# Project 2: Image Processing

20127479 – Lê Nhất Duy

MTH051 - FIT-HCMUS

## 1. Giới thiệu và yêu cầu:

Trong đồ án 1, bạn đã được giới thiệu rằng ảnh được lưu trữ dưới dạng ma trận các điểm ảnh. Mỗi điểm ảnh có thể là một giá trị (ảnh xám) hoặc một vector (ảnh màu).

Trong đồ án này, bạn được yêu cầu thực hiện các chức năng xử lý ảnh cơ bản sau:

1. Thay đổi độ sáng cho ảnh
2. Thay đổi độ tương phản
3. Lật ảnh (ngang - dọc)
4. Chuyển đổi ảnh RGB thành ảnh xám
5. Chồng 2 ảnh cùng kích thước (chỉ làm trên ảnh xám)
6. Làm mờ ảnh

Các thư viện **được phép** sử dụng là: PIL, numpy, matplotlib.

## 2. Môi trường thực hiện:

Đồ án sử dụng Google Colab để chạy và test.

## 3. Liệt kê các chức năng đã hoàn thành:

| Chức năng | Độ hoàn thành |
|---|---|
| 1. Thay đổi độ sáng cho ảnh | 100% |
| 2. Thay đổi độ tương phản | 100% |
| 3. Lật ảnh (ngang - dọc) | 100% |
| 4. Chuyển đổi ảnh RGB thành ảnh xám | 100% |
| 5. Chồng 2 ảnh cùng kích thước (chỉ làm trên ảnh xám) | 100% |
| 6. Làm mờ ảnh | 100% |

## 4. Ý tưởng thực hiện và mô tả các hàm chức năng:

Trước khi đi vào chi tiết các hàm chức năng ta có một hàm bên ngoài để đảm nhiệm vai trò mở ảnh, chuyển ảnh thành mảng và trả ra giá trị để các hàm khác có thể thực hiện.

```
def convert_img(path):
  return np.array(Image.open(path))
```

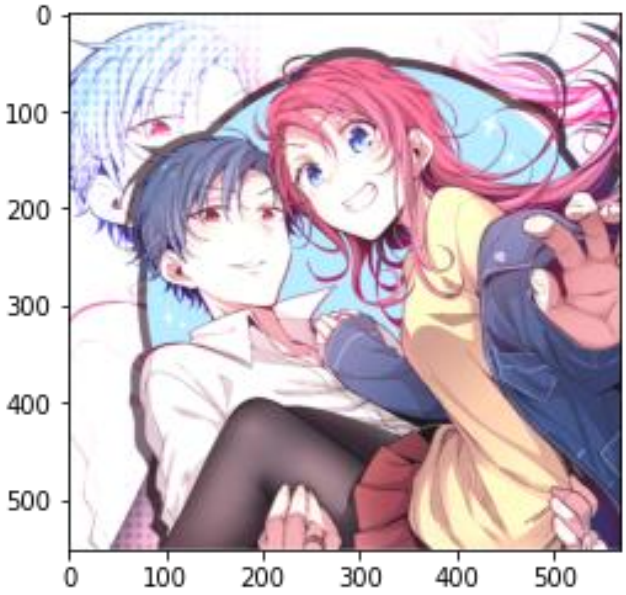
Ảnh gốc tôi dùng để test trong project: có kích thước 569 x 552

### 4.1 Thay đổi độ sáng cho ảnh:

- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: để thay đổi độ sáng cho ảnh ta chỉ cần cộng thêm một giá trị alpha nào đó để tăng hay giảm giá trị các điểm ảnh, giá trị càng gần 255 thì ảnh càng sáng, giá trị càng gần 0 thì ảnh càng tối. Để tránh trường hợp tràn số ngoài khoảng (0, 255) thì ta dùng thêm hàm clip của thư viện numpy để biến các giá trị vượt qua 255 thành 255 và nhỏ hơn 0 thành 0. Tuy nhiên trước khi cộng alpha vào thì ta phải chuyển alpha thành kiểu dữ liệu có dấu chấm động để hàm thực hiện chính xác. Chú ý này được tham khảo trong đường link ở dưới [1]. Vì để cho tránh phức tạp, tôi sẽ gán luôn giá trị alpha để người dùng không cần nhập

```
def change_brightness(path):
  img_result = Image.fromarray((np.clip((convert_img(path) + floa
t(25)), 0, 255)).astype(np.uint8))
  name = path[:(path.find("."))] + "_brightness" + path[(path.fin
d("."))]
  img_result.save(name)
  return img_result
```

- Hình ảnh kết quả:



### 4.2 Thay đổi độ tương phản:

- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: để thay đổi độ tương phản cho ảnh ta chỉ cần nhân thêm một giá trị alpha nào đó để tăng hay giảm khoảng chênh lệch giữa các điểm ảnh gần nhau.
  + Ta sẽ thay đổi giá trị các điểm ảnh theo công thức:

    new_pixel = old_pixel * f + 128 ( 1 – f)

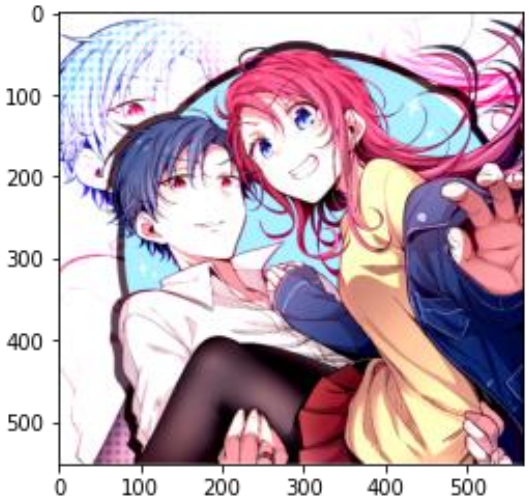
  + Giá trị của f được tính theo công thức:

    f = (float)((259 * (c + 255)) / (255 * (259 - c))) (với c có sẵn)

  + Để tránh trường hợp tràng số ngoài khoảng (0, 255) thì ta dùng thêm hàm clip của thư viện numpy để biến các giá trị vượt qua 255 thành 255 và nhỏ hơn 0 thành 0. Tuy nhiên trước khi nhân f vào thì ta phải chuyển f thành kiểu dữ liệu có dấu chấm động để hàm thực hiện chính xác. Chú ý này được tham khảo trong đường link ở dưới [2]. Vì để cho tránh phức tạp, tôi sẽ gán luôn giá trị c ban đầu để người dùng không cần nhập.
  + 0 < c < 1 → lower contrast [3]
  + c > 1 → higher contrast

```
def change_contrast(path):
  c = 40
  f = (float)((259 * (c + 255)) / (255 * (259 - c)))
  img = convert_img(path)
  img = f * img + 128 * (1 - f)
  img = np.clip(img, 0, 255)
  img_result = Image.fromarray(img.astype(np.uint8))
  name = path[:(path.find("."))] + "_contrast" + path[(path.find(
".")):]
  img_result.save(name)
  return img_result
```

- Hình ảnh kết quả:



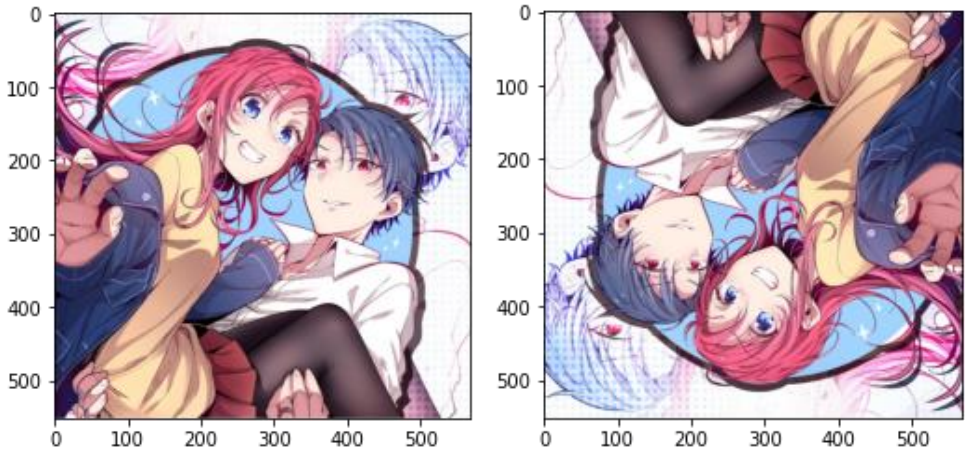
### 4.3 Lật ảnh (ngang - dọc):

- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: để lật ảnh theo chiều ngang hay dọc, ta chỉ cần hoán đổi các cột của ma trận các điểm ảnh (đối với lật ảnh ngang), và hoán đổi các dòng của ma trận các điểm ảnh (đối với lật ảnh dọc). Ý tưởng là sử dụng slicing và từ bài giảng lab03 của cô *Phan Thị Phương Uyên*.

```
def flip_vertically(path):
  img_result = convert_img(path)[::-1]
  name = path[:(path.find("."))] + "_flip_vertically" + path[(pat
h.find(".")):]
  plt.imsave(name, img_result)
  return img_result

def flip_horizontally(path):
  img_result = convert_img(path)[:, ::-1]
  name = path[:(path.find("."))] + "_flip_horizontally" + path[(p
ath.find(".")):]
  plt.imsave(name, img_result)
  return img_result
```

- Hình ảnh kết quả:



### 4.4 Chuyển đổi ảnh RGB thành ảnh xám:

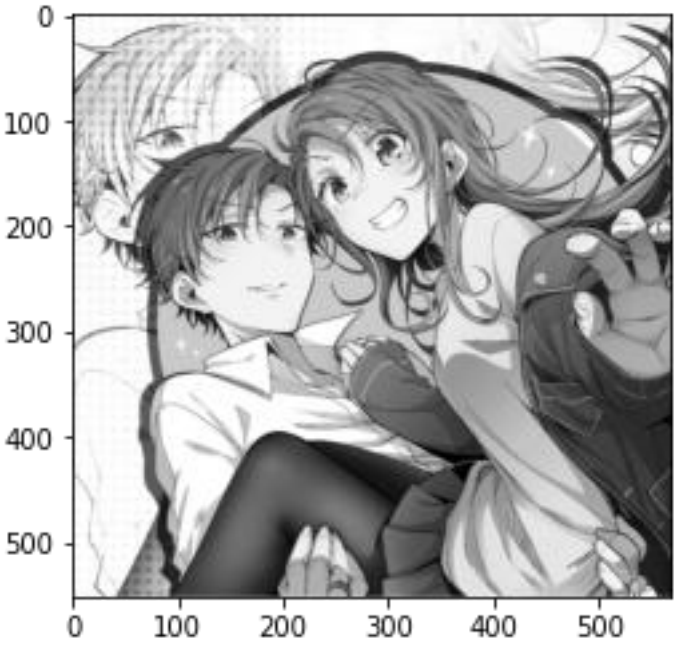
- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: đối với từng điểm ảnh ta gộp 3 giá trị thành một giá trị duy nhất trong khoảng (0, 255) theo một công thức tỷ lệ nào đó. Trong đồ án này ta sử dụng công thức sau:

  new_pixel = 0.2989 Red + 0.5870 Green + 0.1140 Blue [4]

  + Để tránh hàm chạy thời gian dài, cũng như tính toán nhiều ta sẽ dùng kiến thức toán khi nhân 2 ma trận với nhay, ta sẽ nhân ma trận các điểm ảnh với vector [0.2989, 0.5870, 0.1140] thay vì dùng vòng for để duyệt từng pixel của ma trận các điểm ảnh và tính theo công thức.

```
def grayscale(path, isSave = 0):
  img_result = np.dot(convert_img(path)[...,:3], [0.2989, 0.5870,
 0.1140])
  if isSave == 0:
    name = path[:(path.find("."))] + "_grayscale" + path[(path.fi
nd("."))):]
    plt.imsave(name, img_result, cmap = 'gray')
  return img_result
```

- Hình ảnh kết quả:



### 4.5 Chồng 2 ảnh cùng kích thước (chỉ làm trên ảnh xám):

- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: để chồng 2 ảnh cùng kích thước, ta chỉ cần cộng 2 ma trận các điểm ảnh của 2 ảnh lại với nhau theo một tỷ lệ cụ thể nào đó mà chúng ta muốn. Ảnh nào có tỷ lệ lớn hơn thì sẽ rõ hơn. Ở đây để cân bằng giữa 2 ảnh, cũng như không làm phức tạp khi yêu cầu người dùng phải nhập vào tỷ lệ cho các ảnh, ta sẽ chọn luôn tỷ lệ cho 2 ảnh là 0.5 để đều nhau.
  + Vì chọn tỷ lệ đều nhau là 0.5 nên code bên dưới có thể viết gọn thành:

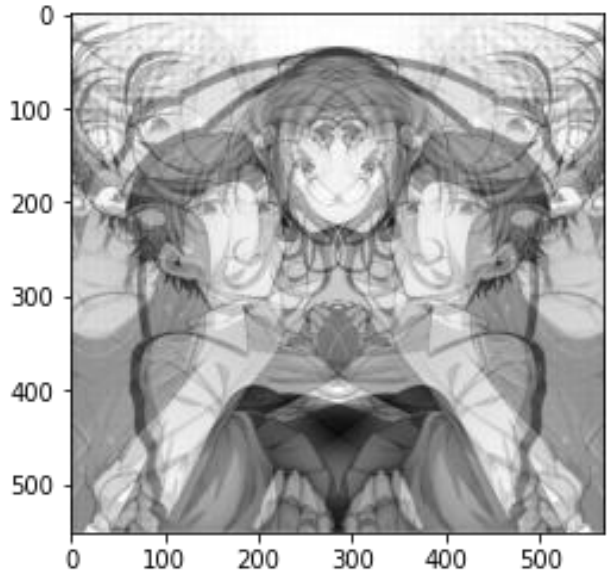
  new_img = (img1 + img2) / 2 [5]

  nếu ta chọn tỷ lệ khác cho các bức ảnh thì ta có thể theo công thức tổng quát sau:

  new_img = img1 * f1 + img2 * f2

```
def add_img(path1, path2):
  img1 = grayscale(path1, 1)
  img2 = grayscale(path2, 1)
  img_result = Image.fromarray(np.clip(((img1 + img2) / 2), 0, 25
5).astype(np.uint8))
  name = path1[:(path1.find("."))] + "_" + path2[:(path2.find("."
))]+ "_blend" + path1[(path1.find(".")):]
  img_result.save(name, cmap = 'gray')
  return img_result
```

- Hình ảnh kết quả: Vì không tìm được ảnh nào có cùng kích thước với ảnh cũ nên tôi xin dùng ảnh đã lật ngang của nó để làm ảnh thứ 2 để chồng lên ảnh gốc. Khi đó 2 ảnh sẽ cùng kích thước và kết quả sẽ được một tấm ảnh đối xứng 2 bên.



### 4.6 Làm mờ ảnh:

- Đầu vào: đường dẫn của ảnh cần thực hiện (path)
- Đầu ra: mảng ảnh kiểu dữ liệu image đã được xử lý xong
- Mô tả ý tưởng và hàm: ta sử dụng ma trận kernel theo Gaussian blur 3 x 3. Đầu tiên ta duyệt hết từng pixel của ma trận các điểm ảnh. Tiếp theo ta làm mờ các điểm ảnh bằng cách gán lại giá trị mới cho điểm ảnh đó, giá trị mới đó là trung bình cộng các điểm ảnh xung quanh điểm ảnh đang xét cùng với chính nó theo một tỷ lệ hợp lý nào đó mà chúng ta muốn.
  + Vì ta dùng kernel theo Gaussian blur 3 x 3 nên ta sẽ có 8 điểm ảnh xung quanh điểm ảnh đang xét, đồng thời, ta sẽ có tỷ lệ điểm ảnh đang xét là 4, các điểm ảnh ngay sát với điểm ảnh đang xét là 2 và các điểm ảnh ở góc của ma trận kernel sẽ có tỷ lệ là 1.
  + Cũng giống như hàm grayscale, ta tận dụng phép nhân ma trận để thực hiện thay vì dùng vòng lặp for. Tuy nhiên ta sẽ gặp một chút vấn đề là ma trận kernel là 3 * 3 khi ta duyệt ma trận ảnh của ta theo ma trận kernel thì các điểm ảnh nằm bên ngoài rìa của ảnh sẽ không được xét, chính vì vậy ta cần chỉnh sửa lại ma trận điểm ảnh để phù hợp với thuật toán, sau khi tính toán xong ta sẽ trả lại ma trận ảnh ban đầu. Giải pháp là ta sẽ thêm các điểm ảnh có toàn giá trị 0 vào bên rìa của ma trận ban đầu (tức là bọc ma trận các điểm ảnh trong một khung hình chữ nhật các điểm ảnh có giá trị toàn 0). Sau khi tham khảo trên một số nguồn trên internet thì ta đã có cách để tạo ra ma trận này bằng hàm pad của thư viện numpy. [6]
  + Dựa theo công thực thông thường thì kernel của gaussian blur sẽ là một ma trận nhân với 1/16. Trong trường hợp này ta lấy kernel theo wiki [7] nên kernel sẽ không bị lẻ. Tuy nhiên một số trường hợp khác người dùng muốn thay đổi tỷ lệ trong kernel. Thì việc chia cho tổng sẽ làm ma kernel "có khả năng" bị lẻ số và bị làm tròn. Tiếp tục vòng lặp làm mờ bằng nhân ma trận thì lại tiếp tục có khả năng làm kết quả pixel bị tiếp tục làm tròn. Và kết quả bị làm tròn 2 lần. Để tránh trường hợp hi hữu này, cũng như tổng quát cho một số trường hợp kernel bị lẻ số. Thì ta sẽ tính kết quả pixel trước rồi mới chia cho tổng để tính trung bình sau.
  Việc này không hề làm thay đổi kết quả ta cần tính.
  + Với mỗi pixel dòng i cột j ta sẽ có ma trận img_g[i:i+3, j:j+3] tương ứng (ma trận 3 * 3 lấy điểm ảnh này làm trung tâm).
  + Sau khi nhân 2 ma trận xong ta sẽ được ma trận 3 * 3, để tính trung bình thì ta chỉ cần lấy tổng các điểm ảnh trong, để tính ta chỉ cần dùng hàm sum để tính tổng các dòng và các cột của ma trận kết quả [8]. Vector kết quả sẽ là điểm ảnh mới.

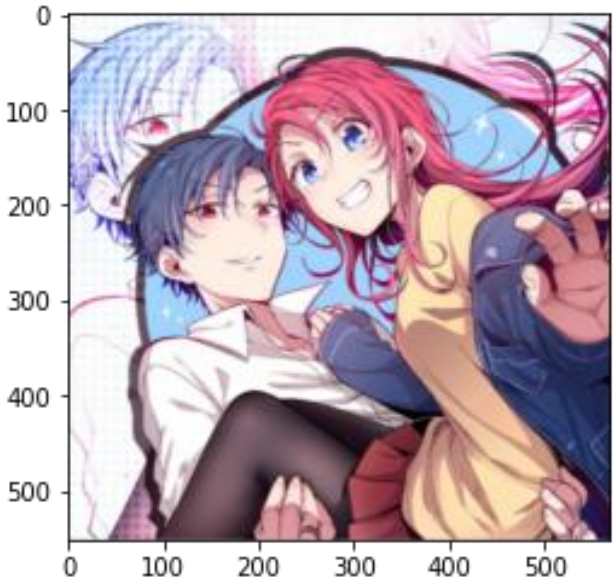
```
def blur(path):
    kernel_gs = np.array([[[1], [2], [1]],
                          [[2], [4], [2]],
                          [[1], [2], [1]]])

    img = convert_img(path)
    img_g = np.pad(img, ((1, 1), (1, 1), (0, 0)))

    for i in range(len(img)):
        for j in range(len(img[0])):
            img[i][j] = (img_g[i:i+3, j:j+3] * kernel_gs).sum(axis = 1).sum(axis = 0) / 16

    img_result = Image.fromarray(img.astype(np.uint8))
    name = path[:(path.find("."))] + "_blur" + path[(path.find("."")):]
    img_result.save(name)
    return img_result
```

- Hình ảnh kết quả:



**4. Tài liệu tham khảo:**


1. https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-4-brightness-adjustment/

2. https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

3. https://stackoverflow.com/questions/39308030/how-do-i-increase-the-contrast-of-an-image-in-python-opencv

4. https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-3-greyscale-conversion/

5. https://stackoverflow.com/questions/32272693/using-python-and-numpy-to-blend-2-images-into-1

6. https://numpy.org/doc/stable/reference/generated/numpy.pad.html

7. https://en.wikipedia.org/wiki/Kernel_(image_processing)

8. https://stackoverflow.com/questions/37142135/sum-numpy-ndarray-with-3d-array-along-a-given-axis-1