BÀI THỰC HÀNH 3

- *Xử lý ảnh cơ bản với OpenCV*

Nội dung:

Thực hành trên Visual Studio Code (vscode) với môi trường ảo Python

- Vẽ trực tiếp trên ảnh sử dụng chuột
- Chuyển đổi giữa các không gian màu (color spaces)
- Trộn và dán ảnh (các phép toán số học trên ảnh)

1. VẼ TRỰC TIẾP TRÊN ẢNH SỬ DỤNG CHUỘT

Trong phần này chúng ta sẽ thực hiện vẽ trực tiếp trên ảnh bằng chuột. Chúng ta có thể sử dụng callbacks để kết nối ảnh đến các hàm sự kiện bằng OpenCV. Điều này sẽ cho phép tương tác trực tiếp với các bức ảnh và xa hơn nữa là tương tác với các video.

Yêu cầu:

1. Nghiên cứu và bổ sung ý nghĩa các lệnh cho phần nội dung ghi chú "#...." trong các đoạn chương trình sau?

Script 1: Kết nối đến hàm cho thao tác Drawing

```
import cv2
import numpy as np
# ...
def draw_circle(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(img,(x,y),10,(0,255,0),-1)

# ...
img = np.zeros((512,512,3), np.uint8)
# ....
cv2.namedWindow(winname='my_drawing')
# ....
cv2.setMouseCallback('my_drawing',draw_circle)

while True: #....
    #...
    cv2.imshow('my_drawing',img)
    # EXPLANATION FOR THIS LINE OF CODE:
    # https://stackoverflow.com/questions/35372700/whats-0xff-for-in-cv2-waitkey1/39201163
    if cv2.waitKey(20) & 0xFF == 27:
        break
#...
cv2.destroyAllWindows()
```

[6] ✓ 11.1s

Script 2: Bổ sung chức năng bằng chọn lựa sự kiện

```
import cv2
import numpy as np

#...
def draw_circle(event,x,y,flags,param):
    if event == cv2.EVENT_LBUTTONDOWN:
        cv2.circle(img,(x,y),100,(0,255,0),-1)
    elif event == cv2.EVENT_RBUTTONDOWN:
        cv2.circle(img,(x,y),100,(0,0,255),-1)
```

```
#...
img = np.zeros((512,512,3), np.uint8)
#...
cv2.namedWindow(winname='my_drawing')
#...
cv2.setMouseCallback('my_drawing',draw_circle)

while True: #...
    #...
    cv2.imshow('my_drawing',img)
    #...
    if cv2.waitKey(20) & 0xFF == 27:
        break
#...
cv2.destroyAllWindows()
```

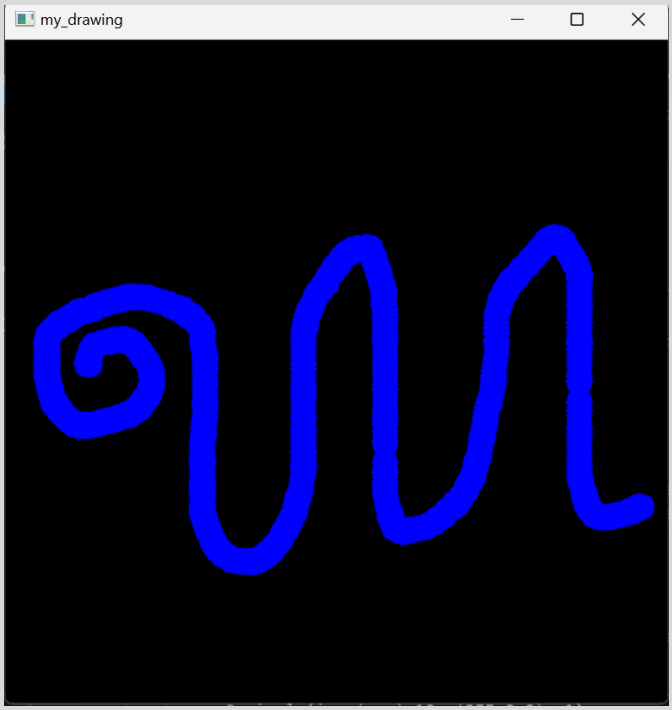
Script 3: Drag chuột

```
import cv2
import numpy as np
#...
drawing = False # True khi chuột được nhấn
ix,iy = -1,-1
```

```
#...
def draw_rectangle(event,x,y,flags,param):
    global ix,iy,drawing,mode
    if event == cv2.EVENT_LBUTTONDOWN:
        #...
        drawing = True
        #...
        ix,iy = x,y
    elif event == cv2.EVENT_MOUSEMOVE:
        #...
        if drawing == True:
            # Nếu drawing bằng True, nghĩa là...
            #...
            cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
    elif event == cv2.EVENT_LBUTTONUP:
        #...
        drawing = False
        #...
        cv2.rectangle(img,(ix,iy),(x,y),(0,255,0),-1)
```

```
#...
cv2.namedWindow(winname='my_drawing')
#...
cv2.setMouseCallback('my_drawing',draw_rectangle)
while True: #...
    #...
    cv2.imshow('my_drawing',img)
    #...
    if cv2.waitKey(1) & 0xFF == 27:
        break
#...
cv2.destroyAllWindows()
```

2. *Dựa trên đoạn chương trình Script 3, xây dựng hàm draw_circle cho phép vẽ liên tục các hình tròn có bán kính 10, màu xanh lam (blue) khi người dùng drag chuột như minh họa trong hình sau:*



2. CHUYỂN ĐỔI GIỮA CÁC KHÔNG GIAN MÀU

Hàm **`cv2.cvtColor()`** cho phép thực hiện chuyển đổi giữa các không gian màu. Có hơn 150 phương thức chuyển đổi không gian màu dựng sẵn trong openCV. Chúng ta sẽ làm quen với một số phương thức chuyển đổi không gian màu sau đây.

❖ **Cú pháp chung:**

```
cv2.cvtColor(src, code[, dst[, dstCn]])
```

Trong đó:

src: ảnh đầu vào, toàn bộ không gian màu của ảnh sẽ thay đổi

code: mã không gian màu sẽ chuyển

dst: ảnh đầu ra có cùng kích thước và độ sâu bit với ảnh đầu vào, (tùy chọn).

dstCn: số lượng kênh trong ảnh đầu ra. Nếu tham số có giá trị 0 thì số lượng kênh được lấy tự động từ src và code, (tùy chọn).

Giá trị trả về: trả về một hình ảnh

Yêu cầu:

1. Thực hiện thứ tự các yêu cầu sau:

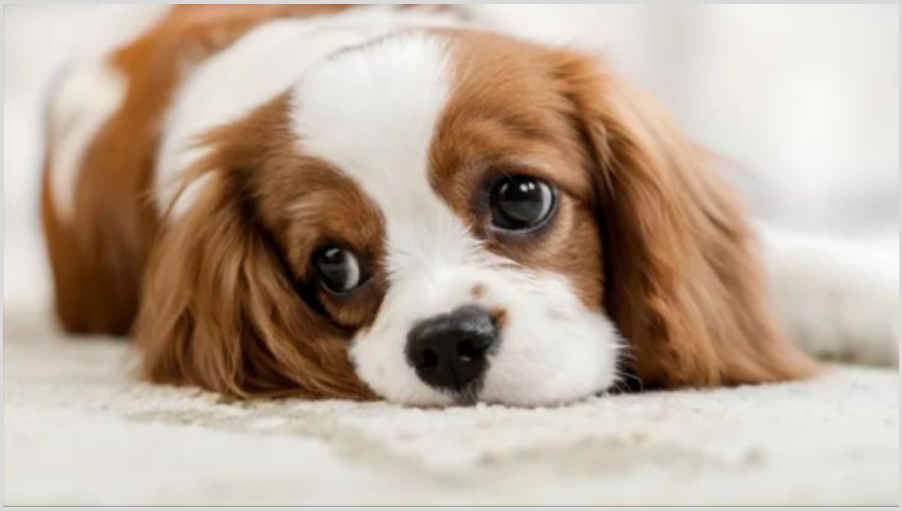
1.1. Tải ảnh bất kỳ về từ Internet

1.2. Đọc ảnh bằng hàm cv2.imread()

1.3. Viết đoạn chương trình thực hiện chuyển đổi ảnh đọc được sang các không gian màu RGB, HSV, HLS. Lưu ý, ảnh trong openCV sử dụng không gian màu BGR

1.4. So sánh các kết quả chuyển đổi không gian màu

2. Đọc vào ảnh sau:



```
img=cv2.imread('images/puppy1.jpg')
```
✓ 0.0s

2.1. Thực hiện chuyển ảnh sang không gian màn HSV với 2 câu lệnh như sau:

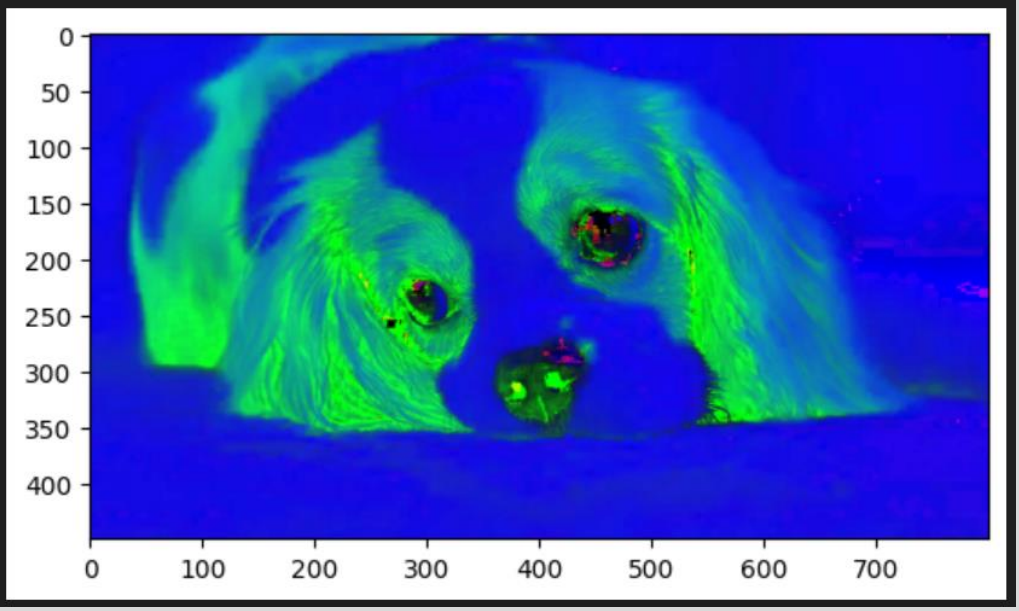
Câu lệnh 1:

```
[17] hsv_img=cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
```
✓ 0.0s

- Kết quả thu được:

```
[18] plt.imshow(hsv_img)
```
✓ 0.2s

```
<matplotlib.image.AxesImage at 0x15339355160>
```

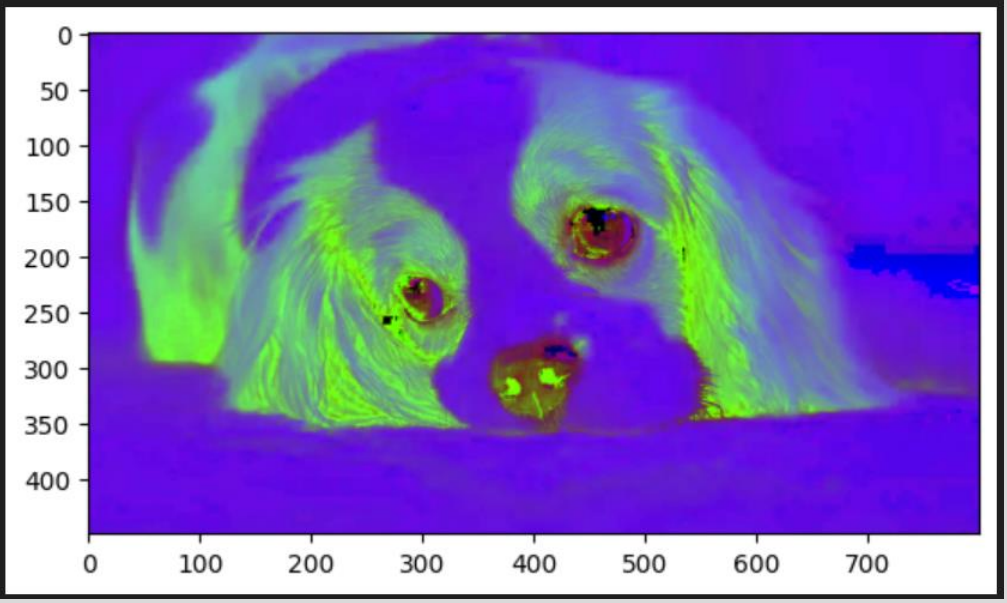


Câu lệnh 2:

```
[26]  hsv_img2=cv2.cvtColor(img,cv2.COLOR_RGB2HSV)
      ✓ 0.0s
```

- Kết quả thu được:

```
[27]  plt.imshow(hsv_img2)
      ✓ 0.1s
...   <matplotlib.image.AxesImage at 0x1533c64df70>
```



2.2. Hãy phân tích và giải thích nguyên nhân sự khác biệt giữa 2 kết quả chuyển đổi không gian màu trên? Đâu sẽ là ảnh kết quả đúng trong không gian màu HSV?

2.3. Đề xuất hướng giải quyết cho phương pháp chuyển đổi không gian màu còn lại để thu được ảnh đúng trong không gian màu HSV? (Lưu ý, Câu lệnh 1(2) không thay đổi)

3. TRỘN VÀ DÁN ẢNH (BLENDING AND PASTING) – CÁC PHÉP TOÁN SỐ HỌC TRÊN ẢNH

3.1. Trộn ảnh với hàm `cv2.addWeighted()`

Ý tưởng của trộn ảnh chỉ đơn giản là thực hiện thêm trọng số cho mỗi pixel trên cả 2 ảnh và kết hợp chúng lại với nhau.

Để trộn ảnh, chúng ta sử dụng công thức đơn giản sau:

$$new\_pixel = \boldsymbol{\alpha} \times pixel\_1 + \boldsymbol{\beta} \times pixel\_2 + \boldsymbol{\gamma}$$

- Đọc vào 2 ảnh:

```
[17]  # Two images
      img1 = cv2.imread('images/dog_backpack.png')
      img2 = cv2.imread('images/watermark_no_copy.png')
      ✓ 0.0s
```

- Kiểm tra kích thước 2 ảnh:

```
[15]  img1.shape
      ✓ 0.0s
...   (1401, 934, 3)
```

```
[18]  img2.shape
      ✓ 0.0s
...   (1280, 1277, 3)
```

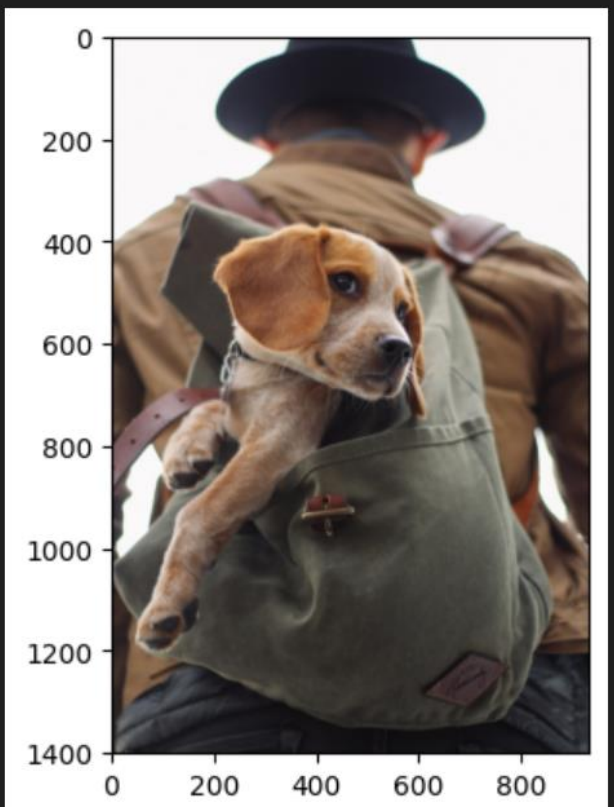
Ảnh img1 có kích thước lớn hơn so với img2

- Điều chỉnh không gian màu của ảnh để hiển thị đúng với matplotlib

```
[19]  img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
      img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
      ✓ 0.0s
```

- Hiển thị ảnh img1:

```
[20]  plt.imshow(img1)
      ✓ 0.4s
...   <matplotlib.image.AxesImage at 0x1e0e487b590>
```

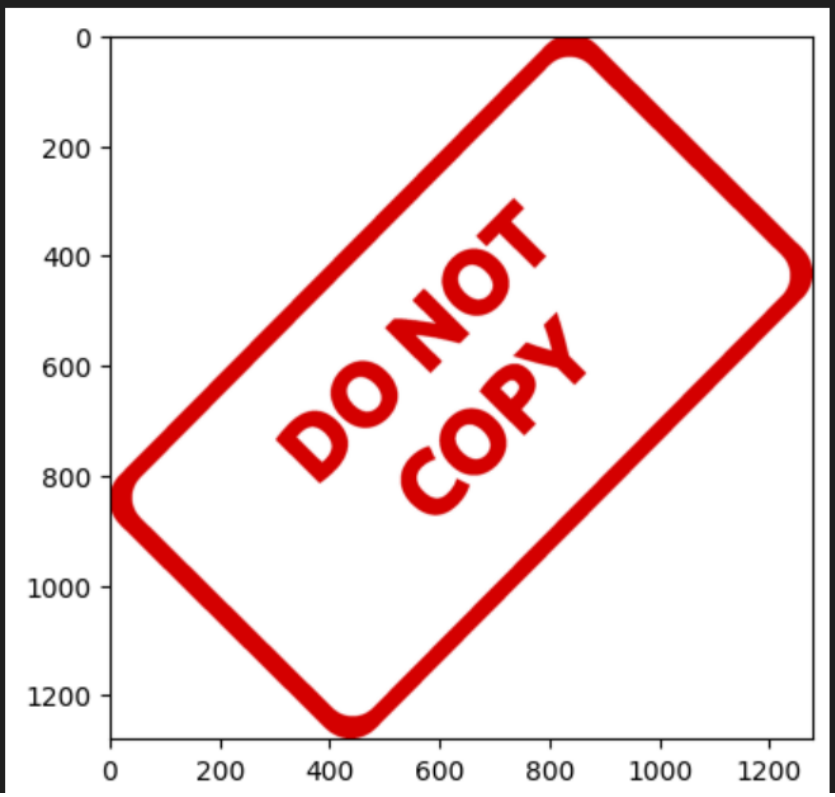


- Hiển thị ảnh img2:

```
plt.imshow(img2)
```

[21] ✓ 0.4s

```
<matplotlib.image.AxesImage at 0x1e0e885e780>
```



- Chuyển 2 ảnh về cùng kích thước:

```
img1 =cv2.resize(img1,(1200,1200))
img2 =cv2.resize(img2,(1200,1200))
```

[10]

- Kiểm tra kết quả thay đổi kích thước ảnh:

```
img1.shape
```

[13]

```
(1200, 1200, 3)
```

```
img2.shape
```

[14]

```
(1200, 1200, 3)
```

- Hiển thị lại ảnh img1 và img2:

```
plt.imshow(img1)
```

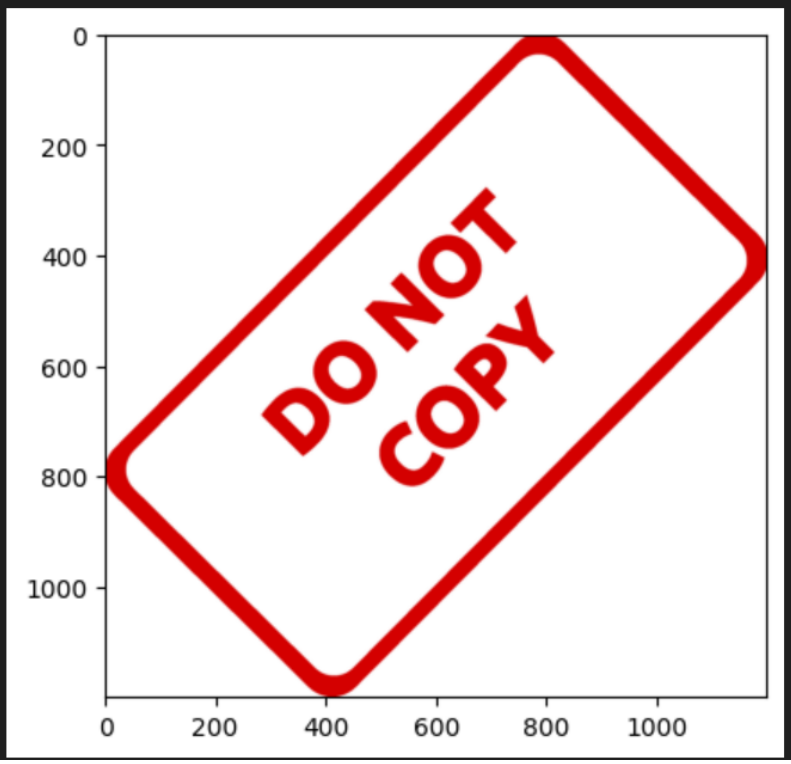
[24] ✓ 0.3s

```
<matplotlib.image.AxesImage at 0x1e0e4a529f0>
```


```
plt.imshow(img2)
```

[25] ✓ 0.3s

```
<matplotlib.image.AxesImage at 0x1e0e4ac70b0>
```



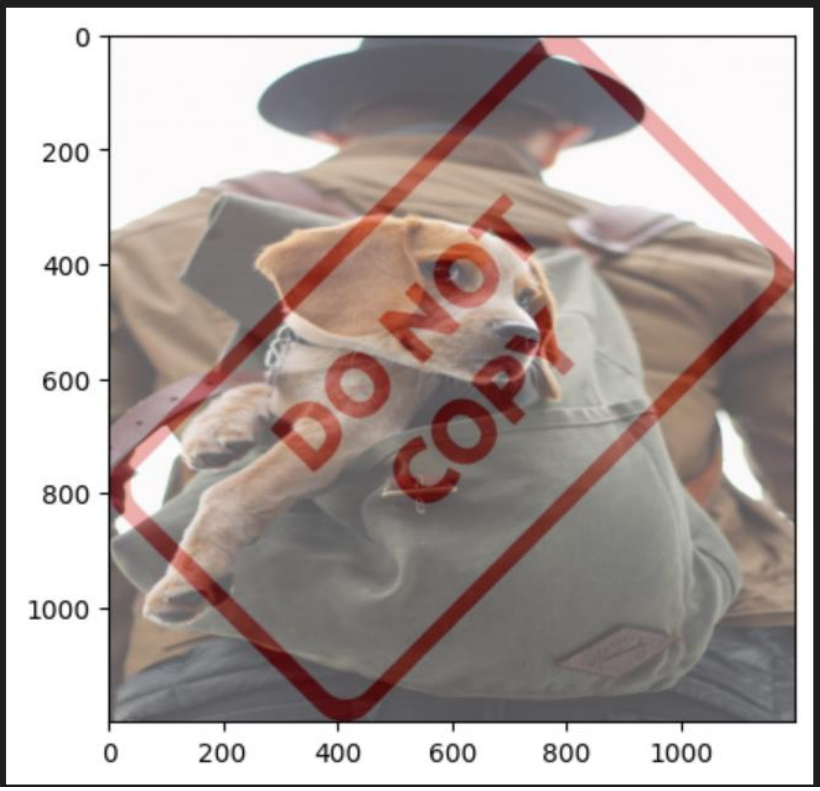
- Trộn ảnh với hàm **`cv2.addWeighted()`** theo công thức:

$$new\_pixel = \boldsymbol{\alpha} \times pixel\_1 + \boldsymbol{\beta} \times pixel\_2 + \boldsymbol{\gamma}$$

```
[26]  blended = cv2.addWeighted(src1=img1,alpha=0.7,src2=img2,beta=0.3,gamma=0)
      ✓ 0.0s
```

- Kết quả thu được:

```
[27]  plt.imshow(blended)
      ✓ 0.4s
...   <matplotlib.image.AxesImage at 0x1e0e886d310>
```



Yêu cầu:
Thay đổi giá trị các trọng số alpha, beta và gamma của hàm **`cv2.addWeighted()`** *và nhận xét về ảnh kết quả thu được?*

3.2. Xếp chồng các hình ảnh khác kích thước lên nhau

Chúng ta có thể sử dụng thủ thuật nhanh này để nhanh chóng xếp chồng các hình ảnh có kích thước khác nhau lên nhau, bằng cách chỉ định lại giá trị của hình ảnh lớn hơn để khớp với hình ảnh nhỏ hơn.

- Đọc lại 2 ảnh
- Thay đổi kích thước ảnh img2 về kích thước 600×600
- Xử lý chuyển đổi không gian màu sang RGB
- Gán ảnh large_img (ảnh lớn) bằng ảnh img1 và ảnh small_img (ảnh nhỏ) bằng ảnh img2

```
img1 = cv2.imread('images/dog_backpack.png')
img2 = cv2.imread('images/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

large_img = img1
small_img = img2
```
[17]

- Định tọa độ x, y bắt đầu xếp chồng ảnh nhỏ lên ảnh lớn (góc trên bên trái)

```
x_offset=0
y_offset=0
```
[18]

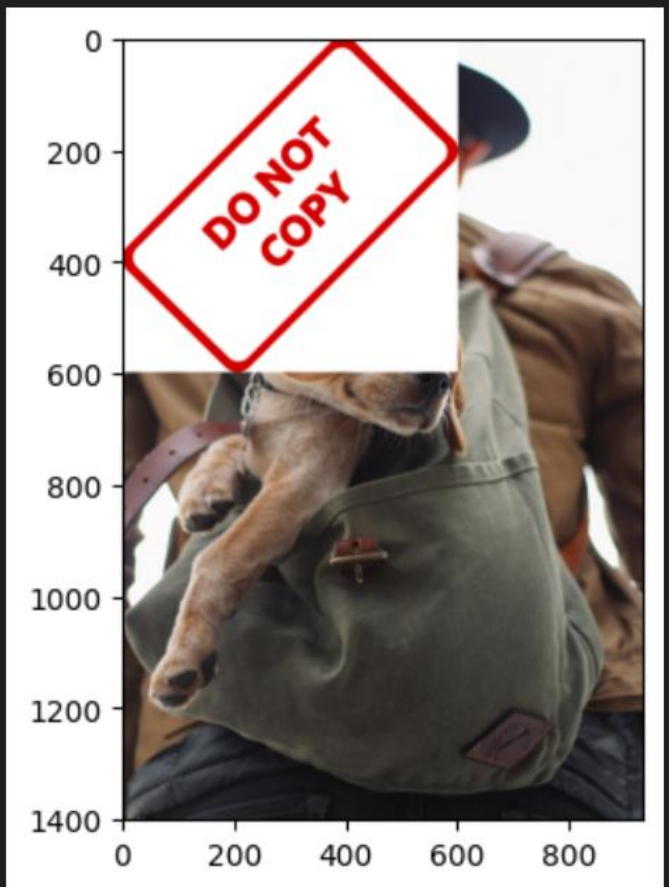
- Chỉnh định lại vùng pixel trên ảnh lớn bằng ảnh nhỏ

```
large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img
```
[19]

- Kết quả thu được:

```
plt.imshow(large_img)
```
[10] ✓ 0.2s

```
<matplotlib.image.AxesImage at 0x1df5975d6d0>
```



Yêu cầu:
Thay đổi vị trí bắt đầu xếp chồng ảnh nhỏ lên ảnh lớn trong ví dụ trên và ghi nhận các kết quả thu được?

3.3. Trộn ảnh với kích thước khác nhau

Yêu cầu:
Nghiên cứu và trình bày các phép toán trên bit trong openCV-Python (Cú pháp, Ví dụ minh họa) thông qua link tham khảo sau đây:
https://www.geeksforgeeks.org/arithmetic-operations-on-images-using-opencv-set-2-bitwise-operations-on-binary-images/

3.3.1. Đọc các ảnh cần trộn

- Import thêm numpy

```
import numpy as np
```

- Đọc các ảnh cần trộn, resize ảnh **img2** (ảnh muốn dán vào ảnh lớn hơn) và thực hiện chuyển đổi các ảnh sang không gian màu RGB:

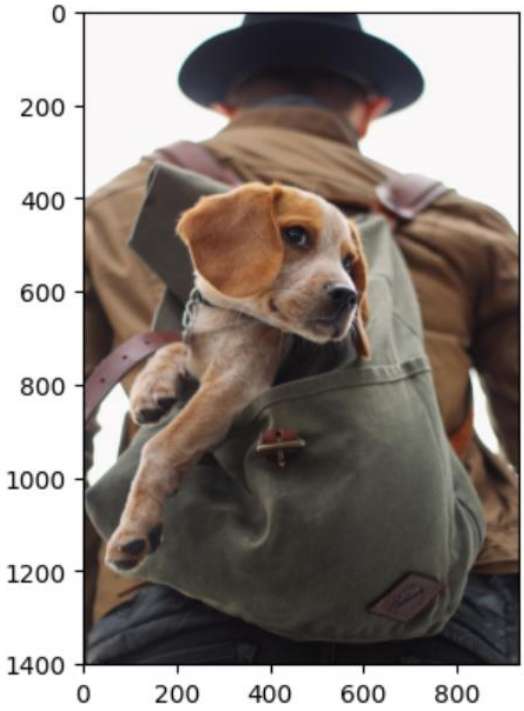
```
img1 = cv2.imread('images/dog_backpack.png')
img2 = cv2.imread('images/watermark_no_copy.png')
img2 =cv2.resize(img2,(600,600))

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2RGB)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)
```

[3] ✓ 0.0s

- Kiểm tra kết quả:

```
plt.imshow(img1)
```



```
plt.imshow(img2)
```

3.3.2. Tạo vùng quan tâm (Region of Interest - ROI)

- Kiểm tra kích thước ảnh img1 (ảnh lớn):

```
img1.shape
```
[7] ✓ 0.0s

```
(1401, 934, 3)
```

- Thiết lập vị trí góc trên bên trái của ROI:

```
x_offset=934-600
y_offset=1401-600
```
[8] ✓ 0.0s

- Tạo vùng quan tâm có kích thước bằng kích thước của ảnh foreground (img2, ảnh có kích thước nhỏ hơn sẽ nằm ở phía trên ảnh lớn hơn)

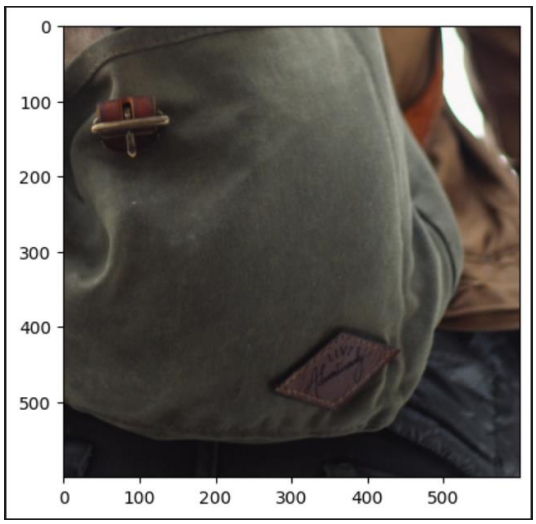
```
rows,cols,channels = img2.shape
# Tạo vùng quan tâm ở góc dưới bên phải ảnh lớn
roi = img1[y_offset:1401,x_offset:943]
```

- Kết quả thu được:

```
roi.shape
```
[12] ✓ 0.0s

```
(600, 600, 3)
```

```
plt.imshow(roi)
```
[11] ✓ 0.1s



3.3.3. Tạo mặt nạ (Mask)

Tạo một mặt nạ của logo và tạo mặt nạ bù của nó

- Tạo măt nạ là ảnh đơn sắc từ ảnh img2

```
img2gray = cv2.cvtColor(img2,cv2.COLOR_BGR2GRAY)
```

- Kết quả thu được

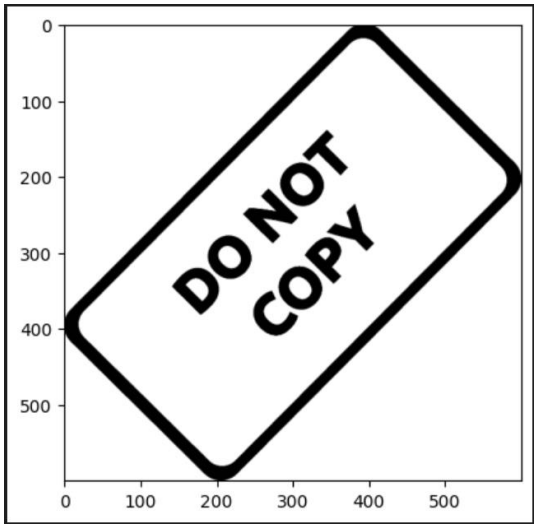
```
img2gray.shape
```

```
[14] ✓ 0.0s
```

```
(600, 600)
```

```
plt.imshow(img2gray,cmap='gray')
```

```
[15] ✓ 0.1s
```




```
mask_inv = cv2.bitwise_not(img2gray)
```

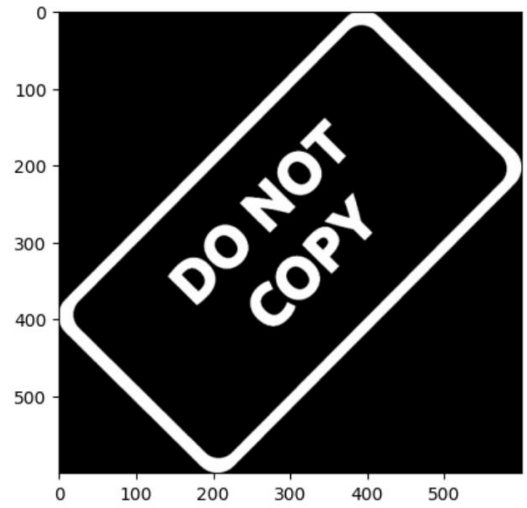
- Kết quả thu được:

```
mask_inv.shape
```

```
[17] ✓ 0.0s
```

```
(600, 600)
```

```
plt.imshow(mask_inv,cmap='gray')
```
[18] ✓ 0.1s



3.3.4. Chuyển mặt nạ về ảnh có ba kênh

- Tạo nền màu trắng:

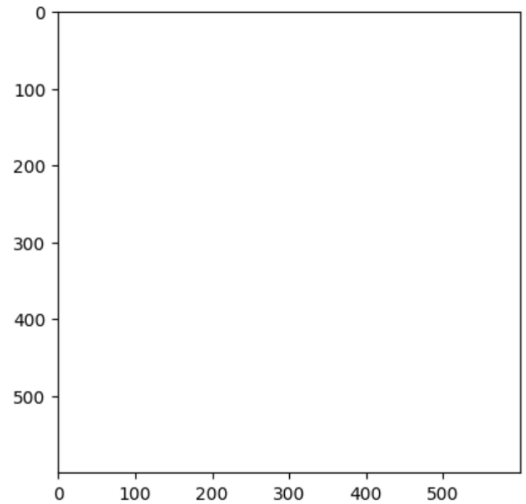
```
white_background = np.full(img2.shape, 255, dtype=np.uint8)
```
[19] ✓ 0.0s

- Kiểm tra kích thước white_background, hiện giờ chúng ta đã có một ảnh nền màu trắng có 3 kênh

```
white_background.shape
```
[38] ✓ 0.0s

```
(600, 600, 3)
```

```
plt.imshow(white_background)
```
[39] ✓ 0.1s



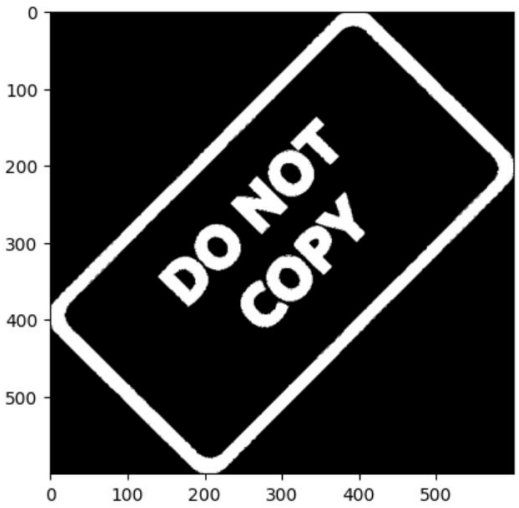
- Công việc tiếp theo là sử dụng hàm **`cv2.bitwise_or()`** với mặt nạ sử dụng là mask_inv để giữ lại những thành phần của ảnh mặt nạ mà chúng ta mong muốn (các đường nét của chữ và đường viền) trên nền trắng đã tạo.

```
[37]  bk = cv2.bitwise_or(white_background, white_background, mask=mask_inv)
```

- Kiểm tra kết quả:

```
[22]  bk.shape
      ✓ 0.0s
...   (600, 600, 3)
```

```
      plt.imshow(bk)
```



3.3.5. Đặt ảnh foreground gốc lên trên mặt nạ

- Hiển thị lại ảnh bù của mặt nạ:

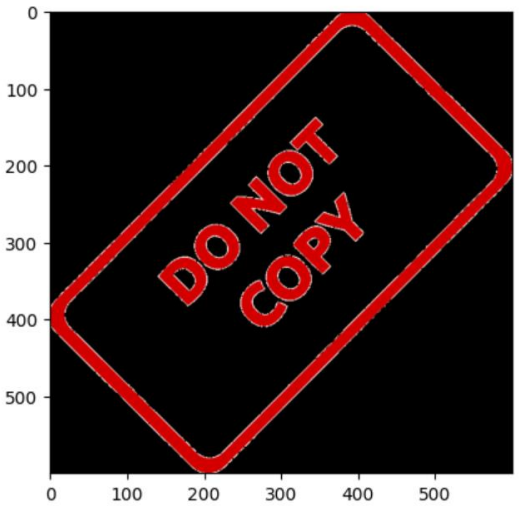
```
      plt.imshow(mask_inv,cmap='gray')
```

- Phân tách thành phần của ảnh img2 bằng mặt nạ mask_inv

```
      fg = cv2.bitwise_or(img2, img2, mask=mask_inv)
```

- Kết quả thu được:

```
      plt.imshow(fg)
```



```
fg.shape
```
[28] ✓ 0.0s

```
(600, 600, 3)
```

3.3.6. Lấy ROI và trộn mặt nạ với ROI

- Phân tách bit vùng quan tâm bởi ảnh foreground đã xử lý

```
final_roi = cv2.bitwise_or(roi,fg)
```
[31] ✓ 0.0s

- Kết quả thu được:

```
plt.imshow(final_roi)
```
[30] ✓ 0.1s

3.3.7. Thêm ROI vào lại phần còn lại của ảnh ban đầu

Lúc này:

- Ảnh lớn tương ứng với ảnh gốc ban đầu
- final_roi (vùng quan tâm đã được xử lý trộn với mặt nạ) đóng vai trò là ảnh nhỏ

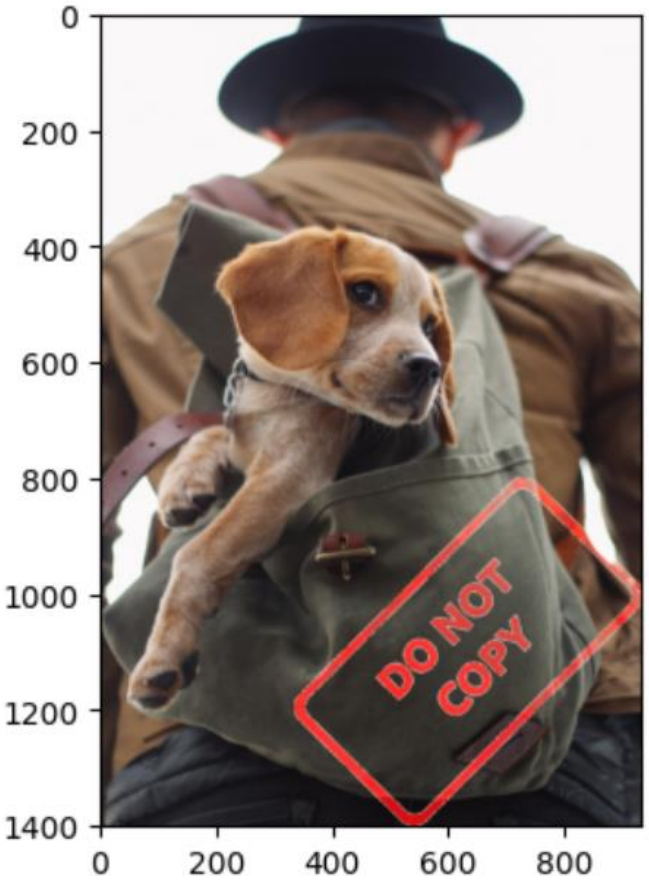
```
large_img = img1
small_img = final_roi
```
[33] ✓ 0.0s

- Chúng ta thực hiện xếp chồng ảnh nhỏ lên ảnh lớn:

```
large_img[y_offset:y_offset+small_img.shape[0], x_offset:x_offset+small_img.shape[1]] = small_img
```

- Kết quả thu được:

```
plt.imshow(large_img)
```
[36] ✓ 0.2s



Yêu cầu:

1. Điều chỉnh kích thước ảnh nhỏ về 300×400.

2. Dựa vào các bước thực hiện đã trình bày, thay đổi vị trí vùng quan tâm trên ảnh lớn (góc trên bên trái, góc dưới bên trái, góc trên bên phải, chính giữa ảnh), thực hiện lại quá trình trộn ảnh nhỏ (theo kích thước mới) với ảnh lớn tại các vị trí này? Ghi nhận quá trình thực hiện và kết quả thu được?