
```

```
#cv2.waitKey(0) # ảnh hiển thị mãi trừ khi user tự bấm 1  
phím nào đó
```

```
cv2.destroyAllWindows() #đóng tất cả cửa sổ
```

Kết quả:



Bây giờ, ta thử chuyển đổi ảnh thành màu khác và lưu trữ nó:

# chuyển đổi ảnh

```
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY) #gõ
```

COLOR\_ rồi tự chọn mã màu

```
img_2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB) # thử  
mã màu khác
```

```
cv2.imshow('base',img)
```

```
cv2.imshow('COLOR_BGR2GRAY', gray_img)
```

```
cv2.imshow('COLOR_BGR2RGB', img_2)
cv2.waitKey(15000)
cv2.destroyAllWindows()
```

Kết quả:

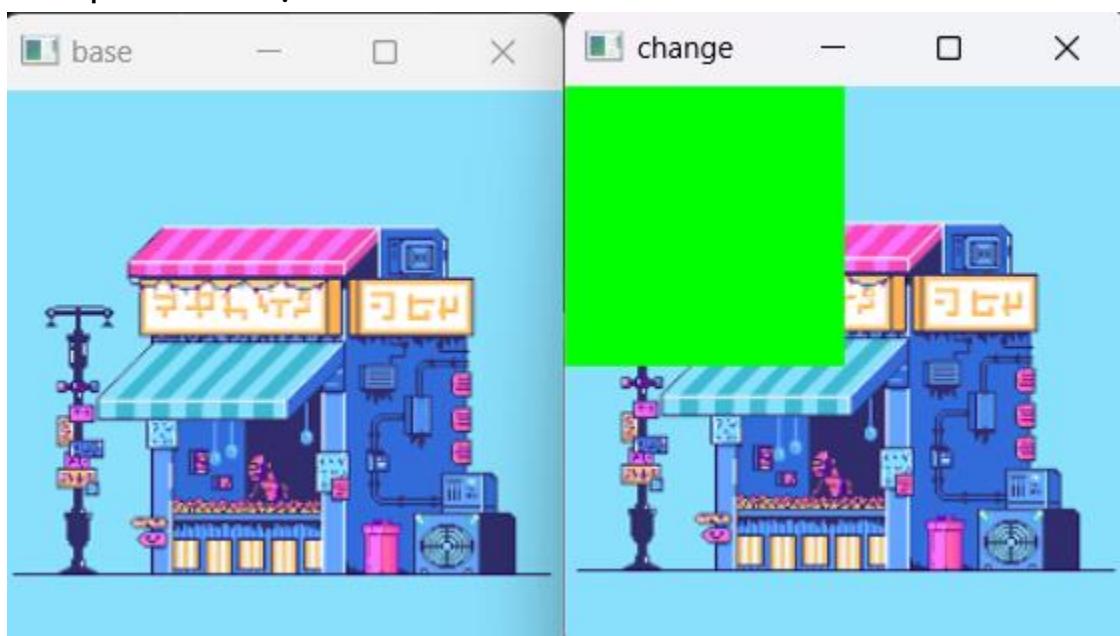


Bây giờ thử tô màu sắc vào 1 khu vực nào đó của ảnh:

```
#tô màu 1 phần ảnh
#thay đổi thông số của điểm ảnh
height, width, z = img.shape
quater_h = height/2
quater_w = width/2
#định nghĩa màu sắc
green_color = (0,255,0)
#copy hình ảnh
img2 = img.copy()
# thay đổi màu góc phần tư bên trái trên của ảnh thành màu
xanh lá
#ép kiểu về int vì mặc định là float thì bị lỗi TypeError: 'float'
object cannot be interpreted as an integer
for y in range(int(quater_h)):
    for x in range(int(quater_w)):
        img2[y,x] = green_color
```

```
#hiển thị  
cv2.imshow('base',img)  
cv2.imshow('change', img2)  
cv2.waitKey(15000)  
cv2.destroyAllWindows()
```

Kết quả thu được:



#ta có thể tạo 1 hàm để thay cho việc hiển thị ảnh rườm rà  
#tuy nhiên như thế này ảnh sẽ hiển thị lần lượt chứ ko cùng  
lúc

# sử dụng display('base', img) để hiển thị

```
def display(title, img):
```

```
    cv2.imshow(title, img)
```

```
    cv2.waitKey(0)
```

```
    cv2.destroyAllWindows()
```

## 22. Sử dụng openCV để vẽ các hình cơ bản

OpenCV cung cấp một loạt các hàm để vẽ các hình học cơ bản trên hình ảnh.

- Vẽ đường thẳng trên hình ảnh: `cv2.line()`  
`cv2.line(image,start_point, end_point, color, thickness)`
- Vẽ hình chữ nhật lên hình ảnh: `cv2.rectangle()`  
`cv2.rectangle(image, top_left_point, bottom_right_point, color, thickness)`
- Vẽ hình tròn trên hình ảnh: `cv2.circle()`  
`cv2.circle(image, center, radius, color, thickness)`
- Vẽ hình elip trên hình ảnh: `cv2.ellipse()`  
`cv2.ellipse(image, center, axes, angle, start_angle, end_angle, color, thickness)`
- Vẽ đa giác trên hình ảnh: `cv2.polyline()`  
`cv2.polyline(image, [array_of_points], is_closed, color, thickness)`

```
import cv2
```

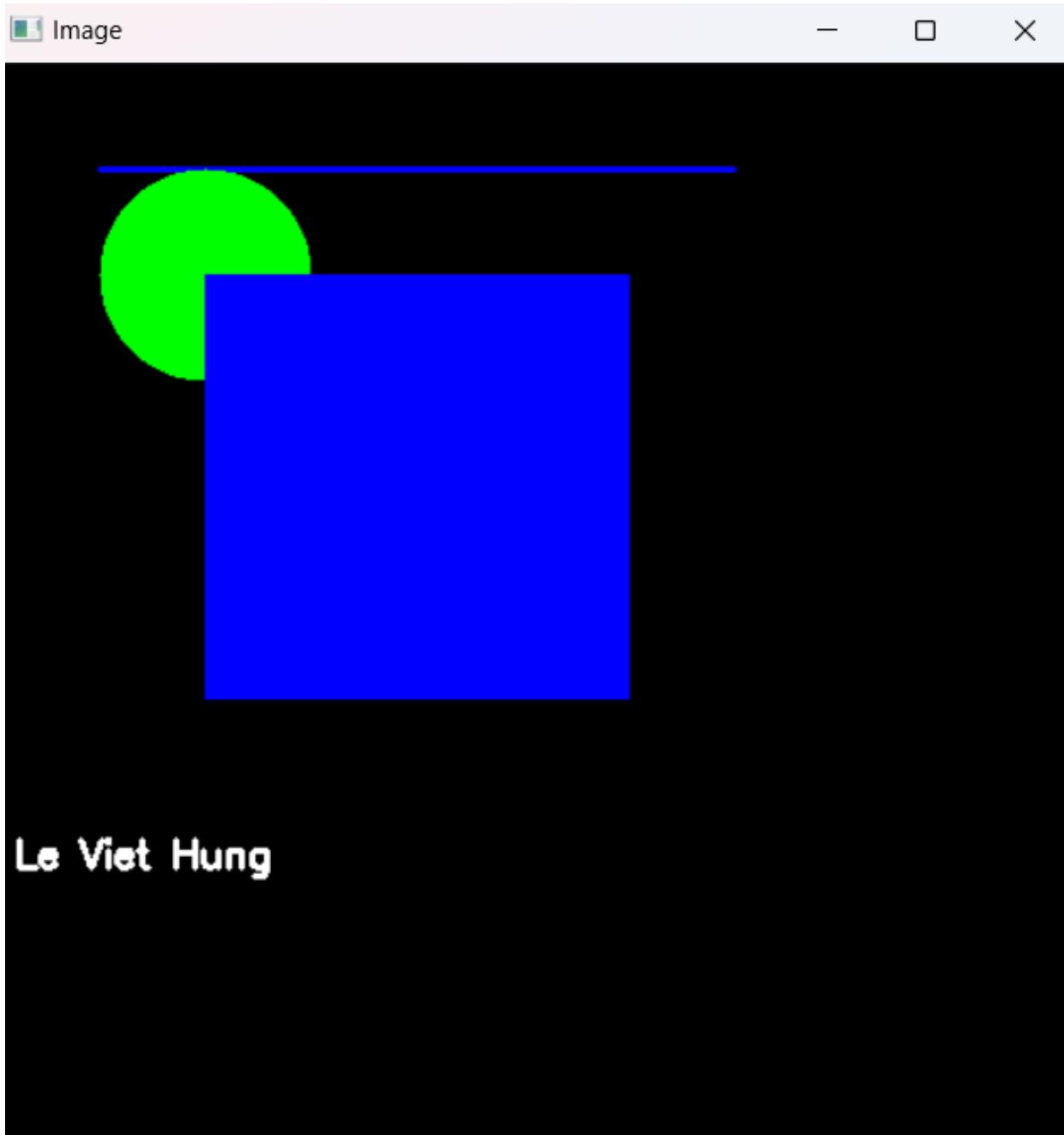
```
import numpy as np
```

```
def display(title, img):  
    cv2.imshow(title, img)  
    cv2.waitKey(0)  
    cv2.destroyAllWindows()
```

```
#tạo 1 ảnh
```

```
img = np.zeros((512, 512, 3), np.uint8) #ảnh toàn điểm 0 nên ảnh sẽ  
đen hoàn toàn  
  
#vẽ đường thẳng xuất phát từ tọa độ (50,50), end ở tọa độ (350,50),  
màu là xanh lam mã(255,0,0), độ dày 2  
  
cv2.line(img, (50,50),(350,50), (255,0,0), 2)  
  
#vẽ hình tròn  
  
cv2.circle(img, (100,100), 50, (0,255,0), -1)  
  
#vẽ hình chữ nhật  
  
cv2.rectangle(img, (100,100), (300,300), (255,0,0), -1)  
  
#ghi text lên hình  
  
content = 'Le Viet Hung'  
  
cv2.putText(img, content, (10, 380), cv2.FONT_HERSHEY_SIMPLEX,  
0.6, (255,255,255)) #kích cỡ chữ: 0,6 độ dày 2  
  
display('Image', img)
```

Kết quả tạo thành như sau:



## 23. OpenCV và các thao tác cơ bản trên video

Video là một chuỗi các hình ảnh tĩnh được thể hiện theo một tốc độ cố định để tạo ra một dãy hình ảnh động.

Mỗi hình ảnh trong video được gọi là “frame” và được hiển thị liên tiếp với tốc độ nhất định để tạo ra hiệu ứng hình ảnh động.

Tốc độ này thường được đo bằng số lượng frame hiển thị mỗi giây và được đo bằng đơn vị “khung hình trên giây” (FPS).

## Các lệnh cơ bản

- Đọc Video từ tệp:

```
video_capture = cv2.VideoCapture('ten_file_video.mp4')
```

- Đọc Frame từ video:

```
ret, frame = video_capture.read()
```

- **ret** là một biến **boolean** cho biết việc đọc frame thành công hay không
- **frame** là khung hình được đọc từ video.

```
import cv2
```

```
#đọc video từ file
```

```
my_video =
```

```
cv2.VideoCapture('C:/computer_vision/vid/video.mp4')
```

```
#tạo cửa sổ để hiển thị
```

```
cv2.namedWindow('Video Player', cv2.WINDOW_NORMAL)
```

```
#hiển thị từng khung ảnh
```

```
while True:
```

```
#đọc 1 frame (đọc từng khung hình một)
```

```
ret, frame = my_video.read()
```

```

#nếu không thể đọc frame nào nữa thì thoát
if not ret:
    break
#hiển thị
cv2.imshow('Video Player', frame)
## khi nào hết video (10s) hoặc gõ q-> enter thì thoát
if (cv2.waitKey(10)== ord('q')):
    break
#hủy bỏ player
my_video.release() #ko đọc video đó nữa

```

cv2.destroyAllWindows()      Kết quả:



Screen Recording  
2024-09-03 165633.m

FPS (frame per second): khung hình trên giây

Nó đo lường tốc độ mà hình ảnh thay đổi trong một trò chơi hoặc ứng dụng đa phương tiện.

FPS được đo bằng số lượng hình ảnh xuất hiện trong một giây.

Bây giờ ta sẽ cho video chạy kèm theo thông số fps:

```

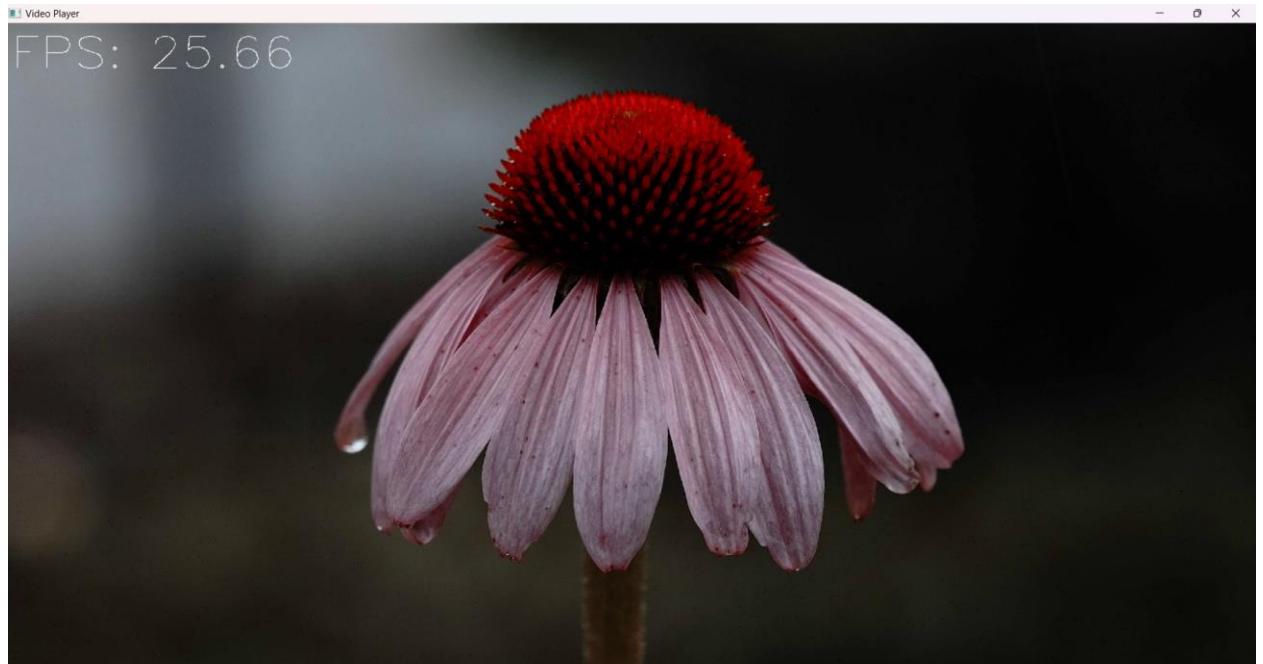
import cv2
import time #phải import để lấy time

```

#đọc video từ file

```
my_video =  
cv2.VideoCapture('C:/computer_vision/vid/video.mp4')  
  
#tạo cửa sổ để hiển thị  
  
cv2.namedWindow('Video Player', cv2.WINDOW_NORMAL)  
  
  
# định nghĩa 1 số thông số của text để hiển thị fps:  
  
font = cv2.FONT_HERSHEY_SIMPLEX  
  
font_color= (255, 255, 255)  
  
font_scale = 5 #chữ to hơn 5 lần so với cỡ chữ mặc định của  
font hershey_simplex  
  
font_thickness = 2  
  
  
  
#hiển thị từng khung ảnh  
  
while True:  
  
    #thời gian trước khi đọc 1 ảnh  
  
    start_time = time.time()  
  
  
  
    #đọc 1 frame (đọc từng khung hình một)  
  
    ret, frame = my_video.read()  
  
    #nếu không thể đọc frame nào nữa thì thoát  
  
    if not ret:  
  
        break
```

```
#thời gian sau khi đọc 1 ảnh  
end_time = time.time()  
  
# tính fps  
fps = 1/(end_time - start_time)  
  
#ghi số lượng fps  
cv2.putText(frame, f'FPS: {fps:.2f}', (10,150), font, font_scale,  
font_color, font_thickness)  
  
#hiển thị  
cv2.imshow('Video Player', frame)  
  
## khi nào hết video (10s) hoặc gõ q-> enter thì thoát  
if (cv2.waitKey(10)== ord('q')):  
    break  
  
#hủy bỏ player  
my_video.release() #ko đọc video đó nữa  
cv2.destroyAllWindows()  
Kết quả thu được:
```



## 24. Sử dụng openCV để lấy hình ảnh từ webcam

Trước tiên, ta sẽ code để mở webcam và hiển thị fps trên đó:

```
import cv2  
  
import time #phải import để lấy time  
  
#đọc video từ cam  
cam = cv2.VideoCapture(0) # số 0 là cam đầu tiên trên  
máy tính  
# nếu nhiều camera thì thay 1,2,... ứng với số cam  
#tạo cửa sổ để hiển thị  
cv2.namedWindow('Video Player',  
cv2.WINDOW_NORMAL)
```

```
# định nghĩa 1 số thông số của text để hiển thị fps:  
font = cv2.FONT_HERSHEY_SIMPLEX  
font_color= (255, 255, 255)  
font_scale = 1  
font_thickness = 2  
  
#hiển thị từng khung ảnh  
while True:  
    #thời gian trước khi đọc 1 ảnh  
    start_time = time.time()  
  
    #đọc 1 frame (đọc từng khung hình một)  
    ret, frame = cam.read()  
    #nếu không thể đọc frame nào nữa thì thoát  
    if not ret:  
        break  
  
    #thời gian sau khi đọc 1 ảnh  
    end_time = time.time()  
  
    # tính fps
```

```
fps = 1/(end_time - start_time)

#ghi số lượng fps
cv2.putText(frame, f'FPS: {fps:.2f}', (10,30), font,
font_scale, font_color, font_thickness)

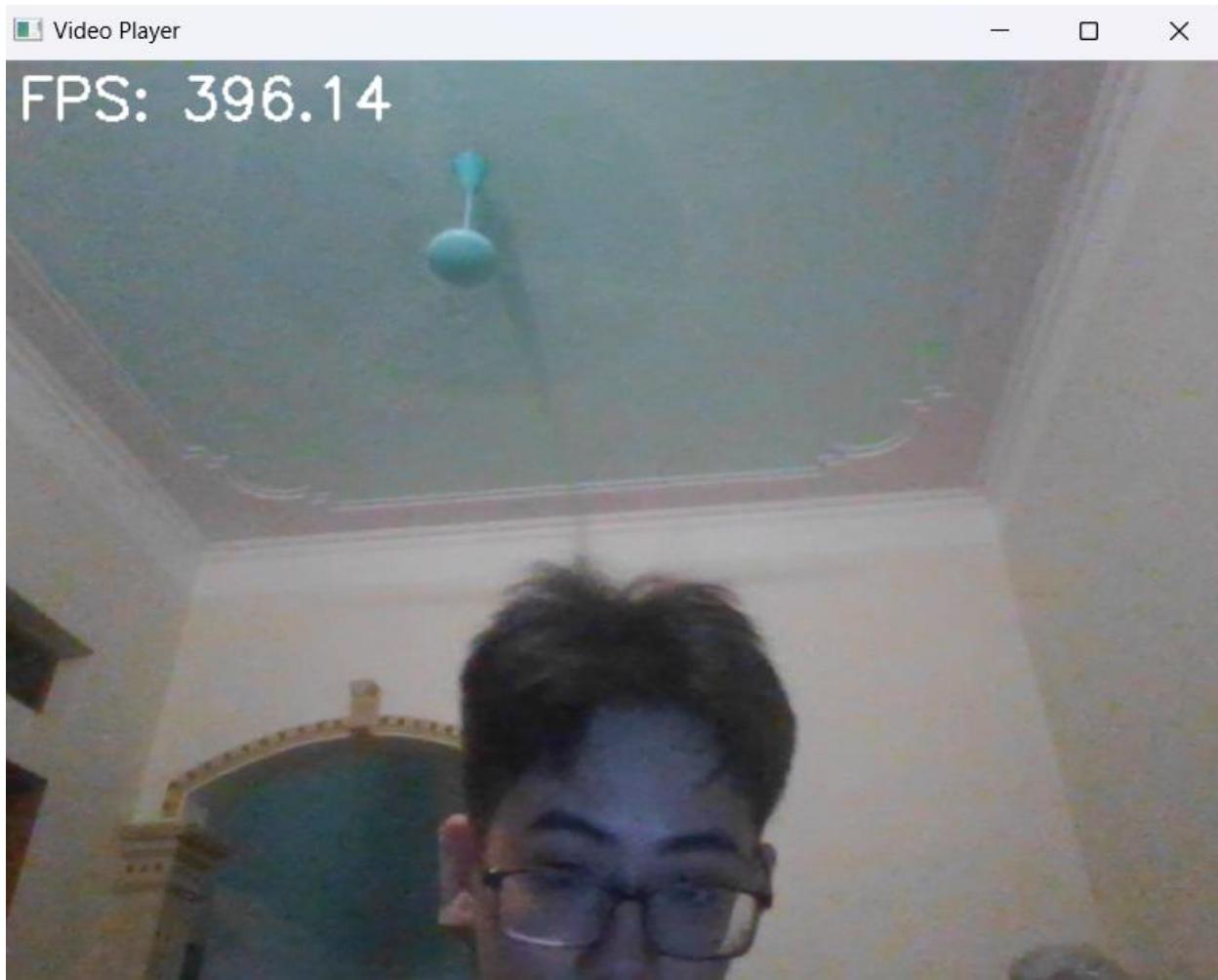
#hiển thị
cv2.imshow('Video Player', frame)

## khi nào hết video (10s) hoặc gõ q-> enter thì thoát
if (cv2.waitKey(10)== ord('q')):

    break

#hủy bỏ player
cam.release() #ko đọc video đó nữa
cv2.destroyAllWindows()
```

Kết quả thu được như sau:



Khi đã thành công, ta sẽ chụp ảnh liên tục từ webcam và lưu trữ:

```
import cv2
```

```
import time #phải import để lấy time
```

```
#đọc video từ cam
```

```
cam = cv2.VideoCapture(0) # số 0 là cam đầu tiên trên máy tính
```

```
# nếu nhiều camera thì thay 1,2,... ứng với số cam
```

```
#tạo cửa sổ để hiển thị
```

```
cv2.namedWindow('Video Player', cv2.WINDOW_NORMAL)
interval = 5 # đặt biến thể hiện cứ 5s (5 khung hình) thì lưu ảnh xuống
count = 0
```

```
# định nghĩa 1 số thông số của text để hiển thị fps:
```

```
font = cv2.FONT_HERSHEY_SIMPLEX
font_color = (255, 255, 255)
font_scale = 1
font_thickness = 2
```

```
#hiển thị từng khung ảnh
```

```
while True:
```

```
    #thời gian trước khi đọc 1 ảnh
```

```
    start_time = time.time()
```

```
#đọc 1 frame (đọc từng khung hình một)
```

```
    ret, frame = cam.read()
```

```
#nếu không thể đọc frame nào nữa thì thoát
```

```
    if not ret:
```

```
        break
```

#tăng count lên nếu count chia hết cho 5 thì lưu xuống một cách  
định kì

```
count +=1  
  
if (count% interval ==0):  
    # lưu ảnh xuống đường dẫn  
    cv2.imwrite(f'C:/computer_vision/img/image_{count}.jpg',  
frame)
```

#thời gian sau khi đọc 1 ảnh

```
end_time = time.time()
```

# tính fps

```
fps = 1/(end_time - start_time)
```

#ghi số lượng fps

```
cv2.putText(frame, f'FPS: {fps:.2f}', (10,30), font, font_scale,  
font_color, font_thickness)
```

#hiển thị

```
cv2.imshow('Video Player', frame)
```

## khi nào hết video (10s) hoặc gõ q-> enter thì thoát

```
if (cv2.waitKey(10)== ord('q')):
```

```
    break
```

#hủy bỏ player

```
cam.release() #ko đọc video đó nữa
```

```
cv2.destroyAllWindows()
```

Như vậy, cứ 5s là tự động chụp và lưu vào img.

## 25. Sử dụng openCV phát hiện khuôn mặt

Haar Cascade Classifier:

- Haar Cascade là một phương pháp nhận dạng đặc trưng dựa trên máy học (machine learning).
- Được phát triển bởi các nhà nghiên cứu Viola và Jones vào năm 2001.
- Dựa trên việc sử dụng các “Haar-like features” (các đặc trưng giống như Haar), là các kích thước và hình dạng khác nhau.
- Haar-like features có thể được sử dụng để nhận dạng các đối tượng cụ thể trong hình ảnh.

Để sử dụng haarcascades thì ta sẽ lấy từ github của openCV:

<https://github.com/opencv/opencv/tree/master/data/haarcascades>

Ví dụ mình cần nhận diện khuôn mặt (face) thì sẽ download 1 file liên quan tới face, ở đây dùng:

haarcascade\_frontalface\_default.xml

Hoặc nhận dạng con mắt (có đeo kính) thì ta có thể dùng  
haarcascade\_eye\_tree\_eyeglasses.xml

Ta sẽ tải 2 file xml này để thực hành.

Trước tiên, ta sẽ nhận diện khuôn mặt bằng cách vẽ 1 khung hình chữ nhật bao quanh khuôn mặt

```
import cv2
```

```

# hàm để hiển thị

def display(title, img):
    cv2.imshow(title, img)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

#khởi tạo Haar cascade classifier cho nhận diện khuôn mặt

face_cascade =
cv2.CascadeClassifier('C:/computer_vision/data/haarcascade_frontalface_default.xml')

#đọc ảnh

img = cv2.imread('C:/computer_vision/img/face.jpg')

#chuyển ảnh sang màu xám

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1,
minNeighbors = 5, minSize=(30,30))      #nhận diện nhiều ảnh
cùng 1 lúc

#scaleFactor kéo dãn, minNeighbors k/c gần nhất, minSize = kích cỡ
mặt min

#vẽ hộp chứa khuôn mặt

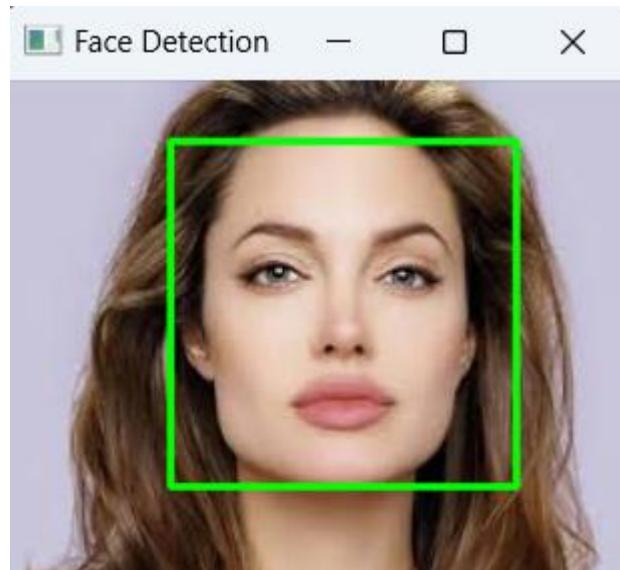
for (x,y,w,h) in faces:

```

```
cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0),2)
```

```
#hiển thị
```

```
display('Face Detection', img)
```



Kết quả thu được như sau:

Thử bổ sung nhận diện mắt, đồng thời ta sẽ gói gọn các thao tác nhận diện lại thành 1 hàm detect để sau tái sử dụng :

```
import cv2
```

```
# hàm để hiển thị
```

```
def display(title, img):
```

```
    cv2.imshow(title, img)
```

```
    cv2.waitKey(0)
```

```
    cv2.destroyAllWindows(title)
```

```
def detect(img):
```

```

#khởi tạo Haar cascade classifier cho nhận diện khuôn mặt và
mắt có đeo kính

face_cascade =
cv2.CascadeClassifier('C:/computer_vision/data/haarcascade_fron
talface_default.xml')

eye_cascade =
cv2.CascadeClassifier('C:/computer_vision/data/haarcascade_eye_
tree_eyeglasses.xml')

#chuyển ảnh sang màu xám

gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

#tìm khuôn mặt

faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1,
minNeighbors = 5, minSize=(30,30))      #nhận diện nhiều ảnh
cùng 1 lúc

#scaleFactor kéo dãn, minNeighbors k/c gần nhất, minSize = kích
cỡ mặt min

#tìm mắt

eyes = eye_cascade.detectMultiScale(gray, scaleFactor = 1.1,
minNeighbors = 1, minSize=(5,5))      #nhận diện nhiều ảnh cùng 1
lúc

#vẽ hộp chứa khuôn mặt

for (x,y,w,h) in faces:

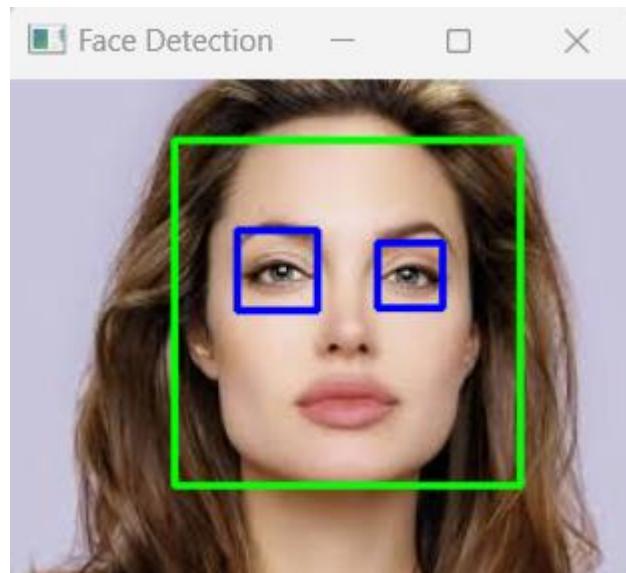
    cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0),2)

#vẽ hộp chứa mắt

for (x,y,w,h) in eyes:

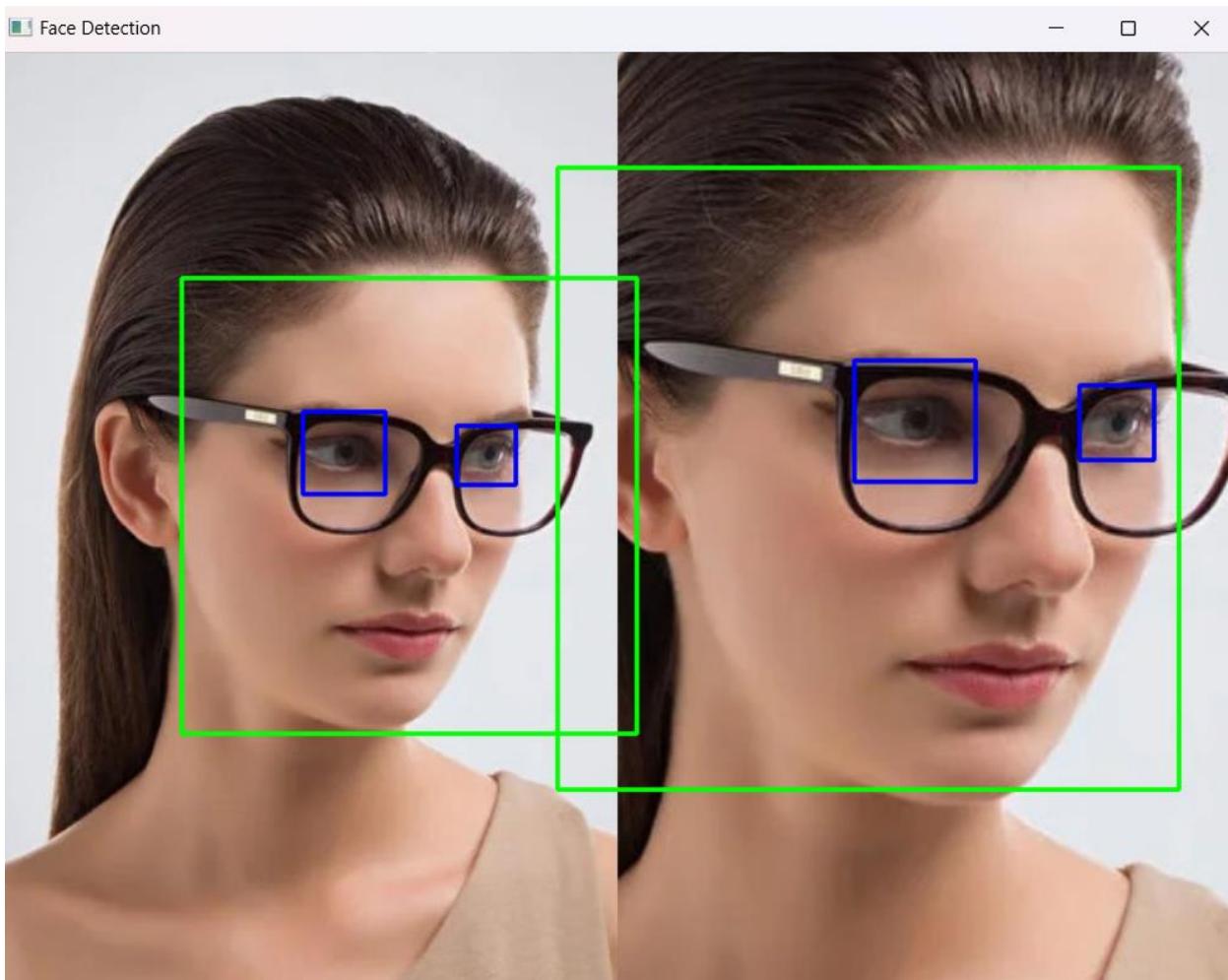
```

```
cv2.rectangle(img, (x,y), (x+w, y+h), (255,0,0),2)  
#trả về ảnh  
return img  
  
#đọc ảnh  
img = cv2.imread('C:/computer_vision/img/face.jpg')  
  
#hiển thị  
display('Face Detection', detect(img))
```



Kết quả ( không đeo kính):

Kết quả (có đeo kính và có nhiều khuôn mặt):



Bây giờ, ta sẽ nhận diện khuôn mặt và mắt trực tiếp từ trên webcam:

```
import cv2
```

```
import time
```

```
# hàm để hiển thị
```

```
def display(title, img):
```

```
    cv2.imshow(title, img)
```

```
    cv2.waitKey(10000)
```

```
cv2.destroyAllWindows()

def detect(img):
    #khởi tạo Haar cascade classifier cho nhận diện khuôn mặt và
    #mắt có đeo kính

    face_cascade =
    cv2.CascadeClassifier('C:/computer_vision/data/haarcascade_frontalface_default.xml')

    eye_cascade =
    cv2.CascadeClassifier('C:/computer_vision/data/haarcascade_eye_tree_eyeglasses.xml')

    #chuyển ảnh sang màu xám
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    #tìm khuôn mặt
    faces = face_cascade.detectMultiScale(gray, scaleFactor = 1.1,
                                           minNeighbors = 5, minSize=(30,30))      #nhận diện nhiều ảnh
    cùng 1 lúc

    #scaleFactor kéo dãn, minNeighbors k/c gần nhất, minSize = kích
    #cỡ mặt min

    #tìm mắt
    eyes = eye_cascade.detectMultiScale(gray, scaleFactor = 1.1,
                                         minNeighbors = 1, minSize=(5,5))      #nhận diện nhiều ảnh cùng 1
    lúc

    #vẽ hộp chứa khuôn mặt
    for (x,y,w,h) in faces:
```

```
cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0),2)
#vẽ hộp chứa mắt
for (x,y,w,h) in eyes:
    cv2.rectangle(img, (x,y), (x+w, y+h), (255,0,0),2)
#trả về ảnh
return img
```

```
#đọc ảnh
img = cv2.imread('C:/computer_vision/img/kinh.jpg')

#hiển thị ảnh
display('Face Detection', detect(img))

# nhận diện từ video
my_video = cv2.VideoCapture(0)
#tạo cửa sổ để hiển thị
cv2.namedWindow('Video Player', cv2.WINDOW_NORMAL)
```

```
# định nghĩa 1 số thông số của text để hiển thị fps:
font = cv2.FONT_HERSHEY_SIMPLEX
font_color= (255, 255, 255)
```

```
font_scale = 1
font_thickness = 2

#hiển thị từng khung ảnh
while True:
    #thời gian trước khi đọc 1 ảnh
    start_time = time.time()

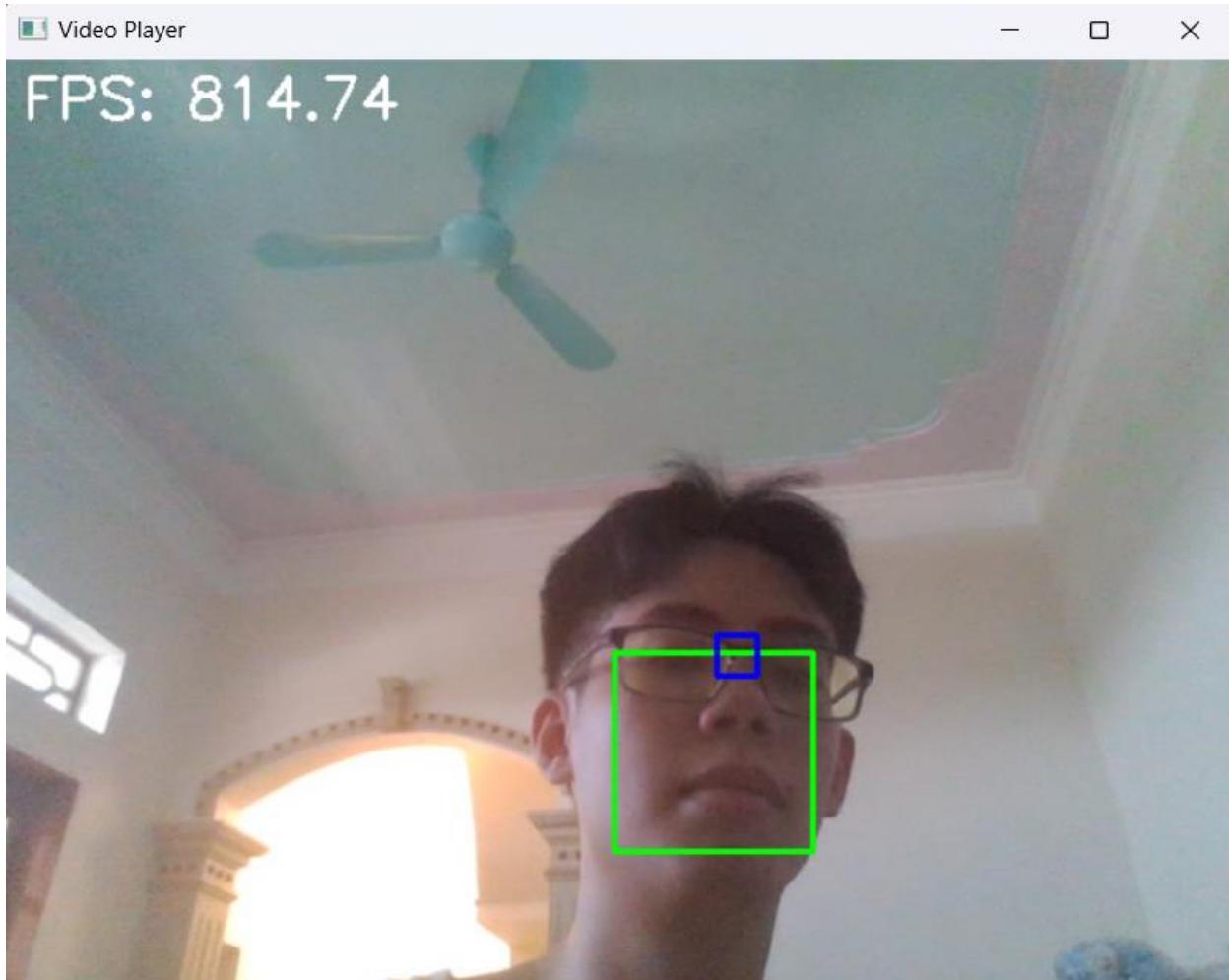
    #đọc 1 frame (đọc từng khung hình một)
    ret, frame = my_video.read()
    #nếu không thể đọc frame nào nữa thì thoát
    if not ret:
        break

    #thời gian sau khi đọc 1 ảnh
    end_time = time.time()

    # tính fps
    fps = 1/(end_time - start_time)
    # detect khuôn mặt và mắt
    frame = detect(frame)
```

```
#ghi số lượng fps  
cv2.putText(frame, f'FPS: {fps:.2f}', (10,30), font, font_scale,  
font_color, font_thickness)  
  
#hiển thị  
cv2.imshow('Video Player', frame)  
  
## khi nào hết video (10s) hoặc gõ q-> enter thì thoát  
if (cv2.waitKey(10)== ord('q')):  
    break  
  
#hủy bỏ player  
my_video.release() #ko đọc video đó nữa  
cv2.destroyAllWindows()
```

Kết quả:



Nhìn chung, do nhiều yếu tố khách quan nên nhận diện còn nhiều sai sót.

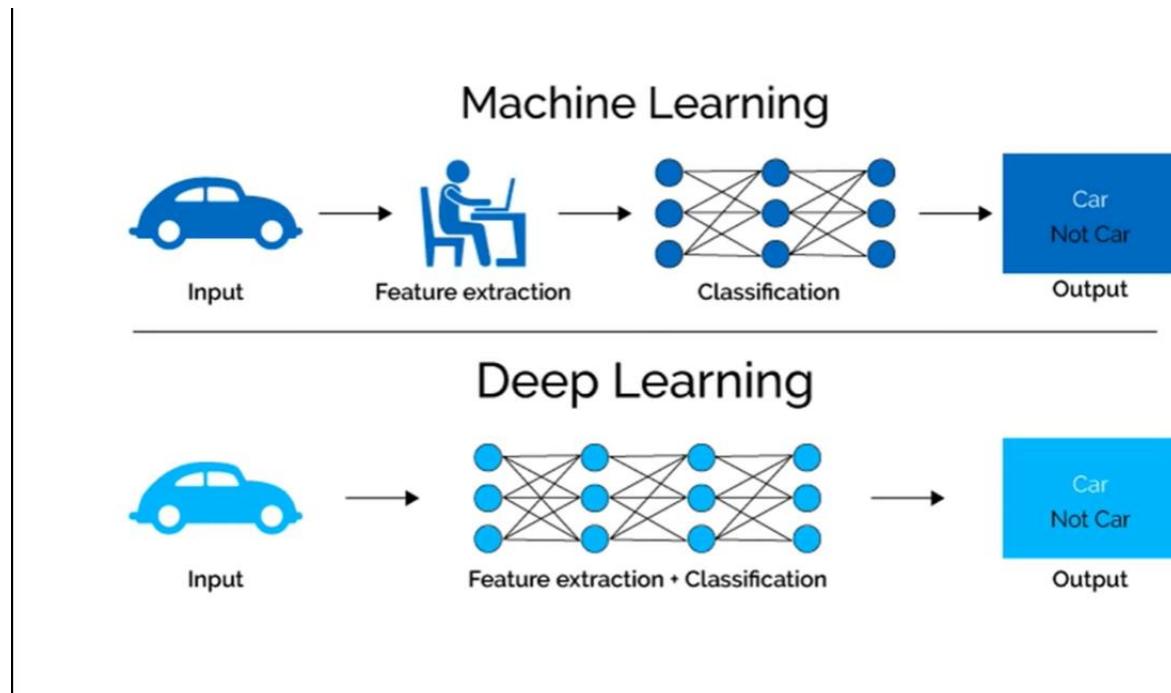
## 26. Giới thiệu DNN và openCV DNN

*Ứng dụng trong thực tế:* Các ứng dụng của bài toán phát hiện khuôn mặt rất đa dạng, từ việc xác định khuôn mặt trong ảnh chân dung, ghi dấu vùng khuôn mặt để tạo hiệu ứng mặt đẹp trên các ứng dụng chỉnh sửa ảnh, đến việc xác định khuôn mặt để kiểm tra danh tính, an ninh hoặc điểm danh trong hệ thống nhận diện khuôn mặt.

DNN: Deep Neural Network thuộc Deep Learning (học sâu)

Deep Learning là một lĩnh vực trong trí tuệ nhân tạo (AI) và máy tính, tập trung vào việc sử dụng mạng neural sâu để học và áp dụng kiến thức từ dữ liệu. Deep Learning được gọi là “sâu” vì nó sử dụng nhiều lớp neural hoặc các tầng ẩn để biểu diễn dữ liệu và học các đặc trưng phức tạp và trừu tượng từ dữ liệu đầu vào.

Sự khác nhau giữa Machine Learning và Deep Learning:



Một số đặc điểm quan trọng của Deep Learning:

- Học biểu diễn dữ liệu: cho phép mô hình học tự động các biểu diễn cấu trúc từ dữ liệu, giúp tạo ra các đặc trưng phức tạp và trừu tượng từ dữ liệu đầu vào.
- Sử dụng mạng neural: thường sử dụng mạng neural, bao gồm mạng neural tích chập (CNN), mạng neural tái khám phá (RNN), và các kiến trúc khác để thực hiện học và dự đoán.

- Tích hợp lớp ẩn: sử dụng nhiều lớp neural ẩn liên tiếp để biểu diễn dữ liệu. Điều này tạo ra sự phức tạp và linh hoạt trong việc học và áp dụng kiến thức.

Ứng dụng:

- Thị giác máy tính (computer vision)
- Xử lý ngôn ngữ tự nhiên (natural language processing)
- Điều khiển tự động (autonomous control)
- Dự đoán dữ liệu

*OpenCV Deep Neural Network (DNN):*

- OpenCV DNN là một phần mở rộng của openCV (open source computer vision library) cho phép tích hợp và sử dụng mạng neural học sâu trong các ứng dụng thị giác máy tính.
- OpenCV là một thư viện phổ biến trong lĩnh vực xử lý hình ảnh và thị giác máy tính, trong khi DNN module cho phép bạn sử dụng các mô hình Deep Learning như CNN để thực hiện nhiều nhiệm vụ liên quan đến thị giác máy tính, chẳng hạn như phát hiện khuôn mặt, phân loại đối tượng, và thậm chí là phát hiện đối tượng trong thời gian thực.

Ứng dụng của openCV DNN:

- Phát hiện khuôn mặt: Sử dụng mô hình đã được đào tạo để xác định vị trí và ranh giới của khuôn mặt trong hình ảnh



- hoặc trong video.
- Phân loại đối tượng: Sử dụng mạng neural để phân loại các đối tượng trong hình ảnh, chẳng hạn phát hiện người và các



- đối tượng khác.
- Phát hiện đối tượng: Sử dụng mạng neural để phát hiện các đối tượng cụ thể trong thời gian thực, chẳng hạn như phát



hiện xe hơi trên đường.

- Đọc văn bản: Sử dụng mô hình để trích xuất và đọc văn bản



từ hình ảnh.

*DNN module trong openCV:* DNN module trong openCV cung cấp chức năng cho phép tải và sử dụng các mô hình học sâu đã được đào tạo từ các nguồn sau:

- Caffe
- TensorFlow
- PyTorch
- Darknet

Caffe (Convolutional Architecture  
for Fast Feature Embedding)

# Caffe

- Caffe (Convolutional Architecture for Fast Feature Embedding) là một framework mã nguồn mở mạnh mẽ được phát triển để hỗ trợ việc xây dựng và huấn luyện các mô hình học máy cho việc thị giác máy tính và xử lý hình ảnh.
- Được phát triển bởi Caffe Project và ban đầu được phát hành vào năm 2014. Caffe đã trở thành một trong những công cụ quan trọng trong lĩnh vực Deep Learning và Computer Vision.

# TensorFlow



- TensorFlow là một thư viện phần mềm mã nguồn mở dành cho máy học trong nhiều loại hình tác vụ nhận thức và hiểu ngôn ngữ.
- Hiện đang được sử dụng cho cả nghiên cứu lẫn sản xuất bởi các đội khác nhau trong các sản phẩm thương mại của Google, như nhận dạng giọng nói, Gmail, Google Photos, và tìm kiếm ...
- TensorFlow nguyên thủy được phát triển bởi đội Google Brain cho mục đích nghiên cứu và sản xuất của Google và sau đó được phát hành theo giấy phép mã nguồn mở Apache 2.0 vào ngày 9/11/2015.

# PyTorch



- PyTorch là một framework học máy dựa trên thư viện Torch được sử dụng trong lĩnh vực thị giác máy tính và xử lý ngôn ngữ tự nhiên, do Meta AI phát triển và ngày nay là một phần của Linux Foundation.
- Là phần mềm tự do nguồn mở phát hành dựa trên giấy phép BSD đã qua sửa đổi. Mặc dù được phát triển chính yếu ở dạng giao diện ngôn ngữ Python, PyTorch cũng hỗ trợ giao diện C++.

# Darknet



- Phát triển bởi Joseph Redmon: Darknet được phát triển bởi Joseph Redmon, một nhà nghiên cứu trong lĩnh vực thị giác máy tính, người đã đóng góp đáng kể vào việc phát triển các mô hình object detection, bao gồm YOLO (You Only Look Once).
- Hỗ trợ các mô hình object detection: Darknet được sử dụng rộng rãi để huấn luyện và triển khai các mô hình object detection, trong đó nổi bật là YOLO (You Only Look Once). YOLO là một mô hình object detection nổi tiếng có khả năng phát hiện và phân loại nhiều đối tượng trong một khung hình với tốc độ nhanh.
- Tốc độ và hiệu suất cao: Darknet được thiết kế để đạt hiệu suất cao và có khả năng xử lý nhanh trên các thiết bị với tài nguyên hạn chế như máy tính cá nhân hoặc thiết bị nhúng. Điều này làm cho nó phù hợp cho các ứng dụng thời gian thực và ứng dụng nhận diện đối tượng trên camera.
- Mã nguồn mở: Darknet là một framework mã nguồn mở, cho phép các nhà phát triển tùy chỉnh và mở rộng mã nguồn để phù hợp với các yêu cầu cụ thể của họ.
- Darknet được sử dụng rộng rãi trong nhiều ứng dụng thị giác máy tính, bao gồm các hệ thống giám sát an ninh, ứng dụng xe tự hành, nhận diện đối tượng trong thời gian thực và nhiều lĩnh vực khác đòi hỏi khả năng phát hiện và nhận diện đối tượng hiệu quả.

## 27. Nhận diện khuôn mặt bằng openCV DNN

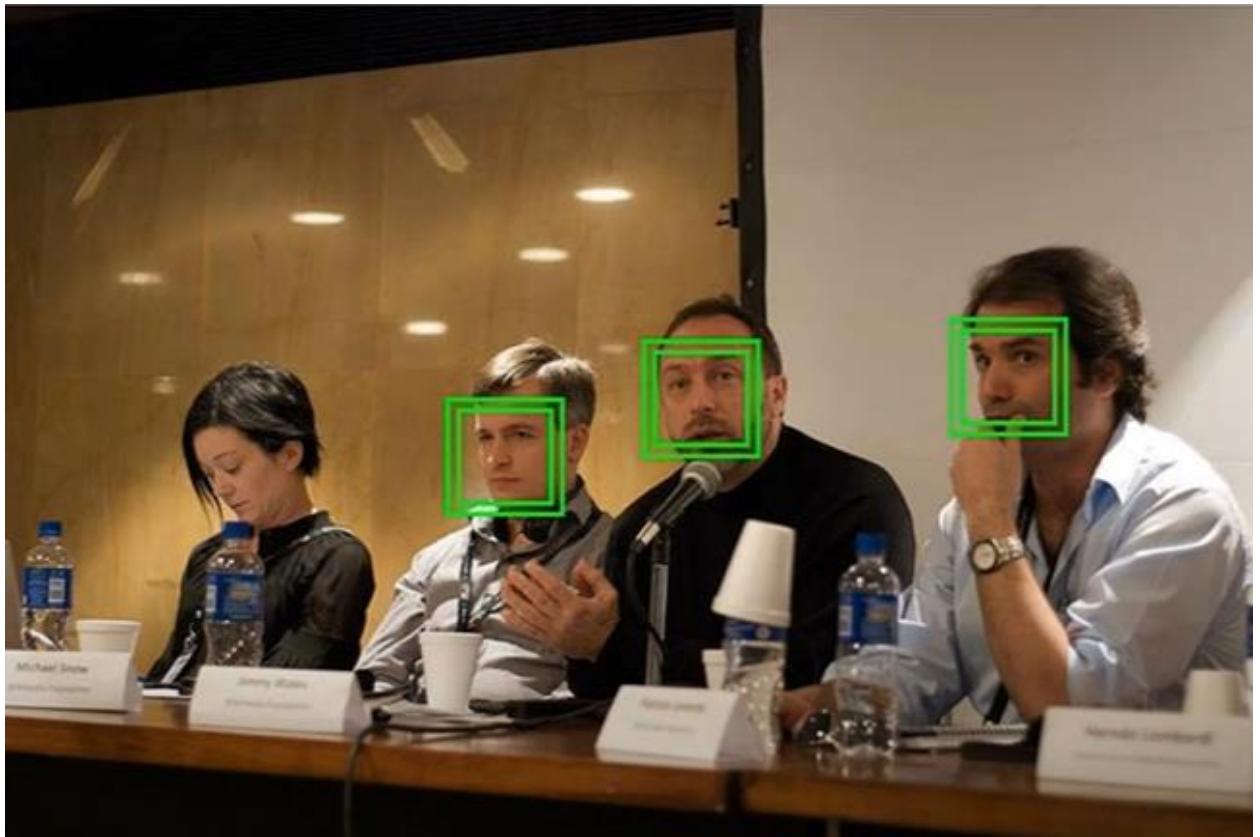
Lần này thay vì dùng CascadeClassifier có sẵn ở openCV thì ta sẽ dùng DNN để nhận diện.

Bài toán phát hiện khuôn mặt (detect face) là một trong những nhiệm vụ quan trọng trong lĩnh vực thị giác máy tính (computer vision).

Nó liên quan đến việc tự động xác định vị trí và ranh giới của khuôn mặt trong một hình ảnh hoặc video.

Mục tiêu chính của bài toán này là tìm ra các vùng trong hình ảnh chứa khuôn mặt của con người và xác định các đường biên (bounding box) xác định vị trí của khuôn mặt đó.

Quá trình phát hiện khuôn mặt thường bắt đầu bằng việc sử dụng các thuật toán và mô hình máy học để tìm các đặc điểm, cấu trúc hoặc mẫu có liên quan đến khuôn mặt trong hình ảnh.



Các phương pháp phổ biến:

- Haar Cascades: sử dụng các phân lớp AdaBoost và bộ lọc Haar để phát hiện khuôn mặt. Đây là một phương pháp truyền thống nhưng vẫn được sử dụng rộng rãi trong ứng dụng thời gian thực.
- Mạng neural tích chập (Convolutional Neural Networks – CNN): sử dụng mạng CNN để học đặc trưng của khuôn mặt và dự đoán vị trí chúng trong hình ảnh. Một số kiến trúc CNN nổi tiếng như Viola-Jones, Single Shot MultiBox Detector (SSD) và Faster R-CNN đã được sử dụng cho bài toán này.
- Mạng học sâu (Deep Learning): sử dụng mạng học sâu như mạng neural tích chập (CNN), mạng phát triển(DNN) hoặc các mô hình đặc biệt như mạng thần kinh đa mục tiêu

(Multi-task Neural Networks) để tăng cường khả năng phát hiện khuôn mặt.

## SSD: Single Shot MultiBox Detector

Sử dụng ResNet-10 network

The image shows a screenshot of an arXiv preprint page. The title is "SSD: Single Shot MultiBox Detector" by Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. The abstract discusses a method for detecting objects in images using a single deep neural network, named SSD. It discretizes output boxes over different aspect ratios and scales per feature map location. At prediction time, the network generates scores for the presence of each object and produces adjustments to the box to better match the object shape. Additionally, the network combines predictions from multiple feature maps without a resampling stage and encapsulates all computation in a single network. This makes SSD easy to train and straightforward to integrate into systems. Experimental results on the PASCAL VOC, MS COCO, and ILSVRC datasets confirm that SSD has comparable accuracy to methods that utilize a multi-stage architecture, while being much faster, and providing a unified framework for both training and inference. Compared to other single stage methods, SSD has much better performance. For 300 × 300 input, SSD achieves 72.1% mAP on VOC2007 test at 58 FPS on a Nvidia Titan X and for 500 × 500 input, SSD achieves a comparable state-of-the-art Faster R-CNN model. Code is available at [this URL](https://github.com/weiliu/SSD TensorFlow).

## ResNet-10

- ResNet-10 là một mô hình mạng nơ-ron sâu trong lĩnh vực thị giác máy tính, dựa trên kiến trúc Residual Networks (ResNet) được giới thiệu ban đầu bởi Kaiming He, Xiangyu Zhang, Shaoqing Ren và Jian Sun vào năm 2015. Mạng ResNet được thiết kế để giải quyết vấn đề sự mất mát đạo hàm đối với các mạng nơ-ron sâu, cho phép bạn xây dựng các mô hình sâu hơn mà vẫn có khả năng học tốt.
- Số "10" trong ResNet-10 chỉ ra rằng mô hình này có một số lượng lớp (hoặc các đơn vị tích chập) cố định là 10. Cụ thể, ResNet-10 bao gồm 10 lớp convolutional và một lớp fully connected ở cuối để thực hiện phân loại.
- ResNet-10 không phải là mô hình sâu nhất trong gia đình ResNet, mà nó là một biến thể đơn giản hơn của ResNet được sử dụng để thực hiện các nhiệm vụ thị giác, chẳng hạn như phân loại hình ảnh. Các biến thể khác của ResNet như ResNet-18, ResNet-50, và ResNet-101 cung cấp sự sâu hơn và hiệu suất tốt hơn, nhưng cũng đòi hỏi nhiều tài nguyên tính toán hơn.

Ta sẽ sử dụng các model sau:

Face Detection (FP16): phiên bản dấu phẩy động 16 của triển khai Caffe:

[https://github.com/opencv/opencv\\_3rdparty/raw/19512576c112aa2c7b6328cb0e8d589a4a90a26d/res10\\_300x300\\_ssd\\_iter\\_140000\\_fp16.caffemodel](https://github.com/opencv/opencv_3rdparty/raw/19512576c112aa2c7b6328cb0e8d589a4a90a26d/res10_300x300_ssd_iter_140000_fp16.caffemodel)

Kiến trúc của models:

[https://github.com/opencv/opencv/blob/master/samples/dnn/face\\_detector/deploy.prototxt](https://github.com/opencv/opencv/blob/master/samples/dnn/face_detector/deploy.prototxt)

Face Detector (UINT8): phiên bản 8 bit sử dụng TensorFlow:

[https://github.com/opencv/opencv\\_3rdparty/raw/8033c2bc31b3256f0d461c919ecc01c2428ca03b/opencv\\_face\\_detector\\_uint8.pb](https://github.com/opencv/opencv_3rdparty/raw/8033c2bc31b3256f0d461c919ecc01c2428ca03b/opencv_face_detector_uint8.pb)

Kiến trúc của models:

[https://github.com/opencv/opencv\\_extra/blob/master/testdata/dnn/opencv\\_face\\_detector.pbtxt](https://github.com/opencv/opencv_extra/blob/master/testdata/dnn/opencv_face_detector.pbtxt)

Sau khi tải xong các mô hình đã được huấn luyện từ trước thì ta thử tiến hành code:

```
import cv2  
import numpy as np
```

```
#tải ảnh đầu vào  
img = cv2.imread('C:/computer_vision/img/nhieu_face.jpg')  
#hiển thị ảnh gốc  
cv2.imshow('Base image', img)  
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()

#tải mô hình đã được huấn luyện từ trước
net =
cv2.dnn.readNetFromCaffe('C:/computer_vision/models/deploy.prototxt',
                         'C:/computer_vision/models/res10_300x300_ssd_iter_140000_fp16.caffemodel')

#chuẩn bị dữ liệu đầu vào
#1.0 scale ảnh về tỉ lệ 1.0, tức là ảnh ko bị co giãn
#(300, 300) là kích thước mà mô hình yêu cầu cho đầu vào
# (104,177,123) màu sắc trung bình
# swapRB = False ko hoán đổi kênh màu đỏ và xanh
blob = cv2.dnn.blobFromImage(img, 1.0, (300,300), (104,177, 123),
                             swapRB= False)

#đặt dữ liệu đầu vào cho mạng
net.setInput(blob)

#chạy mạng để phát hiện khuôn mặt
#hàm sẽ chạy dần trên ảnh để tìm khuôn mặt
faces= net.forward()
```

```
#lấy kích thước của ảnh đầu vào  
h = img.shape[0]  
w = img.shape[1]  
  
#in thông tin  
print(faces.shape) #(1, 1, 200, 7)  
#1: số lượng ảnh đầu vào, 1: số lượng ảnh đầu ra, 200 khuôn mặt đc  
phát hiện, 7 thông tin  
print(faces[0, 0, 0, :]) #in ra 7 trường thông tin  
#[0. 1. 0.99763405 0.45528927 0.71117806 0.5378257  
# 0.8057307 ]
```

Các thông số thể hiện cho những điều sau:

```
print(detections.shape)
```

(1, 1, 200, 7)

- Chiều đầu tiên (đầu tiên từ trái sang phải) có kích thước là 1, đại diện cho số lô (batch size) của các ảnh đầu vào. Trong trường hợp này, bạn đang xử lý một ảnh nên batch size là 1.
- Chiều thứ hai cũng có kích thước là 1, đại diện cho số lớp (classes) hoặc số kích thước (size dimensions) trong dữ liệu đầu ra. Trong trường hợp này, bạn đang thực hiện nhận dạng khuôn mặt nên số lớp là 1.

```
print(detections.shape)
```

(1, 1, 200, 7)

- Chiều thứ ba có kích thước là 200, đại diện cho số khuôn mặt được phát hiện trong một ảnh. Điều này có nghĩa rằng mô hình có khả năng phát hiện tối đa 200 khuôn mặt trong ảnh đầu vào.

print(faces[0, 0, 0, :]) #in ra 7 trường thông tin

# [0. 1. 0.99763405 0.45528927 0.71117806 0.5378257

# 0.8057307 ] thì 7 thông tin thể hiện cho những điều sau:

```
print(detections.shape)
```

(1, 1, 200, 7)

7 giá trị của chiều cuối cùng là:

- Số thứ tự của ảnh
- Số thứ tự lớp (Class ID): Đây là số nguyên đại diện cho lớp hoặc loại đối tượng mà mô hình đã phát hiện. Trong trường hợp này, vì bạn đang nhận dạng khuôn mặt, nên giá trị này thường là 1 để chỉ khuôn mặt. (Có giá trị là 0 hoặc 1)
- Độ tin cậy (Confidence Score): Đây là một giá trị thực (float) thể hiện mức độ tin tưởng của mô hình rằng khuôn mặt đã được phát hiện đúng. Giá trị này thường nằm trong khoảng từ 0 đến 1, và càng cao thì độ tin cậy càng lớn. Trong trường hợp của bạn, bạn đã kiểm tra nếu confidence >= 0.5 để xác định xem một khuôn mặt có được coi là được phát hiện hay không.
- Start X: Đây là tọa độ x (hoành độ) của góc trái trên của hình chữ nhật giới hạn khuôn mặt phát hiện trong ảnh.
- Start Y: Đây là tọa độ y (tung độ) của góc trái trên của hình chữ nhật giới hạn khuôn mặt phát hiện trong ảnh.
- End X: Đây là tọa độ x của góc phải dưới của hình chữ nhật giới hạn khuôn mặt phát hiện trong ảnh.
- End Y: Đây là tọa độ y của góc phải dưới của hình chữ nhật giới hạn khuôn mặt phát hiện trong ảnh.

*Độ tin cậy >0.5 mới được coi là khuôn mặt.*

```
#duyệt từng khuôn mặt đã được phát hiện  
for i in range(0, faces.shape[2]):    #vị trí thứ 2 của faces là độ tin cậy  
    confidence = faces[0,0, i,2]    #vị trí độ tin cậy  
    #kiểm tra xem độ tin cậy có > 0.5 hay không  
    if confidence > 0.5:  
        #trích xuất tọa độ  
        print(faces[0,0,i,3:7]) # lấy từ pt 3 tới 6
```

Kết quả :

```
[0.45528927 0.71117806 0.5378257 0.8057307 ]  
[0.25925028 0.87112087 0.3433196 0.95433 ]  
[0.44929308 0.5330197 0.5303895 0.6261024 ]  
[0.6643052 0.5299996 0.7457144 0.6382607]  
[0.06174818 0.3707326 0.13863042 0.46135658]  
[0.65804493 0.8672387 0.74079645 0.9549449 ]  
[0.64594036 0.19858676 0.73240644 0.2934181 ]  
[0.46065223 0.3629729 0.5412905 0.4517927 ]  
[0.2607263 0.36812454 0.3429086 0.4495259 ]  
[0.06300719 0.54019254 0.14296739 0.6476098 ]  
[0.2669791 0.19890074 0.34135824 0.29016858]  
[0.87356746 0.70246387 0.9497957 0.79511225]  
[0.26038226 0.71399206 0.33963236 0.7996692 ]  
[0.8516298 0.2007192 0.93304306 0.29023075]
```

```
[0.66956246 0.3738941 0.7467531 0.47004962]  
[0.06811853 0.86045426 0.15179047 0.9670953 ]  
[0.86753476 0.8741953 0.949963 0.9590749 ]  
[0.6655427 0.6978978 0.74399185 0.78895605]  
[0.85688806 0.5363432 0.9365624 0.6360047 ]  
[0.8699623 0.03510578 0.95232874 0.12755029]  
[0.86702687 0.36667222 0.9396897 0.45276904]  
[0.46897623 0.03335896 0.54400414 0.1356858 ]  
[0.66382587 0.03196649 0.74162054 0.12095462]  
[0.06083808 0.1940463 0.14144242 0.2848812 ]  
[0.4665645 0.8728285 0.5375992 0.9532938]  
[0.06964158 0.6969877 0.15532547 0.79014665]  
[0.47345555 0.20721377 0.5397464 0.28813732]  
[0.26380777 0.03153073 0.3354565 0.11999477]  
[0.06269085 0.02701677 0.1426423 0.12377939]
```

29 dữ liệu tương ứng với 29 khuôn mặt, ta thử tăng độ tin cậy đến 0.9 để xem kết quả

#duyệt từng khuôn mặt đã được phát hiện

```
for i in range(0, faces.shape[2]):  #vị trí thứ 2 của faces là độ tin cậy  
    confidence = faces[0,0, i,2]  #vị trí độ tin cậy  
    #kiểm tra xem độ tin cậy có > 0.5 hay không  
    if confidence > 0.9:
```

```
#trích xuất tọa độ
```

```
print(faces[0,0,i,3:7]) # lấy từ pt 3 tới 6
```

Kết quả:

```
[0.45528927 0.71117806 0.5378257 0.8057307 ]  
[0.25925028 0.87112087 0.3433196 0.95433 ]  
[0.44929308 0.5330197 0.5303895 0.6261024 ]  
[0.6643052 0.5299996 0.7457144 0.6382607]  
[0.06174818 0.3707326 0.13863042 0.46135658]  
[0.65804493 0.8672387 0.74079645 0.9549449 ]  
[0.64594036 0.19858676 0.73240644 0.2934181 ]  
[0.46065223 0.3629729 0.5412905 0.4517927 ]  
[0.2607263 0.36812454 0.3429086 0.4495259 ]  
[0.06300719 0.54019254 0.14296739 0.6476098 ]  
[0.2669791 0.19890074 0.34135824 0.29016858]  
[0.87356746 0.70246387 0.9497957 0.79511225]  
[0.26038226 0.71399206 0.33963236 0.7996692 ]  
[0.8516298 0.2007192 0.93304306 0.29023075]  
[0.66956246 0.3738941 0.7467531 0.47004962]  
[0.06811853 0.86045426 0.15179047 0.9670953 ]  
[0.86753476 0.8741953 0.949963 0.9590749 ]  
[0.6655427 0.6978978 0.74399185 0.78895605]  
[0.85688806 0.5363432 0.9365624 0.6360047 ]
```

[0.8699623 0.03510578 0.95232874 0.12755029]

[0.86702687 0.36667222 0.9396897 0.45276904]

[0.46897623 0.03335896 0.54400414 0.1356858 ]

[0.66382587 0.03196649 0.74162054 0.12095462]

Vậy là chỉ còn 23 khuôn mặt, so sánh với ảnh gốc (20 khuôn mặt)



Thì rõ ràng đã có sự cải thiện đáng kể.

Tiếp theo ta sẽ vẽ hình chữ nhật lên khuôn mặt đã phát hiện

Để trích xuất tọa độ hình chữ nhật ta làm như sau:

# đưa tọa độ về số nguyên để dễ vẽ hình chữ nhật

```
startx= int(faces[0,0,i,3] *w)
starty = int(faces[0,0,i,4] * h)
endx = int(faces[0,0,i,5] * w)
endy = int(faces[0,0,i,6] * h)
print(startx, starty, endx, endy)
```

Kết quả:

[0.45528927 0.71117806 0.5378257 0.8057307 ]

437 819 516 928

[0.25925028 0.87112087 0.3433196 0.95433 ]

248 1003 329 1099

[0.44929308 0.5330197 0.5303895 0.6261024 ]

431 614 509 721

[0.6643052 0.5299996 0.7457144 0.6382607]

637 610 715 735

[0.06174818 0.3707326 0.13863042 0.46135658]

59 427 133 531

[0.65804493 0.8672387 0.74079645 0.9549449 ]

631 999 711 1100

[0.64594036 0.19858676 0.73240644 0.2934181 ]

620 228 703 338

[0.46065223 0.3629729 0.5412905 0.4517927 ]

442 418 519 520

[0.2607263 0.36812454 0.3429086 0.4495259 ]

250 424 329 517

[0.06300719 0.54019254 0.14296739 0.6476098 ]

60 622 137 746

[0.2669791 0.19890074 0.34135824 0.29016858]

256 229 327 334

[0.87356746 0.70246387 0.9497957 0.79511225]

838 809 911 915

[0.26038226 0.71399206 0.33963236 0.7996692 ]

249 822 326 921

[0.8516298 0.2007192 0.93304306 0.29023075]

817 231 895 334

[0.66956246 0.3738941 0.7467531 0.47004962]

642 430 716 541

[0.06811853 0.86045426 0.15179047 0.9670953 ]

65 991 145 1114

[0.86753476 0.8741953 0.949963 0.9590749 ]

832 1007 911 1104

[0.6655427 0.6978978 0.74399185 0.78895605]

638 803 714 908

[0.85688806 0.5363432 0.9365624 0.6360047 ]

822 617 899 732

[0.8699623 0.03510578 0.95232874 0.12755029]

835 40 914 146

[0.86702687 0.36667222 0.9396897 0.45276904]

832 422 902 521

[0.46897623 0.03335896 0.54400414 0.1356858 ]

450 38 522 156

[0.66382587 0.03196649 0.74162054 0.12095462]

637 36 711 139

Từ đó, ta sẽ vẽ được hình bao quanh khuôn mặt đã phát hiện, code hoàn chỉnh như sau:

```
import cv2
```

```
import numpy as np
```

```
#tải ảnh đầu vào
```

```
img = cv2.imread('C:/computer_vision/img/nhieu_face.jpg')
```

```
#hiển thị ảnh gốc
```

```
cv2.imshow('Base image', img)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

```
#tải mô hình đã được huấn luyện từ trước
```

```
net =  
cv2.dnn.readNetFromCaffe('C:/computer_vision/models/deploy.pro  
totxt',  
                          'C:/computer_vision/models/res10_300x300_ssd_iter  
_140000_fp16.caffemodel')
```

```
#chuẩn bị dữ liệu đầu vào  
#1.0 scale ảnh về tỉ lệ 1.0, tức là ảnh ko bị co giãn  
#(300, 300) là kích thước mà mô hình yêu cầu cho đầu vào  
# (104,177,123) màu sắc trung bình  
# swapRB = False ko hoán đổi kênh màu đỏ và xanh  
blob = cv2.dnn.blobFromImage(img, 1.0, (300,300), (104,177, 123),  
swapRB= False)
```

```
#đặt dữ liệu đầu vào cho mạng  
net.setInput(blob)
```

```
#chạy mạng để phát hiện khuôn mặt  
#hàm sẽ chạy dần trên ảnh để tìm khuôn mặt  
faces= net.forward()
```

```
#lấy kích thước của ảnh đầu vào  
h = img.shape[0]
```

```
w = img.shape[1]

#in thông tin
print(faces.shape) #(1, 1, 200, 7)
#1: số lượng ảnh đầu vào, 1: số lượng ảnh đầu ra, 200 khuôn mặt dc
phát hiện, 7 thông tin
print(faces[0, 0, 0, :]) #in ra 7 trường thông tin
#[0.      1.      0.99763405 0.45528927 0.71117806 0.5378257
# 0.8057307 ]

#duyệt từng khuôn mặt đã được phát hiện
for i in range(0, faces.shape[2]): #vị trí thứ 2 của faces là độ tin cậy
    confidence = faces[0,0,i,2] #vị trí độ tin cậy
    #kiểm tra xem độ tin cậy có > 0.5 hay không
    if confidence > 0.9:
        #trích xuất tọa độ
        print(faces[0,0,i,3:7]) # lấy từ pt 3 tới 6
        # đưa tọa độ về số nguyên để dễ vẽ hình chữ nhật
        startx= int(faces[0,0,i,3] *w)
        starty = int(faces[0,0,i,4] * h)
        endx = int(faces[0,0,i,5] * w)
        endy = int(faces[0,0,i,6] * h)
```

```
print(startx, starty, endx, endy)
```

```
#vẽ hình chữ nhật quanh khuôn mặt đã phát hiện
```

```
cv2.rectangle(img, (startx, starty), (endx, endy),(0,255,0),2)
```

```
#hiển thị độ tin cậy
```

```
text = 'Face: {:.2f}%'.format(confidence*100) #2 con số sau dấu  
,
```

```
cv2.putText(img, text, (startx, starty-10),  
cv2.FONT_HERSHEY_PLAIN,1 ,(255,255,255))
```

```
#hiển thị ảnh:
```

```
cv2.imshow('Image detect face', img)
```

```
cv2.waitKey(0)
```

```
cv2.destroyAllWindows()
```

```
Kết quả:
```



Bây giờ, ta sẽ thử nhận diện khuôn mặt từ web cam:

```
import cv2
```

```
import numpy as np
```

```
#tải mô hình đã được huấn luyện từ trước
```

```
net =  
cv2.dnn.readNetFromCaffe('C:/computer_vision/models/deploy.pro  
totxt',
```

```
'C:/computer_vision/models/res10_300x300_ssd_iter  
_140000_fp16.caffemodel')
```

```
#mở webcam
```

```
cap = cv2.VideoCapture(0) #cam mặc định
```

```
while True:
```

```
    #đọc khung hình từ cam
```

```
    ret, frame = cap.read()
```

```
    #nếu ko đọc được
```

```
    if not ret:
```

```
        break
```

```
#chuẩn bị dữ liệu đầu vào
```

```
#1.0 scale ảnh về tỉ lệ 1.0, tức là ảnh ko bị co giãn
```

```
 #(300, 300) là kích thước mà mô hình yêu cầu cho đầu vào
```

```
# (104,177,123) màu sắc trung bình
```

```
# swapRB = False ko hoán đổi kênh màu đỏ và xanh
```

```
blob = cv2.dnn.blobFromImage(frame, 1.0, (300,300), (104,177,  
123), swapRB= False)
```

```
#đặt dữ liệu đầu vào cho mạng
```

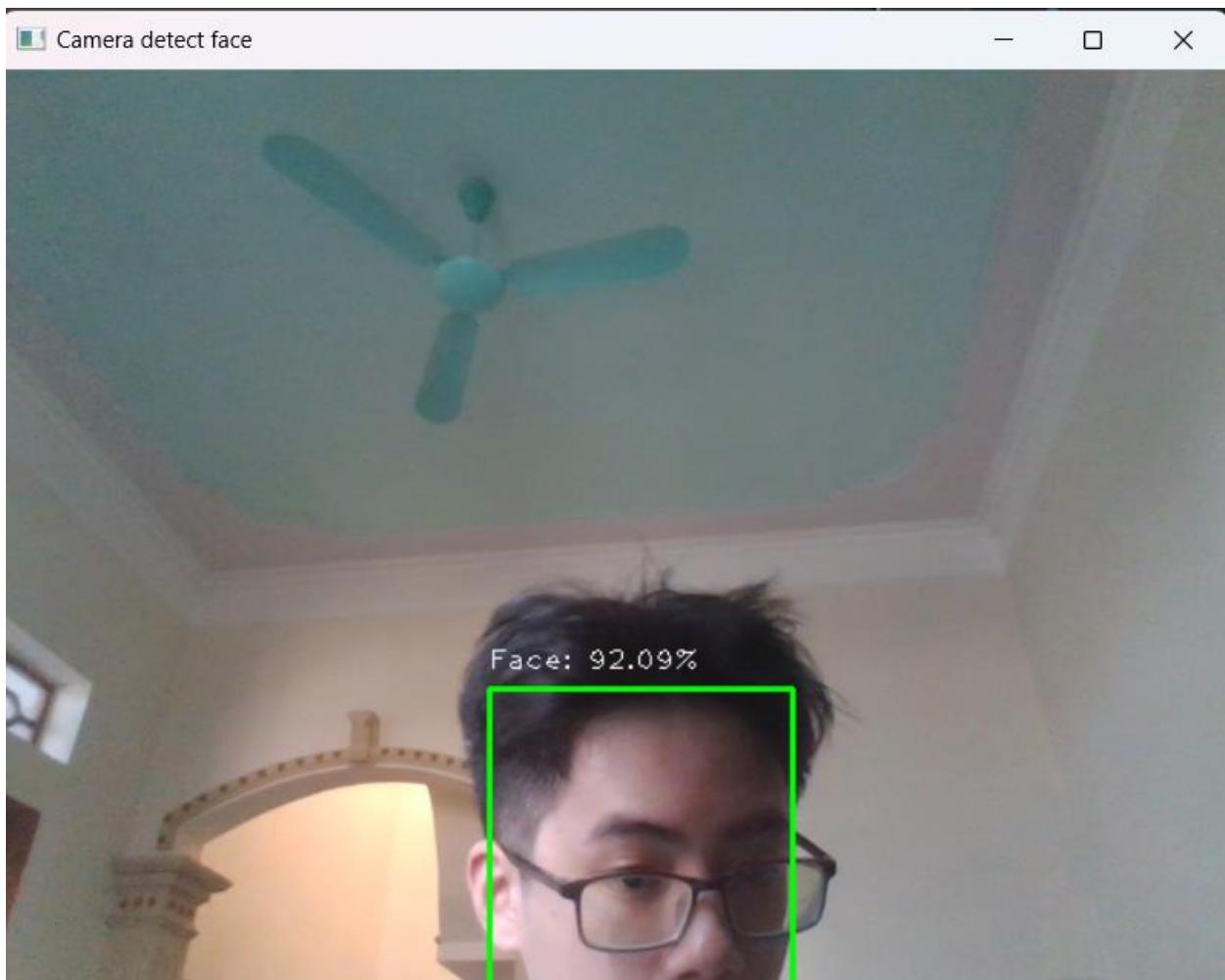
```
net.setInput(blob)
```

```
#chạy mạng để phát hiện khuôn mặt
#hàm sẽ chạy dần trên ảnh để tìm khuôn mặt
faces= net.forward()

#lấy kích thước của ảnh đầu vào
h = frame.shape[0]
w = frame.shape[1]

#duyệt từng khuôn mặt đã được phát hiện
for i in range(0, faces.shape[2]):  #vị trí thứ 2 của faces là độ tin
cậy
    confidence = faces[0,0, i,2]  #vị trí độ tin cậy
    #kiểm tra xem độ tin cậy có > 0.5 hay không
    if confidence > 0.9:
        #trích xuất tọa độ
        #print(faces[0,0,i,3:7]) # lấy từ pt 3 tới 6
        # đưa tọa độ về số nguyên để dễ vẽ hình chữ nhật
        startx= int(faces[0,0,i,3] *w)
        starty = int(faces[0,0,i,4] * h)
        endx = int(faces[0,0,i,5] * w)
        endy = int(faces[0,0,i,6] * h)
        #print(startx, starty, endx, endy)
```

```
#vẽ hình chữ nhật quanh khuôn mặt đã phát hiện  
cv2.rectangle(frame, (startx, starty), (endx, endy),(0,255,0),2)  
  
#hiển thị độ tin cậy  
text = 'Face: {:.2f}%'.format(confidence*100) #2 con số sau  
dấu ,  
cv2.putText(frame, text, (startx, starty-10),  
cv2.FONT_HERSHEY_PLAIN,1 ,(255,255,255))  
  
#hiển thị:  
cv2.imshow('Camera detect face', frame)  
if (cv2.waitKey(10) == ord('q')):  
    break  
  
cap.release()      #giải phóng camera  
cv2.destroyAllWindows()  
Kết quả:
```



## 28. Giới thiệu về Google colab



Google colab là một dạng Jupyter Notebook tùy biến cho phép thực thi Python trên nền tảng đám mây, được cung cấp bởi Google.

Sử dụng Google Colab có những lợi ích ưu việt như: sẵn sàng chạy python ở bất kỳ thiết bị nào có kết nối internet mà không cần cài đặt,

chia sẻ và làm việc nhóm dễ dàng, sử dụng miễn phí GPU cho các dự án về AI.

Một số tính năng:

- Tạo mục lục dựa trên các Heading viết bằng ngôn ngữ markdown
- Thêm hình ảnh, biểu mẫu dễ dàng với markdown
- Kết nối dễ dàng với Google Drive, Google Sheets
- Chạy Python trên Cloud hay Local Runtime (Python trên máy tính cá nhân) của bạn đều cho kết quả tốt.
- Tự động lưu lịch sử chỉnh sửa thành các phiên bản giúp bạn dễ dàng khôi phục lại phiên bản gần nhất nếu cần khi bạn gặp lỗi.
- Cho phép tìm kiếm và chèn các đoạn mã được soạn thảo sẵn trong các Template (bởi chính mình) vào Notebook.
- Tạo dashboard viết bằng Python và chia sẻ với team dễ dàng.

Link truy cập: <https://colab.google/>

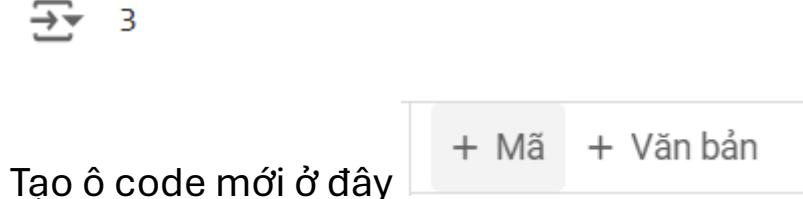
New Notebook để tạo file mới hoặc open Colab để mở file cũ.

Thử thực hiện 1 phép tính python trên colab:

The screenshot shows a single code cell in a Google Colab notebook. The cell contains the following Python code:

```
a= 1  
b=2  
c= a+b  
print(c)
```

To the left of the code is a play button icon. To the right, the output of the code is shown: "3".



Bây giờ, ta sẽ thử connect với google drive để lấy các file từ 1 đường dẫn và sau đó sẽ hiển thị thử một ảnh:

```
[1] # kết nối với google drive
from google.colab import drive
drive.mount('/content/drive')    #lệnh này sẽ yêu cầu kết nối drive
```

→ Mounted at /content/drive

```
[5] import cv2
import numpy as np
import os
from google.colab.patches import cv2_imshow
```

```
[4] path = '/content/drive/MyDrive/test_gg_colab'    #đường dẫn bắt kí trên drive
#hiển thị tất cả thư mục hoặc file
os.listdir(path)
```

['ssstik.io\_1687409377198.mp3',
 'z5208104843458\_b1f942a78b5b7ef4aca1d82f4d030ed.jpg',
 'z5208104843558\_ace3673d9ad5a3ec0ca35419d5c65cf2.jpg',
 'z5208351346020\_58ed5991cd58c67c09553c2f54fcfaa41.jpg',
 'z52161274966692\_5d01cd898d27c06693bf2741cb0ee836.jpg',
 'z5230227978776\_5644d3c25c9aad6ff31a075315e76910.jpg',
 'z5239112963639\_7a9f3b728e9ce6723e90376b6e88c4c.jpg',
 'z5252351596535\_6895da8ab85d58aaeef8eb589827b87.jpg',
 'X-WORLD-PRIVATE-FILE-DON'T MOVE AND DELETE-1714329224225',
 'Qunitichnh.zip']

```
#hiển thị một ảnh
img = cv2.imread(path + '/z5208104843458_b1f942a78b5b7ef4aca1d82f4d030ed.jpg')
cv2_imshow(img)
```

→



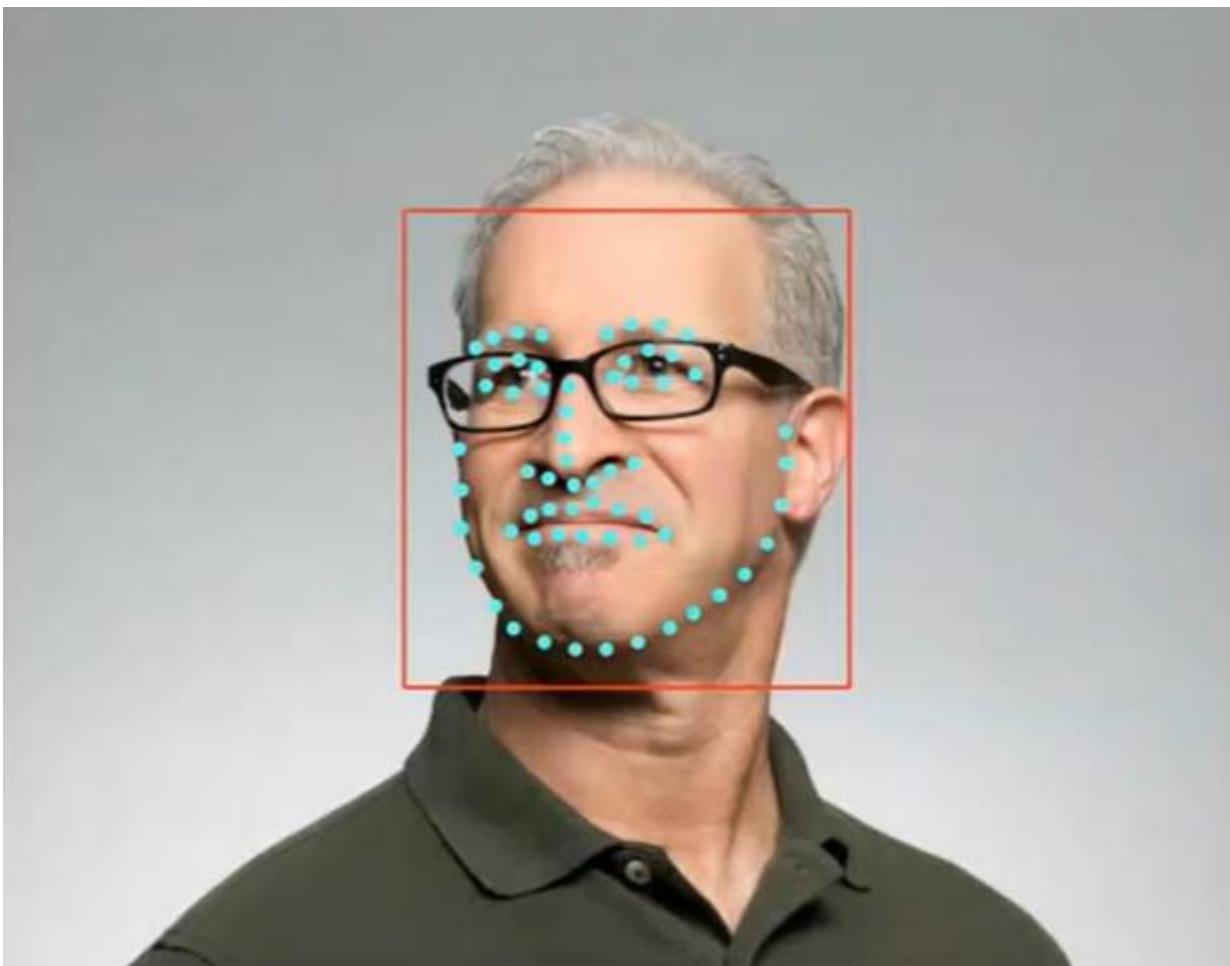
Ta có thể thử code lại các bài trước ở google colab.

## 29. Trích xuất đặc trưng khuôn mặt

Điểm đặc trưng là một điểm hoặc vùng trên hình ảnh mà có tính chất độc đáo hoặc quan trọng.

Điểm đặc trưng thường được sử dụng để mô tả và xác định các đặc điểm quan trọng của đối tượng trong hình ảnh. Điều quan trọng là chúng thường được sử dụng để nhận biết hoặc theo dõi các đối tượng trong các ứng dụng như nhận diện khuôn mặt, nhận dạng biểu

cảm, nhận dạng vật thể, và nhiều nhiệm vụ khác trong lĩnh vực thị giác máy tính.



Công nghệ nhận dạng khuôn mặt thường tìm kiếm những đặc trưng sau:

- Khoảng cách giữa hai mắt.
- Khoảng cách từ trán tới cằm.
- Khoảng cách giữa mũi và miệng.
- Độ sâu của hốc mắt.
- Hình dạng của xương gò má.
- Đường viền của môi, tai và cằm.

Ta sẽ sử dụng thư viện sau để trích xuất đặc trưng khuôn mặt:

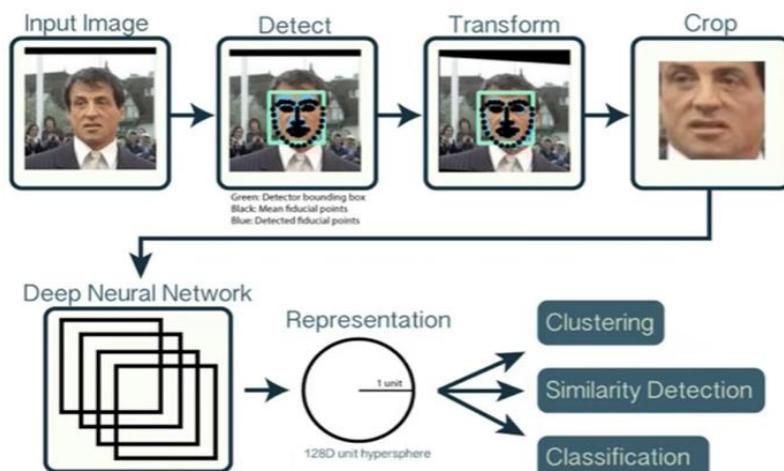
## Dlib Shape Predictors

- Dlib Shape Predictors là một công cụ và thư viện mã nguồn mở trong lĩnh vực thị giác máy tính và xử lý ảnh, được phát triển bởi Davis E. King. Dlib Shape Predictors chủ yếu được sử dụng để nhận diện và dự đoán vị trí của các điểm đặc trưng trên khuôn mặt, đặc biệt là trong việc nhận diện và theo dõi các điểm trên khuôn mặt của con người.
- Được sử dụng để dự đoán và xác định vị trí của các điểm đặc trưng trên khuôn mặt, như mắt, miệng, mũi, và các điểm trên mô hình 3D của khuôn mặt. Điều này cho phép nhận diện và phân tích khuôn mặt một cách chi tiết.

Để cài đặt dlib: `pip install dlib`

Cài đặt thư viện này khá khó, tuy nhiên google collab đã cài thư viện này sẵn cho rồi, cho nên bài này ta sẽ thực hiện trên colab.

### Các bước xử lý



Ta tiến hành download model sau và sử dụng bằng cách upload ngược lên google drive để sử dụng:

[https://github.com/ageitgey/face\\_recognition\\_models/tree/master/face\\_recognition\\_models/models](https://github.com/ageitgey/face_recognition_models/tree/master/face_recognition_models/models)

Ở bài này ta sẽ sử dụng 2 file là  
shape\_predictor\_68\_face\_landmarks.dat

dlib\_face\_recognition\_resnet\_model\_v1.dat

Sau khi xong tiến hành tải lên google drive, tất cả các file này sẽ  
được lưu trữ ở mục Facial\_feature\_extraction trên drive.

Tiến hành code:

```
✓ 1 giây  #import các thư viện
          import os
          import cv2
          import numpy as np
          from google.colab.patches import cv2_imshow
          import dlib
          from imutils import face_utils

✓ 28 giây [2] #kết nối google drive
          from google.colab import drive
          drive.mount('/content/drive')

→ Mounted at /content/drive
```

```

0 giây
[1] #thiết lập đường dẫn
path = '/content/drive/MyDrive/Facial_feature_extraction'
os.listdir(path)

[2] ['nhieu_face.jpg',
     'dlib_face_recognition_resnet_model_v1.dat',
     'shape_predictor_68_face_landmarks.dat']

[3] #thứ hiển thị một ảnh
img = cv2.imread(path + '/nhieu_face.jpg')
cv2_imshow(img)

```

The image shows a 2x5 grid of 10 different human faces, each with a neutral or slightly smiling expression. The faces are of various ethnicities and hair colors, including men and women.

# các đường dẫn đến model

```

face_landmarks_model_path =
"/content/drive/MyDrive/Facial_feature_extraction/shape_predictor_
68_face_landmarks.dat"

```

```

face_recognition_model_path =
"/content/drive/MyDrive/Facial_feature_extraction/dlib_face_recogn
ition_resnet_model_v1.dat"

```

#đường dẫn đến ảnh

```

img_path =
"/content/drive/MyDrive/Facial_feature_extraction/nhieu_face.jpg"

```

#tạo bộ nhận diện khuôn mặt

```

face_detector = dlib.get_frontal_face_detector()

```

#đọc hình ảnh và phát hiện ra các khuôn mặt

```

img= cv2.imread(img_path)

```

```
#nhận diện các khuôn mặt bên trong ảnh
faces = face_detector(img)

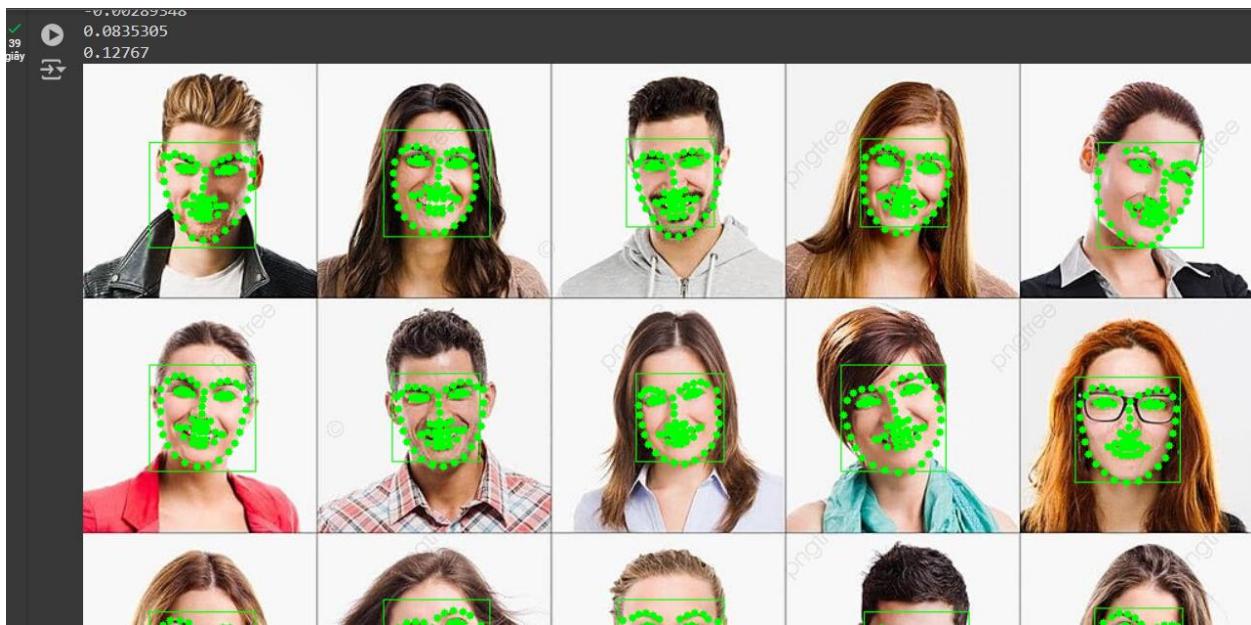
#vòng lặp qua từng khuôn mặt
for face in faces:
    point_a = face.left(), face.top()
    point_b = face.right(), face.bottom()
    #phát hiện ra các điểm đặc trưng trên khuôn mặt
    shape_predictor =
    dlib.shape_predictor(face_landmarks_model_path)
    face_shape = shape_predictor(img, face)
    face_shape_array = face_utils.shape_to_np(face_shape) #mảng
    chứa các đặc trưng khuôn mặt
    #tính toán đặc trưng của khuôn mặt
    shape_descriptor =
    dlib.face_recognition_model_v1(face_recognition_model_path)
    face_descriptors =
    shape_descriptor.compute_face_descriptor(img, face_shape)
    print(face_descriptors)

    #đoạn này cần chạy ctr = runtime -> thay đổi loại thời gian chạy ->
    T4 GPU -> lưu và run lại từ đầu
    #để run lại từ đầu -> thời gian chạy -> khởi động lại phiên và chạy
    tất cả ô
    # kết quả có thể lên tới 128 con số đặc trưng cho từng đặc trưng
    khuôn mặt gì đó
```

```
#vẽ hình chữ nhật và các điểm đặc trưng lên hình ảnh  
cv2.rectangle(img, point_a, point_b, (0, 255, 0))  
  
#vẽ điểm đặc trưng  
for point in face_shape_array:  
    cv2.circle(img, tuple(point), 3, (0,255,0), -1)  
  
  
cv2_imshow(img) #alt+enter để run nhanh
```

Kết quả:

```
✓ 39 iây  cv2_imshow(img)  #alt+enter để run nhanh
-0.039184
0.0345879
-0.0818732
0.112198
-0.343743
-0.0584958
0.274618
-0.0352734
0.0779343
0.064078
-0.10342
-0.0969494
0.17956
-0.216909
0.0344175
0.00277083
-0.123427
0.0241302
-0.192661
0.134941
0.0640449
-0.112243
-0.0765319
0.190952
-0.134271
-0.0461973
0.185285
-0.0870732
-0.225203
-0.234688
0.0298181
```



Như vậy, khuôn mặt đã được bo bằng hình chữ nhật và các đặc trưng thì được chấm các chấm xanh.

### 30. Tiền xử lí ảnh cho bài toán nhận diện khuôn mặt

Ta sẽ tiếp tục sử dụng google colab để làm cho đỡ phải cài đặt nhiều thư viện.

Ta sẽ tải lên drive những thư mục cần thiết để làm, bao gồm những tệp sau( cách tải ở những bài trước):

res10\_300x300\_ssd\_iter\_140000\_fp16.caffemodel

deploy.prototxt

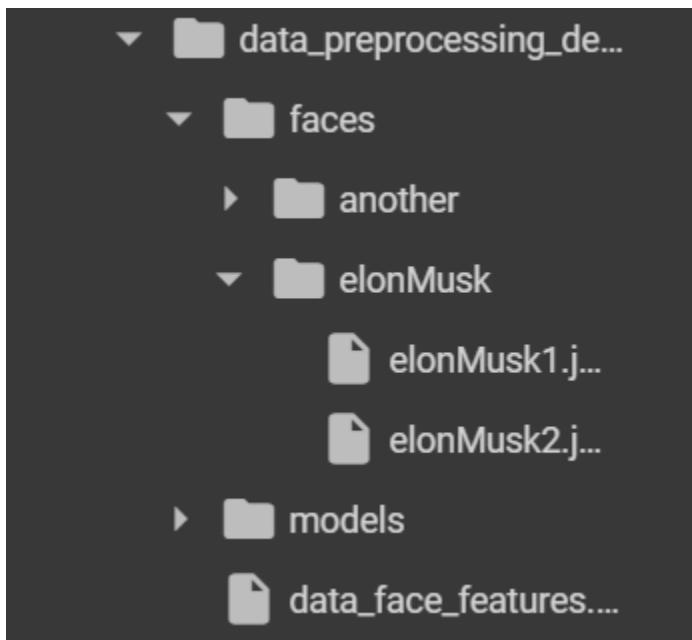
đồng thời tải openface.nn4.small2.v1.v7 để up lên drive ở link sau:

<https://github.com/pyannote/pyannote-data/blob/master/openface.nn4.small2.v1.t7>

Sau đó, ta cần lấy rất nhiều hình ảnh về một số đối tượng để trích xuất đặc trưng khuôn mặt.

Lưu ý, ở mục faces, ta chỉ chứa thư mục (bên trong thư mục sẽ chứa ảnh) bởi ta sẽ truy xuất đến thư mục để đọc ảnh trong nó.

Cấu trúc trên drive sẽ như sau:



Tiến hành code:

```
[1] #import các thư viện cần thiết
import numpy as np
import cv2
import pandas as pd
import os
import pickle    #giúp lưu data và tải data lên
from google.colab.patches import cv2_imshow
from google.colab import drive

[2] #kết nối google colab
drive.mount('/content/drive')
path = '/content/drive/MyDrive/data_preprocessing_detect_face'
#hiển thị tất cả thư mục và file
os.listdir(path)

→ Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
['models', 'faces']
```

```
0 # lấy model
faceDetectionModel = '/content/drive/MyDrive/data_preprocessing_detect_face/models/res10_300x300_ssd_iter_140000_fp16.caffemodel'
## mô tả kiến trúc
faceDetectionProto = '/content/drive/MyDrive/data_preprocessing_detect_face/models/deploy.prototxt'
## mô hình mô tả khuôn mặt
faceDescriptor = '/content/drive/MyDrive/data_preprocessing_detect_face/models/openface_nn4.small12.v1.t7'
## sử dụng openCV DNN đọc mô hình nhận diện khuôn mặt
detectorModel = cv2.dnn.readNetFromCaffe(faceDetectionProto, faceDetectionModel)
## đọc mô hình mô tả khuôn mặt từ file Torch
descriptorModel = cv2.dnn.readNetFromTorch(faceDescriptor)

[4] # danh sách hình ảnh chuẩn bị
faces = '/content/drive/MyDrive/data_preprocessing_detect_face/faces'
os.listdir(faces)

→ ['elonMusk', 'another']
```

```
✓ [5] #hiển thị một hình ảnh
giây imgFace = '/content/drive/MyDrive/data_preprocessing_detect_face/faces/another/mat.jpg'
      img = cv2.imread(imgFace)
      cv2.imshow(img)
```



#trích xuất đặc trưng cho 1 ảnh

```
path =
'/content/drive/MyDrive/data_preprocessing_detect_face/faces/ano
ther/mat.jpg'

img1 = cv2.imread(imgFace)

#copy cái ảnh

image = img1.copy()

#lấy chiều cao và chiều rộng ảnh

h, w = image.shape[:2]

# chuẩn bị dữ liệu đầu vào cho mô hình nhận diện khuôn mặt

imgBlob = cv2.dnn.blobFromImage(image, 1,(300,300),
(104,177,123), swapRB = False, crop = False)

# thiết lập đầu vào cho mô hình

detectorModel.setInput(imgBlob)

# thực hiện việc nhận diện khuôn mặt
```

```
detections = detectorModel.forward()

# kiểm tra xem có khuôn mặt nào hay không

if (len(detections)> 0):

    #chọn khuôn mặt có độ tin cậy cao nhất

    i = np.argmax(detections[0,0 ,:, 2])  #giá trị thứ 2 trong 7 giá trị cuối
    cùng là confidence(độ tin cậy)

    confidence = detections[0,0, i,2]

    # kiểm tra xem độ tin cậy có lớn hơn 0.9 không (0.5 là được nhưng
    để chính xác hơn thì lấy 0.9)

    if (confidence > 0.9):

        #tính toán hộp bao quanh khuôn mặt

        box = detections[0,0, i, 3:7] * np.array([w, h, w, h])
        (startX , startY, endX, endY) = box.astype('int')

        #trích xuất vùng ảnh chứa khuôn mặt ra

        roi = image[startY: endY, startX: endX]

        cv2_imshow(roi)

        #chuẩn bị dữ liệu đầu vào cho mô hình trích xuất đặc trưng

        faceBlob = cv2.dnn.blobFromImage(roi, 1/255, (96,96), (0,0,0),
        swapRB = True, crop = True)

        #thiết lập đầu vào cho mô hình

        descriptorModel.setInput(faceBlob)

        #thực hiện việc trích xuất đặc trưng

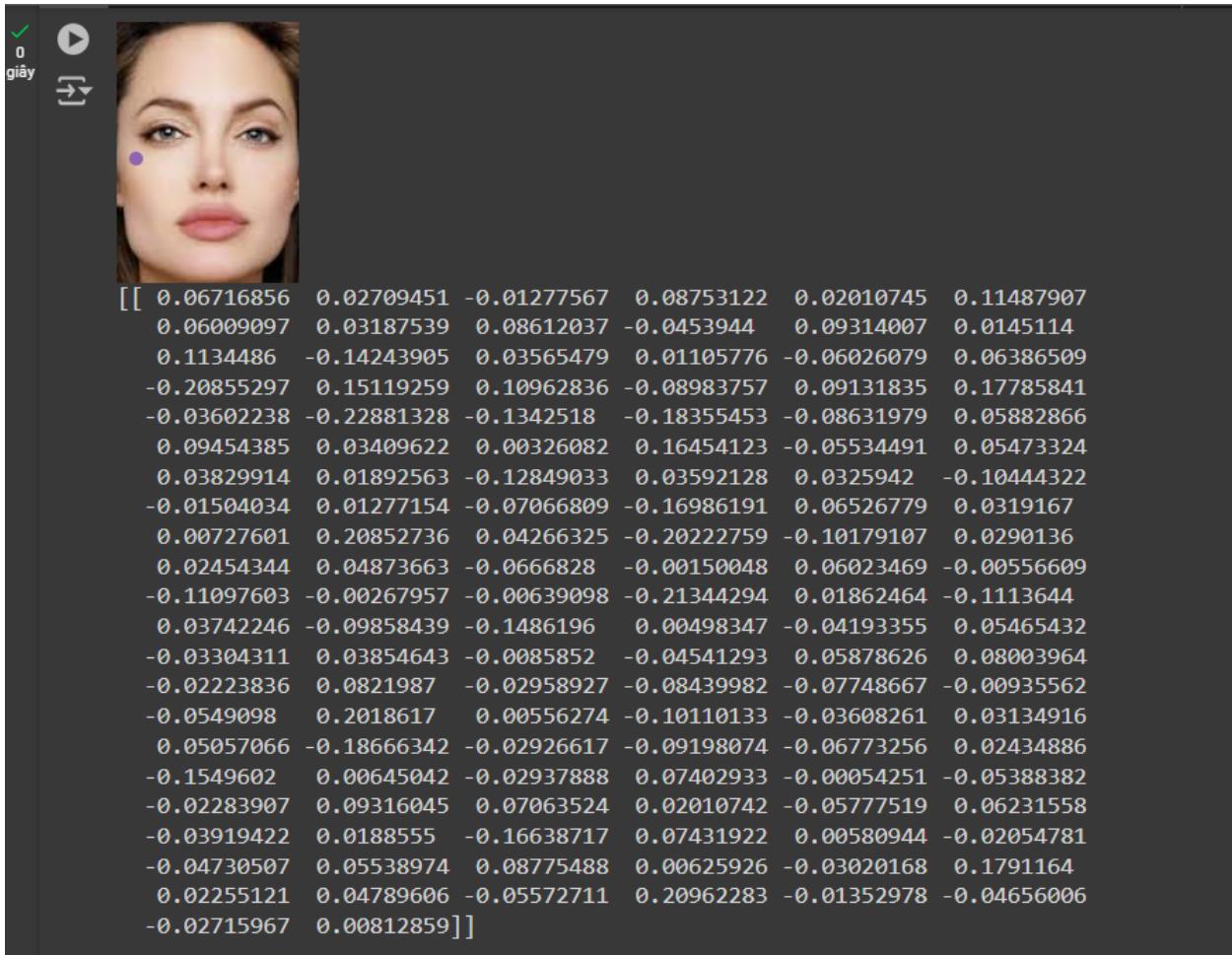
        vectors = descriptorModel.forward()
```

```
#in vectors
print(vectors)

# tất nhiên, thực tế sẽ lấy số liệu đã trích xuất để so sánh với csdl
có sẵn -> nhận diện

# vấn đề là nếu người đó có nhiều ảnh -> duyệt tất cả các ảnh đó ->
tính trung bình vectors -> số liệu sát nhất có thể
```

Kết quả:



```
[[ 0.06716856  0.02709451 -0.01277567  0.08753122  0.02010745  0.11487907
  0.06009097  0.03187539  0.08612037 -0.0453944   0.09314007  0.0145114
  0.11344486 -0.14243905  0.03565479  0.01105776 -0.06026079  0.06386509
  -0.20855297  0.15119259  0.10962836 -0.08983757  0.09131835  0.17785841
  -0.03602238 -0.22881328 -0.1342518   -0.18355453 -0.08631979  0.05882866
  0.09454385  0.03409622  0.00326082  0.16454123 -0.05534491  0.05473324
  0.03829914  0.01892563 -0.12849033  0.03592128  0.0325942   -0.10444322
  -0.01504034  0.01277154 -0.07066809 -0.16986191  0.06526779  0.0319167
  0.00727601  0.20852736  0.04266325 -0.20222759 -0.10179107  0.0290136
  0.02454344  0.04873663 -0.0666828   -0.00150048  0.06023469 -0.00556609
  -0.11097603 -0.00267957 -0.00639098 -0.21344294  0.01862464 -0.1113644
  0.03742246 -0.09858439 -0.1486196   0.00498347 -0.04193355  0.05465432
  -0.03304311  0.03854643 -0.0085852   -0.04541293  0.05878626  0.08003964
  -0.02223836  0.0821987  -0.02958927 -0.08439982 -0.07748667 -0.00935562
  -0.0549098   0.2018617   0.00556274 -0.10110133 -0.03608261  0.03134916
  0.05057066 -0.18666342 -0.02926617 -0.09198074 -0.06773256  0.02434886
  -0.1549602   0.00645042 -0.02937888  0.07402933 -0.00054251 -0.05388382
  -0.02283907  0.09316045  0.07063524  0.02010742 -0.05777519  0.06231558
  -0.03919422  0.0188555  -0.16638717  0.07431922  0.00580944 -0.02054781
  -0.04730507  0.05538974  0.08775488  0.00625926 -0.03020168  0.1791164
  0.02255121  0.04789606 -0.05572711  0.20962283 -0.01352978 -0.04656006
  -0.02715967  0.00812859]]
```

# Tạo ra một functions để áp dụng cho nhiều ảnh

```
def myDetect(image_path):
```

# Trích xuất đặc trưng cho một ảnh

```
img1 = cv2.imread(image_path)

# Copy cái ảnh
image = img1.copy()

# Lấy chiều cao và chiều rộng của ảnh
h, w = image.shape[:2]

# Chuẩn bị dữ liệu đầu vào cho mô hình nhận diện khuôn mặt
imgBlob = cv2.dnn.blobFromImage(image, 1, (300,300), (104, 177,
123), swapRB=False, crop=False)

# Thiết lập đầu vào cho mô hình
detectorModel.setInput(imgBlob)

# Thực hiện việc nhận diện khuôn mặt
detections = detectorModel.forward()

# Kiểm tra xem có khuôn mặt nào hay không?
if (len(detections)>0):
    # Chọn khuôn mặt có độ tin cậy cao nhất (confidence) cao nhất
    i = np.argmax(detections[0, 0, :, 2]) # Giá trị thứ 2 trong 7 giá trị
    cuối cùng => confidence
```

```
confidence = detections[0, 0, i, 2]

# Kiểm tra độ tin cậy có lớn hơn 0.5
if (confidence>0.5):
    # Tính toán hộp bao quanh khuôn mặt
    box = detections[0, 0, i, 3:7]* np.array([w, h, w, h])
    (startX, startY, endX, endY) =box.astype('int')

    # Trích xuất vùng ảnh chứa khuôn mặt ra
    roi = image[startY:endY, startX:endX]
    # cv2_imshow(roi)

    # Chuẩn bị dữ liệu đầu vào cho mô hình trích xuất đặc trưng
    faceBlob = cv2.dnn.blobFromImage(roi, 1/255, (96,96), (0,0,0),
swapRB=True, crop=True)

    # Thiết lập đầu vào cho mô hình
    descriptorModel.setInput(faceBlob)

    # Thực hiện việc trích xuất đặc trưng
    vectors = descriptorModel.forward()
```

```
# Print vectors
return vectors

# Apply for all images

# Khởi tạo một dictionary
data = dict(data=[], label=[])

myDir =
'/content/drive/MyDrive/data_preprocessing_detect_face/faces'
folders = os.listdir(myDir)

# Duyệt qua từng thư mục
for folder in folders:
    path = myDir + "/" + folder
    print(path)

    files = os.listdir(path)
    for fileName in files:
        try:
            # Gọi hàm trích xuất đặc trưng
            vector = myDetect(path + '/' + fileName)
```

```
# Kiểm tra việc trích xuất đặc trưng thành công:  
if vector is not None:  
    #Thêm vector đặc trưng vào trong từ điển với nhãn là tên của  
    #thư mục  
    data['data'].append(vector)  
    data['label'].append(folder)  
    print('Trích xuất đặc trưng thành công')  
  
except:  
    # Bỏ qua nếu gặp lỗi  
    Pass
```

Kết quả:

```
↳ /content/drive/MyDrive/data_preprocessing_detect_face/faces/elonMusk  
Trích xuất đặc trưng thành công  
Trích xuất đặc trưng thành công  
/content/drive/MyDrive/data_preprocessing_detect_face/faces/another  
Trích xuất đặc trưng thành công  
Trích xuất đặc trưng thành công  
Trích xuất đặc trưng thành công
```

```
✓ 0 [9] data.keys()
giây ➔ dict_keys(['data', 'label'])

✓ 0 [10] # Tạo một series từ dictionary
giây    labelSeries = pd.Series(data['label'])
        # Đếm số lần xuất hiện của mỗi nhãn
        labelCounts = labelSeries.value_counts()
        print(labelCounts)

➔ another      3
      elonMusk   2
      Name: count, dtype: int64

✓ 0 ➡ # lưu trữ file vì việc chạy sẽ tốn time
      # Save data -> file
      print('Lưu file data thành công')
      fileName = '/content/drive/MyDrive/preprocessing_detect_face/data_face_features.pickle'
      pickle.dump(data, open(fileName, mode='wb'))

➔ Lưu file data thành công
```

Tổng kết, ta đã trích xuất đặc điểm khuôn mặt thành công và đã lưu data thành file như sau trên drive:

 **data\_face\_features.pickle**

Tuy vậy, việc train này đòi hỏi nhiều ảnh đầu vào và thời gian.

# Source code

Update tại: [https://github.com/noname1711/computer\\_vision](https://github.com/noname1711/computer_vision)