BÀI TẬP VỀ NHÀ

07, Lê Phạm Công - 20KTMT1

1. **Buổi 1 (05-01-2023):**

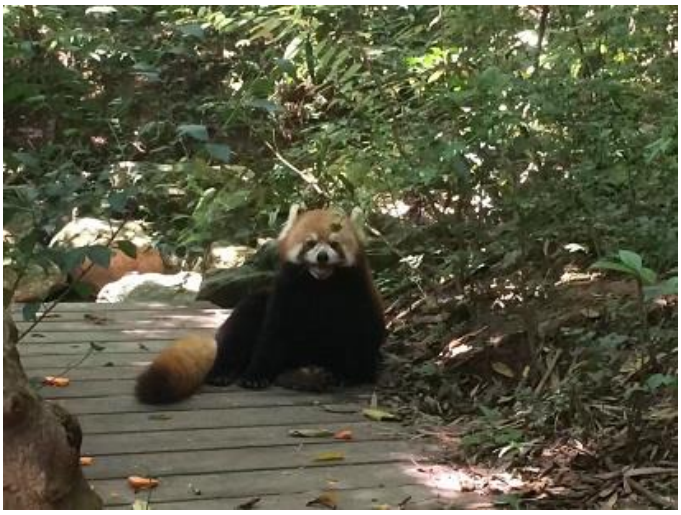
a. **BT1: viết chương trình xoay ảnh một góc Alpha**

SOURCE CODE:

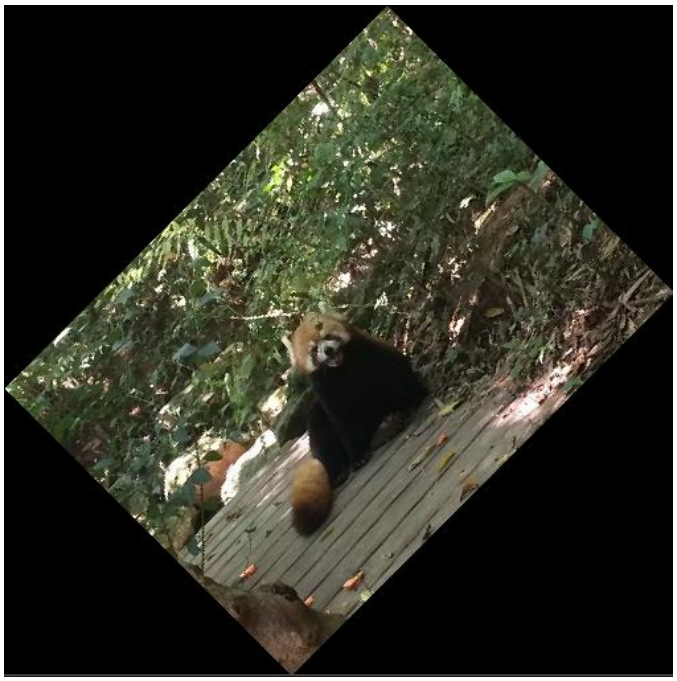
```
import cv2

import numpy as np

import math

img = cv2.imread("Image_processing/image/5.bmp")

alpha = 45

(h, w) = img.shape[:2]

new_w = int((abs(math.sin(math.radians(alpha))) * h) +
(abs(math.cos(math.radians(alpha))) * w))

new_h = int((abs(math.cos(math.radians(alpha))) * h) +
(abs(math.sin(math.radians(alpha))) * w))

rotated_img = np.zeros((new_h, new_w, 3), dtype=np.uint8)

center_x = w // 2

center_y = h // 2

rotated_center_x = new_w // 2

rotated_center_y = new_h // 2

for i in range(new_h):

    for j in range(new_w):

        translated_x = j - rotated_center_x

        translated_y = i - rotated_center_y

        rotated_x = (translated_x * math.cos(math.radians(alpha))) - (translated_y *
math.sin(math.radians(alpha)))

        rotated_y = (translated_x * math.sin(math.radians(alpha))) + (translated_y *
math.cos(math.radians(alpha)))

        rotated_x = rotated_x + center_x

        rotated_y = rotated_y + center_y
```

```
        if 0 <= rotated_x < w and 0 <= rotated_y < h:
            rotated_img[i, j] = img[int(round(rotated_y)), int(round(rotated_x))]
cv2.imshow('Original image', img)
cv2.imshow('Rotated image', rotated_img)
cv2.waitKey(0)
```

KẾT QUẢ



Hình . Ảnh gốc



Hình . Ảnh xoay 45^0

b. BT2: viết chương trình tạo ảnh gif

SOURCE CODE

```
from PIL import Image
# Tạo một list chứa tên các file ảnh
path="Image_processing/image/frames/"
imageFiles = [path +"0.png",path+"1.png",path+"2.png",path+"3.png"]
images = []
# Đọc các ảnh và lưu vào list
for filename in imageFiles:
    with Image.open(filename) as image:
        images.append(image.convert('P'))
# Tạo ảnh GIF đầu ra
output_gif = "Image_processing/image/frames/animation.gif"
images[0].save(output_gif, save_all=True,
               append_images=images[1:], duration=500, loop=0)
print('GIF đã được tạo thành công.')
```

KẾT QUẢ

Hình . Kết quả

2. Buổi 2 (12-01-2023):

a. BT: chia ảnh thành 16 phần rồi xáo trộn nó

SOURCE CODE:

```
import cv2
import numpy as np
import random
def shuffle_image(img):
    img_parts = []
    for i in range(4):
        for j in range(4):
            img_parts.append(img[i*int(img.shape[0]/4):(i+1)*int(img.shape[0]/4),
                        j*int(img.shape[1]/4):(j+1)*int(img.shape[1]/4)])
    random.shuffle(img_parts)
    new_img = np.zeros(img.shape, dtype=np.uint8)
    for i in range(4):
        for j in range(4):
            new_img[i*int(img.shape[0]/4):(i+1)*int(img.shape[0]/4), j *
                int(img.shape[1]/4):(j+1)*int(img.shape[1]/4)] = img_parts[i*4+j]
    return new_img
Img = cv2.imread("Image_processing/image/motor.jpg")
cv2.imshow("Original", Img)
new_img = shuffle_image(Img)
cv2.imshow("Shuffle", new_img)
cv2.waitKey(0)
```

KẾT QUẢ:

Hình . Ảnh gốc

Hình . Ảnh đã xáo trộn

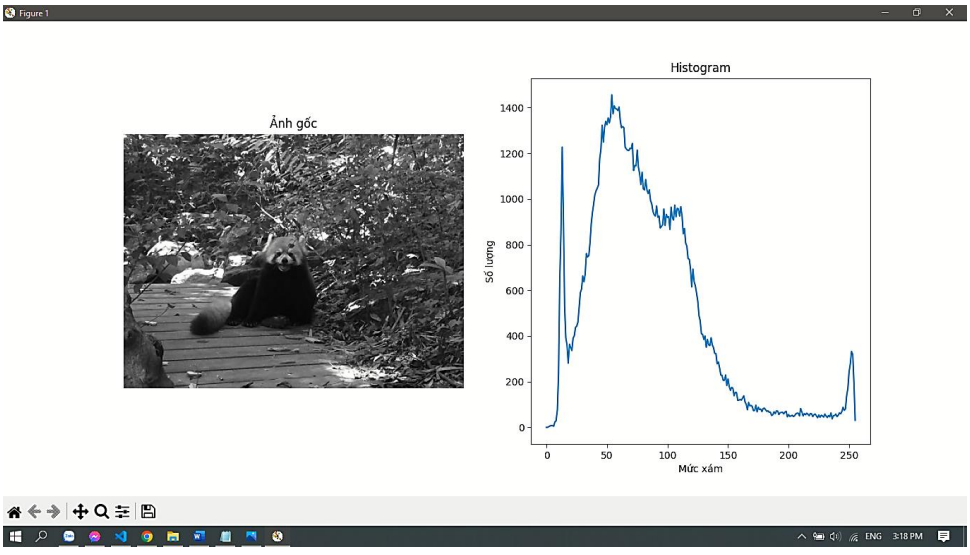
3. Buổi 3 (02-02-2023):

a. BT1: tìm và vẽ histogram của một ảnh xám

SOURCE CODE:

```
# Đề: tìm và vẽ histogram của ảnh bất kỳ (hình xám)
import cv2
import numpy as np
import matplotlib.pyplot as plt
def find_hist(img):
    hist = np.zeros(256)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            hist[img[i, j]] += 1
    return hist
image = cv2.imread("Image_processing/image/5.bmp", 0)
hist = find_hist(image)
plt.plot(hist)
plt.xlabel("Intensity")
plt.ylabel("Number of pixels")
plt.show()
```

KẾT QUẢ:



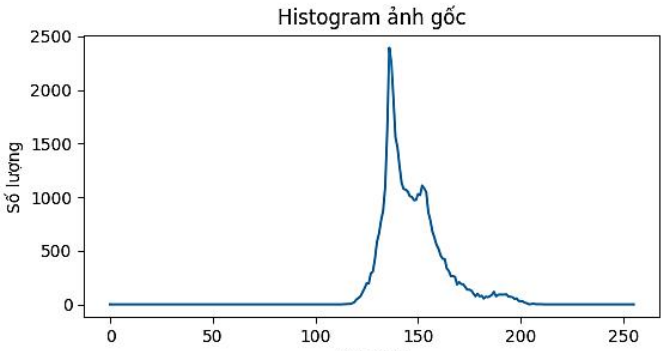
b. BT2: cân bằng histogram, so sánh với hàm có sẵn, nhận xét

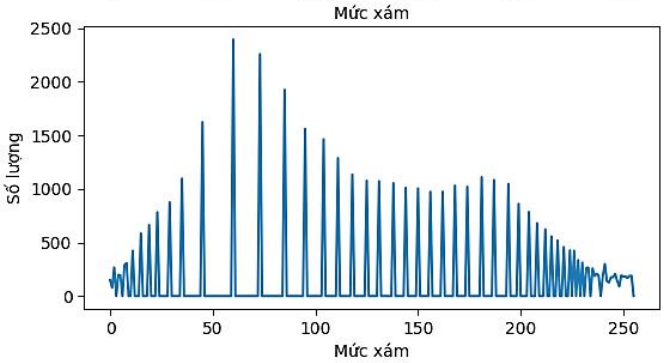
SOURCE CODE:

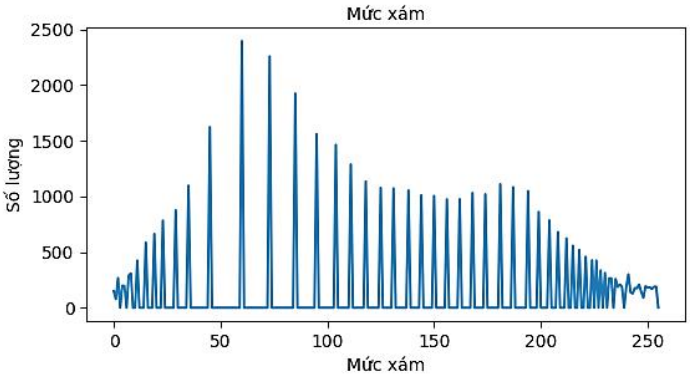
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def find_hist(img):
    hist = np.zeros(256)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            hist[img[i, j]] += 1
    return hist
def equalize_hist(img):
    # tính tổng pixels
    total_pixels = img.shape[0] * img.shape[1]
    # tính xác suất
    prob = find_hist(img) / total_pixels
    # tính phân bố đồ thị tuyệt đối
    cdf = np.zeros(256)
    for i in range(256):
        cdf[i] = np.sum(prob[:i + 1])
    # tính phân bố đồ thị tuyệt đối sau khi cân bằng
    cdf = (cdf * 255).astype(np.uint8)
    # thay đổi giá trị pixel
    new_img = np.zeros(img.shape, np.uint8)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            new_img[i, j] = cdf[img[i, j]]
    return new_img
```

```
image = cv2.imread("Image_processing/image/histeq_numpy1.jpg", 0)
hist_img = find_hist(image)
equalize_img = equalize_hist(image)
hist_equalize_img = find_hist(equalize_img)
equalize_img_opencv = cv2.equalizeHist(image)
hist_equalize_img_opencv = find_hist(equalize_img_opencv)
plt.subplot(3, 2, 1)
plt.imshow(image, cmap="gray"), plt.title(
    "Original image"), plt.xticks([]), plt.yticks([])
plt.subplot(3, 2, 2)
plt.plot(hist_img), plt.title(
    "Histogram of original image"), plt.xlabel("Mức xám"), plt.ylabel("Số lượng")
plt.subplot(3, 2, 3)
plt.imshow(equalize_img, cmap="gray"), plt.title(
    "Equalize image"), plt.xticks([]), plt.yticks([])
plt.subplot(3, 2, 4)
plt.plot(hist_equalize_img), plt.title(
    "Histogram of equalize image"), plt.xlabel("Mức xám"), plt.ylabel("Số lượng")
plt.subplot(3, 2, 5)
plt.imshow(equalize_img_opencv, cmap="gray"), plt.title(
    "Equalize image opencv"), plt.xticks([]), plt.yticks([])
plt.subplot(3, 2, 6)
plt.plot(hist_equalize_img_opencv), plt.title(
    "Histogram of equalize image opencv"), plt.xlabel("Mức xám"), plt.ylabel("Số lượng")
plt.show()
```

KẾT QUẢ:







NHẬN XÉT:

– Dựa trên hình ảnh và biểu đồ histogram của ảnh trước và sau khi cân bằng, ta có thể nhận xét như sau:

+ Ảnh trước khi cân bằng histogram có sự phân bố giá trị pixel chưa đều, dẫn đến một số vùng ảnh bị thiếu sáng hoặc bị chìm đen.

+ Sau khi cân bằng histogram, ảnh đã được điều chỉnh lại độ sáng, màu sắc trở nên cân bằng hơn, màu sắc trở nên trung thực hơn. Cụ thể hơn, biểu đồ histogram sau khi cân bằng phân bố đều hơn và các đỉnh và đáy của biểu đồ được tách ra rõ ràng hơn, chỉ ra sự cân bằng giá trị pixel trên toàn bộ ảnh.

+ Sự khác biệt giữa ảnh gốc và ảnh đã được cân bằng rõ ràng hơn khi xem trực tiếp hai ảnh kèm theo biểu đồ histogram của chúng.

– Kết quả ảnh cân bằng bằng hàm tự viết và ảnh cân bằng dùng OpenCV đều cho thấy một sự cải thiện đáng kể trong việc cân bằng histogram của ảnh gốc. Cả hai phương pháp đều đã thực hiện cân bằng histogram một cách hiệu quả và đưa ra kết quả tương đối giống nhau.

– Tuy nhiên, phương pháp sử dụng hàm cân bằng histogram tự viết yêu cầu một số lượng mã lệnh nhiều hơn so với phương pháp sử dụng OpenCV. Điều này có thể khiến nó trở nên khó khăn hơn để quản lý và duy trì nếu có nhiều tác vụ xử lý ảnh khác nhau cần sử dụng cân bằng histogram. Trong khi đó, sử dụng hàm của OpenCV rất thuận tiện và nhanh chóng.

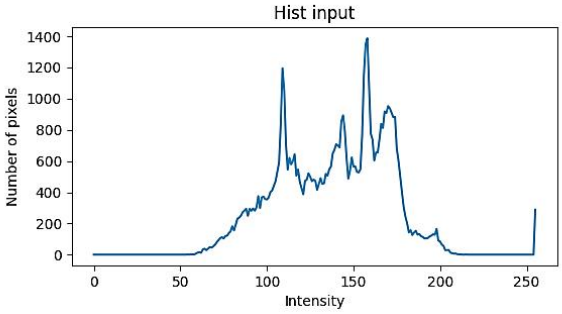
c. BT3: cân bằng histogram của ảnh màu

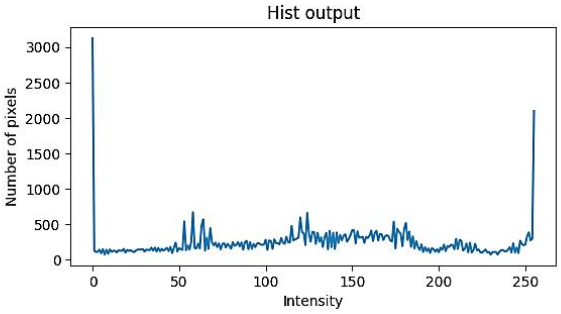
SOURCE CODE

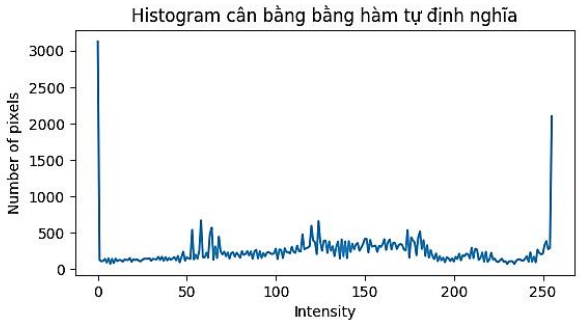
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def find_hist(img):
    hist = np.zeros(256)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            hist[img[i, j]] += 1
    return hist
def equalize_hist(img):
    # tính tổng pixels
    total_pixels = img.shape[0] * img.shape[1]
    # tính xác suất
    prob = find_hist(img) / total_pixels
    # tính phân bố đồ thị tuyệt đối
    cdf = np.zeros(256)
    for i in range(256):
        cdf[i] = np.sum(prob[:i + 1])
    # tính phân bố đồ thị tuyệt đối sau khi cân bằng
```

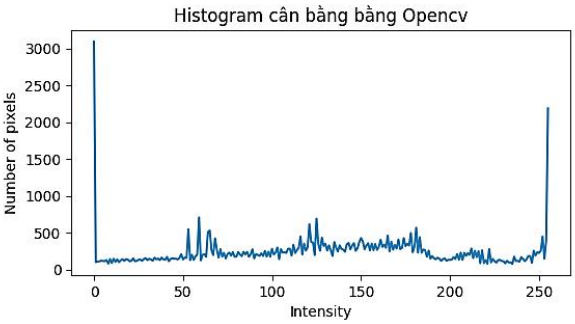
```
    cdf = (cdf * 255).astype(np.uint8)
    # thay đổi giá trị pixel
    new_img = np.zeros(img.shape, np.uint8)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            new_img[i, j] = cdf[img[i, j]]
    return new_img
img = cv2.imread("Image_processing/image/HE.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_YCrCb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)
hist_img = find_hist(img_YCrCb[:, :, 0])
test=img_YCrCb.copy()
img_YCrCb[:, :, 0] = equalize_hist(img_YCrCb[:, :, 0])
img_YCrCb = cv2.cvtColor(img_YCrCb, cv2.COLOR_YCrCb2BGR)
hist_output = find_hist(img_YCrCb[:, :, 0])
plt.subplot(2, 2, 1)
plt.imshow(img), plt.title("Input"), plt.axis("off")
plt.subplot(2, 2, 2)
plt.imshow(img_YCrCb), plt.title("Output"), plt.axis("off")
plt.subplot(2, 2, 3)
plt.plot(hist_img), plt.title("Hist input"), plt.xlabel(
    "Intensity"), plt.ylabel("Number of pixels")
plt.subplot(2, 2, 4)
plt.plot(hist_output), plt.title("Hist output"), plt.xlabel(
    "Intensity"), plt.ylabel("Number of pixels")
plt.show()
test[:, :, 0] = cv2.equalizeHist(test[:, :, 0])
test = cv2.cvtColor(test, cv2.COLOR_YCrCb2BGR)
hist_test = find_hist(test[:, :, 0])
```

KẾT QUẢ:









NHẬN XÉT

– Đối với hình ảnh thang độ xám, mỗi pixel được biểu thị bằng giá trị cường độ (độ sáng); đó là lý do tại sao chúng ta có thể cung cấp trực tiếp các giá trị pixel cho hàm Histogram Equalization. Tuy nhiên, đó không phải là cách nó hoạt động đối với hình ảnh màu có định dạng *RGB* . Mỗi kênh của *R* , *G* và *B* đại diện cho cường độ của màu liên quan, không phải cường độ/độ sáng của toàn bộ hình ảnh. Và vì vậy**,** chạy **Histogram Equalization** trên các kênh màu này KHÔNG phải là cách phù hợp .

- Trước tiên chúng ta nên tách độ sáng của hình ảnh khỏi màu sắc và sau đó chạy HE trên độ sáng. Giờ đây, đã có các không gian màu được tiêu chuẩn hóa mã hóa độ sáng và màu sắc riêng biệt, như- *YCbCr* , *HSV* , v.v.; vì vậy, chúng ta có thể sử dụng chúng ở đây để tách và sau đó hợp nhất lại độ sáng. Cách thích hợp:
- Chuyển đổi không gian màu từ *RGB* sang *YCbCr* >> Chạy Histogram Equalization trên kênh *Y* (kênh này biểu thị độ sáng) >> Chuyển đổi lại không gian màu thành *RGB*
- Đối với không gian màu *HSV* , Histogram Equalization phải được chạy trên kênh *V*. Tuy nhiên, kênh *Y* của *YCbCr* là đại diện tốt hơn cho độ sáng so với kênh *V* của *HSV* . Vì vậy, sử dụng định dạng ***YCbCr*** sẽ tạo ra kết quả chính xác hơn cho **Histogram Equalization .**

4. Buổi 4 (09-02-2023):

a. BT1

ĐỀ BÀI

Cho I là ảnh xám:

h1 = 1/9 * ones(3, 3)

h2 = 1/25 * ones(5, 5)

h3 = 1/81 * ones(9, 9)

Lọc I bởi bộ lọc h1, h2, h3

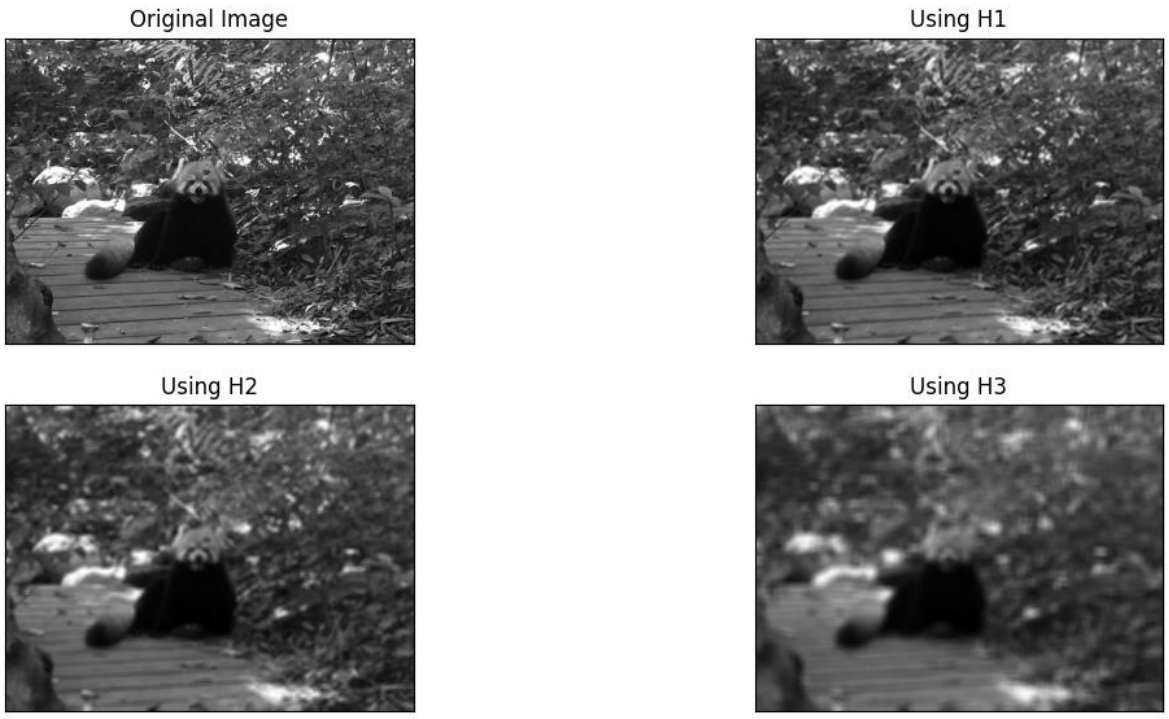
Hiển thị kết quả & nhận xét

SOURCE CODE

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img=cv2.imread("Image_processing/image/5.bmp",0)
H1=1/9*np.ones((3,3))
H2=1/25*np.ones((5,5))
H3=1/81*np.ones((9,9))
Y1=cv2.filter2D(img,-1,H1)
Y2=cv2.filter2D(img,-1,H2)
```

```
Y3=cv2.filter2D(img,-1,H3)
plt.subplot(2,2,1),plt.imshow(img,cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,2),plt.imshow(Y1,cmap='gray')
plt.title('Using H1'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,3),plt.imshow(Y2,cmap='gray')
plt.title('Using H2'), plt.xticks([]), plt.yticks([])
plt.subplot(2,2,4),plt.imshow(Y3,cmap='gray')
plt.title('Using H3'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ



Hình . Kết quả

NHẬN XÉT

- Y1, Y2, Y3 là kết quả khi sử dụng bộ lọc H1, H2, H3 lên Y.
- Việc áp dụng bộ lọc là một phương pháp xử lý ảnh phổ biến để làm mịn hoặc làm nét ảnh. Kết quả thu được phụ thuộc vào kích thước và giá trị của bộ lọc được sử dụng.

- Trong đoạn mã trên, chúng ta đã áp dụng ba bộ lọc khác nhau trên một ảnh grayscale và hiển thị kết quả. Dưới đây là một số nhận xét về kết quả thu được:

+ Bộ lọc H1 (kích thước 3x3) có tác dụng làm mịn ảnh, giảm nhiễu. Ảnh kết quả Y1 có vẻ mịn hơn ảnh gốc, các chi tiết nhỏ bị mất đi.

+ Bộ lọc H2 (kích thước 5x5) làm mịn hơn nữa và có tác dụng làm giảm các cạnh trong ảnh. Ảnh kết quả Y2 có vẻ mịn hơn Y1, các đường cạnh trên ảnh ban đầu đã bị mờ đi.

+ Bộ lọc H3 (kích thước 9x9) làm mịn ảnh nhiều hơn và mất đi nhiều chi tiết hơn. Ảnh kết quả Y3 có vẻ mịn nhất trong ba ảnh được xử lý, tuy nhiên nó đã mất đi nhiều thông tin hơn so với ảnh gốc và các ảnh kết quả trước đó.

- Tóm lại, khi áp dụng bộ lọc, chúng ta phải cân nhắc kích thước và giá trị của bộ lọc để đảm bảo độ mịn của ảnh thu được phù hợp với mục đích sử dụng và đồng thời giữ được các chi tiết quan trọng trên ảnh.

b. BT2

ĐỀ BÀI

Làm lại câu 1 thay I bằng ảnh I có nhiễu gauss (mean=0, var=theta^2)

I<-I+theta.randn(M,N)

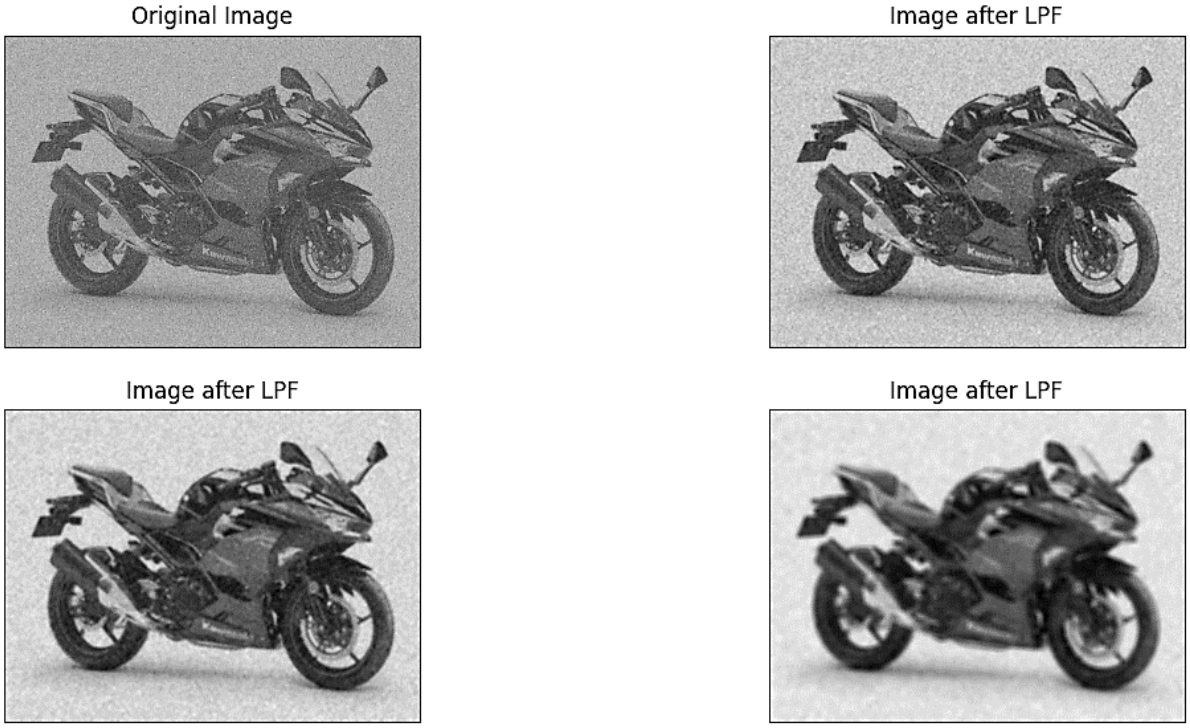
randn(M,N) tạo ma trận ngẫu nhiêu MxN giá tri thuộc N(0,1) (phân phối chuẩn)

SOURCE CODE

```
import cv2

import numpy as np

from matplotlib import pyplot as plt

img = cv2.imread("Image_processing/image/motor.jpg", 0)

img = img.astype(np.float64) + 50*np.random.randn(*img.shape)

H1 = 1/9*np.ones((3, 3))

H2 = 1/25*np.ones((5, 5))

H3 = 1/81*np.ones((9, 9))

Y2 = cv2.filter2D(img, -1, H1)

Y3 = cv2.filter2D(img, -1, H2)
```

```
Y4 = cv2.filter2D(img, -1, H3)
plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(Y2, cmap='gray')
plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(Y3, cmap='gray')
plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(Y4, cmap='gray')
plt.title('Image after LPF'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ



Hình . Nhận xét

NHẬN XÉT

- Kết quả đầu ra của đoạn code trên hiển thị 4 ảnh với các tiêu đề tương ứng. Ảnh gốc là một bức ảnh màu xám bị nhiễu Gaussian với độ lệch chuẩn là 50. Các ảnh còn lại được tạo ra bằng cách áp dụng bộ lọc thông thấp (LPF) với các kernel khác nhau: H1 (3x3), H2 (5x5) và H3 (9x9). Kết quả cho thấy khi ta áp dụng lọc trên ảnh nhiễu thì các nhiễu được giảm đáng kể, nhưng đồng thời

cũng làm mất đi một số chi tiết của bức ảnh gốc. Nên tùy vào mục đích sử dụng, ta có thể chọn kernel phù hợp để tạo ra kết quả tốt nhất.

c. BT3

ĐỀ BÀI

Cho H = 1/4 * [1, 2, 1](1 cột), H2 = H1' (chuyển vị / transpose)

I là ảnh xám

a. tính Y1 = I*H1, Y2 = Y1*H2

b. gọi H = 1/16 * [1, 2, 1

2, 4, 2

1, 2, 1]. tính Y3 = I*H

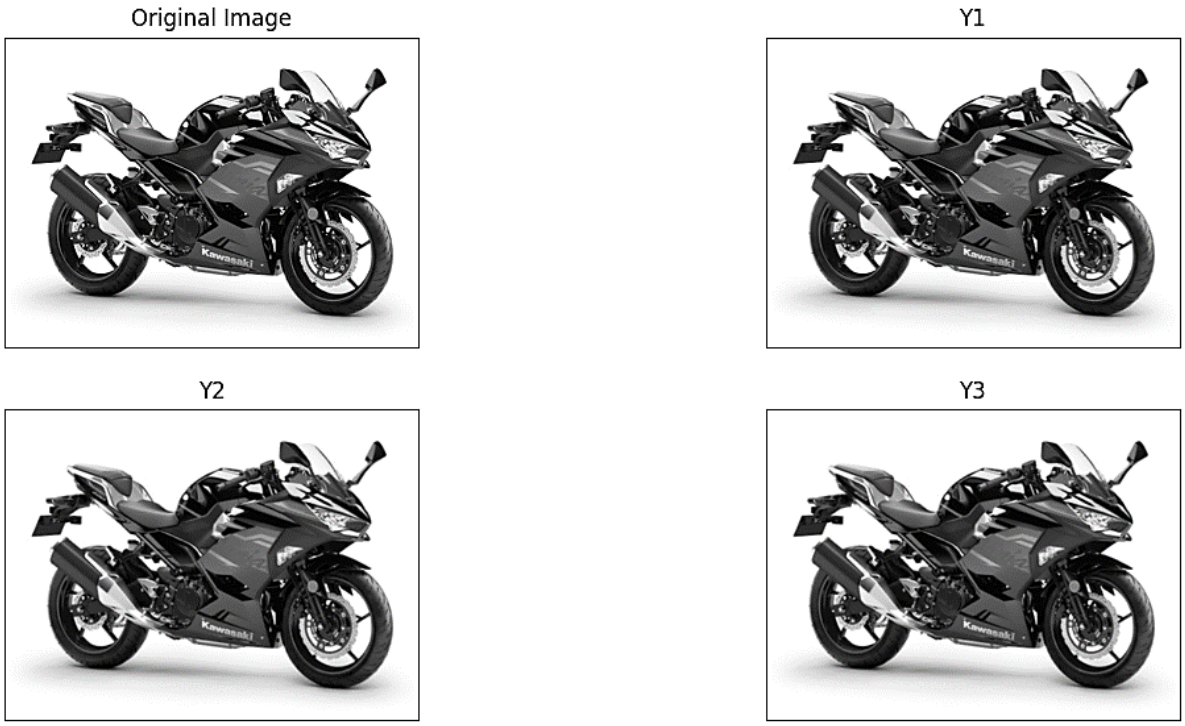
c. so sánh Y2 & Y3. Nhận xét.

SOURCE CODE

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread("Image_processing/image/motor.jpg", 0)
H1 = np.array([[1, 2, 1]])/4
H2 = np.transpose(H1)
H3 = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/16
# câu a
Y1 = cv2.filter2D(img, -1, H1)
Y2 = cv2.filter2D(Y1, -1, H2)
# câu b
Y3 = cv2.filter2D(img, -1, H3)
plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(Y1, cmap='gray')
plt.title('Y1'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(Y2, cmap='gray')
```

```
plt.title('Y2'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(Y3, cmap='gray')
plt.title('Y3'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ



NHẬN XÉT

- Y2 và Y3 là 2 kết quả khác nhau sau khi áp dụng bộ lọc. Y2 được tạo ra bằng cách áp dụng bộ lọc giữa H1 và H2 lần lượt trên ảnh gốc, trong khi Y3 được tạo ra bằng cách áp dụng bộ lọc H3.
- Y2 là kết quả của việc lọc thông tin bên trong ảnh để giảm nhiễu và làm mờ cạnh, trong khi Y3 là kết quả của việc áp dụng bộ lọc trung bình trên toàn bộ ảnh để làm mịn ảnh.
- Nhìn chung, Y2 được áp dụng để giảm nhiễu và Y3 để làm mịn ảnh. Tuy nhiên, Y2 có xu hướng làm mất một số chi tiết nhỏ và cạnh, trong khi Y3 làm mất nhiều chi tiết hơn. Do đó, việc chọn bộ lọc phụ thuộc vào mục đích sử dụng và đặc tính của ảnh.

5. **Buổi 5 (16-02-2023):**

ĐỀ BÀI: Tạo ảnh mosaic

SOURCE CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import os
img = cv2.imread("Image_processing/image/Kodak/19.png")
img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
(r, c, _) = img.shape
mr = np.zeros((r, c))
mg = np.zeros((r, c))
mb = np.zeros((r, c))
mr[::2, ::2] = 1
mb[1::2, 1::2] = 1
mg = 1 - mb - mr
I_red = img[:, :, 2].astype(np.float64)
I_green = img[:, :, 1].astype(np.float64)
I_blue = img[:, :, 0].astype(np.float64)
red = mr * I_red
green = mg * I_green
blue = mb * I_blue
demos_picture = np.zeros((r, c, 3), dtype=np.uint8)
demos_picture[:, :, 0] = blue
demos_picture[:, :, 1] = green
demos_picture[:, :, 2] = red
plt.subplot(1,2,1)
plt.imshow(img), plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,2,2)
```

plt.imshow(demos_picture), plt.title('Mosaic image'), plt.xticks([]), plt.yticks([])

plt.show()

KẾT QUẢ

6. **Buổi 6 (23-02-2023):**

ĐỀ BÀI: khôi phục ảnh màu bằng phương pháp nội suy tuyến tính

SOURCE CODE

import cv2

import numpy as np

import matplotlib.pyplot as plt

import os

img = cv2.imread("Image_processing/image/Kodak/19.png")

img=cv2.cvtColor(img,cv2.COLOR_BGR2RGB)

(r, c, _) = img.shape

mr = np.zeros((r, c))

mg = np.zeros((r, c))

mb = np.zeros((r, c))

mr[::2, ::2] = 1

```
mb[1::2, 1::2] = 1
mg = 1 - mb - mr
I_red = img[:, :, 2].astype(np.float64)
I_green = img[:, :, 1].astype(np.float64)
I_blue = img[:, :, 0].astype(np.float64)
red = mr * I_red
green = mg * I_green
blue = mb * I_blue
demos_picture = np.zeros((r, c, 3), dtype=np.uint8)
demos_picture[:, :, 0] = blue
demos_picture[:, :, 1] = green
demos_picture[:, :, 2] = red
w_R = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]])/4
w_G = np.array([[0, 1, 0], [1, 4, 1], [0, 1, 0]])/4
w_B = w_R
blue = cv2.filter2D(blue, -1, w_B)
green = cv2.filter2D(green, -1, w_G)
red = cv2.filter2D(red, -1, w_R)
picture_restored = np.zeros((r, c, 3), dtype=np.uint8)
picture_restored[:, :, 0] = blue
picture_restored[:, :, 1] = green
picture_restored[:, :, 2] = red
plt.subplot(1,3,1)
plt.imshow(img), plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,2)
plt.imshow(demos_picture), plt.title('Demosic'), plt.xticks([]), plt.yticks([])
plt.subplot(1,3,3)
plt.imshow(picture_restored), plt.title('Restored'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ

7. **Buổi 7 (02-03-2023):**

ĐỀ BÀI: khôi phục ảnh màu bằng phương pháp Aleysson

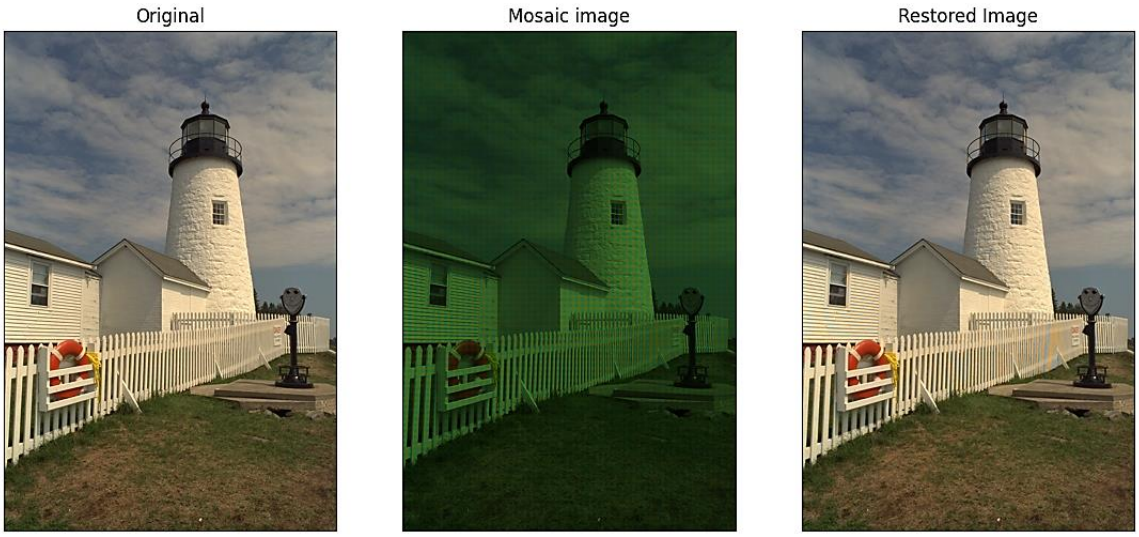
SOURCE CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("Image_processing/image/Kodak/19.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
(row, col, rgb) = img.shape
mr = np.zeros((row, col))
mg = np.zeros((row, col))
mb = np.zeros((row, col))
mr[::2, ::2] = 1
mb[1::2, 1::2] = 1
mg = 1 - mr - mb
red = img[:, :, 2]
green = img[:, :, 1]
blue = img[:, :, 0]
red = np.multiply(red.astype(float), mr)
```

```
green = np.multiply(green.astype(float), mg)
blue = np.multiply(blue.astype(float), mb)
multi_img = np.zeros((row, col, 3), dtype=np.uint8)
multi_img[:, :, 0] = blue.astype(np.uint8)
multi_img[:, :, 1] = green.astype(np.uint8)
multi_img[:, :, 2] = red.astype(np.uint8)
F_L = np.array([[-2, 3, -6, 3, -2],
                [3, 4,  2, 4,  3],
                [-6, 2, 48, 2, -6],
                [3, 4,  2, 4,  3],
                [-2, 3, -6, 3, -2]]) * (1/64)
out = red + green + blue
lum = cv2.filter2D(out, -1, F_L)
multi_chr = out - lum
redd = np.zeros((row, col))
greenn = np.zeros((row, col))
bluee = np.zeros((row, col))
redd[::2, ::2] = multi_chr[::2, ::2]
bluee[1::2, 1::2] = multi_chr[1::2, 1::2]
greenn = multi_chr - redd - bluee
smp_chr = np.zeros((row, col, 3), dtype=np.float64)
smp_chr[:, :, 0] = redd
smp_chr[:, :, 1] = greenn
smp_chr[:, :, 2] = bluee
wrb = np.array([[1, 2, 1],
                [2, 4, 2],
                [1, 2, 1]])/4
wg = np.array([[0, 1, 0],
               [1, 4, 1],
```

```
            [0, 1, 0]])/4
redd = cv2.filter2D(redd, -1, wrb)
greenn = cv2.filter2D(greenn, -1, wg)
bluee = cv2.filter2D(bluee, -1, wrb)
chr = np.zeros((row, col, 3), dtype=np.float64)
chr[:, :, 0] = bluee
chr[:, :, 1] = greenn
chr[:, :, 2] = redd
picture_res = np.zeros((row, col, 3), dtype=np.uint8)
picture_res[:, :, 0] = np.clip(chr[:, :, 0] + lum, 0, 255)
picture_res[:, :, 1] = np.clip(chr[:, :, 1] + lum, 0, 255)
picture_res[:, :, 2] = np.clip(chr[:, :, 2] + lum, 0, 255)
plt.subplot(1, 3, 1)
plt.imshow(img), plt.title('Original'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 3, 2)
plt.imshow(multi_img), plt.title(
    'Mosaic image'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 3, 3)
plt.imshow(picture_res), plt.title(
    'Restored Image'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ



NHẬN XÉT:

- Phương pháp nội suy tuyến tính và phương pháp Alleyson là hai trong số các phương pháp khôi phục ảnh màu từ ảnh mosaic. Cả hai phương pháp đều được sử dụng để tái tạo màu của ảnh bằng cách sử dụng thông tin màu từ ảnh gốc.
- Tuy nhiên, phương pháp nội suy tuyến tính chỉ sử dụng một cách đơn giản là sử dụng các thông tin màu của các pixel lân cận nhất trong ảnh mosaic để xác định giá trị màu của các pixel tương ứng trong ảnh màu tái tạo. Phương pháp này thường được sử dụng trong các ứng dụng đơn giản như việc tái tạo màu của các ảnh được chụp từ các máy ảnh cũ hoặc các bức ảnh cổ.
- Trong khi đó, phương pháp Alleyson sử dụng một mô hình màu để định vị các thông tin màu và khôi phục màu của các pixel trong ảnh mosaic. Điều này có nghĩa là phương pháp này sử dụng một mô hình màu để tính toán màu của các pixel trong ảnh màu tái tạo. Phương pháp này cho phép tái tạo ảnh màu chính xác hơn và đạt được kết quả tốt hơn đối với các ảnh phức tạp.
- Tóm lại, phương pháp nội suy tuyến tính là một phương pháp đơn giản và được sử dụng rộng rãi trong các ứng dụng đơn giản, trong khi phương pháp Alleyson cung cấp kết quả tốt hơn đối với các ảnh phức tạp hơn, nhưng cũng đòi hỏi nhiều tính toán hơn.

8. **Buổi 8 (09-03-2023):** ***thi giữa kỳ***
9. **Buổi 9 (16-03-2023):**

ĐỀ BÀI:

2D Dicrete Fourier Transform and Inverse Discrete Fourier Transform cho ảnh xám như hình anh1, anh2

1.tìm và hiển thị phổ biên độ của ảnh (dùng lệnh fft2)

2. tịnh tiến thanh màu trắng đến vị trí khác. làm lại câu 1 và nhận xét

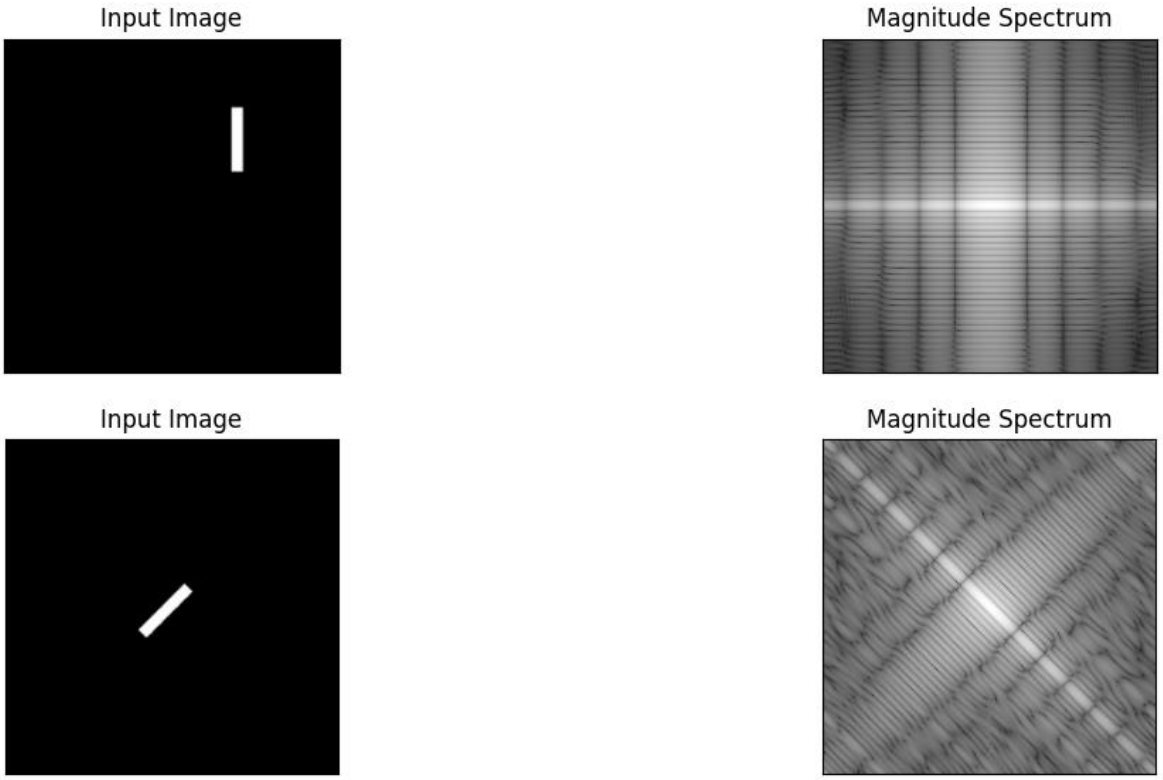
3. xoay ảnh để tạo ra ảnh mới

SOURCE CODE

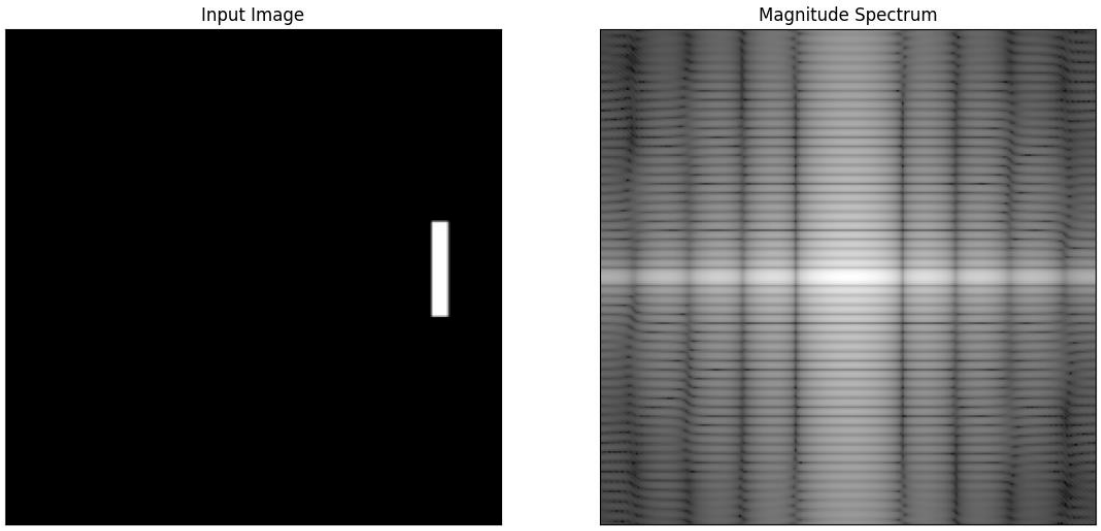
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.ndimage import rotate
from scipy.ndimage import shift
# 1. tìm và hiển thị phổ biên độ của ảnh (dùng lệnh fft2)
anh1 = cv2.imread("Image_processing/image/anh1.png", 0)
anh2 = cv2.imread("Image_processing/image/anh2.png", 0)
anh3 = cv2.imread("Image_processing/image/cameraman.jpg", 0)
f1 = np.fft.fft2(anh1)
f1shift = np.fft.fftshift(f1)
magnitude_spectrum1 = 20*np.log(np.abs(f1shift))
f2 = np.fft.fft2(anh2)
f2shift = np.fft.fftshift(f2)
magnitude_spectrum2 = 20*np.log(np.abs(f2shift))
plt.subplot(221), plt.imshow(anh1, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(222), plt.imshow(magnitude_spectrum1, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(223), plt.imshow(anh2, cmap='gray')
```

```
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(224), plt.imshow(magnitude_spectrum2, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
# 2. tịnh tiến thanh màu trắng đến vị trí khác. làm lại câu 1 và nhận xét
anh1_shift = shift(anh1, (50, 50))
f3 = np.fft.fft2(anh1_shift)
f3shift = np.fft.fftshift(f3)
magnitude_spectrum3 = 20*np.log(np.abs(f3shift))
plt.subplot(121), plt.imshow(anh1_shift, cmap='gray')
plt.title('Input Image'), plt.xticks([]), plt.yticks([])
plt.subplot(122), plt.imshow(magnitude_spectrum3, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.show()
# ảnh - trung bình ảnh => trung tâm ảnh có giá trị =0
anh3 = anh3-np.mean(anh3)
f4 = np.fft.fft2(anh3)
f4shift = np.fft.fftshift(f4)
magnitude_spectrum4 = 20*np.log(np.abs(f4shift))
```

KẾT QUẢ



Hình . Kết quả câu 1 và câu 3



Hình . Kết quả câu 2

NHẬN XÉT

- Khi xoay ảnh góc bao nhiêu độ thì phổ biên độ xoay bấy nhiêu
- Khi tịnh tiến ảnh thì phổ biên độ không đổi

+ Ở câu 1, chúng ta đã tìm và hiển thị phổ biên độ của ảnh, thấy rằng phổ biên độ tập trung chủ yếu ở các tần số thấp.

+ Ở câu 2, chúng ta đã tịnh tiến ảnh và thực hiện lại phần 1. Kết quả cho thấy phổ biên độ của ảnh tịnh tiến cũng tập trung ở các tần số thấp và không có sự thay đổi đáng kể.

+ Ở câu 3, chúng ta đã xoay ảnh và quan sát thấy rằng phổ biên độ của ảnh sau khi xoay bị biến đổi đáng kể, với một số tần số cao hơn.

- Từ đó, ta nhận thấy rằng phổ biên độ của ảnh cung cấp thông tin về tần số và hướng của các đặc trưng trong ảnh, giúp chúng ta hiểu hơn về tính chất của ảnh và thực hiện các phép biến đổi cơ bản trên ảnh.

10. Buổi 10 (23-03-2023):

a. BT1

ĐỀ BÀI:

phổ biên độ và phổ pha:

Cho I là 1 ảnh xám:

S_I=DFT{I}=|S_I|*exp(j*phi_I)

|S_I| là phổ biên độ, phi_I là phổ pha

|S_I| và phi_I là hàm theo (u,v)

1) Hiển thị |S_I| và phi_I

2) Gọi S_1=1*exp(j*phi_I), Hiển thị ảnh I_1 tương ứng với phổ S_1

3) Gọi S_2=|S_I|*exp(j*phi_2), với phi_2(u,v)=1, Hiển thị ảnh I_2 tương ứng với phổ S_2, nhận xét

4) Cho Y là một ảnh xám khác với phổ

S_Y=DFT{Y}=|S_Y|*exp(j*phi_Y)

Gọi S_3=|S_I|*exp(j*phi_Y)

Gọi S_4=|S_Y|*exp(j*phi_I) ;(I và Y có cùng kích thước)

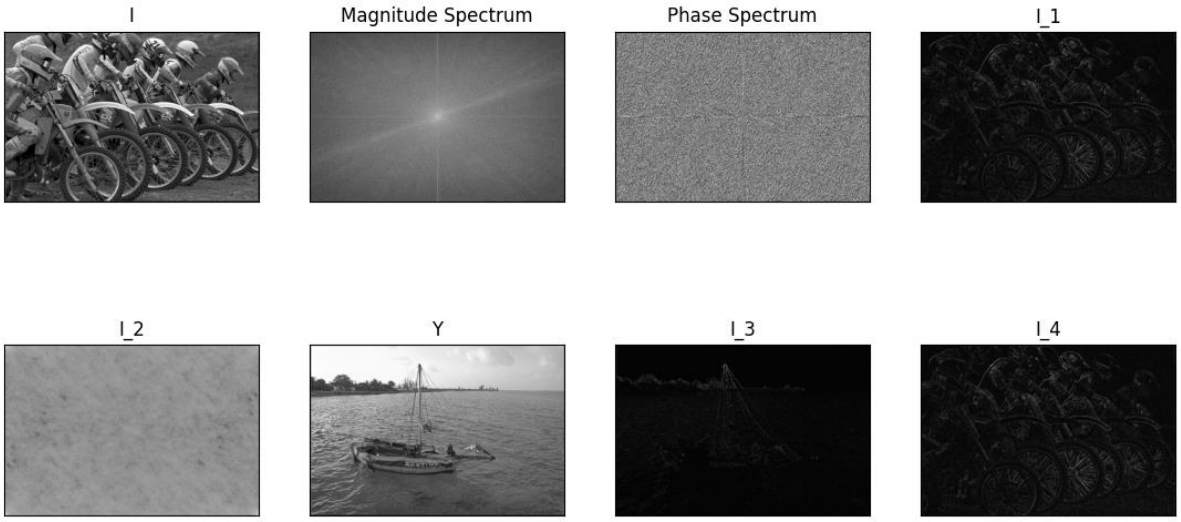
Hiển thị ảnh I_3 & I_4 tương ứng với phổ S_3 & S_4, nhận xét

SOURCE CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
I = cv2.imread("Image_processing/image/Kodak/5.png", 0)
rows, cols = I.shape
# 1)
S_I = np.fft.fft2(I)
S_I_shift = np.fft.fftshift(S_I)
magnitude_spectrum_I = np.abs(S_I_shift)
magnitude_spectrum_II = 20*np.log(magnitude_spectrum_I)
phase_spectrum_I = np.angle(S_I_shift)
# 2)
S_1 = 1*np.exp(1j*phase_spectrum_I)
S_1_shift = np.fft.ifftshift(S_1)
I_1 = np.fft.ifft2(S_1_shift)
I_1 = np.abs(I_1)
# 3)
S_2 = magnitude_spectrum_I*np.exp(1j)
S_2=np.fft.ifftshift(S_2)
I_2 = np.fft.ifft2(S_2)
I_2 = np.abs(I_2)
I_2=np.log(I_2)
# 4)
Y = cv2.imread("Image_processing/image/Kodak/6.png", 0)
S_Y = np.fft.fft2(Y)
magnitude_spectrum_Y = 20*np.log(np.abs(S_Y))
phase_spectrum_Y = np.angle(S_Y)
S_3 = magnitude_spectrum_II*np.exp(1j*phase_spectrum_Y)
```

```
S_4 = magnitude_spectrum_Y*np.exp(1j*phase_spectrum_I)
S_3=np.fft.ifftshift(S_3)
S_4=np.fft.ifftshift(S_4)
I_3 = np.fft.ifft2(S_3)
I_3 = np.abs(I_3)
I_4 = np.fft.ifft2(S_4)
I_4 = np.abs(I_4)
# Hiển thị kết quả
plt.subplot(241), plt.imshow(I, cmap='gray')
plt.title('I'), plt.xticks([]), plt.yticks([])
plt.subplot(242), plt.imshow(magnitude_spectrum_II, cmap='gray')
plt.title('Magnitude Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(243), plt.imshow(phase_spectrum_I, cmap='gray')
plt.title('Phase Spectrum'), plt.xticks([]), plt.yticks([])
plt.subplot(244), plt.imshow(I_1, cmap='gray')
plt.title('I_1'), plt.xticks([]), plt.yticks([])
plt.subplot(245), plt.imshow(I_2, cmap='gray')
plt.title('I_2'), plt.xticks([]), plt.yticks([])
plt.subplot(246), plt.imshow(Y, cmap='gray')
plt.title('Y'), plt.xticks([]), plt.yticks([])
plt.subplot(247), plt.imshow(I_3, cmap='gray')
plt.title('I_3'), plt.xticks([]), plt.yticks([])
plt.subplot(248), plt.imshow(I_4, cmap='gray')
plt.title('I_4'), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ



NHẬN XÉT

- Kết quả I_1 là kết quả của phép dịch chuyển pha của phổ ảnh I. I_1 có cùng hình dạng với ảnh gốc I, nhưng có màu sắc khác vì pha của phổ đã bị thay đổi.
- Kết quả I_2 là kết quả của phép thay đổi phổ của ảnh I. I_2 có hình dạng khác với ảnh gốc I và được trông thấy như một ảnh phản chiếu của ảnh gốc I, tạo ra một hiệu ứng gương.
- Kết quả I_3 là kết quả của phép thay đổi phổ của ảnh Y bằng phổ biên độ của ảnh I. I_3 có hình dạng tương tự như ảnh gốc Y, nhưng có một số hiệu chỉnh do thay đổi phổ.
- Kết quả I_4 là kết quả của phép thay đổi phổ của ảnh Y bằng phổ biên độ của ảnh I và ngược lại. I_4 có hình dạng tương tự như ảnh gốc Y, nhưng có màu sắc khác do thay đổi phổ.
- Phổ biên độ được sử dụng để phân tích đặc trưng của ảnh và làm cho chúng ta hiểu được cấu trúc của nó. Phổ phase thường được sử dụng để phục hồi hình ảnh hoặc để thực hiện các phép biến đổi không gian khác trên ảnh.
- Phase giúp giữ được đường nét của ảnh

b. BT2

ĐỀ BÀI

Phổ của ảnh mosaic

Cho một ảnh màu I, tạo ra ảnh mosaic tương ứng (1 lớp, 3 màu xen kẽ)

Hiển thị phổ biên độ của ảnh mosaic

Chỉ ra thành phần luminance và chrominance của ảnh mosaic

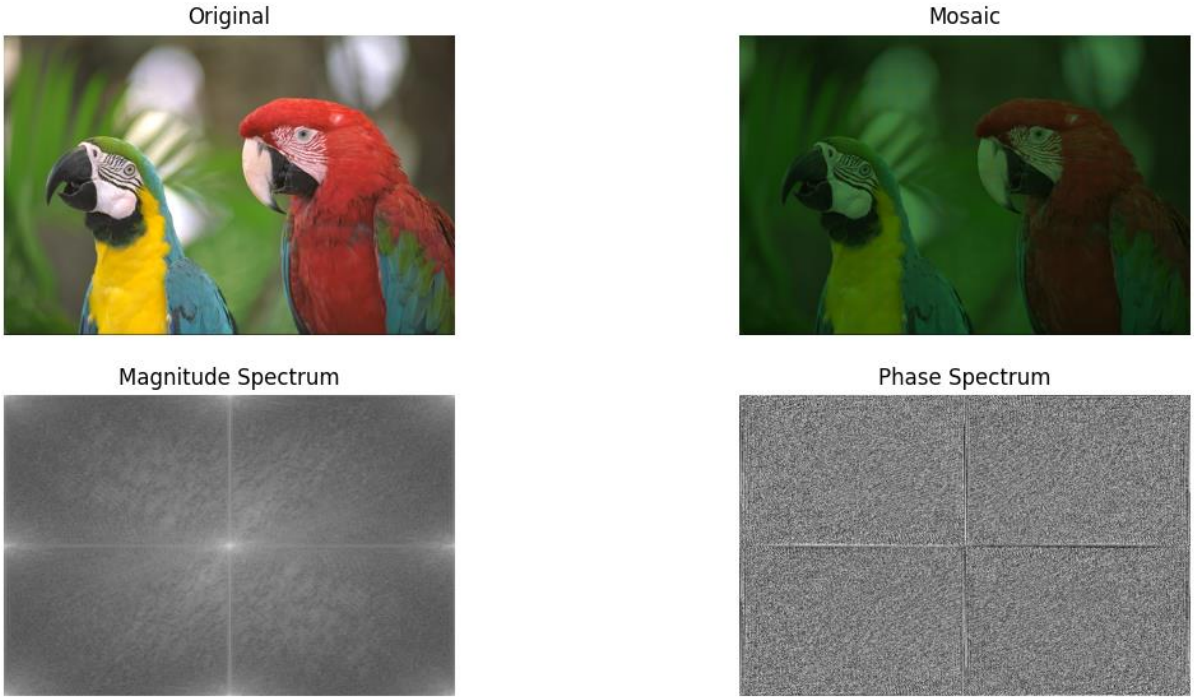
So sánh với pp trong bài báo. Nhận xét

SOURCE CODE

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("Image_processing/image/Kodak/23.png")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
(row, col, rgb) = img.shape
mr = np.zeros((row, col))
mg = np.zeros((row, col))
mb = np.zeros((row, col))
mr[::2, ::2] = 1
mb[1::2, 1::2] = 1
mg = 1 - mr - mb
red = img[:, :, 2]
green = img[:, :, 1]
blue = img[:, :, 0]
red = np.multiply(red.astype(float), mr)
green = np.multiply(green.astype(float), mg)
blue = np.multiply(blue.astype(float), mb)
multi_img = np.zeros((row, col, 3), dtype=np.uint8)
multi_img[:, :, 0] = blue.astype(np.uint8)
multi_img[:, :, 1] = green.astype(np.uint8)
multi_img[:, :, 2] = red.astype(np.uint8)
mosaic = red + green + blue
# Phổ của ảnh mosaic
mosaic_fft = np.fft.fft2(mosaic)
```

```
mosaic_fft_shift = np.fft.fftshift(mosaic_fft)
mosaic_fft_mag = np.log(np.abs(mosaic_fft_shift))
mosaic_fft_phase = np.angle(mosaic_fft_shift)
plt.subplot(221)
plt.imshow(img, cmap='gray'), plt.title('Original'), plt.axis('off')
plt.subplot(222)
plt.imshow(multi_img, cmap='gray'), plt.title('Mosaic'), plt.axis('off')
plt.subplot(223)
plt.imshow(mosaic_fft_mag, cmap='gray'), plt.title(
    'Magnitude Spectrum'), plt.axis('off')
plt.subplot(224)
plt.imshow(mosaic_fft_phase, cmap='gray'), plt.title(
    'Phase Spectrum'), plt.axis('off')
plt.show()
```

KẾT QUẢ



Hình . Kết quả

NHẬN XÉT

- Chỉ ra thành phần Luminance và Chrominance (dựa vào phổ biên độ)

  Luminance: phần sáng ở giữa

  Chrominance: phần sáng4 góc và biên

11. Buổi 11 (30-03-2023):

a. BT1: tìm biến đổi Fourier của $\mathbf{h(x) = e^{-\frac{x^2}{2\sigma^2}}}$

- Công thức biến đổi fourier:

$$X(\omega) = \int_{-\infty}^{+\infty} x(t).e^{-j\omega t}dt \quad (1)$$

- Áp dụng với h(x):

$$H(\omega) = F\left[e^{-\frac{x^2}{2\sigma^2}}\right] = \int_{-\infty}^{+\infty} h(x).e^{-j\omega x}dx \quad (2)$$

$$H(\omega) = F\left[e^{-\frac{x^2}{2\sigma^2}}\right] = \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2\sigma^2}}.e^{-j\omega x}dx \quad (3)$$

$$H(\omega) = F\left[e^{-\frac{x^2}{2\sigma^2}}\right] = \int_{-\infty}^{+\infty} e^{-\frac{x^2}{2\sigma^2}-j\omega x}dx \quad (4)$$

Đặt $a = \frac{1}{2\sigma^2}$

- Suy ra:

$$H(\omega) = \int_{-\infty}^{+\infty} e^{-(ax^2+j\omega x)}dx \quad (5)$$

$$H(\omega) = e^{-\left(\frac{\omega^2}{4a}\right)} \int_{-\infty}^{+\infty} e^{-\left(x\sqrt{a}+\frac{j\omega}{2\sqrt{a}}\right)^2}dx \quad (6)$$

Đặt $u = x\sqrt{a} + \frac{j\omega}{2\sqrt{a}}$

- Suy ra:

$$du = \sqrt{a}dx \rightarrow dx = \frac{du}{\sqrt{a}} \quad (7)$$

$$H(\omega) = e^{-\left(\frac{\omega^2}{4a}\right)} \int_{-\infty}^{+\infty} \frac{e^{-u^2}}{\sqrt{a}} du \quad (8)$$

- Với:

$$\int_{-\infty}^{+\infty} e^{-u^2} du = \sqrt{\pi} \quad (9)$$

- Suy ra:

$$H(\omega) = \frac{e^{-\left(\frac{\omega^2}{4a}\right)}}{\sqrt{a}} . \sqrt{\pi} = \sqrt{\frac{\pi}{a}} . e^{-\left(\frac{\omega^2}{4a}\right)} = \sigma\sqrt{2\pi} . e^{-\frac{\omega^2\sigma^2}{2}} \quad (10)$$

- Kết luận:

$$\boldsymbol{h(x) = e^{-\frac{x^2}{2\sigma^2}} \overset{FT}{\leftrightarrow} H(\omega) = \sigma\sqrt{2\pi} . e^{-\frac{\omega^2\sigma^2}{2}}} \quad (\mathbf{11})$$

b. BT2: gọi I là ảnh Mosaic thu được từ một ảnh màu cho trước (I gồm 1 lớp 3 màu xen kẽ). Sử dụng bộ lọc thông thấp lý tưởng, Butterworth, Gaussian một cách hợp lí để lọc I. Hiển thị kết quả và nhận xét

SOURCE CODE

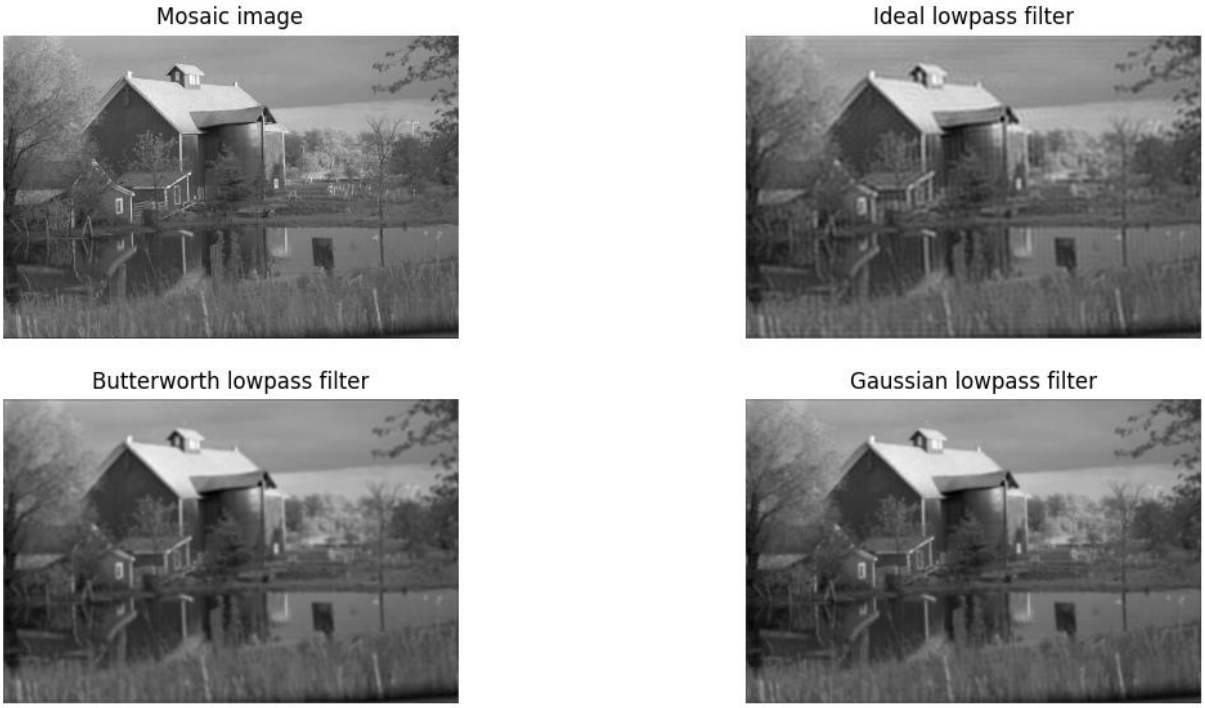
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def ideal_lowpass_filter(img, D0):
    img_double = np.double(img)
    row, col = img_double.shape
    img_fft = np.fft.fft2(img_double)
    img_fft_shift = np.fft.fftshift(img_fft)
    u, v = np.meshgrid(np.arange(col), np.arange(row))
    u = u-(1+col)/2
    v = v-(1+row)/2
    u = u/col
    v = v/row
    D = np.sqrt(u**2+v**2)
```

```
    H = np.zeros_like(D)
    H[D <= D0] = 1
    img_fft_shift_filtered = img_fft_shift*H
    img_fft_filtered = np.fft.ifftshift(img_fft_shift_filtered)
    img_filtered = np.fft.ifft2(img_fft_filtered)
    img_filtered = np.abs(img_filtered)
    return img_filtered
def butterworth_lowpass_filter(img, D0, n):
    img_double = np.double(img)
    row, col = img_double.shape
    img_fft = np.fft.fft2(img_double)
    img_fft_shift = np.fft.fftshift(img_fft)
    u, v = np.meshgrid(np.arange(col), np.arange(row))
    u = u-(1+col)/2
    v = v-(1+row)/2
    u = u/col
    v = v/row
    D = np.sqrt(u**2+v**2)
    H = np.zeros_like(D)
    H = 1/(1+(D/D0)**(2*n))
    img_fft_shift_filtered = img_fft_shift*H
    img_fft_filtered = np.fft.ifftshift(img_fft_shift_filtered)
    img_filtered = np.fft.ifft2(img_fft_filtered)
    img_filtered = np.abs(img_filtered)
    return img_filtered
def gaussian_lowpass_filter(img, D0):
    img_double = np.double(img)
    row, col = img_double.shape
    img_fft = np.fft.fft2(img_double)
```

```
    img_fft_shift = np.fft.fftshift(img_fft)
    u, v = np.meshgrid(np.arange(col), np.arange(row))
    u = u-(1+col)/2
    v = v-(1+row)/2
    u = u/col
    v = v/row
    D = np.sqrt(u**2+v**2)
    H = np.zeros_like(D)
    H = np.exp(-(D**2)/(2*(D0**2)))
    img_fft_shift_filtered = img_fft_shift*H
    img_fft_filtered = np.fft.ifftshift(img_fft_shift_filtered)
    img_filtered = np.fft.ifft2(img_fft_filtered)
    img_filtered = np.abs(img_filtered)
    return img_filtered
def make_mosaic(img):
    row, col = img.shape[:2]
    mr = np.zeros((row, col))
    mg = np.zeros((row, col))
    mb = np.zeros((row, col))
    mr[::2, ::2] = 1
    mb[1::2, 1::2] = 1
    mg = 1 - mr - mb
    red = img[:, :, 2]
    green = img[:, :, 1]
    blue = img[:, :, 0]
    red = np.multiply(red.astype(float), mr)
    green = np.multiply(green.astype(float), mg)
    blue = np.multiply(blue.astype(float), mb)
    # multi_img = np.zeros((row, col, 3), dtype=np.uint8)
```

```
        # multi_img[:, :, 0] = blue.astype(np.uint8)
        # multi_img[:, :, 1] = green.astype(np.uint8)
        # multi_img[:, :, 2] = red.astype(np.uint8)
        mosaic_image = red+green+blue
        return mosaic_image
    I = cv2.imread("Image_processing/image/Kodak/22.png")
    I_mosaic = make_mosaic(I)
    I_filtered_by_ilpf = ideal_lowpass_filter(I_mosaic, 0.1)
    I_filtered_by_butterworth = butterworth_lowpass_filter(I_mosaic, 0.1, 2)
    I_filtered_by_gaussian = gaussian_lowpass_filter(I_mosaic, 0.1)
    plt.figure()
    plt.subplot(2, 2, 1)
    plt.imshow(I_mosaic, cmap="gray"), plt.title("Mosaic image"), plt.axis("off"), plt.xticks([]), plt.yticks([])
    plt.subplot(2, 2, 2)
    plt.imshow(I_filtered_by_ilpf, cmap="gray"), plt.title("Ideal lowpass filter"), plt.axis("off"), plt.xticks([]), plt.yticks([])
    plt.subplot(2, 2, 3)
    plt.imshow(I_filtered_by_butterworth, cmap="gray"), plt.title("Butterworth lowpass filter"), plt.axis("off"), plt.xticks([]), plt.yticks([])
    plt.subplot(2, 2, 4)
    plt.imshow(I_filtered_by_gaussian, cmap="gray"), plt.title("Gaussian lowpass filter"), plt.axis("off"), plt.xticks([]), plt.yticks([])
    plt.show()
```

KẾT QUẢ



Hình . Kết quả

NHẬN XÉT

- Nếu ảnh mosaic chứa các chi tiết nhỏ nhưng không có nhiễu thì nên sử dụng bộ lọc thông thấp lý tưởng hoặc Butterworth với n=2 hoặc n=3 để giảm thiểu hiện tượng răng cưa và lọc các tần số cao không mong muốn.
- Nếu ảnh mosaic chứa nhiễu Gaussian thì nên sử dụng bộ lọc Gaussian để giảm thiểu nhiễu và lọc các tần số cao không mong muốn. D0 được chọn bằng giá trị độ rộng của hàm Gaussian tương ứng với mức độ mờ muốn của ảnh.
- Nếu ảnh mosaic chứa nhiễu muối tiêu hoặc tiếng động nhiễu khác thì nên sử dụng bộ lọc Butterworth với n>3 để lọc các tần số cao không mong muốn và giảm thiểu nhiễu.

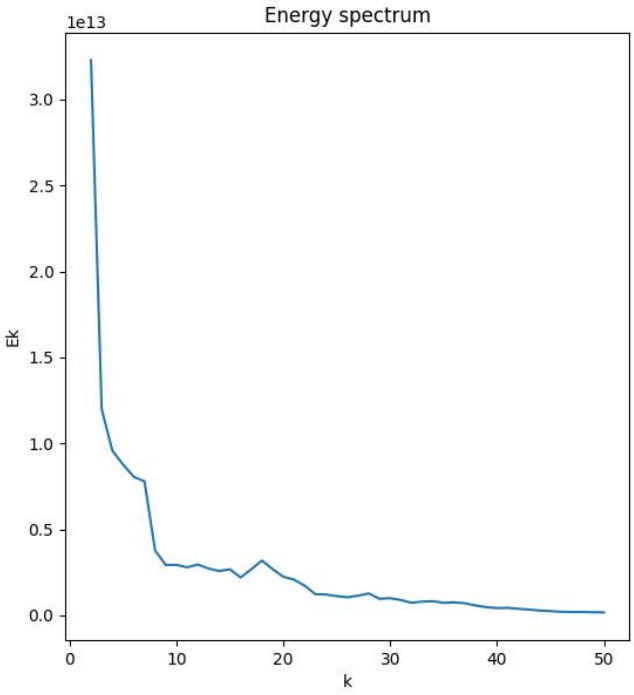
c. BT3: gọi I là một ảnh xám. SI=DFT{I}. Trong miền tần số chuẩn hóa, ta thiết kế bộ lọc $H_k(u,v) = \begin{cases} 1 & R_{k-1} \le D(u,v) \le R_k \\ 0 & \neq \end{cases}$. Tính S_k

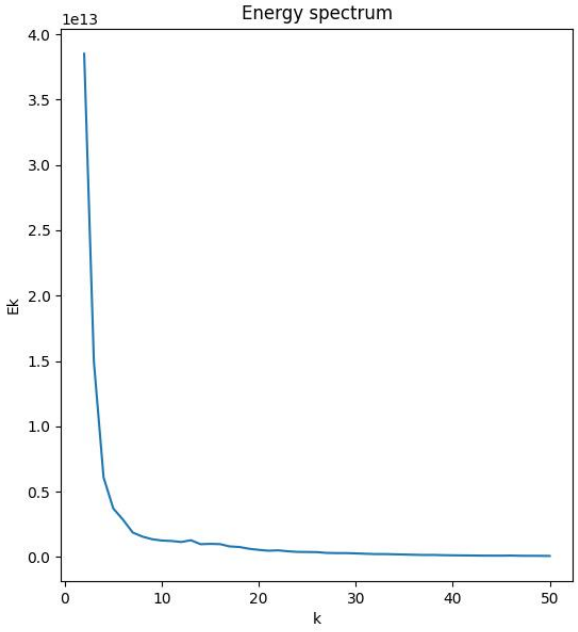
SOURCE CODE

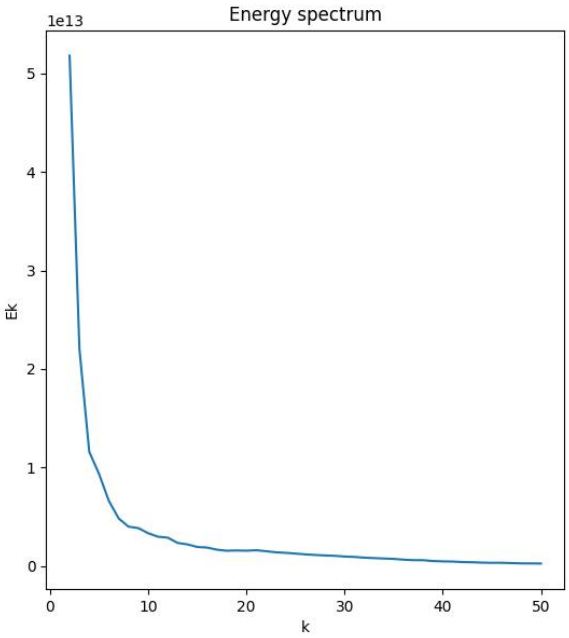
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread("Image_processing/image/Kodak/19.png", 0)
img_double = img.astype(np.float64)
row, col = img_double.shape
si = np.fft.fft2(img_double)
si_shift = np.fft.fftshift(si)
u, v = np.meshgrid(np.arange(col), np.arange(row))
u = u-(1+col)/2
v = v-(1+row)/2
u = u/col
v = v/row
D = np.sqrt(u**2+v**2)
N = 50
Ek = []
for k in range(2, N+1):
    H = np.zeros_like(D)
    R_k = 0.5*k/N
    R_k_1 = 0.5*(k-1)/N
    H[(R_k_1 <= D) & (D < R_k)] = 1
    si_shift_filtered = si_shift*H
    sum = 0
    for i in range(row):
        for j in range(col):
            sum += np.abs(si_shift_filtered[i, j])**2
```

```
    Ek.append(sum)
print(Ek)
plt.plot(Ek)
plt.show()
```

KẾT QUẢ







NHẬN XÉT

- Biến E_k trong đoạn mã trên là danh sách các giá trị năng lượng tương ứng với mỗi bộ lọc H_k. Mỗi giá trị năng lượng được tính bằng cách tính tổng bình phương độ lớn của các hệ số Fourier sau khi áp dụng bộ lọc tương ứng lên miền tần số.
- Các giá trị E_k cho thấy mức độ thông tin bị lọc đi khi áp dụng các bộ lọc thông qua băng H_k lên hình ảnh gốc. Nếu E_k giảm rõ rệt ở một khoảng k tương đối nhỏ, điều đó có thể cho thấy rằng các tần số nằm trong khoảng đó đóng góp mạnh mẽ vào hình ảnh.
- Từ đó, ta có thể điều chỉnh các bộ lọc thông qua băng H_k để giữ lại các tần số quan trọng hơn trong miền tần số.

12. Buổi 12 (06-04-2023)

ĐỀ BÀI

Ví dụ 1: Detection and Segmentation

1) Cho ảnh I sau: E:\SOURCE_CODE\Image_processing\image\test.png

Bộ lọc H=[-1 1]

Tìm Y1=I*H

Tìm Y2=I*H' <<<<< H' là ma trận chuyển vị của H

Hiển thị Y1, Y2, |Y1| và |Y2|. Nhận xét

2) Làm lại bài 1 với H=[1 -2 1]. Nhận xét về tổng các hệ số của H, giải thích

3) Thêm nhiễu vào ảnh I như sau I <- I+sigma*randn(size(I)). tự chọn sigma, ví dụ sigma=3, sigma=5, Làm lại bài 1 và 2. Nhận xét

4) làm lại bài 3 với "cameraman.tif" (có sẵn trong matlab hoặc google). xét sigma=0, sigma=5

5) I la ảnh cameraman.tif. Thêm nhiễu vào ảnh I như trên.

H=[-1 0 1]

G=[-1 0 1

-2 0 2

-1 0 1]

Hiển thị Y1=I*H, Y2=I*G, Y3=|Y1|+|Y2|. Tương tự với bộ lọc G. So sánh và giải thích

SOURCE CODE

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
# bài 1
img = cv2.imread('E:/SOURCE_CODE/Image_processing/image/test.png', 0)
img = img.astype(np.float64)
H1 = np.array([[-1, 1]])
Y1 = cv2.filter2D(img, -1, H1)
H1_T = np.transpose(H1)
Y2 = cv2.filter2D(img, -1, H1_T)
Y1_abs = np.abs(Y1)
Y2_abs = np.abs(Y2)
plt.figure
plt.subplot(2, 2, 1), plt.imshow(Y1, cmap='gray'), plt.title(
   'Y1'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(Y2, cmap='gray'), plt.title(
```

```
    'Y2'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(Y1_abs, cmap='gray'), plt.title(
    'Y1_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(Y2_abs, cmap='gray'), plt.title(
    'Y2_abs'), plt.xticks([]), plt.yticks([])
# plt.show()
# Bài 2:
H2 = np.array([[1, -2, 1]])
Y_H2 = cv2.filter2D(img, -1, H2)
Y_H2_abs = np.abs(Y_H2)
H2_T = np.transpose(H2)
Y_H2_T = cv2.filter2D(img, -1, H2_T)
Y_H2_T_abs = np.abs(Y_H2_T)
plt.figure
plt.subplot(2, 2, 1), plt.imshow(Y_H2, cmap='gray'), plt.title(
    'Y_H2'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 2), plt.imshow(Y_H2_abs, cmap='gray'), plt.title(
    'Y_H2_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 3), plt.imshow(Y_H2_T, cmap='gray'), plt.title(
    'Y_H2_T'), plt.xticks([]), plt.yticks([])
plt.subplot(2, 2, 4), plt.imshow(Y_H2_T_abs, cmap='gray'), plt.title(
    'Y_H2_T_abs'), plt.xticks([]), plt.yticks([])
# plt.show()
#  Nhận xét bài 2:
# tại sao tổng hệ số của H lại bằng 0? : loại bỏ thành phần dc, lọc tần số cao
# Bài 3:
img_noise = img.astype(np.float64) + 3*np.random.randn(*img.shape)
Y1_noise = cv2.filter2D(img_noise, -1, H1)
Y1_noise_abs = np.abs(Y1_noise)
```

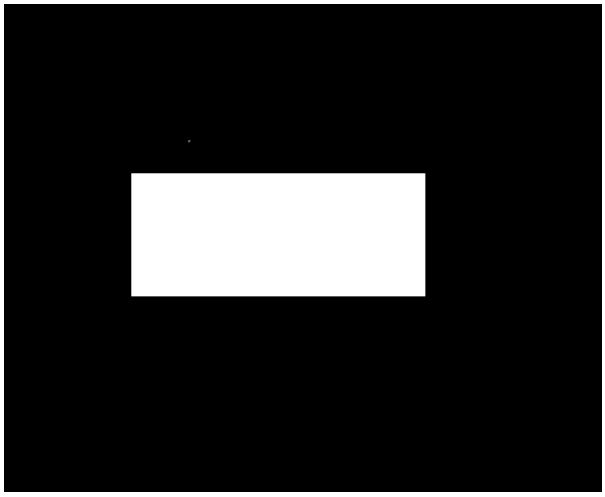
```
Y1_T_noise = cv2.filter2D(img_noise, -1, H1_T)
Y1_T_noise_abs = np.abs(Y1_T_noise)
Y2_noise = cv2.filter2D(img_noise, -1, H2)
Y2_noise_abs = np.abs(Y2_noise)
Y2_T_noise = cv2.filter2D(img_noise, -1, H2_T)
Y2_T_noise_abs = np.abs(Y2_T_noise)
plt.figure
plt.subplot(4, 2, 1), plt.imshow(Y1_noise, cmap='gray'), plt.title(
    'Y1_noise'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 2), plt.imshow(Y1_noise_abs, cmap='gray'), plt.title(
    'Y1_noise_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 3), plt.imshow(Y1_T_noise, cmap='gray'), plt.title(
    'Y1_T_noise'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 4), plt.imshow(Y1_T_noise_abs, cmap='gray'), plt.title(
    'Y1_T_noise_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 5), plt.imshow(Y2_noise, cmap='gray'), plt.title(
    'Y2_noise'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 6), plt.imshow(Y2_noise_abs, cmap='gray'), plt.title(
    'Y2_noise_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 7), plt.imshow(Y2_T_noise, cmap='gray'), plt.title(
    'Y2_T_noise'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 8), plt.imshow(Y2_T_noise_abs, cmap='gray'), plt.title(
    'Y2_T_noise_abs'), plt.xticks([]), plt.yticks([])
# plt.show()
# Bài 4:
img_cameraman = cv2.imread(
    'E:/SOURCE_CODE/Image_processing/image/cameraman.tif', 0)
img_cameraman = img_cameraman.astype(np.float64)
img_cameraman_noise_sigma_0 = img_cameraman + \
```

```
    0*np.random.randn(*img_cameraman.shape)
img_cameraman_noise_sigma_5 = img_cameraman.astype(
    np.float64) + 5*np.random.randn(*img_cameraman.shape)
Y1_cameraman_noise_sigma_0 = cv2.filter2D(img_cameraman_noise_sigma_0, -1, H1)
Y1_cameraman_noise_sigma_0_abs = np.abs(Y1_cameraman_noise_sigma_0)
Y1_T_cameraman_noise_sigma_0 = cv2.filter2D(
    img_cameraman_noise_sigma_0, -1, H1_T)
Y1_T_cameraman_noise_sigma_0_abs = np.abs(Y1_T_cameraman_noise_sigma_0)
Y2_cameraman_noise_sigma_0 = cv2.filter2D(img_cameraman_noise_sigma_0, -1, H2)
Y2_cameraman_noise_sigma_0_abs = np.abs(Y2_cameraman_noise_sigma_0)
Y2_T_cameraman_noise_sigma_0 = cv2.filter2D(
    img_cameraman_noise_sigma_0, -1, H2_T)
Y2_T_cameraman_noise_sigma_0_abs = np.abs(Y2_T_cameraman_noise_sigma_0)
plt.figure
plt.subplot(4, 2, 1), plt.imshow(Y1_cameraman_noise_sigma_0,
                        cmap='gray'), plt.title('Y1_sigma_0'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 2), plt.imshow(Y1_cameraman_noise_sigma_0_abs,
                        cmap='gray'), plt.title('Y1sigma_0_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 3), plt.imshow(Y1_T_cameraman_noise_sigma_0,
                        cmap='gray'), plt.title('Y1_T_sigma_0'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 4), plt.imshow(Y1_T_cameraman_noise_sigma_0_abs,
                        cmap='gray'), plt.title('Y1_T_sigma_0_abs'), plt.xticks([]),
plt.yticks([])
plt.subplot(4, 2, 5), plt.imshow(Y2_cameraman_noise_sigma_0,
                        cmap='gray'), plt.title('Y2_sigma_0'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 6), plt.imshow(Y2_cameraman_noise_sigma_0_abs,
                        cmap='gray'), plt.title('Y2_sigma_0_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 7), plt.imshow(Y2_T_cameraman_noise_sigma_0,
                        cmap='gray'), plt.title('Y2_T_sigma_0'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 8), plt.imshow(Y2_T_cameraman_noise_sigma_0_abs,
```

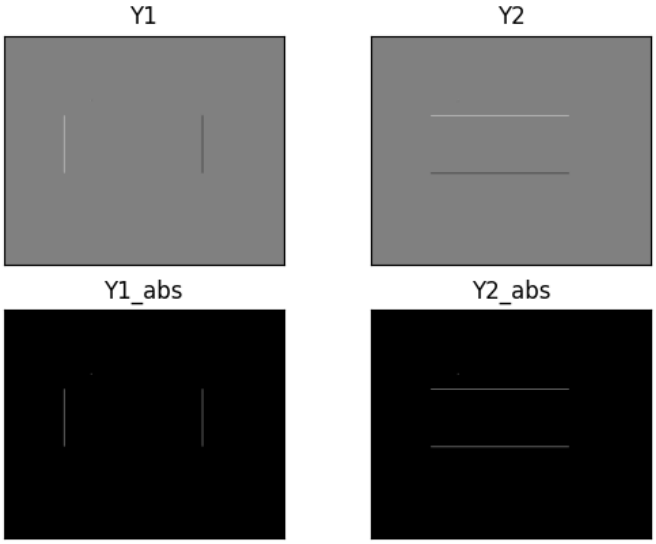
```
                    cmap='gray'), plt.title('Y2_T_sigma_0_abs'), plt.xticks([]),
plt.yticks([])
# plt.show()
# sigma = 5
Y1_cameraman_noise_sigma_5 = cv2.filter2D(img_cameraman_noise_sigma_5, -1, H1)
Y1_cameraman_noise_sigma_5_abs = np.abs(Y1_cameraman_noise_sigma_5)
Y1_T_cameraman_noise_sigma_5 = cv2.filter2D(
   img_cameraman_noise_sigma_5, -1, H1_T)
Y1_T_cameraman_noise_sigma_5_abs = np.abs(Y1_T_cameraman_noise_sigma_5)
Y2_cameraman_noise_sigma_5 = cv2.filter2D(img_cameraman_noise_sigma_5, -1, H2)
Y2_cameraman_noise_sigma_5_abs = np.abs(Y2_cameraman_noise_sigma_5)
Y2_T_cameraman_noise_sigma_5 = cv2.filter2D(
   img_cameraman_noise_sigma_5, -1, H2_T)
Y2_T_cameraman_noise_sigma_5_abs = np.abs(Y2_T_cameraman_noise_sigma_5)
plt.figure
plt.subplot(4, 2, 1), plt.imshow(Y1_cameraman_noise_sigma_5,
                    cmap='gray'), plt.title('Y1_sigma_5'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 2), plt.imshow(Y1_cameraman_noise_sigma_5_abs,
                    cmap='gray'), plt.title('Y1sigma_5_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 3), plt.imshow(Y1_T_cameraman_noise_sigma_5,
                    cmap='gray'), plt.title('Y1_T_sigma_5'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 4), plt.imshow(Y1_T_cameraman_noise_sigma_5_abs,
                    cmap='gray'), plt.title('Y1_T_sigma_5_abs'), plt.xticks([]),
plt.yticks([])
plt.subplot(4, 2, 5), plt.imshow(Y2_cameraman_noise_sigma_5,
                    cmap='gray'), plt.title('Y2_sigma_5'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 6), plt.imshow(Y2_cameraman_noise_sigma_5_abs,
                    cmap='gray'), plt.title('Y2_sigma_5_abs'), plt.xticks([]), plt.yticks([])
plt.subplot(4, 2, 7), plt.imshow(Y2_T_cameraman_noise_sigma_5,
                    cmap='gray'), plt.title('Y2_T_sigma_5'), plt.xticks([]), plt.yticks([])
```

```
plt.subplot(4, 2, 8), plt.imshow(Y2_T_cameraman_noise_sigma_5_abs,
                                 cmap='gray'), plt.title('Y2_T_sigma_5_abs'), plt.xticks([]),
plt.yticks([])
plt.show()
# Bài 5
I = cv2.imread('E:/SOURCE_CODE/Image_processing/image/cameraman.tif', 0)
I_noise = I.astype(np.float64)+5*np.random.normal(0, 5, I.shape)
H = np.array([[-1, 0, 1]])
G = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
Y1_H = cv2.filter2D(I_noise, -1, H)
Y2_H = cv2.filter2D(I_noise, -1, np.transpose(H))
Y3_H = np.abs(Y1_H)+np.abs(Y2_H)
Y1_G = cv2.filter2D(I_noise, -1, G)
Y2_G = cv2.filter2D(I_noise, -1, np.transpose(G))
Y3_G = np.abs(Y1_G)+np.abs(Y2_G)
plt.figure
plt.subplot(231), plt.imshow(Y1_H, cmap='gray'), plt.title(
   'Y1_H'), plt.xticks([]), plt.yticks([])
plt.subplot(232), plt.imshow(Y2_H, cmap='gray'), plt.title(
   'Y2_H'), plt.xticks([]), plt.yticks([])
plt.subplot(233), plt.imshow(Y3_H, cmap='gray'), plt.title(
   'Y3_H'), plt.xticks([]), plt.yticks([])
plt.subplot(234), plt.imshow(Y1_G, cmap='gray'), plt.title(
   'Y1_G'), plt.xticks([]), plt.yticks([])
plt.subplot(235), plt.imshow(Y2_G, cmap='gray'), plt.title(
   'Y2_G'), plt.xticks([]), plt.yticks([])
plt.subplot(236), plt.imshow(Y3_G, cmap='gray'), plt.title(
   'Y3_G'), plt.xticks([]), plt.yticks([])
plt.show()
```

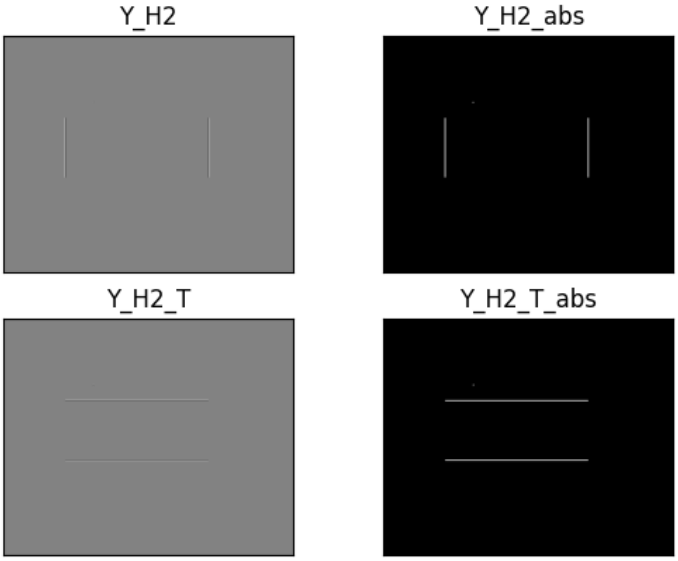
KẾT QUẢ



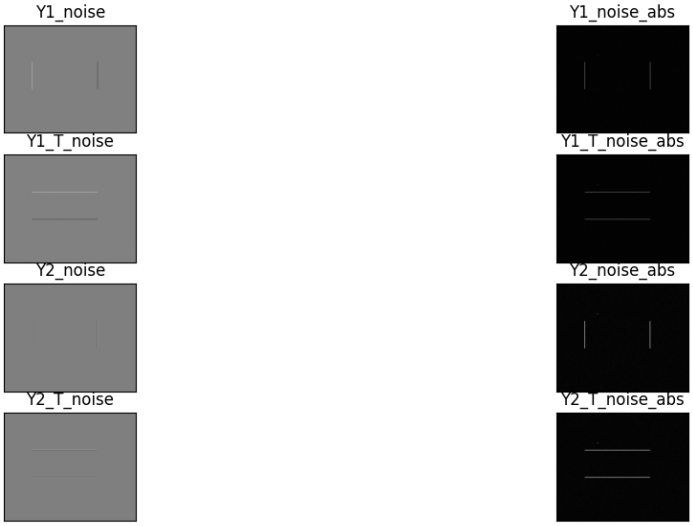
Hình . ảnh test bài 1



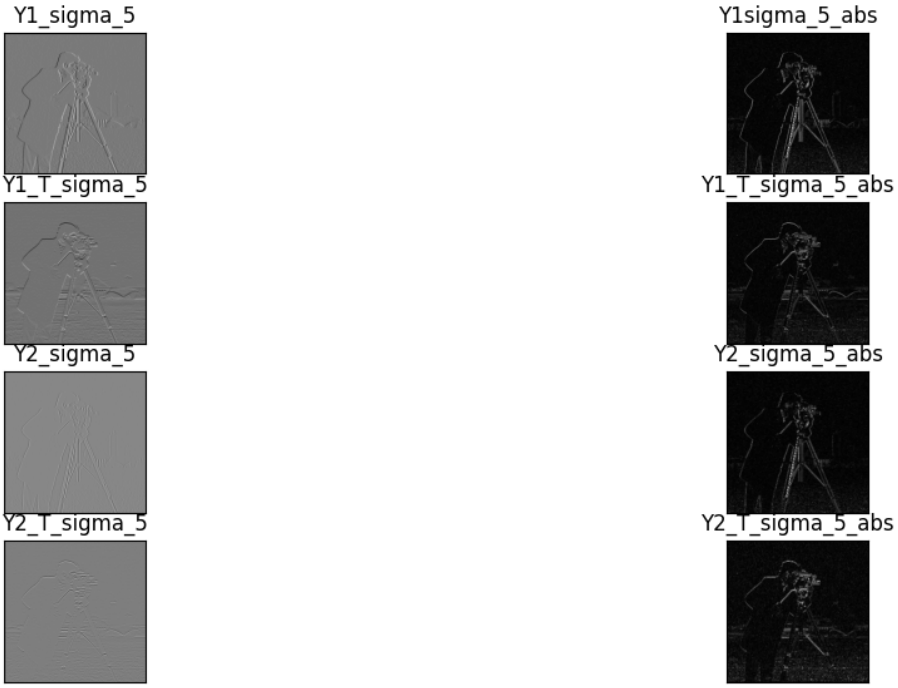
Hình . Kết quả bài 1



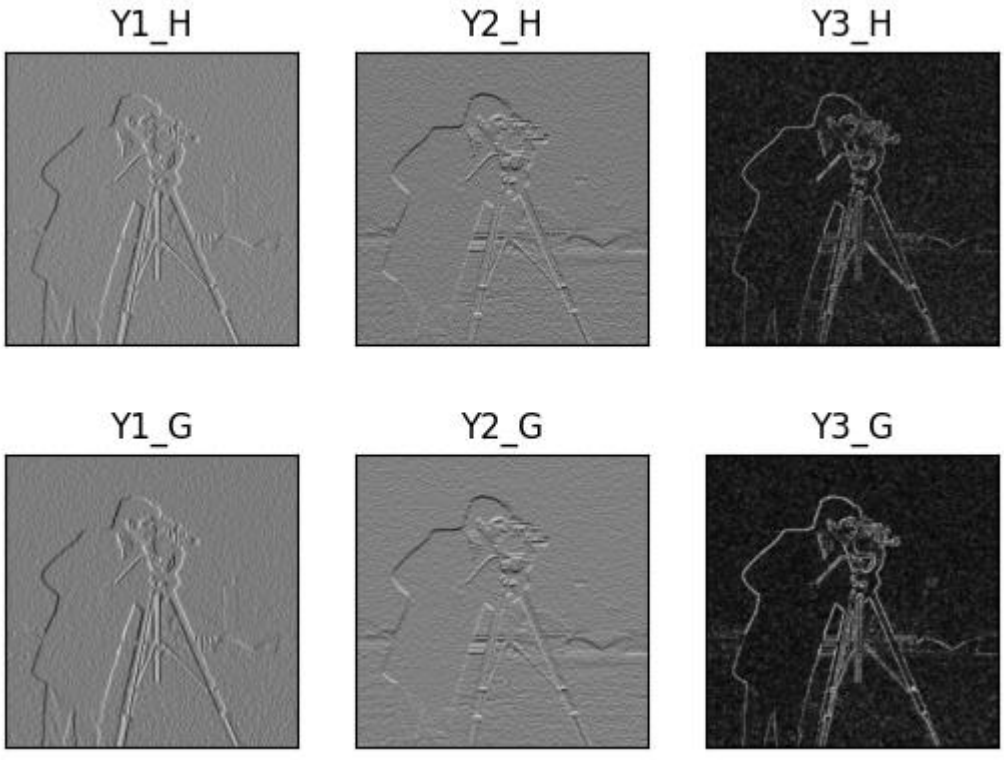
Hình . Kết quả bài 2



Hình . Kết quả bài 3



Hình . Kết quả bài 4



Hình . Kết quả bài 5

NHẬN XÉT

- H và G đều lọc theo ngang, nên lọc theo đường thẳng đứng, ta có thể dùng chuyển vị của H và G
- Khi có nhiễu, G lọc tốt hơn, vì G có thêm 2 hàng nên có thể lọc được nhiễu ở 2 chiều

13. Buổi 13 (13-04-2023)

a. BT1: thực hiện bài toán Segmentation trên ảnh màu bằng phương pháp Kmeans Clustering

SOURCE CODE

```
import numpy as np
import matplotlib.pyplot as plt
import cv2
def kmeans(X, K, max_iters=10):
    centroids = X[np.random.choice(range(len(X)), K, replace=False)]
    for i in range(max_iters):
        # Assign each data point to its closest centroid
        distances = np.linalg.norm(X[:, np.newaxis, :] - centroids, axis=-1)
        labels = np.argmin(distances, axis=-1)
        # Update the centroids based on the assigned data points
        for j in range(K):
            centroids[j] = np.mean(X[labels == j], axis=0)
    return labels, centroids
# Đọc ảnh màu
original_img = cv2.imread("Image_processing/image/Kodak/23.png")
original_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2RGB)
# Chuyen sang khong gian mau HSV
HSV_img = cv2.cvtColor(original_img, cv2.COLOR_BGR2HSV)
# Vector hóa ảnh
vectorized = HSV_img.reshape((-1, 3))
vectorized = np.float32(vectorized)
# Chạy K-means với K=10
```

```
K = 15
labels, centroids = kmeans(vectorized, K)
# Gán nhãn cho từng pixel và lấy giá trị trung bình của mỗi cluster để tạo ảnh kết quả
centroids = np.uint8(centroids)
res = centroids[labels.flatten()]
result_image = res.reshape((HSV_img.shape))
result_image = cv2.cvtColor(result_image, cv2.COLOR_HSV2BGR)
# Hiển thị ảnh gốc và ảnh phân vùng
figure_size = 15
plt.figure(figsize=(figure_size, figure_size))
plt.subplot(1, 2, 1), plt.imshow(original_img)
plt.title('Original Image'), plt.xticks([]), plt.yticks([])
plt.subplot(1, 2, 2), plt.imshow(result_image)
plt.title('Segmented Image when K = %i' % K), plt.xticks([]), plt.yticks([])
plt.show()
```

KẾT QUẢ

Hình . Kết quả

b. BT2: tìm hiểu 1 phương pháp thực hiện bài toán segmentation khác ngoài Kmeans

PHƯƠNG PHÁP MEANSHIFT CLUSTERING

– Meanshift Clustering là một kỹ thuật phân cụm phi tham số, không yêu cầu chỉ định số lượng cụm. Ngoài ra, nó rất mạnh đối với các ngoại lệ vì các cụm không có dạng hình cầu mà nó có hình dạng phi tuyến tính theo quy trình phân cụm.

– Thuật toán:

Extract feature space from image

While number of unvisited points > 0

Select a random point in feature space (Initial mean)

While true:

Get distance between mean and all points in feature space

For uniform window, select points in the range of specified bandwidth and track that points

Get the new mean, it is the mean of points within bandwidth

if distance between new and old means < threshold:

for c in clusters:

#Check merge condition

if distance(c, center) < 0.5* Bandwidth:

mean of cluster c = 0.5*distance(c,center)

cluster all tracked points to cluster c

#No merge

cluster all tracked points to new mean

#Update visited points

update number of visited points

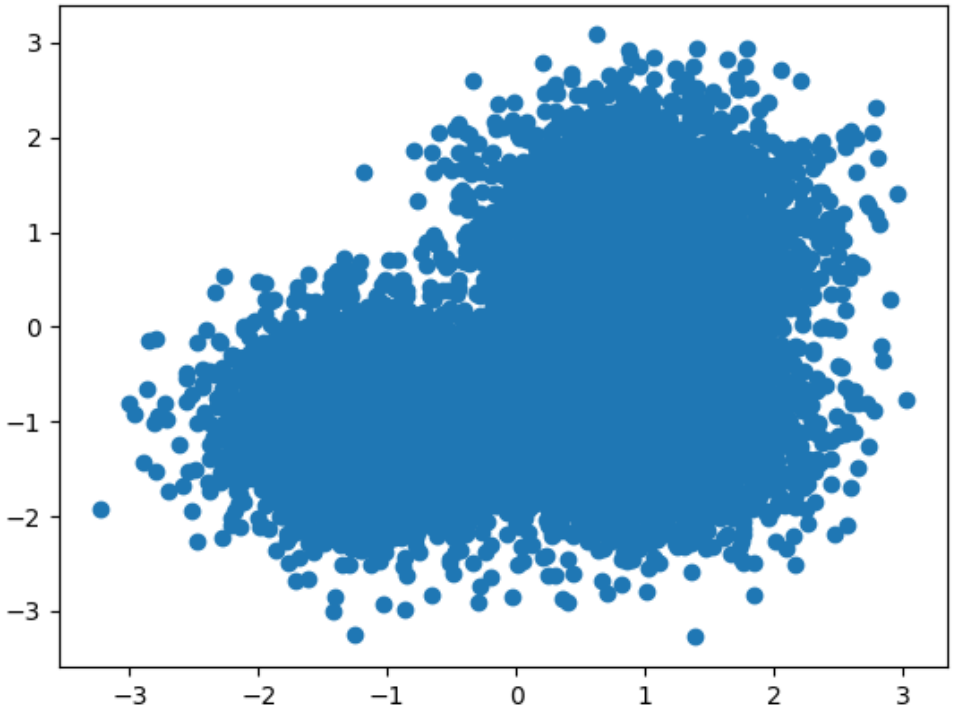
break

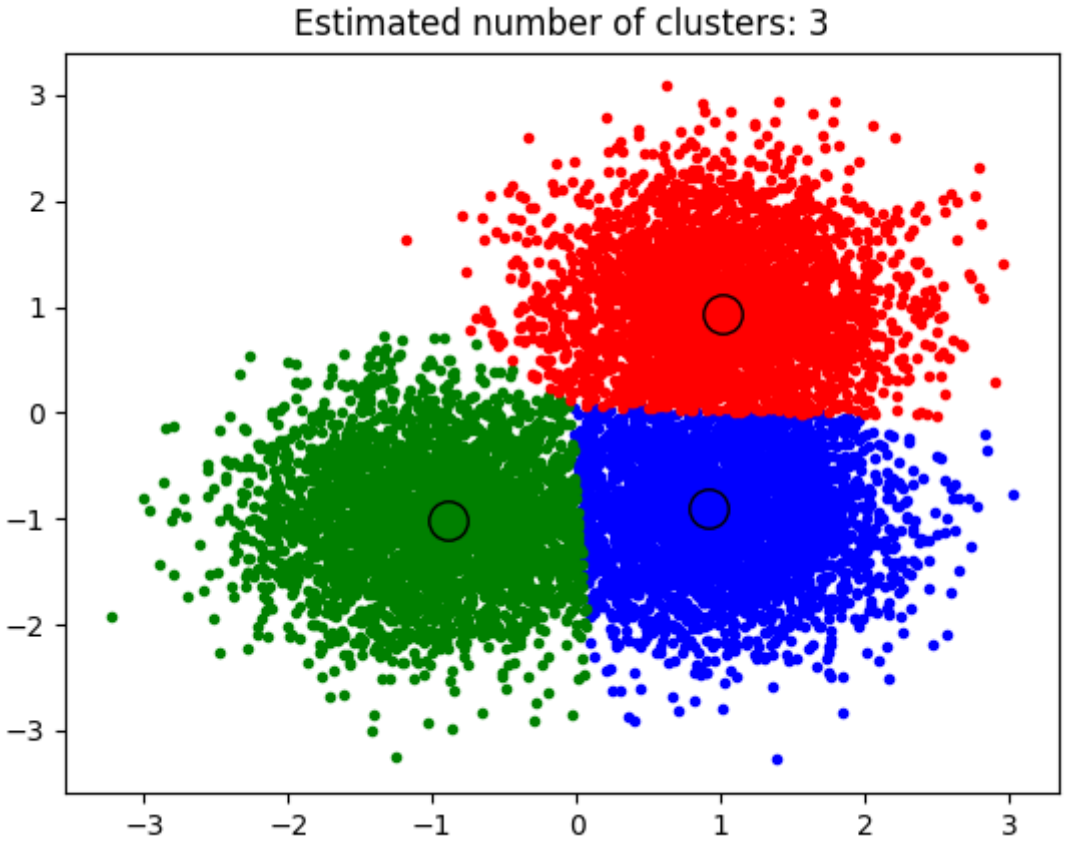
SOURCE CODE:

```
# Meanshift CLustering

import numpy as np
```

```
from sklearn.cluster import MeanShift, estimate_bandwidth
from sklearn.datasets import make_blobs
import matplotlib.pyplot as plt
from itertools import cycle
centers = [[1, 1], [-1, -1], [1, -1]]  # 3 centers
X, _ = make_blobs(n_samples=10000, centers=centers, cluster_std=0.6)
plt.scatter(X[:, 0], X[:, 1])
plt.show()
bandwidth = estimate_bandwidth(X, quantile=0.2, n_samples=500)
ms = MeanShift(bandwidth=bandwidth, bin_seeding=True)
ms.fit(X)
labels = ms.labels_
cluster_centers = ms.cluster_centers_
labels_unique = np.unique(labels)
n_clusters_ = len(labels_unique)
print("number of estimated clusters : %d" % n_clusters_)
plt.figure(1)
plt.clf()
colors = cycle('bgrcmykbgrcmykbgrcmykbgrcmyk')
for k, col in zip(range(n_clusters_), colors):
    my_members = labels == k
    cluster_center = cluster_centers[k]
    plt.plot(X[my_members, 0], X[my_members, 1], col + '.')
    plt.plot(cluster_center[0], cluster_center[1], 'o', markerfacecolor=col,
             markeredgecolor='k', markersize=14)
plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

KẾT QUẢ:


3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3


Estimated number of clusters: 3
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

14. **Buổi 14 (20-04-2023)**
15. **Buổi 15 (27-04-2023)**

ĐỀ BÀI: tìm hiểu nhận dạng biển số xe trong ảnh

SOURCE CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure
I = cv2.imread("Image_processing/image/bienso2.jpg")
cv2.imshow("anh goc", I)
cv2.waitKey(0)
Igray = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
rows, cols = Igray.shape
# cv2.imshow("anh xam", Igray)
# cv2.waitKey(0)
# Dilate and Erode
kernel = np.ones((3, 3), np.uint8)
Iopen = cv2.morphologyEx(Igray, cv2.MORPH_OPEN, kernel, iterations=1)
# cv2.imshow("anh open", Iopen)
# cv2.waitKey(0)
difference = 0
sum = 0
totel_sum = 0
difference = np.uint32(difference)
# process edges in horizontal direction
max_horz = 0
maximum = 0
horz1 = np.zeros((cols, 1), np.uint32)
for i in range(1, cols):
```

```
    sum = 0
    for j in range(1, rows):
        if Iopen[j, i] > Iopen[j-1, i]:
            difference = np.uint32(Iopen[j, i]-Iopen[j-1, i])
        else:
            difference = np.uint32(Iopen[j-1, i]-Iopen[j, i])
        if difference > 20:
            sum = sum+difference
    horz1[i] = sum
    # find peak value
    if sum > maximum:
        max_horz = i
        maximum = sum
    totel_sum = totel_sum+sum
average = totel_sum/cols
# plt.subplot(2, 1, 1)
# plt.plot(horz1)
# plt.title('Horizontal Edge Processing Histogram')
# plt.xlabel('Column Number ->')
# plt.ylabel('Difference ->')
# plt.show()
# Smoothen the Horizontal Histogram by applying Low Pass Filter
sum = 0
horz = horz1
for i in range(20, cols-20):
    sum = 0
    for j in range(i-20, i+20):
        sum = sum + horz1[j]
    horz[i] = sum/40
```

```
# plt.subplot(2, 1, 2)
# plt.plot(horz)
# plt.title('Histogram after passing through Low Pass Filter')
# plt.xlabel('Column Number ->')
# plt.ylabel('Difference ->')
# plt.show()
# filter out horizontal histogram values by applying Dynamic Threshold
print("Filter out horizontal histogram values by applying Dynamic Threshold")
for i in range(0, cols):
    if horz[i] < average:
        horz[i] = 0
        for j in range(0, rows):
            Iopen[j, i] = 0
# plt.subplot(2, 1, 1)
# plt.plot(horz)
# plt.title('Histogram after Filtering')
# plt.xlabel('Column Number ->')
# plt.ylabel('Difference ->')
# plt.show()
# cv2.imshow("anh open", Iopen)
# cv2.waitKey(0)
# process edges in vertical direction
difference = 0
totel_sum = 0
difference = np.uint32(difference)
print("process edges in vertical direction")
maximum = 0
max_vert = 0
vert1 = np.zeros((rows, 1), np.uint32)
```

```
for i in range(1, rows):
    sum = 0
    for j in range(1, cols):
        if Iopen[i, j] > Iopen[i, j-1]:
            difference = np.uint32(Iopen[i, j]-Iopen[i, j-1])
        else:
            difference = np.uint32(Iopen[i, j-1]-Iopen[i, j])
        if difference > 20:
            sum = sum+difference
    vert1[i] = sum
    # find peak value
    if sum > maximum:
        max_vert = i
        maximum = sum
    totel_sum = totel_sum+sum
average = totel_sum/rows
# plt.subplot(2, 1, 1)
# plt.plot(vert1)
# plt.title('Vertical Edge Processing Histogram')
# plt.xlabel('Row Number ->')
# plt.ylabel('Difference ->')
# plt.show()
# smoothen the vertical Histogram by applying Low Pass Filter
print("smoothen the vertical Histogram by applying Low Pass Filter")
sum = 0
vert = vert1
for i in range(20, rows-20):
    sum = 0
    for j in range(i-20, i+20):
```

```
            sum = sum+vert1[j]
        vert[i] = sum/40
    # plt.subplot(2, 1, 2)
    # plt.plot(vert)
    # plt.title('Histogram after passing through Low Pass Filter')
    # plt.xlabel('Row Number ->')
    # plt.ylabel('Difference ->')
    # plt.show()
    # filter out vertical histogram values by applying Dynamic Threshold
    print("filter out vertical histogram values by applying Dynamic Threshold")
    for i in range(0, rows):
        if vert[i] < average:
            vert[i] = 0
            for j in range(0, cols):
                Iopen[i, j] = 0
    # plt.subplot(2, 1, 1)
    # plt.plot(vert)
    # plt.title('Histogram after Filtering')
    # plt.xlabel('Row Number ->')
    # plt.ylabel('Difference ->')
    # plt.show()
    # cv2.imshow("anh open", Iopen)
    # cv2.waitKey(0)
    # find probable candidates for number plate
    j = 0
    column = np.zeros((cols, 1), np.uint32)
    for i in range(1, cols-1):
        if horz[i] != 0 and horz[i-1] == 0 and horz[i+1] == 0:
            column[j] = i
```

```
            column[j+1] = i
            j = j+2
        elif (horz[i] != 0 and horz[i-1] == 0) or (horz[i] != 0 and horz[i+1] == 0):
            column[j] = i
            j = j+1
    j = 1
    row = np.zeros((rows, 1), np.uint32)
    for i in range(1, rows-1):
        if vert[i] != 0 and vert[i-1] == 0 and vert[i+1] == 0:
            row[j] = i
            row[j+1] = i
            j = j+2
        elif (vert[i] != 0 and vert[i-1] == 0) or (vert[i] != 0 and vert[i+1] == 0):
            row[j] = i
            j = j+1
    (temp, column_length) = column.shape
    if column_length % 2 != 0:
        column[column_length] = cols-1
    (temp, row_length) = row.shape
    if row_length % 2 != 0:
        row[row_length] = rows-1
    # check each and every probable candidate
    for i in range(0, row_length-1, 2):
        for j in range(0, column_length-1, 2):
            # crop the candidate
            img = Iopen[row[i]:row[i+1], column[j]:column[j+1]]
            # cv2.imshow("anh", img)
            # cv2.waitKey(0)
            # calculate the density of non-zero pixels in the candidate
```

```
        density = np.count_nonzero(img)
        # if the density is less than 20% then it is not a number plate
        # print(density)
        if density < ((row[i+1]-row[i])*(column[j+1]-column[j])*0.2):
            for m in range(row[i], row[i+1]):
                for n in range(column[j], column[j+1]):
                    Iopen[m, n] = 0
# plt.subplot(2, 1, 1)
# plt.imshow(Iopen)
# plt.title('Number Plate Candidate')
# plt.xlabel('Column Number ->')
# plt.ylabel('Row Number ->')
# plt.show()
# cv2.imshow("anh open", Iopen)
# cv2.waitKey(0)
image = Iopen
# Tìm vùng giữa 2 ngưỡng giá trị
contour, hier = cv2.findContours(
    image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
x, y, w, h = cv2.boundingRect(contour[0])
cv2.rectangle(I, (x, y), (x+w, y+h), (0, 255, 0), 2)
cv2.imshow("anh", I)
cv2.waitKey(0)
```

KẾT QUẢ

Hình . Ảnh gốc

Hình . Kết quả