NỘI DUNG BÁO CÁO

I. THÔNG TIN SINH VIÊN

Đại diện nhóm: Trịnh Nguyễn Hoàng Vũ
Giáo viên hướng dẫn: Võ Quang Hoàng Khang
Nhóm báo cáo: 4
Mã lớp học phần: DHKHDL18A - 420301412201

MSSV	Họ	Tên
22642231	Trịnh Nguyễn Hoàng	Vũ
22634681	Nguyễn Hoàng Ái	Linh
22684251	Trịnh Dương	Hoan

II. THỐNG KÊ MỨC ĐỘ HOÀNH THÀNH

STT	CÁC CHỨC NĂNG	MỨC ĐỘ HOÀN THÀNH	SINH VIÊN THỰC HIỆN
1	Checkered Board	100%	Cả nhóm
2	Color Correction	100%	Cả nhóm
3	Rotate Image	100%	Cả nhóm
4	Color Separation	100%	Cả nhóm
5	Corner Line	100%	Cả nhóm
6	Gradient	100%	Cả nhóm
7	Letter B	100%	Cả nhóm
8	Find secret by subtract	100%	Cả nhóm

III. PHÂN TÍCH VÀ MÔ TẢ THUẬT TOÁN

1. Checkered Board

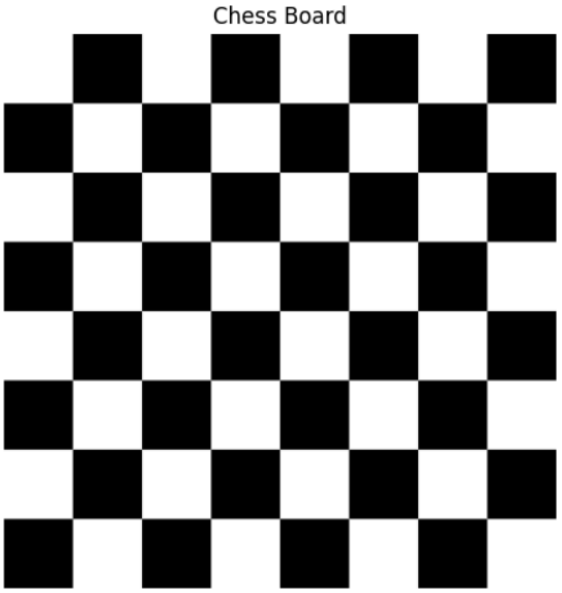
- Ý tưởng: Tạo ma trận đại diện cho bàn cờ.
- Mô tả thuật toán: Tạo ma trận với tất cả các giá trị ban đầu là 0 (màu đen). Những ô có tổng tọa độ chẵn sẽ có giá trị 255 (màu trắng).

+ Hàm $create\_chessboard$: Hàm này với đầu vào là kích thước bàn cờ $size$ và kích thước ô cờ $block\_size$ sẽ tạo ma trận $chess\_board$ có kích thước $size \cdot block\_size, size \cdot block\_size$. Duyệt qua các giá trị của kích thước nếu $(r + c) \% 2 = 0$ thì cập nhật $chess\_board$ bằng slicing gán các phần tử từ $r \cdot block\_size$ đến $(r + 1) \cdot block\_size$ của hàng và phần tử từ $c \cdot block\_size$ đến $(c + 1) \cdot block\_size$ của cột thành 255.

- Code:

```python
def create_chessboard(size, block_size=100):
    ''' Hàm tạo bàn cờ

    size: Kích thước bàn cờ
    block_size: Kích thuốc một ô cờ

    '''

    chess_board = np.zeros((size * block_size, size * block_size), dtype=np.uint8)   # Ma trận dại diện cho bàn cờ

    for r in range(size):
        for c in range(size):
            if (r + c) % 2 == 0:                                                     # Ô cần được tô trắng
                chess_board[r * block_size: (r + 1) * block_size,
                            c * block_size: (c + 1) * block_size] = 255               # Gán giá trị 255 (màu trắng)

    return chess_board                                                               # Hình ảnh bàn cờ
```

- Kết quả:



2. Color Correction

- Ý tưởng: Chúng ta sẽ chuyển đổi ảnh màu về ảnh xám và đảo trắng đen bằng cách lấy 255 (đại diện cho màu trắng) và trừ đi cho các điểm ảnh.

- Mô tả thuật toán:

+ Hàm *convert_RGB_to_Gray*: Hàm này sẽ duyệt qua từng điểm ảnh của ảnh màu (được biểu diễn dưới dạng một mảng 2D với mỗi phần tử là một tuple chứa giá trị RGB). Sau đó, nó tính toán giá trị độ xám của mỗi pixel bằng cách sử dụng công thức tổng trọng số các thành phần đỏ (R), xanh lá (G), và xanh dương (B). Công thức chuyển đổi:

```
gray = 0.2989 * r + 0.5870 * g + 0.1140 * b
```

+ Hàm *reversed_black_white_color* : Hàm này đầu vào sẽ là một ảnh xám, nó sẽ duyệt qua từng điểm ảnh và lấy giá trị 255 trừ cho giá trị tại điểm ảnh đó, điều này sẽ khiến các điểm ảnh gần đen sẽ thành trắng và ngược lại.

- Code:

```python
def convert_BGR_to_gray(image) :
    # Chuyển đổi ảnh từ không gian màu BGR sang RGB.
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

    # Khởi tạo danh sách rỗng để lưu ảnh xám.
    gray_image = []

    # Duyệt qua từng hàng pixel trong ảnh RGB.
    for row in image:
        # Khởi tạo danh sách rỗng để lưu các giá trị xám của hàng hiện tại.
        gray_row = []

        # Duyệt qua từng pixel trong hàng hiện tại.
        for pixel in row:
            # Tách các giá trị màu đỏ (r), xanh lá cây (g), và xanh dương (b) từ pixel.
            r, g, b = pixel

            # Tính giá trị xám sử dụng công thức chuyển đổi từ RGB sang grayscale.
            gray = int(0.2989 * r + 0.5870 * g + 0.1140 * b)

            # Thêm giá trị xám vào hàng xám hiện tại.
            gray_row.append(gray)

        # Thêm hàng xám hiện tại vào danh sách ảnh xám.
        gray_image.append(gray_row)

    # Trả về ảnh xám hoàn chỉnh.
    return gray_image
```

```python
def reversed_black_white_color(image) :
    # Tạo và trả về ảnh với màu đen và trắng đảo ngược.
    return [[255 - pixel for pixel in row] for row in image]
```

- Kết quả:

3. Rotate Image

- Ý tưởng: Áp dụng toán học để biến đổi tọa độ (hàng, cột) của ma trận thành tọa độ mới trên hình kết quả.

- Mô tả thuật toán:

+ Hàm $rotate$: Tạo ảnh mới dựa trên ảnh gốc có góc quay α. Ảnh đầu ra sẽ có kích thước lớn hơn để ảnh khi xoay không bị khuất, kích thước mới được tính bằng các điểm ở góc ảnh sau khi xoay vì các điểm này sẽ luôn nằm trên giới hạn của ảnh mới. Để thay đổi tọa độ x, y cần

xác định điểm trung tâm của ảnh cũ $cx\_old,\ cy\_old$ và ảnh mới $cx\_new,\ cy\_new$ để coi đó là trục quay.

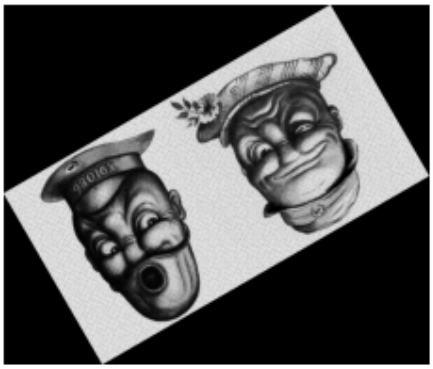
+ Công thức ma trận xoay: $R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$

+ Công thức cho tọa độ của tọa độ mới: $\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$

- Code:

```python
def rotate(img, alpha):
    ''' Hàm xoay ảnh đầu vào theo góc alpha

    - img: Ảnh đầu vào
    - alpha: Góc quay của ảnh đầu ra

    '''
    theta = alpha * pi / 180                                        # Chuyển đơn vị độ sang radian
    R = np.array([                                                  # Tạo ma trận xoay
        [cos(theta), -sin(theta)],
        [sin(theta), cos(theta)]
    ])

    rows, cols = img.shape[:2]                                      # Lấy kích thước gốc của ảnh
    new_rows = int(abs(rows * cos(theta)) + abs(cols * sin(theta)))  # Tính toán kích thước mới của ảnh sau khi xoay
    new_cols = int(abs(rows * sin(theta)) + abs(cols * cos(theta)))

    rotated_img = np.zeros(shape=(new_rows, new_cols, 3), dtype=img.dtype)  # Tạo ma trận mới đại diện cho đầu ra
    cx_old, cy_old = rows / 2, cols / 2                             # Tính toán tọa độ tâm của ảnh gốc
    cx_new, cy_new = new_rows / 2, new_cols / 2                     # Tính toán tọa độ tâm của ảnh mới

    for r in range(rows):
        for c in range(cols):
            new_r, new_c = np.dot(R, [r - cx_old, c - cy_old])      # Tính toán tọa độ mới bằng ma trận xoay
            new_r, new_c = int(new_r + cx_new), int(new_c + cy_new) # Điều chỉnh tọa độ mới về không gian tọa độ của hình ảnh mới

            if 0 <= new_r < new_rows and 0 <= new_c < new_cols:     # Kiểm tra tọa độ nằm trong giới hạn của hình ảnh mới
                rotated_img[new_r, new_c] = img[r, c]               # Thay đổi giá trị của ảnh mới theo ảnh gốc

    return rotated_img                                              # Ảnh đã được xoay
```

- Kết quả:



4. Color Separation

- Ý tưởng: Chúng ta sẽ thử nghiệm từng ngưỡng cho ảnh đầu vào và đối với mỗi ngưỡng thì những điểm ảnh mà có giá trị thấp hơn ngưỡng thì sẽ chuyển về giá trị 0 (màu đen) còn những giá trị còn lại giữ nguyên để làm nổi bật vật thể.

- Mô tả thuật toán: Đầu vào sẽ là một ảnh xám, chúng ta sẽ áp dụng từng ngưỡng lên bức để thử nghiệm xem là ngưỡng nào hợp lý nhất. Trong bài này nhóm chúng em sử dụng các ngưỡng là 32, 64, 127.

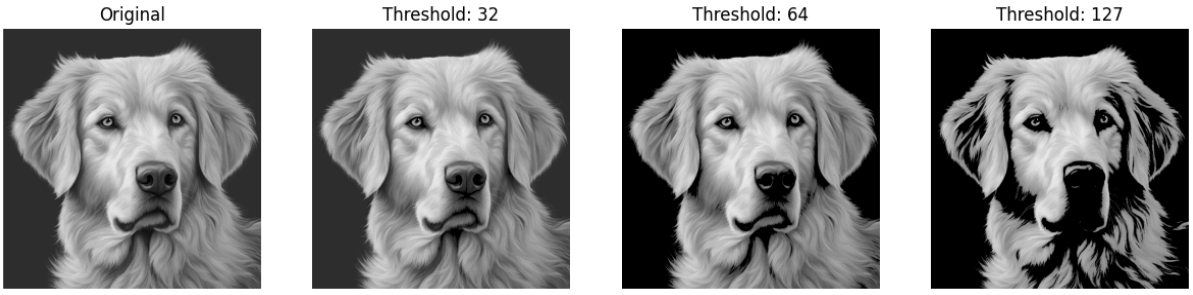
  + Hàm *threshold_image (img_gray, threshold)* : Hàm nhận vào một ảnh xám (img_gray) và một giá trị ngưỡng (thresshold) . Duyệt qua các điểm ảnh của ảnh được truyền vào, nếu điểm ảnh nào mà có giá trị thấp hơn ngưỡng thì sẽ được chuyển thành 0 (màu đen) . Sau khi duyệt xong hết tất cả các điểm ảnh thì trả về ảnh đã được xử lí.
  + Hàm *apply_thresholds(img_gray, thresholds)* : Hàm này nhận vào một ảnh xám và một danh sách các giá trị ngưỡng cần thử nghiệm. Duyệt qua từng giá trị ngưỡng và gọi hàm *threshold_image* để áp dụng giá trị ngưỡng lên ảnh, sau đó trực quan hóa các ảnh thu được bằng thư viện *matplotlib* để dễ dàng trong việc quan sát.

- Code:

```python
def threshold_image(img_gray, threshold):
    ''' Hàm lấy biến đổi ảnh xám dựa trên ngưỡng

    img_gray: Ảnh gốc có kích thước 2D
    threshold: Ngưỡng yêu cầu tùy ý

    '''

    img_thresholded = img_gray.copy()                  # Ảnh Tạo ảnh mới là kết quả đầu ra
    img_thresholded[img_thresholded < threshold] = 0    # Với tất cả các pixel có giá trị nhỏ hơn ngưỡng (threshold), đặt giá trị của chúng bằng 0

    return img_thresholded                             # Trả về ảnh đã được ngưỡng hóa
```

```python
def apply_thresholds(img_gray, thresholds):
    plt.figure(figsize=(15, 5))

    # Hiển thị ảnh gốc ở vị trí đầu tiên (subplot 1).
    plt.subplot(1, 4, 1)
    plt.title("Original")
    plt.axis('off')
    plt.imshow(img_gray, cmap='gray')

    # Lặp qua từng giá trị ngưỡng trong danh sách thresholds.
    for i, threshold in enumerate(thresholds):
        # Áp dụng ngưỡng hóa lên ảnh gốc với giá trị ngưỡng hiện tại.
        img_thresholded = threshold_image(img_gray, threshold)

        # Hiển thị ảnh sau khi ngưỡng hóa ở các vị trí tiếp theo.
        plt.subplot(1, 4, i+2)
        plt.imshow(img_thresholded, cmap='gray')
        plt.title(f"Threshold: {threshold}")
        plt.axis('off')
    plt.show()
```

- Kết quả:



5. Corner Line

- Ý tưởng: Coi hình ảnh như không gian 2D, từ đó có thể viết phương trình đường thẳng và áp dụng toán học cơ bản để biết được những nơi cần thay đổi.

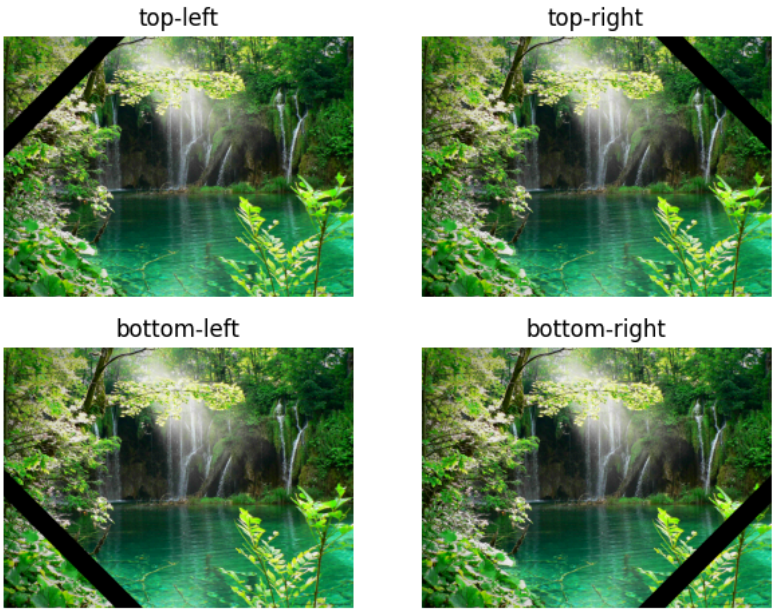
- Mô tả thuật toán: Tạo 2 đường thẳng từ cách trên, những tọa độ x, y sẽ được chỉnh về màu đen nếu chúng nằm giữa 2 đường thẳng này.

  + Phương trình đường thẳng theo góc α bất kì $y = x \cdot \tan(\alpha) + b$ (Coi row, col của ảnh tương tự như x, y).
  + Duyệt qua các row tạo 2 biến đại diện $line1, line2$ với ý nghĩa là cận dưới và cận trên của giá trị col, nếu col nằm trong khoảng giá trị này thì đó là điểm ảnh cần được tô đen.

- Code:

```
def corner_line(img, alpha, x, thickness=200):
    ''' Hàm tạo đường kẻ đen có góc quay alpha tại vị trí x và đường kẻ này có độ dày thickness (mặc định 200)

    - img: ảnh đầu vào
    - alpha: góc quay mong muốn của đường kẻ
    - x: vị trí xuất phát của đường kẻ
    - thickness: độ dày của đường kẻ

    '''
    theta = alpha * pi / 180                    # Chuyển đơn vị độ về radian
    rows, cols = img.shape[:2]                  # Lấy kích thước của ảnh
    result = img.copy()                         # Khởi tạo ảnh đầu ra

    for r in range(rows):                       # Duyệt qua các dòng của ảnh
        line1 = r * tan(theta) + x              # Giá trị đại diện cho line1 tại x = r
        line2 = line1 + thickness               # Giá trị đại diện cho line2 tại x = r

        for c in range(cols):                   # Duyệt qua các cột của ảnh
            if line1 <= c <= line2:             # Những cột có giá trị nằm giữa line1 và line2
                result[r, c] = 0                # Tọa độ cần được chỉnh về màu đen

    return result                               # Hình ảnh đã được kẻ đường chéo
```

- Kết quả:



6. Gradient

- Ý tưởng: Chia các giá trị trong khoảng $[0, 255]$ thành k đoạn cách đều, sau đó tạo ma trận kích thước tùy ý dựa trên các giá trị đó.

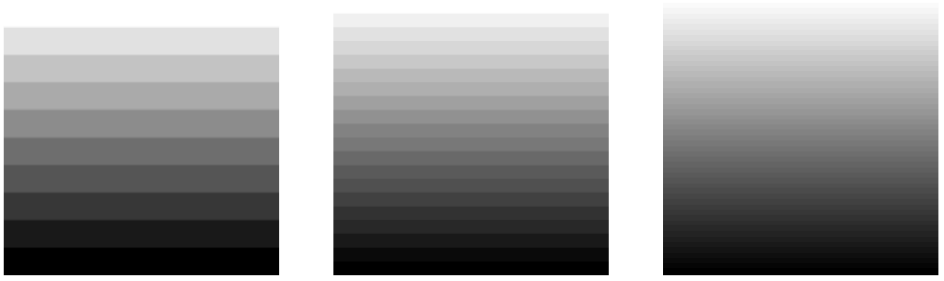
- Mô tả thuật toán:

  + Tạo một mảng $gradient$ chưa k giá trị tăng dần từ 0 đến 255 (đen - trắng).
  + Tạo một ma trận $gradient\_image$ đại diện cho ảnh đầu ra với kích thước (h, w).
  + Duyệt qua k giá trị trong $gradient$ và gán các ô từ hàng $i \cdot h // k$ đến $(i + 1) \cdot h // k$ thành màu hiện tại (Chia k khoảng giá trị cho h dòng).

- Code:

```python
def generate_gradient(w, h, k):
    ''' Hàm tạo ảnh gradient với kích thước (h, w) và có k màu

    w: weight của ảnh mong muốn
    h: height của ảnh mong muốn
    k: số lượng khoảng màu

    '''
    gradient = np.linspace(0, 255, k)                # Tạo mảng chứa k giá trị đại diện cho k màu
    gradient_image = np.zeros((h, w))                # Tạo ma trận là ảnh gradient mông muốn

    for i in range(k):                               # Duyệt qua k màu
        start, end = i * h // k, (i + 1) * h // k    # Hàng bắt đầu và kết quả của màu hiện tại
        gradient_image[start:end, :] = gradient[i]   # Gán màu hiện tại cho các ô xác định

    return gradient_image                            # Ảnh gradient có kích thước (h, w) và có k màu
```

- Kết quả:



7. Letter B

- Ý tưởng: Sử dụng OpenCV và Numpy để tạo một ảnh trắng và vẽ chữ lên đó

- Mô tả thuật toán:

  + Khởi tạo ảnh trắng : Dùng thư viện Numpy để tạo ra một ma trận 500*500 có các giá trị là 255 (đại diện cho màu trắng)
  + Định nghĩa font và chữ : Dùng cv2.FONT_HERSHEY_SIMPLEX để định dạng font, thiết lập các thông số cơ bản của chữ như *font_scale* (độ lớn của chữ) , *font_thickness* (độ dày của chữ)

+ Sử dụng *cv2.getTextSize* để tính kích thước của chữ "B" dựa trên font, kích thước, và độ dày đã chọn.
+ Sử dụng *cv2.putText* để vẽ chữ "B" màu đen (RGB: (0, 0, 0)) lên hình ảnh ở vị trí đã tính toán.

- Code:

```
def draw_letter(letter):
    ''' Hàm vẽ kí tự bất kì (A-Z)

    letter: Chữ cần vẽ

    '''

    width, height = 500, 500                                                    # Xác định chiều rộng và chiều cao của ảnh là 500x500
    image = np.ones((height, width, 3), dtype="uint8") * 255                    # Tạo một ảnh trắng (màu RGB là [255, 255, 255]) với kích thước 500x500
    font = cv2.FONT_HERSHEY_SIMPLEX                                             # Chọn phông chữ kiểu Hershey Simplex
    font_scale = 10                                                             # Đặt kích thước phông chữ là 10
    font_thickness = 30                                                         # Đặt độ dày của chữ là 30

    text_size = cv2.getTextSize(letter, font, font_scale, font_thickness)[0]    # Tính kích thước của chữ cái với phông chữ, kích thước và độ dày đã chọn

    text_x = (image.shape[1] - text_size[0]) // 2                               # Tính toán tọa độ x để đặt chữ cái sao cho nó nằm giữa theo chiều ngang
    text_y = (image.shape[0] + text_size[1]) // 2                               # Tính toán tọa độ y để đặt chữ cái sao cho nó nằm giữa theo chiều dọc

    # Vẽ chữ cái lên ảnh tại vị trí trung tâm đã tính toán với màu đen (RGB là [0, 0, 0]).
    cv2.putText(image, letter, (text_x, text_y), font, font_scale, (0, 0, 0), font_thickness)

    return image                                                                # Trả về ảnh chứa chữ cái đã được vẽ
```

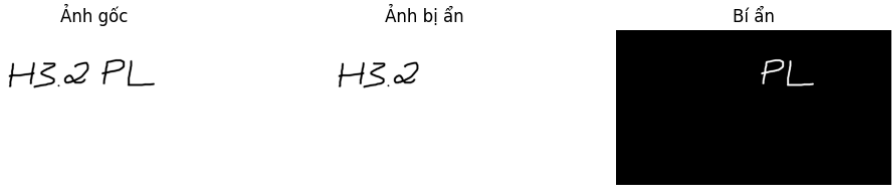
- Kết quả:

8. Find secret by subtract

- Ý tưởng: Trích xuất thông tin ẩn trong một hình ảnh bằng cách so sánh ảnh gốc với ảnh đã bị ẩn bằng cách trừ hai ảnh cho nhau, giúp làm lộ ra dữ liệu bị ẩn đi.
- Mô tả thuật toán: Tính toán sự khác biệt giữa ảnh gốc và ảnh bị ẩn bằng cách trừ hai hình ảnh cho nhau $result\_img = secret\_img - origin\_img$.
- Code:

```
def find_secret(origin_img, secret_img):
    ''' Hàm tìm những điểm ảnh bị ẩn từ ảnh gốc

    origin_img: Ảnh gốc
    secret_img: Ảnh bị khuyết điểm ảnh

    '''

    result_img = secret_img - origin_img     # Ảnh những điểm bị ẩn
    return result_img                        # Kết quả đầu ra mong muốn
```

- Kết quả:



IV. TÀI LIỆU THAM KHẢO

[1] Fundamentals of Image Processing and Computer Vision – Laboratory 1

[2] OpenCV_with_Python_By_Example_123