

TRƯỜNG ĐẠI HỌC BÁCH KHOA
ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
KHOA ĐIỆN – ĐIỆN TỬ



HOMEWORK
THỊ GIÁC MÁY
GIẢNG VIÊN HƯỚNG DẪN: TS. PHẠM VIỆT CƯỜNG
SINH VIÊN THỰC HIỆN

STT	HỌ VÀ TÊN	MSSV	ĐÓNG GÓP
1	Nguyễn Duy Khang	2113665	100%
2	Nguyễn Lâm Anh Vũ	2112670	100%
3	Lê Thanh Tùng	2112613	100%
4	Đặng Nhật Nam	2111791	100%
5	Nguyễn Phú Tiền	2112444	100%
6	Võ Thành Khoa	2111547	100%

TP. HỒ CHÍ MINH

BẢNG PHÂN CÔNG NHIỆM VỤ

HỌ VÀ TÊN	HOMEWORK
Nguyễn Duy Khang	1,3,9
Nguyễn Lâm Anh Vũ	2
Lê Thanh Tùng	7
Đặng Nhật Nam	6,8
Nguyễn Phú Tiên	4
Võ Thành Khoa	5

MỤC LỤC

BẢNG PHÂN CÔNG NHIỆM VỤ	1
MỤC LỤC.....	2
1. Homework 1: Use Matlab/OpenCV functions.....	4
1.1 Làm quen với các lệnh xử lý ảnh trong matlab:	4
1.2 Tìm các lệnh tương ứng dùng OpenCV.....	28
2. Homework 2: Use built-in functions in MATLAB such as vision.CascadeObjectDetector to detect several objects like face, nose, mouth, and eyes, then derive metrics like precision, recall, and accuracy.	47
2.1 Thủ với model nhận diện khuôn mặt.....	47
2.2 Thủ với model nhận diện miệng.....	48
2.3 Thủ với model nhận diện mũi.....	49
2.4 Thủ với model nhận diện mắt.....	50
3. Homework 3: HW1: Viola-Jones algorithm: Training	52
4. Homework 4: Create a mind map for the Viola-Jones Algorithm.....	57
5. Homework 5: 2 questions: gradient vector, exploration - exploitation	59
A. How are exploration and exploitation related to Mini-batch Gradient Descent, Batch Gradient Descent, and Stochastic Gradient Descent?	59
6. Homework 6: ANN playground.....	62
7. Homework 7	75
A. Intuitively, receiving information from multiple sources seems better, so why does CNN use local connectivity?	75
B. In an ANN, each neuron has its own set of parameters to adjust, but why does CNN share the same set of parameters?.....	76

8. Homework 8 Alexnet: size & number of kernels, number of parameters	77
9. Homework 9	78
Alexnet: evaluation (top-1, top-5 error rate) with 100 images, 5 classes	78

1. Homework 1: Use Matlab/OpenCV functions

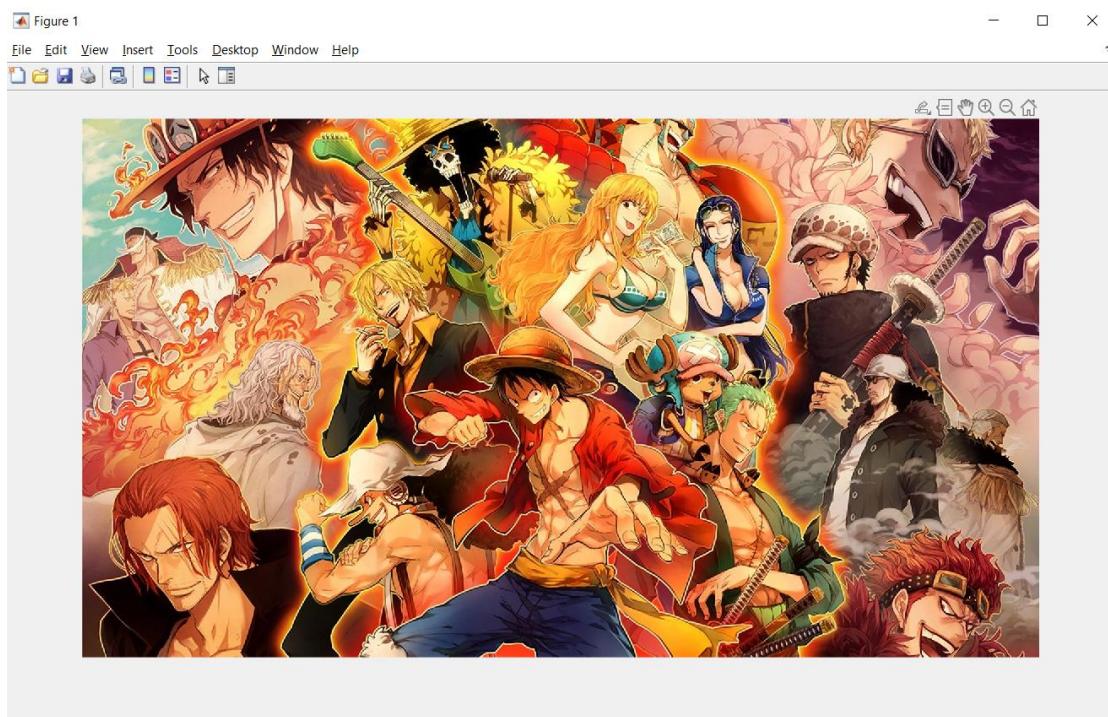
1.1 Làm quen với các lệnh xử lý ảnh trong matlab:

imread(filename): Đọc ảnh từ 1 tệp ảnh có sẵn trong WorkSpace

imshow(x): Hiển thị ảnh ra ngoài màn hình

```
A = imread('HW1_OnePiece.jpg')
```

```
imshow(A)
```

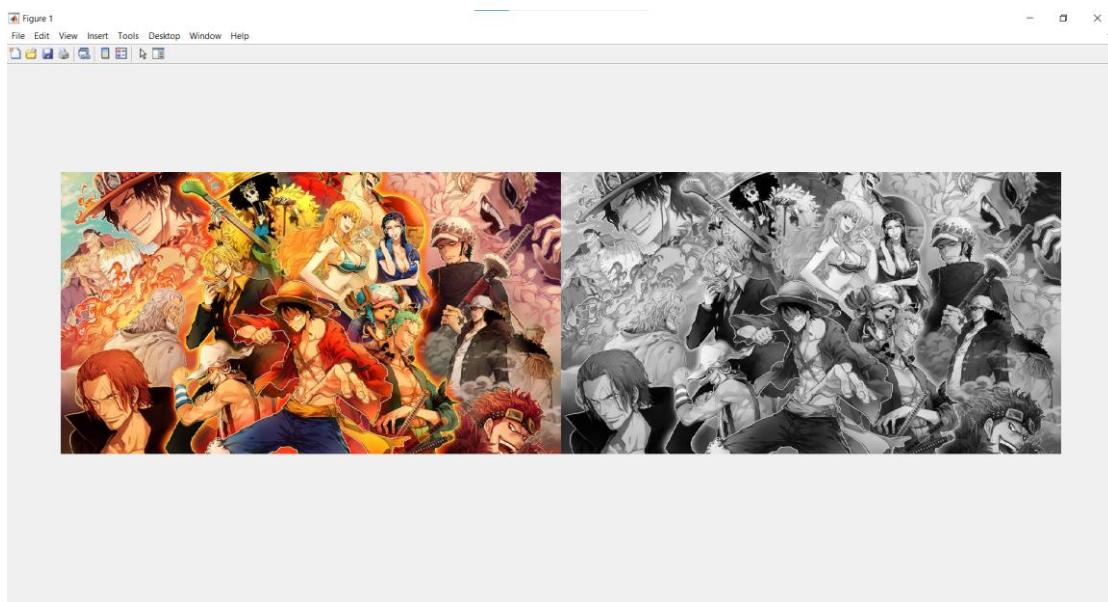


rgb2gray(RGB): Chuyển từ ảnh màu (RGB) sang ảnh xám (grayscale)

imshowpair(x, y, 'montage'): hiển thị nhiều ảnh cùng một lúc

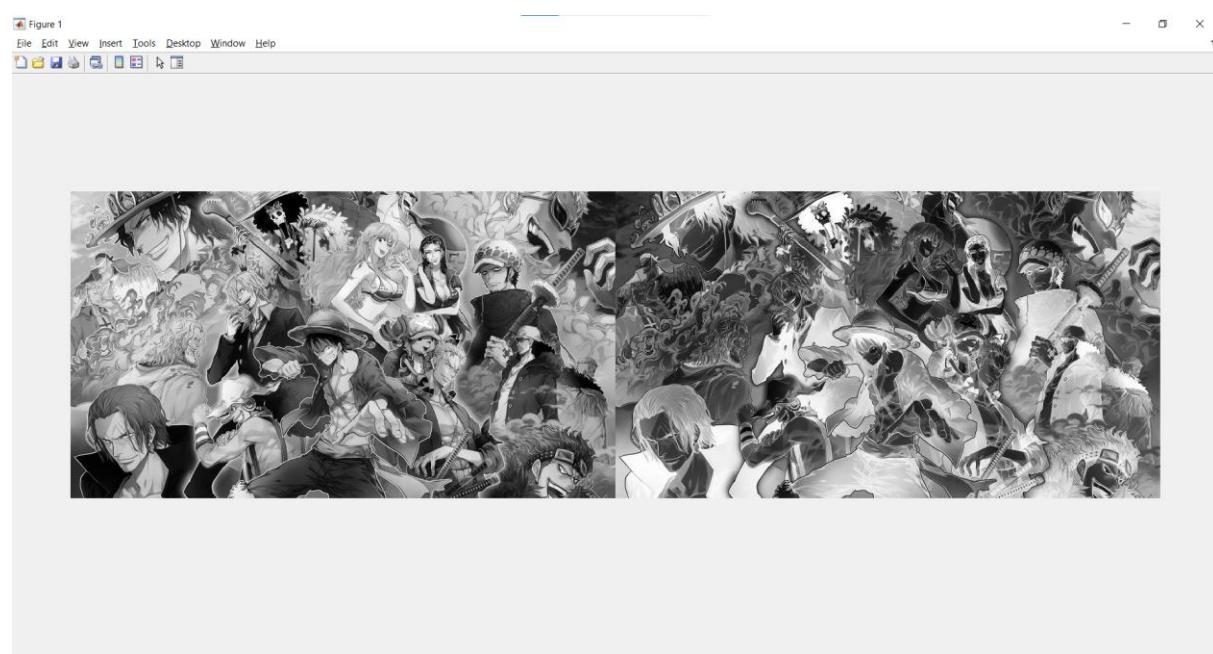
```
gray_x = rgb2gray(x)
```

```
imshowpair(x, gray_x, 'montage')
```



$\text{negative_}(x) = 255 - \text{gray}(x)$: chuyển ảnh màu xám thành ảnh âm bản

```
negative_A = 255 - gray_A  
imshowpair(gray_A, negative_A, 'montage')
```



Các lệnh liên quan đến histogram

✓ Hiển thị histogram của ảnh:

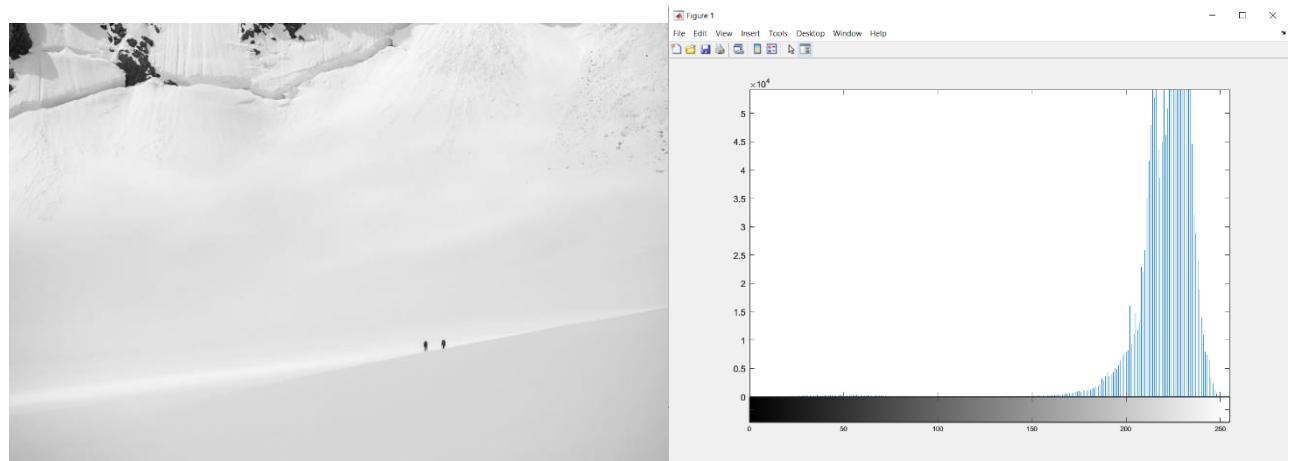
imhist(A, 100): hiển thị histogram của một hình ảnh có 100 vách ngăn

- Dark image: histogram bị lệch về phía bên trái
- Bright image: histogram bị lệch về phía bên tay phải.
- High contrast image: sử dụng toàn bộ dãy thông tin, histogram phân bố trải dài.
- Low contrast image: không sử dụng toàn bộ dãy thông tin, histogram chỉ tập trung 1 khoảng nhỏ.

→ Histogram cung cấp sự mô tả bên ngoài tấm ảnh một cách tổng quát.

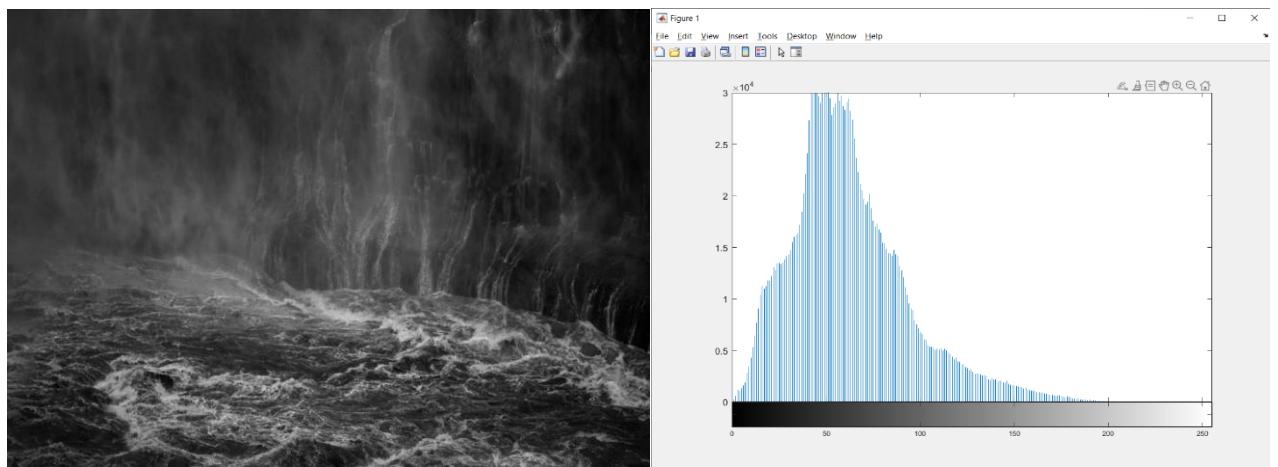
Ảnh sáng

```
A = imread('HW1_BrightImage.jpg')
imshow(A)
title('Bright Image')
imhist(A)
```



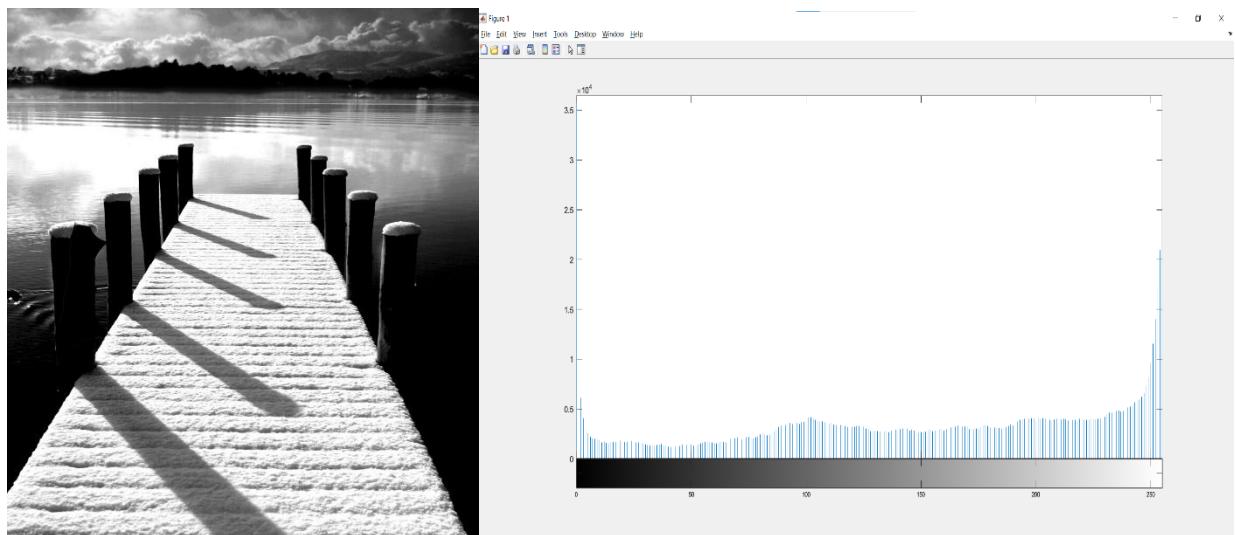
Ảnh tối

```
B = imread('HW1_DarkImage.jpg')  
imshow(B)  
title('Dark image')  
imhist(B)
```



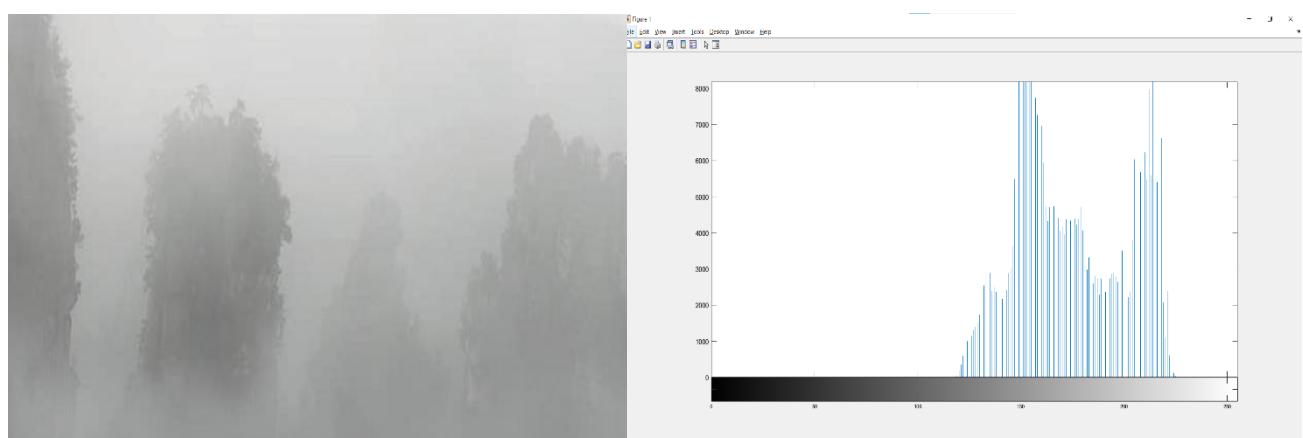
Ảnh tương phản cao:

```
C = imread('HW1_HighContrastImage.jpg')  
imshow(C)  
title('High contrast image')  
imhist(C)
```



Ảnh tương phản thấp:

```
D = imread('HW1_LowContrastImage.jpg')
imshow(D)
title('Low contrast image')
imhist(D)
```

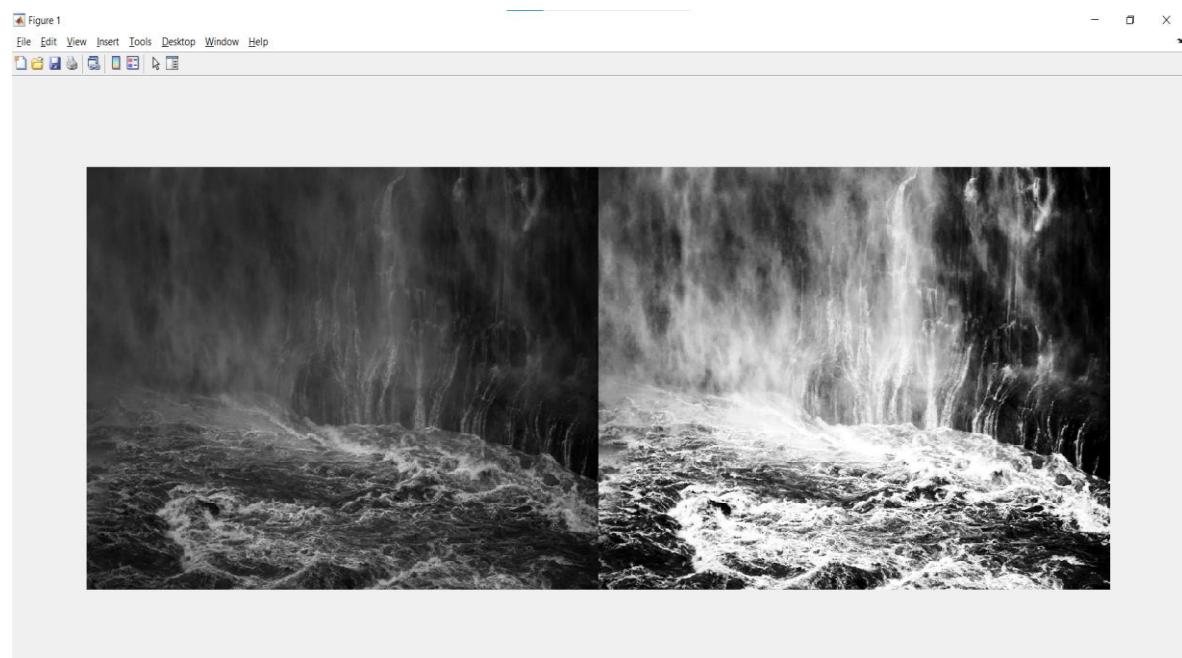


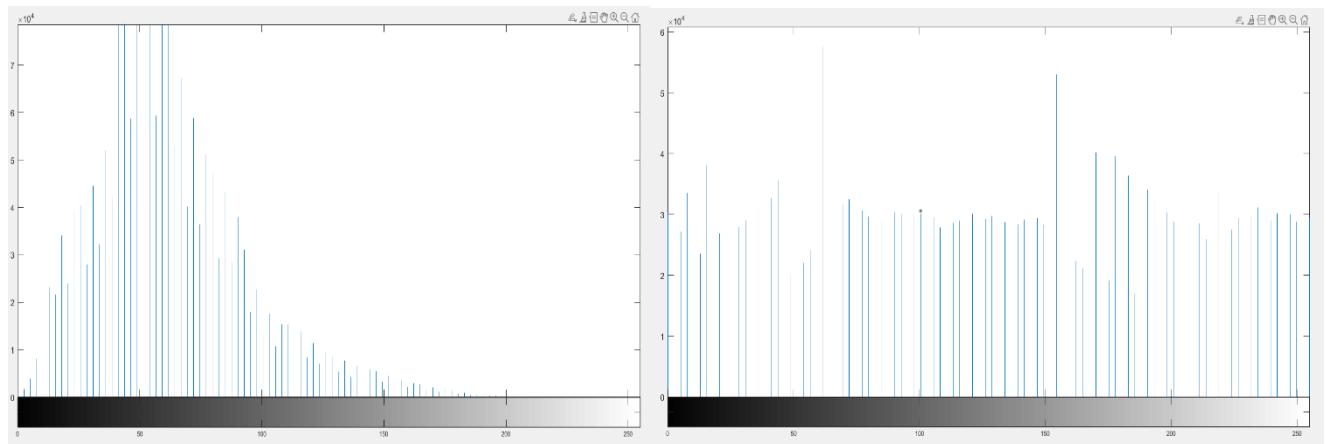
✓ Histogram equalization

imhist(A, 100): hiển thị histogram của một hình ảnh có 100 vách ngăn.

histeq: cân bằng histogram của một ảnh.

```
A = imread('HW1_DarkImage.jpg')  
Heq = histeq(A)  
imshowpair(A, Heq, 'montage')  
imhist(A,100)  
imhist(Heq,100)
```





✓ **Histogram matching**

```

A = imread('HW1_HistogramSourceImage.jpg')

Ref = imread('HW1_HistogramRefImage.jpg')

B = imhistmatch(A,Ref)

imshow(A)

title('Original Image')

imhist(A)

imshow(Ref)

title('Reference Image')

imhist(Ref)

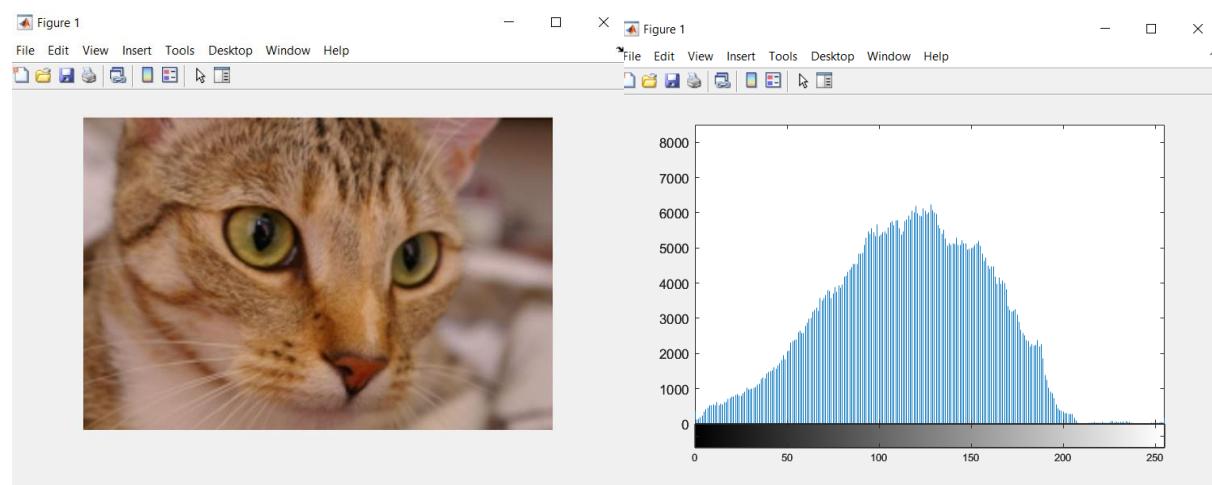
imshow(B)

title('Output Image')

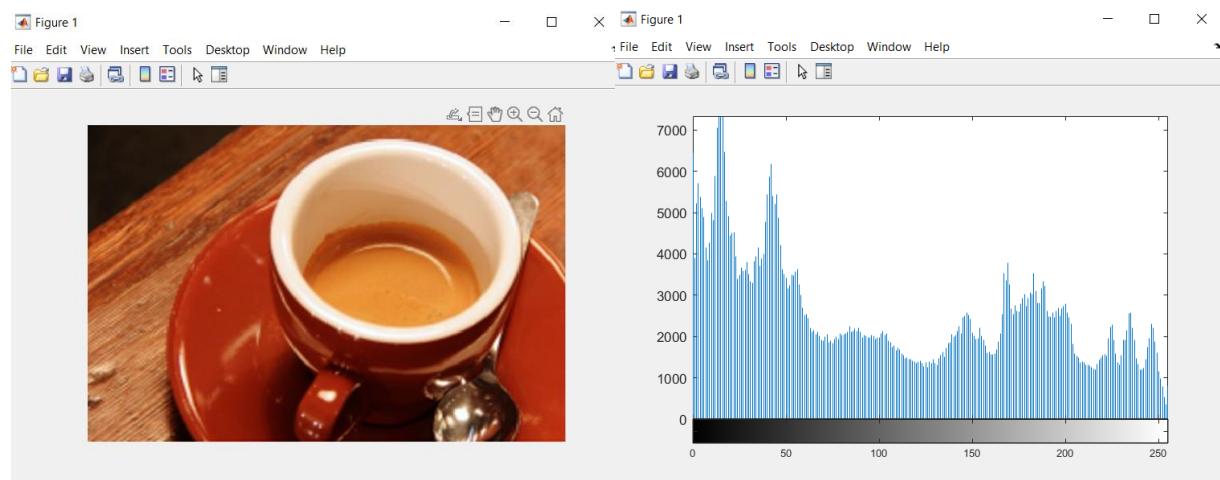
imhist(B)

```

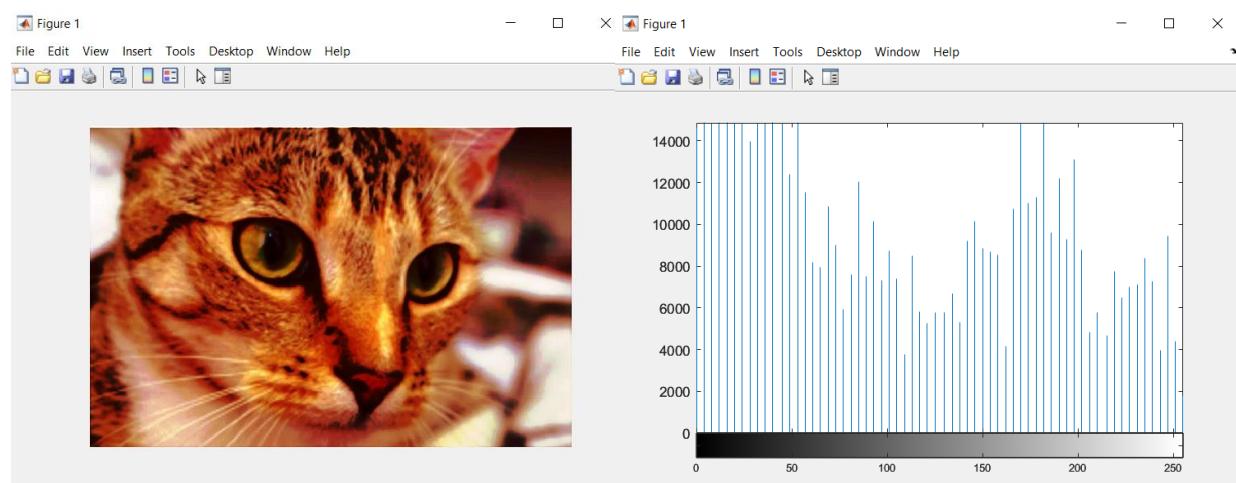
Ảnh gốc:



Ảnh tham chiếu



Ảnh cuối cùng sau khi chỉnh histogram theo ảnh tham chiếu:



Spatial Domain Filtering:

✓ Box Filter:

- **imboxfilt:** để thực hiện một bộ lọc trung bình (mean filter) trên hình ảnh (5 kích thước của kernel 5x5 mà bộ lọc trung bình sử dụng để tính trung bình các giá trị pixel trong vùng xung quanh mỗi pixel trong hình ảnh)

```

A = imread('HW1_BoxFilter.jpg')

imshow(A)

title('Original image')

A_filter = imboxfilt(A,5)

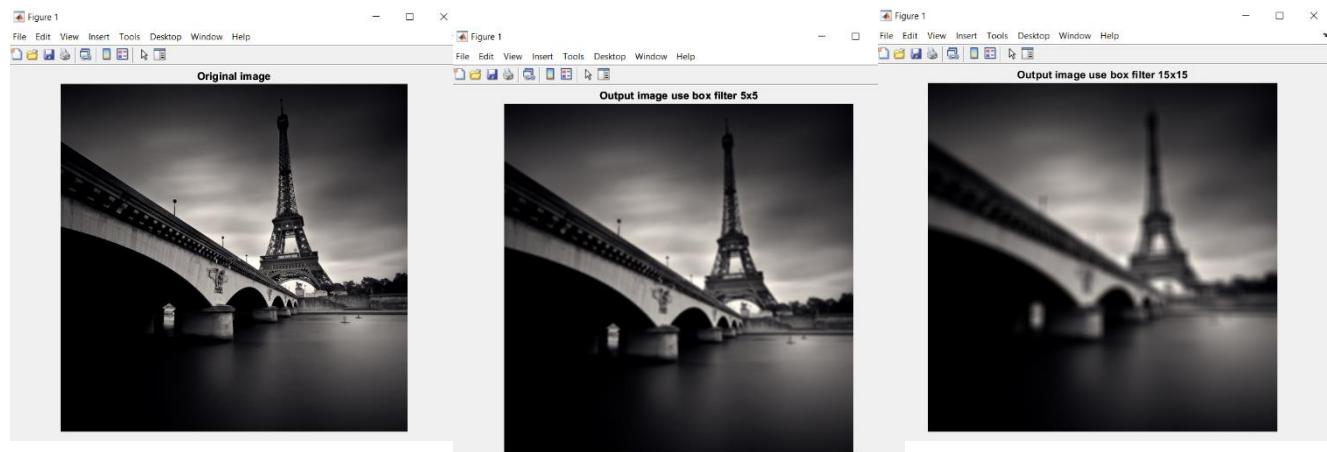
imshow(A_filter)

title('Original image')

A_filter = imboxfilt(A,15)

imshow(A_filter)

```



✓ Weighted Average Filter:

imgaussfilt: để thực hiện một bộ lọc Gaussian trên hình ảnh (2 là tham số sigma của phân phối Gaussian, được sử dụng để xác định độ rộng của hàm Gaussian, sigma lớn → hình ảnh sau khi xử lý sẽ mờ hơn)

```

A = imread('HW1_WeightedAverageFilter.jpg')
A_filter = imgaussfilt(A,2)
imshow(A)

```



✓ **Median Filter:**

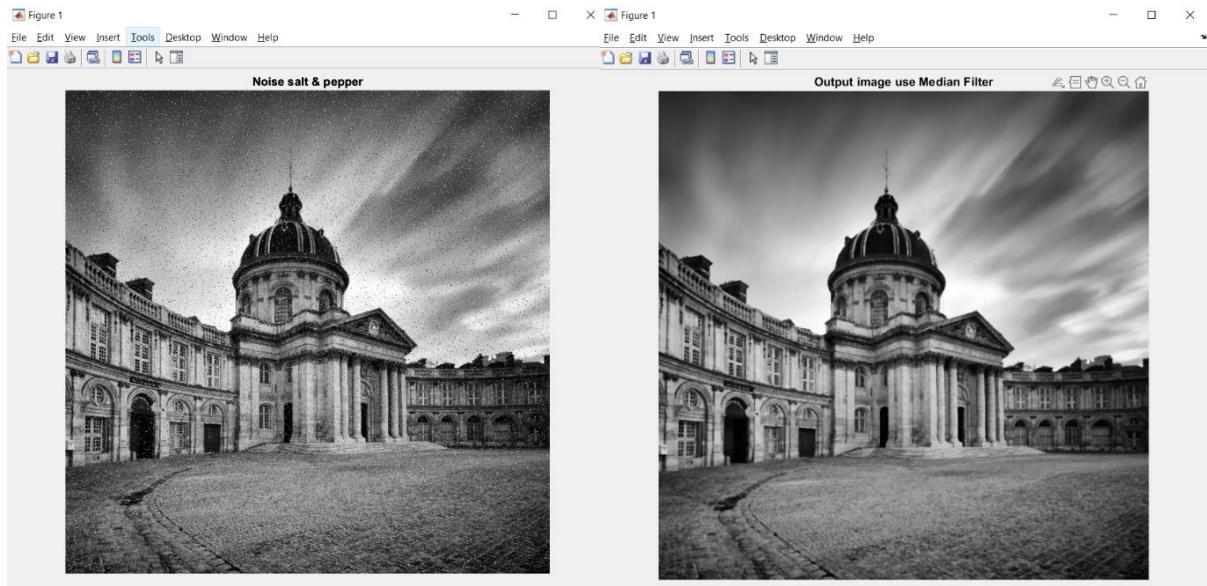
imnoise: Sử dụng để thêm nhiễu vào hình ảnh (0.02 có nghĩa là khoảng 2% điểm ảnh trong hình ảnh sẽ bị nhiễu).

medfilt: Thực hiện bộ lọc trung vị trên hình ảnh.

```

A = imread('HW1_MedianFilter.jpg');
B = imnoise(A, 'salt & pepper', 0.04)
B_gray = rgb2gray(B)
imshow(B_gray)
title('Noise salt & pepper')
C = medfilt2(B_gray)
imshow(C)
title('Output image use Median Filter')

```



✓ **Order-statistic filters:**

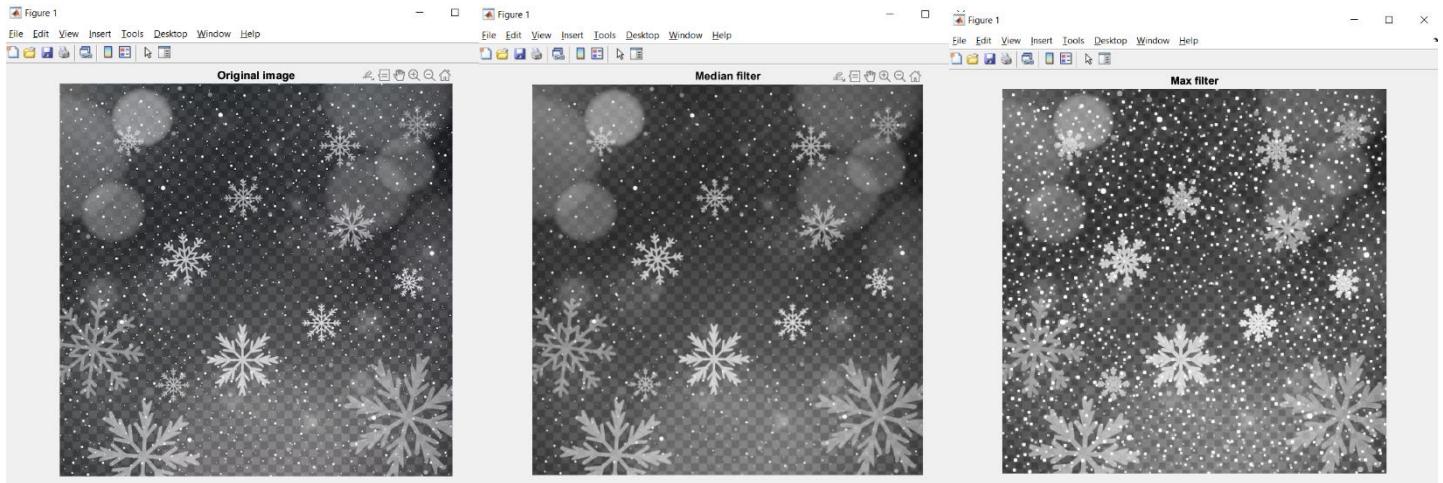
- ***ordfilt2(A,order,domain)*:** Thông số order được xác định bởi kích thước domain, như ở hình trên đây domain là ma trận kích thước 3×3 , khi đó nếu dùng Median filter thì order = 5, nếu dùng Minimum filter thì order = 1, nếu dùng Maximum filter thì order = 9. Vậy nếu domain là ma trận kích thước 5×5 , khi đó nếu dùng Median filter thì order = 13, nếu dùng Minimum filter thì order = 1, nếu dùng Maximum filter thì order = 25.

```

J = imread('HW1_OrderStatisticFilter.jpg');
imshow(J)
title('Original image')
gray_J = rgb2gray(J);
A_med = ordfilt2(gray_J, 5, ones(3,3));
imshow(A_med)
title('Median filter')
% Ảnh sau khi sử dụng max filter:

```

```
A_max = ordfilt2(gray_J, 9, ones(3,3));
imshow(A_max)
title('Max filter')
```



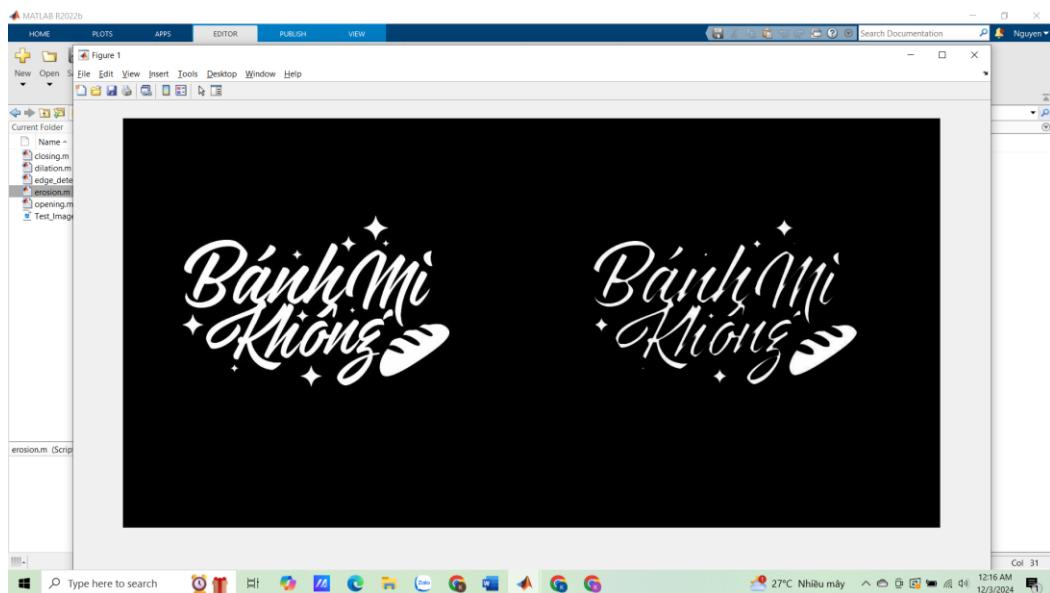
✓ **Morphological Image Processing:**

- **Erosion:**

```
I = imread("Test_Image.jpg");
SE = strel('square', 5);
I_Ero = imerode(I, SE);
imshowpair(I,I_Ero,'montage');
```

+ *strel*: để tạo một phần tử cấu trúc hình vuông (square) với kích thước 5x5 pixel.

+ *imerode*: để thực hiện phép co lại (erosion) trên hình ảnh bằng cách sử dụng phần tử cấu trúc SE.

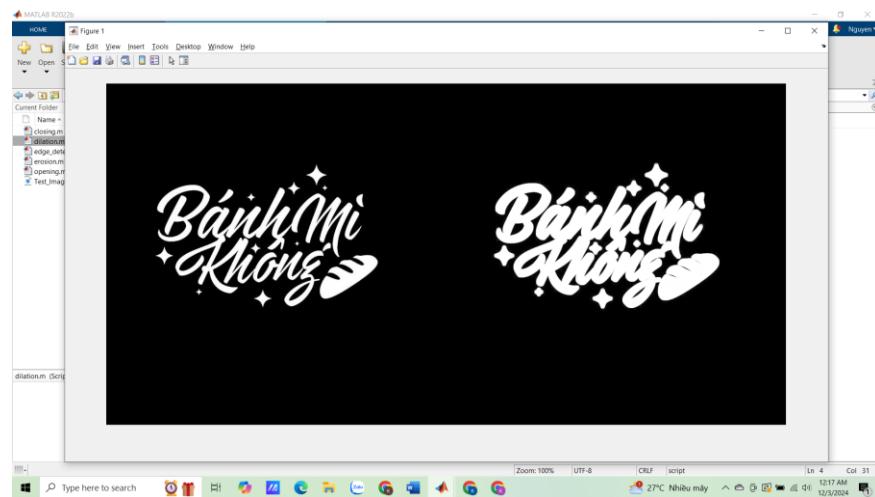


- Dilation:

```
I = imread("Test_Image.jpg");
SE = strel('disk', 5);
I_Dil = imdilate(I, SE);
imshowpair(I,I_Dil,'montage');
```

+ *strel*: để tạo một phần tử cấu trúc dạng hình tròn (disk) với bán kính là 2.5 pixel.

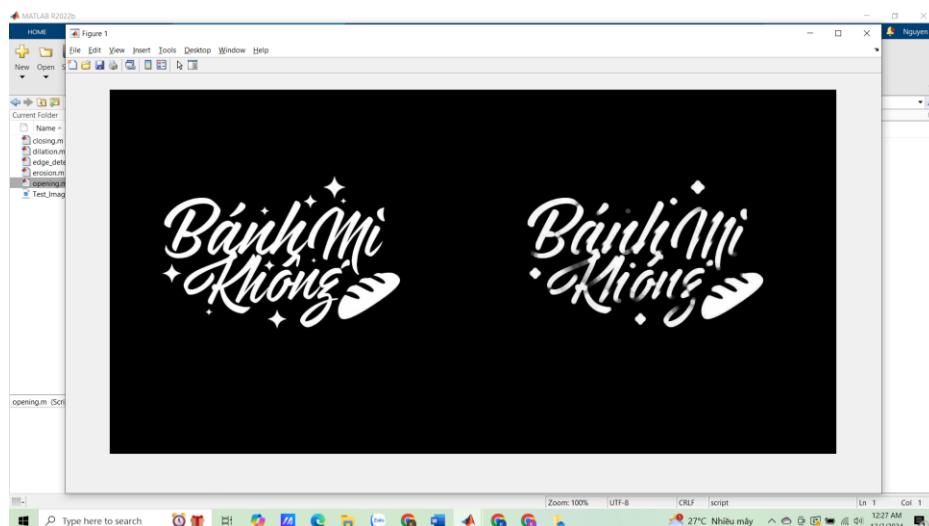
+ *imdilate*: để thực hiện phép co dãn (dilation) trên hình ảnh bằng cách sử dụng phần tử cấu trúc SE.



- Opening:

```
I = imread("Test_Image.jpg");
SE = strel('disk', 5);
I_Op = imopen(I, SE);
imshowpair(I,I_Op,'montage');
```

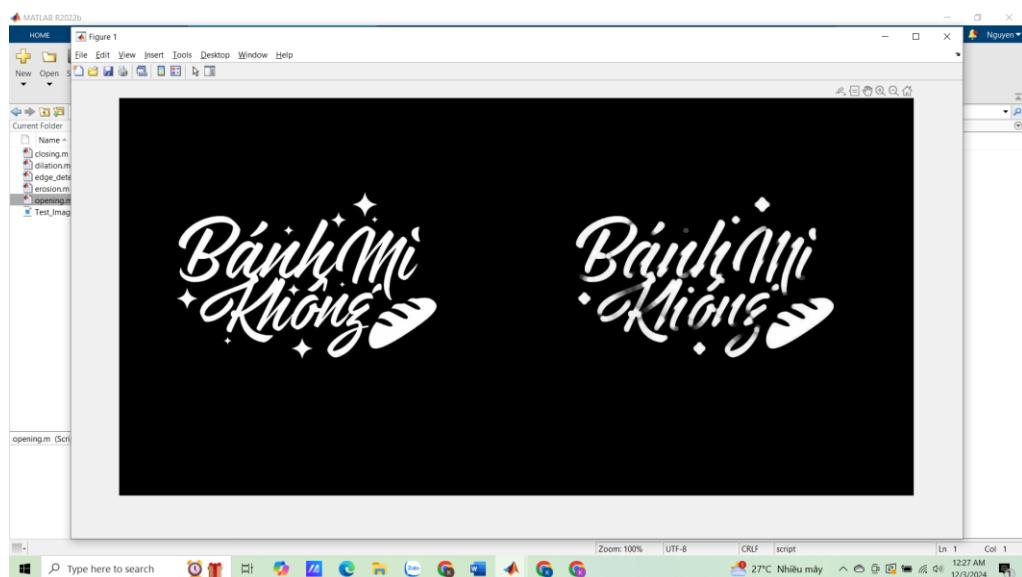
+ *imopen*: để thực hiện phép mở (opening) trên hình ảnh bằng cách sử dụng phần tử cấu trúc SE.



- Closing:

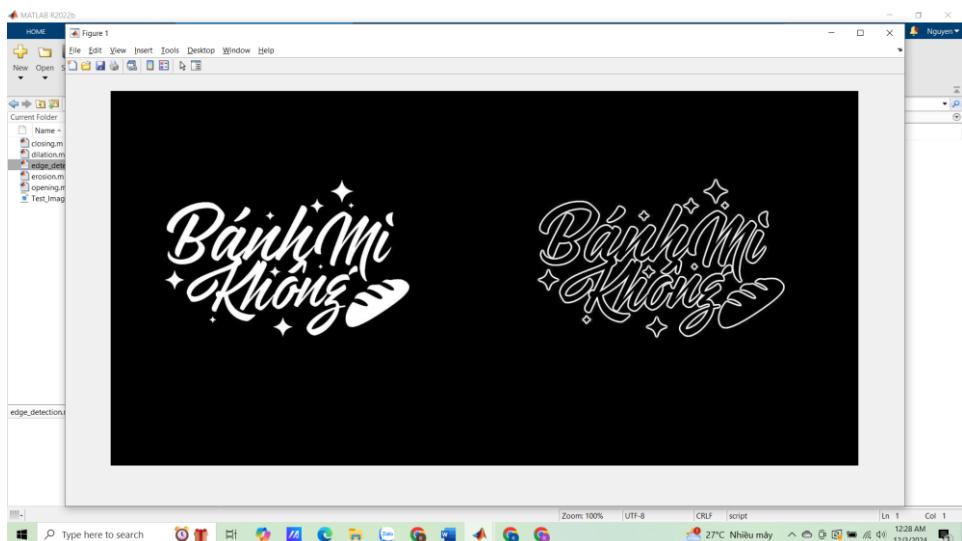
```
I = imread("Test_Image.jpg");
SE = strel('disk', 5);
I_Clo = imopen(I, SE);
imshowpair(I,I_Clo,'montage');
```

+ *imclose*: để thực hiện phép đóng (closing) trên hình ảnh bằng cách sử dụng phần tử cấu trúc SE.



- Edge detection:

```
I = imread("Test_Image.jpg");
SE = strel('disk', 4);
I_edg = imdilate(I, SE);
SE = strel('disk', 4);
I_Dil = imdilate(I, SE);
I_edg = I_Dil - I;
imshowpair(I,I_edg,'montage');
```



✓ *Image Segmentation:*

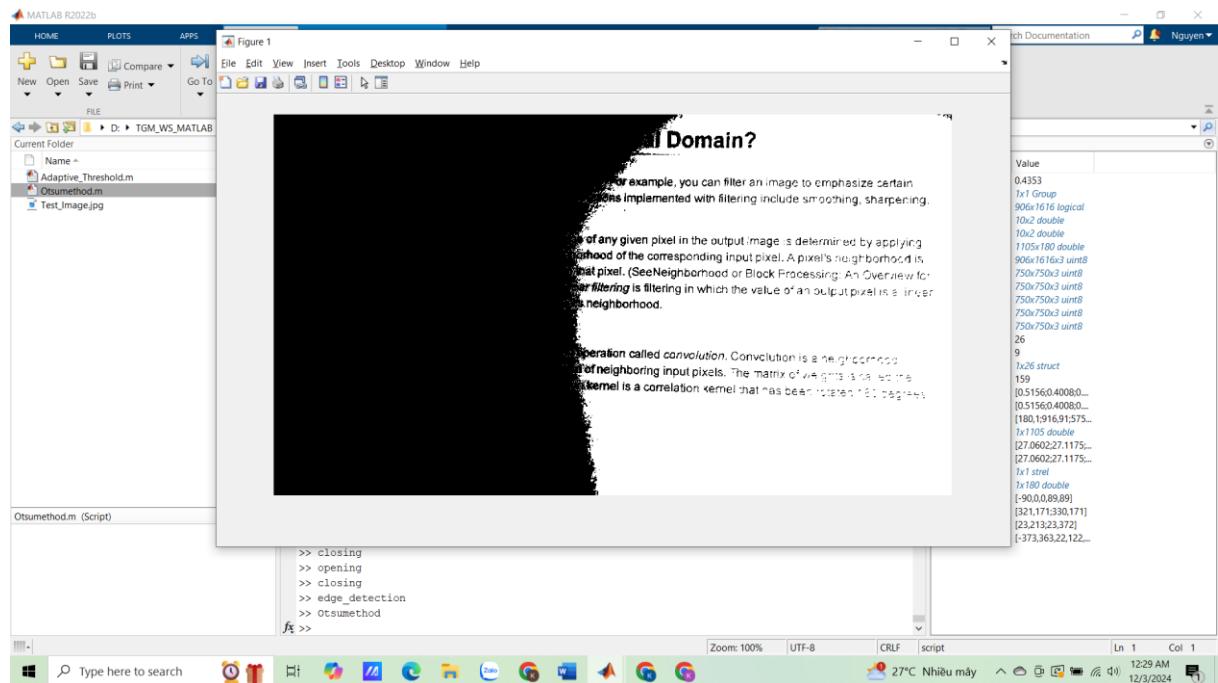
- **Otsu method:**

```
I = imread("Test_Image.jpg");
imshow(I);
A = graythresh(I);
BW = imbinarize(I,A);
BW = squeeze(BW(:,:,1));
imshow(BW);
```

+ *graythresh*: để tính toán một ngưỡng (threshold) nhị phân hóa dựa trên hình ảnh.

+ *imbinarize*: để nhị phân hóa hình ảnh sử dụng ngưỡng level được tính toán trước đó bằng hàm *graythresh*.

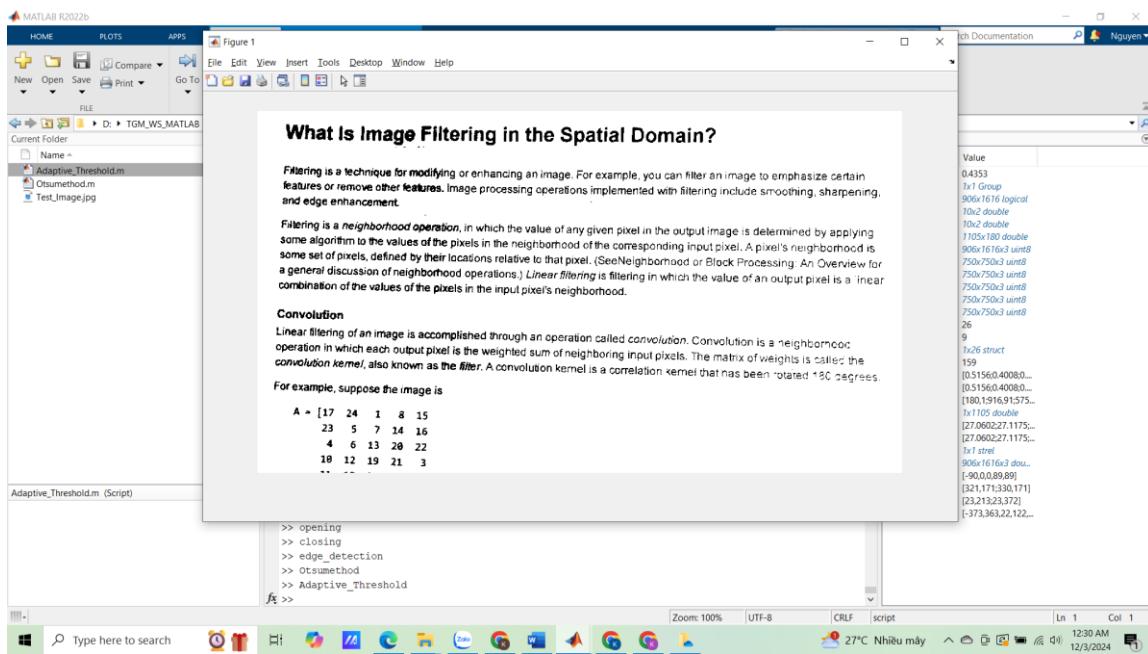
+ *squeeze*: được sử dụng để trích xuất một kênh màu từ một hình ảnh RGB và chuyển nó thành hình ảnh đơn sắc (BW(:,:,1) là một trích xuất dữ liệu từ kênh màu đỏ của hình ảnh ban đầu).



- Adaptive threshold:

```
I = imread("Test_Image.jpg");  
  
imshow(I);  
  
T = adaptthresh(I,0.3,'ForegroundPolarity','dark');  
  
BW = imbinarize(I,T);  
  
BW = squeeze(BW(:,:,1));  
  
imshow(BW);
```

+ `adapththresh`: để tính toán một ngưỡng (threshold) cục bộ dựa trên hình ảnh (0.4 là giá trị ngưỡng toàn cục (global threshold), 'ForegroundPolarity', 'dark': quan tâm đến các đối tượng hoặc vùng có độ sáng thấp hơn so với nền (vùng tối hơn)).



✓ **Edge detection:**

- **Sobel:**

```
I = imread("Test_Image.jpg");

I = squeeze(I(:,:,1));

I_sobel = edge(I, 'sobel',0.1);

subplot(1,2,1);

imshow(I);

title('Original Image');

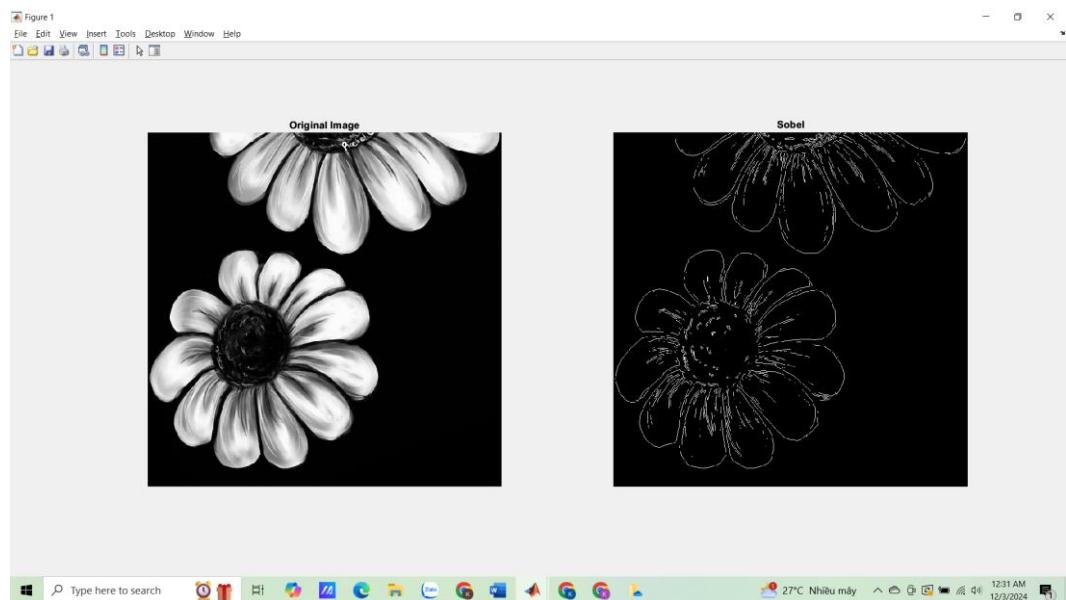
subplot(1,2,2);

imshow(I_sobel);

title('Sobel');
```

+ *edge*: để thực hiện phát hiện biên (edge detection) trên hình ảnh bằng cách sử dụng bộ lọc Sobel và một ngưỡng (threshold) là 0.1. Là một trong các loại bộ lọc có thể được sử dụng để phát hiện biên. Bộ lọc Sobel được sử dụng để xác định

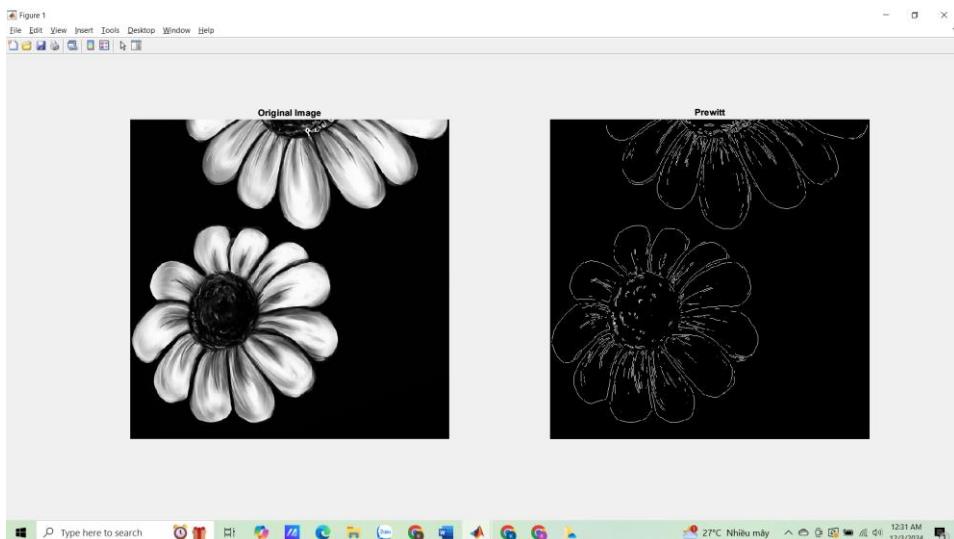
đạo hàm riêng của hình ảnh theo hướng ngang và dọc. Nó thường được sử dụng trong việc phát hiện biên.



- Prewitt:

```
I = imread("Test_Image.jpg");
I = squeeze(I(:,:,1));
I_pre = edge(I, 'prewitt',0.1);
subplot(1,2,1);
imshow(I);
title('Original Image');
subplot(1,2,2);
imshow(I_pre);
title('Prewitt');
```

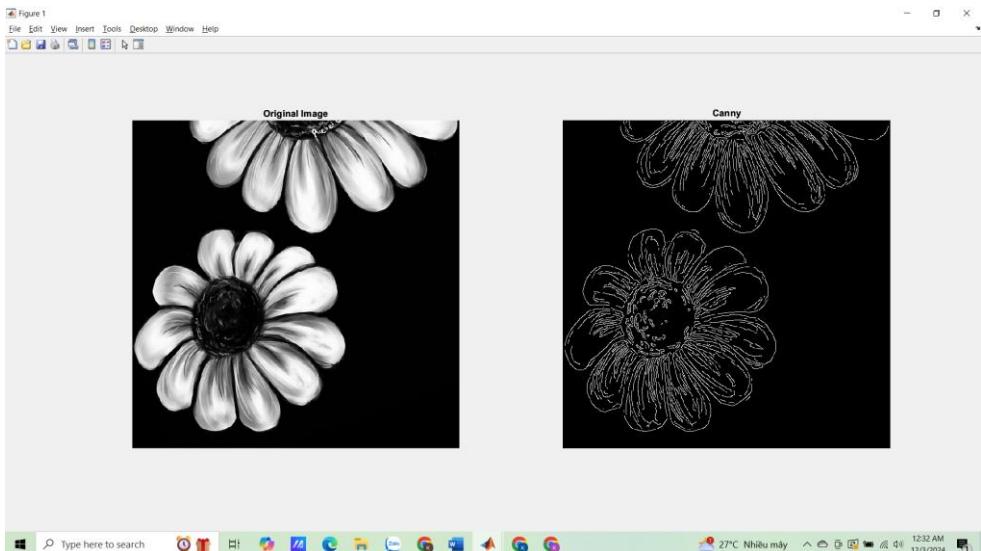
+ *edge*: để thực hiện phát hiện biên (edge detection) trên hình ảnh bằng cách sử dụng bộ lọc Prewitt và một ngưỡng (threshold) là 0.1.



- Canny:

```
I = imread("Test_Image.jpg");
I = squeeze(I(:,:,1));
I_can = edge(I, 'canny',0.1);
subplot(1,2,1);
imshow(I);
title('Original Image');
subplot(1,2,2);
imshow(I_can);
title('Canny');
```

+ *edge*: để thực hiện phát hiện biên (edge detection) trên hình ảnh bằng cách sử dụng bộ lọc Canny và một ngưỡng (threshold) là 0.1.



- Hough – Houghlines – Houghpeaks:

```

I = imread("Test_Image1.jpg");

I = squeeze(I(:,:,1));

BW = edge(I, 'canny');

imshow(BW);

title('Original Image');

[H,T,R] = hough(BW);

imshow(H, [], 'XData', T, 'YData', R, 'InitialMagnification', 'fit');

xlabel('theta'), ylabel('rho');

axis on, axis normal, hold on;

title('Hough Transform');

P = houghpeaks(H, 5, 'threshold', ceil(0.3 * max(H(:))));

x = T(P(:, 2));

y = R(P(:, 1));

plot(x, y, 's', 'color', 'white');

lines = houghlines(BW, T, R, P, 'FillGap', 5, 'MinLength', 7);

```

```

figure, imshow(I), hold on

max_len = 0;

for k = 1:length(lines)

    xy = [lines(k).point1; lines(k).point2];

    plot(xy(:,1), xy(:,2), 'LineWidth', 2, 'Color', 'green');

    % Plot beginnings and ends of lines

    plot(xy(1,1), xy(1,2), 'x', 'LineWidth', 2, 'Color', 'yellow');

    plot(xy(2,1), xy(2,2), 'x', 'LineWidth', 2, 'Color', 'red');

    % Determine the endpoints of the longest line segment

    len = norm(lines(k).point1 - lines(k).point2);

    if (len > max_len)

        max_len = len;

        xy_long = xy;

    end

end

```

+ *squeeze*: chuyển sang ảnh 2 chiều.

+ *edge*: tạo ảnh nhị phân.

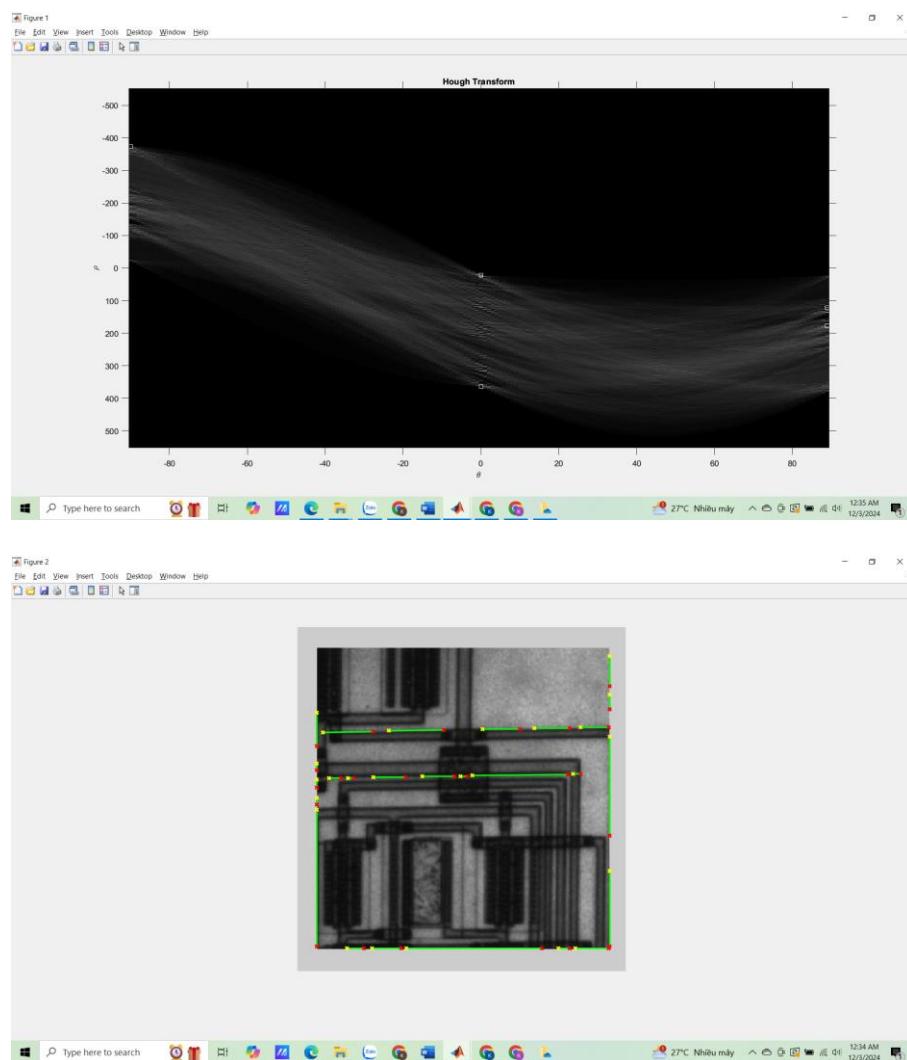
+ *hough*: tạo ma trận Hough transform.

+ *houghpeaks*: để tìm các điểm cao điểm trên biến đổi Hough ‘H’. Cụ thể, tìm 5 điểm cao điểm (điểm tối đa) bằng cách sử dụng độ ngưỡng (threshold) là 30% của giá trị lớn nhất trong ‘H’.

+ *houghlines*: sử dụng phép biến đổi Hough để phát hiện các đường thẳng trong hình ảnh nhị phân.

- *FillGap*, 5: chỉ định một khoảng cách tối đa (5 pixel) giữa các đoạn đường có thể được kết hợp lại thành một đường thẳng duy nhất.
 - *MinLength*, 7: chỉ định chiều dài tối thiểu của một đường thẳng để nó được chấp nhận.
- + *norm*: tính độ dài của đoạn đường thẳng.

→ Đoạn mã trên thực hiện các bước để phát hiện và vẽ đoạn đường thẳng dài nhất trong một hình ảnh.

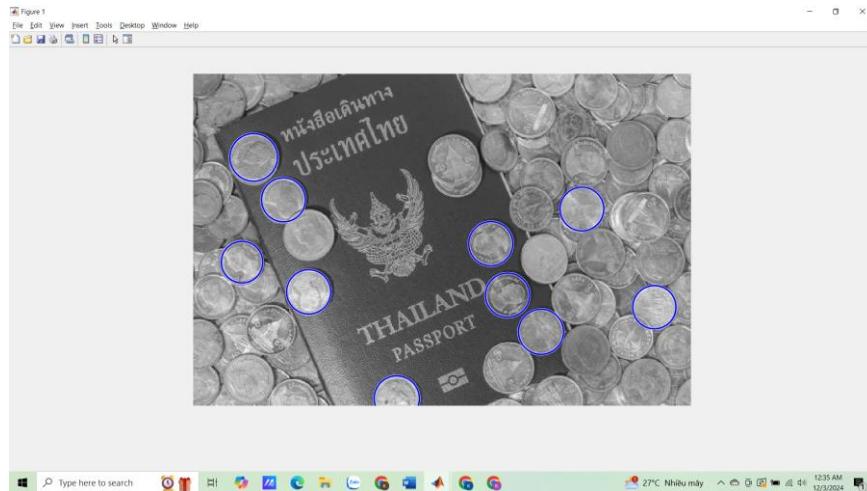


- Hough circle:

```
I = imread('Test_Image2.jpg');
imshow(I)
```

```
[centers, radii, metric] = imfindcircles(I, [10 30]);
centersStrong10 = centers(1:10, :);
radiiStrong10 = radii(1:10);
metricStrong10 = metric(1:10);
viscircles(centersStrong10, radiiStrong10, 'EdgeColor', 'b');
```

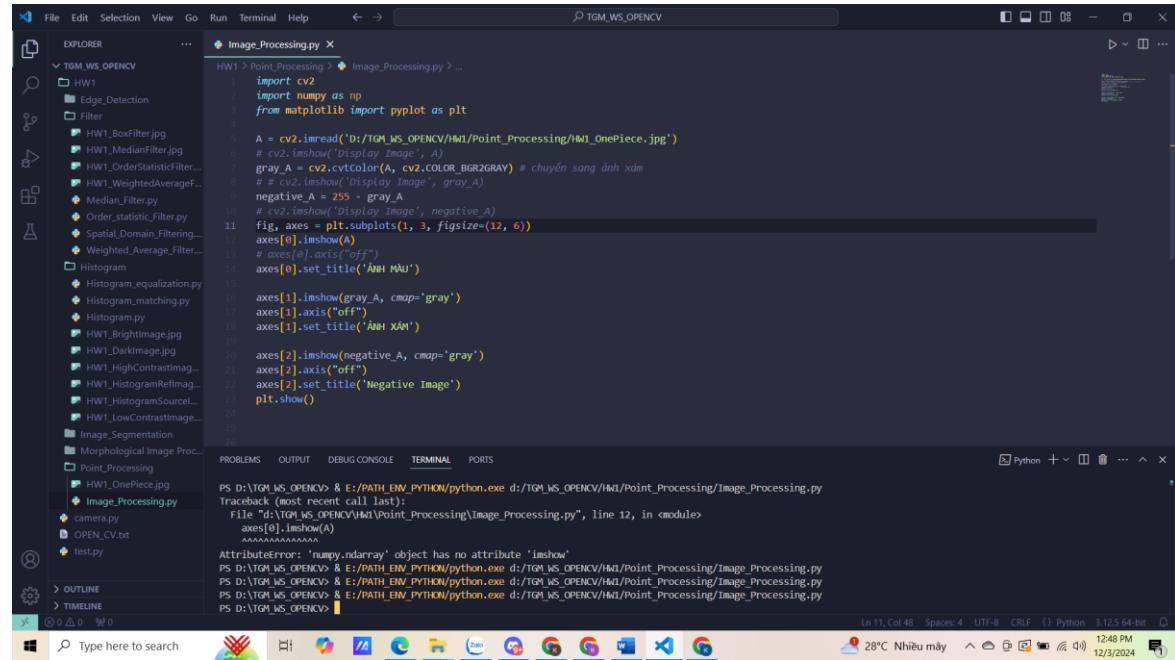
- + *imfindcircles*: để tìm các vòng tròn trong hình ảnh (tìm các vòng tròn có bán kính nằm trong khoảng từ 10 đến 30 điểm ảnh).
- + *centers*: chọn ra 10 tâm (centers) đầu tiên từ biến centers và lưu chúng vào biến centersStrong10.
- + *radii*: chọn ra 10 bán kính đầu tiên từ biến radii và lưu chúng vào biến radiiStrong10.
- + *metric*: chọn ra 10 giá trị độ tin cậy đầu tiên từ biến metric và lưu chúng vào biến metricStrong10.
- + *viscircles*: để vẽ các vòng tròn đã được chọn lên hình ảnh (tọa độ tâm của các vòng tròn, bán kính của các vòng tròn, đặt màu viền của các vòng tròn).



1.2 Tìm các lệnh tương ứng dùng OpenCV

✓ Point Processing:

- Đọc và hiển thị ảnh màu, ảnh xám và negative image.

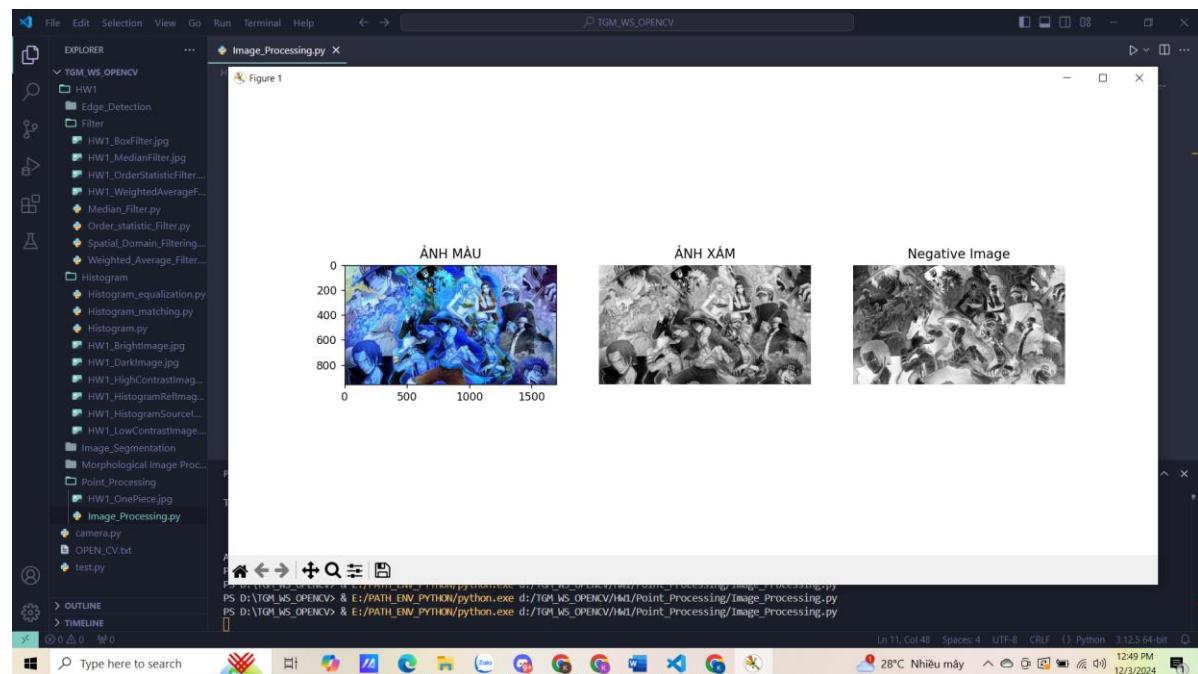


```

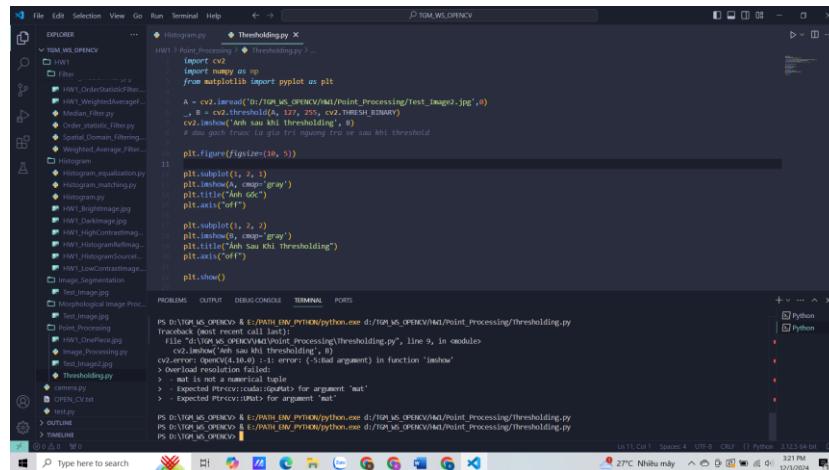
File Edit Selection View Go Run Terminal Help
EXPLORER Image_Processing.py
HW1
  Edge_Detection
  Filter
    HW1_BoxFilter.jpg
    HW1_MedianFilter.jpg
    HW1_OrderStatisticFilter...
    HW1_WeightedAverageF...
    Median_Filter.py
    Order_statistic_Filter.py
    Spatial_Domain_Filtering...
    Weighted_Average_Filter...
  Histogram
    Histogram_equalization.py
    Histogram_matching.py
    Histogram.py
    HW1_BrightImage.jpg
    HW1_DarkImage.jpg
    HW1_HighContrastImage...
    HW1_HistogramRefine...
    HW1_HistogramSource...
    HW1_LowContrastImage...
  Image_Segmentation
  Morphological_Image_Pro...
  Point_Processing
    HW1_OnePiece.jpg
    Image_Processing.py
  camera.py
  OPEN_CV.txt
  test.py
  OUTLINE
  TIMELINE
  Type here to search
  PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
PS D:\TGM_WS_OPENCV & E:/PATH_ENV_PYTHON/python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/Image_Processing.py
Traceback (most recent call last):
  File "d:/TGM_WS_OPENCV/HW1/Point_Processing\Image_Processing.py", line 12, in <module>
    axes[0].imshow(A)
    ~~~~~
AttributeError: 'numpy.ndarray' object has no attribute 'imshow'
PS D:\TGM_WS_OPENCV & E:/PATH_ENV_PYTHON/python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/Image_Processing.py
PS D:\TGM_WS_OPENCV & E:/PATH_ENV_PYTHON/python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/Image_Processing.py
PS D:\TGM_WS_OPENCV & E:/PATH_ENV_PYTHON/python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/Image_Processing.py
PS D:\TGM_WS_OPENCV

```

→ Kết quả:



- Thresholding:



```

File Edit Selection View Go Run Terminal Help <- > Jupyter Notebook Thresholding.py
HWT> Point_Processing > Thresholding.py
import cv2
import numpy as np
from matplotlib import pyplot as plt

A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Point_Processing/test_image1.jpg',0)
B = cv2.threshold(A, 127, 255, cv2.THRESH_BINARY)
cv2.imshow('Anh Goc', A)
cv2.imshow('Anh sau khi thresholding', B)

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(A, cmap='gray')
plt.title("Anh Goc")
plt.xticks([])
plt.yticks([])

plt.subplot(1, 2, 2)
plt.imshow(B, cmap='gray')
plt.title("Anh Sau Khi Thresholding")
plt.xticks([])
plt.yticks([])

plt.show()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS D:\TGM_WS_OPENCV & E:\PATH_EM PYTHON\python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/thresholding.py

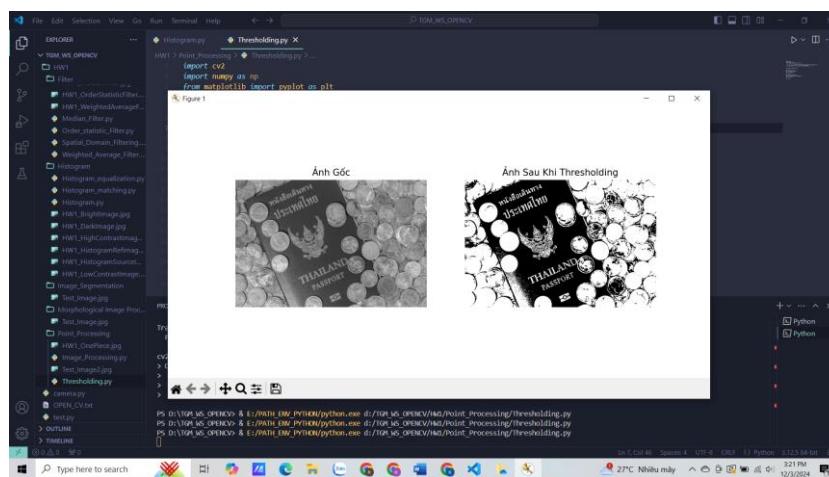
File d:/TGM_WS_OPENCV/HW1/Point_Processing/thresholding.py, line 9, in module
 cv2.imshow('Anh sau khi thresholding', B)
cv2.imshow: argument 1 is not a valid image.
 > mat is not a numerical tuple
 > mat is not an array
 > - Expected Ptr<CvArr<Output> for argument 'mat'
 > - Expected Ptr<CvArr<Input> for argument 'mat'

PS D:\TGM_WS_OPENCV & E:\PATH_EM PYTHON\python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/thresholding.py

PS D:\TGM_WS_OPENCV & E:\PATH_EM PYTHON\python.exe d:/TGM_WS_OPENCV/HW1/Point_Processing/thresholding.py

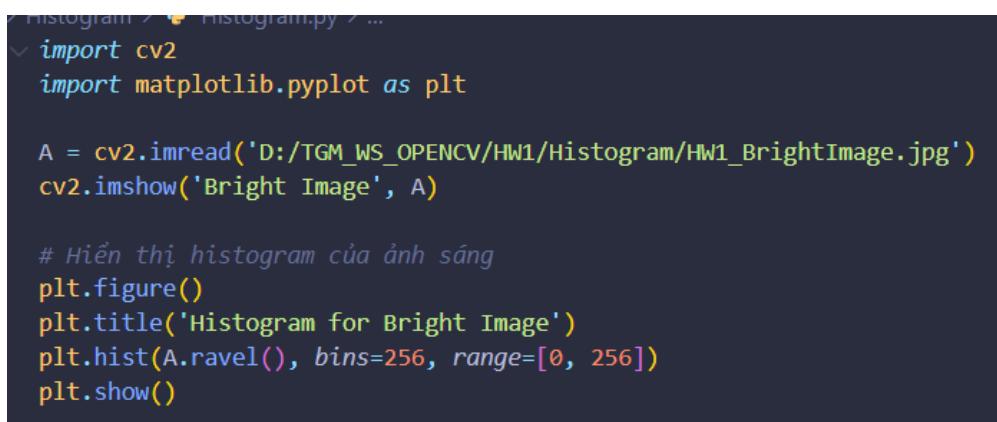
PS D:\TGM_WS_OPENCV

→ Kết quả:



✓ Histogram:

- Vẽ histogram của ảnh sáng.



```

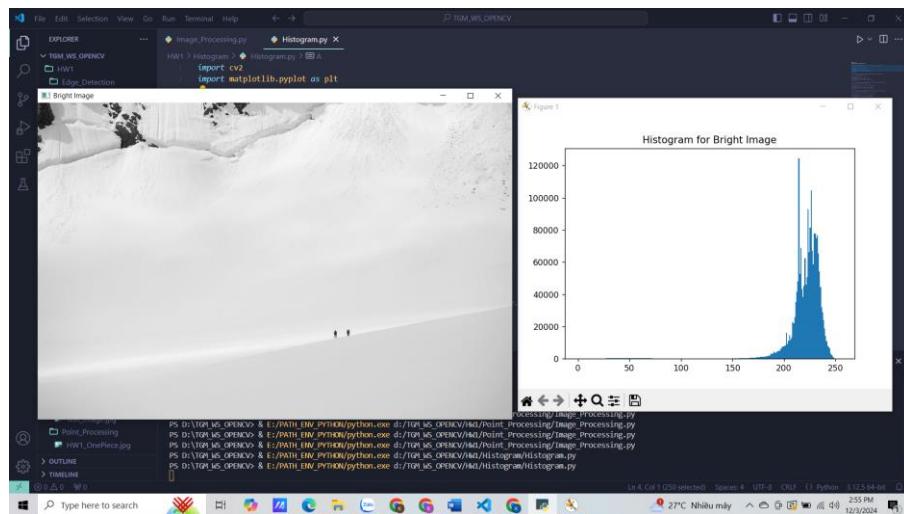
Histogram > Histogram.py > ...
import cv2
import matplotlib.pyplot as plt

A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Histogram/HW1_BrightImage.jpg')
cv2.imshow('Bright Image', A)

# Hiển thị histogram của ảnh sáng
plt.figure()
plt.title('Histogram for Bright Image')
plt.hist(A.ravel(), bins=256, range=[0, 256])
plt.show()

```

→ Kết quả:

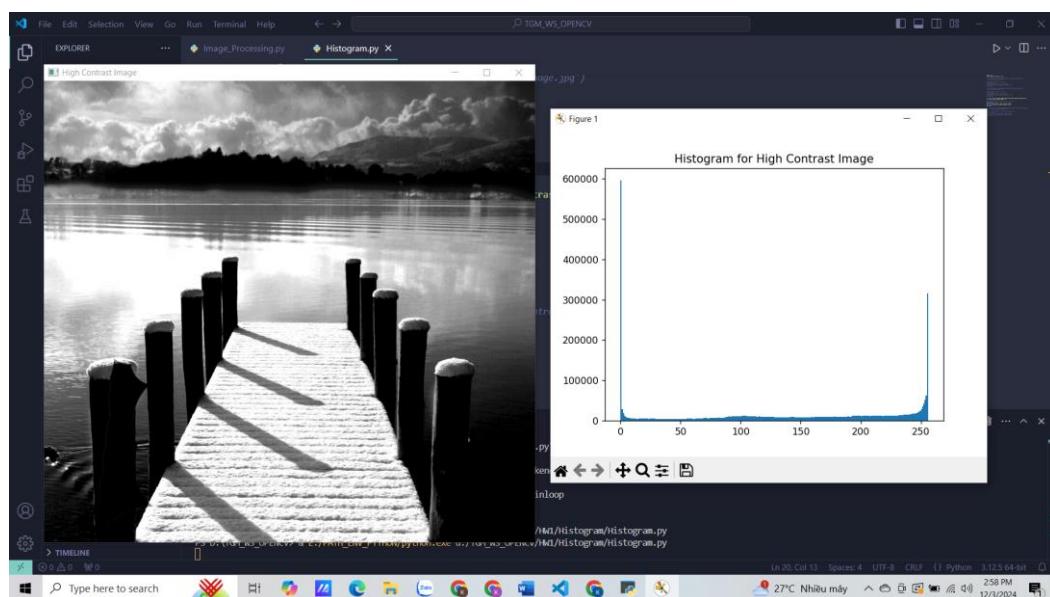


- Vẽ histogram của ảnh có độ tương phản cao.

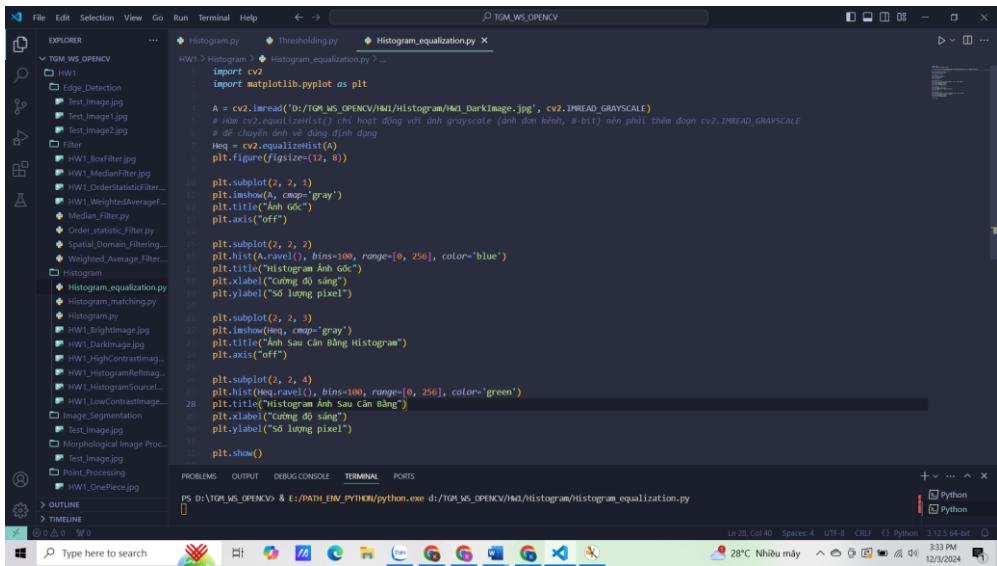
```
C = cv2.imread('D:/TGM_WS_OPENCV/HW1/Histogram/HW1_HighContrastImage.jpg')
cv2.imshow('High Contrast Image', C)

# Hiển thị histogram của ảnh có độ tương phản cao
plt.figure()
plt.title('Histogram for High Contrast Image')
plt.hist(C.ravel(), bins=256, range=[0, 256])
plt.show()
```

→ Kết quả:



- Histogram equalization:



```

File Edit Selection View Go Run Terminal Help
File Histogram.py Thresholding.py Histogram_equalization.py
EXPLORER HW1
HW1
Edge_Detection Test_Image1.jpg Test_Image1.jpg Test_Image2.jpg
Filter HW1_BBoxFilter.jpg HW1_MedianFilter.jpg HW1_OrderStatisticFilter...
HW1_WeightedAverageF... Median_Filter.py Order_statistic_Filter.py
Spatial_Domain_Filtering... Weighted_Average_Filter...
Histogram Histogram_equalization.py Histogram_matching.py
Histogram.py HW1_BrightImage.jpg HW1_DarkImage.jpg HW1_HighContrastImage...
HW1_HistogramRefining... HW1_HistogramSource... HW1_LowContrastImage...
Image_Segmentation Test_Image.jpg Test_Image.jpg Test_Image.jpg
Morphological_Image_Proc... Test_Image.jpg Point_Processing HW1_OnePiece.jpg
OUTLINE TIMELINE
Type here to search

```

```

import cv2
import matplotlib.pyplot as plt

A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Histogram/HW1_DarkImage.jpg', cv2.IMREAD_GRAYSCALE)
# Đọc ảnh (D:/TGM_WS_OPENCV/HW1/Histogram/HW1_DarkImage.jpg) chỉ hoạt động với ảnh grayscale (anh don kênh, 8-bit) nên phải thêm đoạn cv2.IMREAD_GRAYSCALE
# để chuyển đổi về định dạng ảnh

Heq = cv2.equalizeHist(A)
plt.figure(figsize=(12, 8))

plt.subplot(2, 2, 1)
plt.imshow(A, cmap='gray')
plt.title('Ảnh gốc')
plt.xlabel('Cường độ sáng')
plt.ylabel('Số lượng pixel')

plt.subplot(2, 2, 2)
plt.hist(A.ravel(), bins=100, range=[0, 256], color='blue')
plt.title('Histogram Ảnh Gốc')
plt.xlabel('Cường độ sáng')
plt.ylabel('Số lượng pixel')

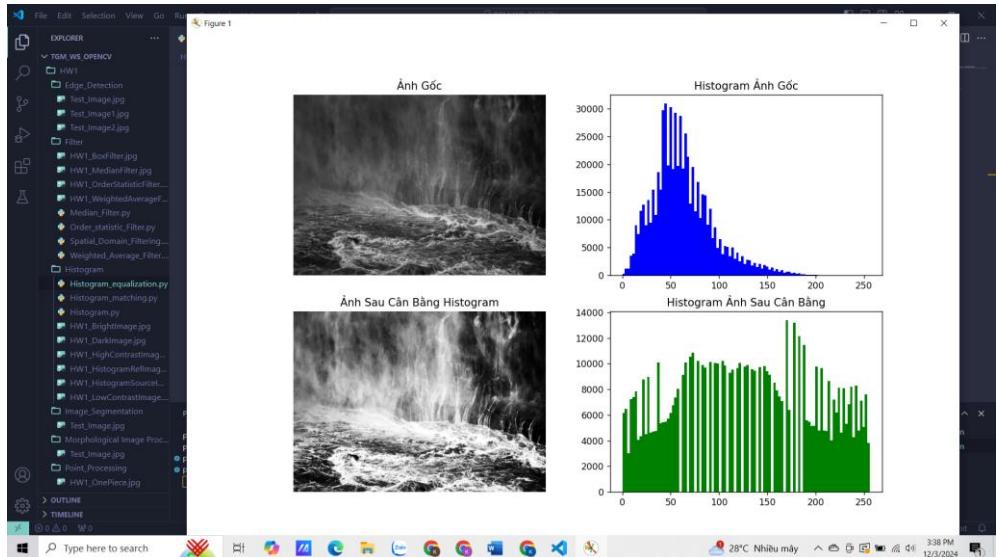
plt.subplot(2, 2, 3)
plt.imshow(Heq, cmap='gray')
plt.title('Ảnh Sau Cân Bằng Histogram')
plt.xlabel('Cường độ sáng')
plt.ylabel('Số lượng pixel')

plt.subplot(2, 2, 4)
plt.hist(Heq.ravel(), bins=100, range=[0, 256], color='green')
plt.title('Histogram Ảnh Sau Cân Bằng')
plt.xlabel('Cường độ sáng')
plt.ylabel('Số lượng pixel')

plt.show()

```

→ Kết quả:



✓ Spatial Domain Filtering:

- Dùng Weighted Average Filter để xử lý ảnh bị nhiễu Gauss.

```

> Filter > Weighted_Average_Filter.py > ...
import cv2
from matplotlib import pyplot as plt

A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Filter/HW1_WeightedAverageFilter.jpg')

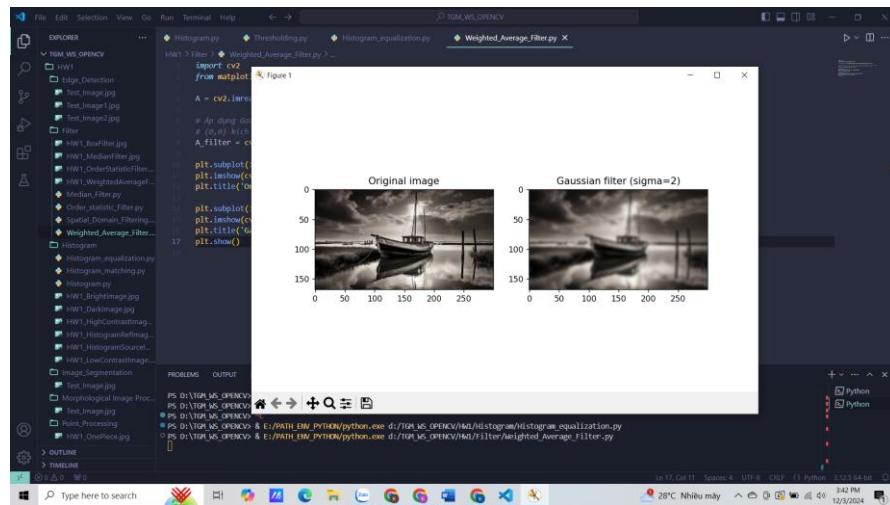
# Áp dụng Gaussian filter với sigma = 2 (tương đương imgaussfilt(A, 2) trong MATLAB)
# (0,0) kích thước cửa sổ kernel dùng để lọc, Sigma càng lớn thì mờ nhiều
A_filter = cv2.GaussianBlur(A, (0, 0), 2)

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(A, cv2.COLOR_BGR2RGB))
plt.title('Original image')

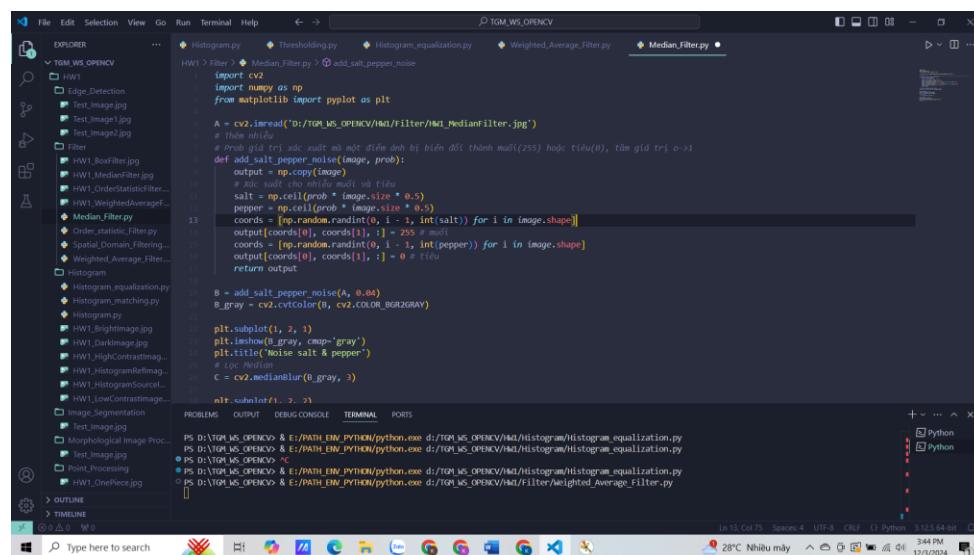
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(A_filter, cv2.COLOR_BGR2RGB))
plt.title('Gaussian filter (sigma=2)')
plt.show()

```

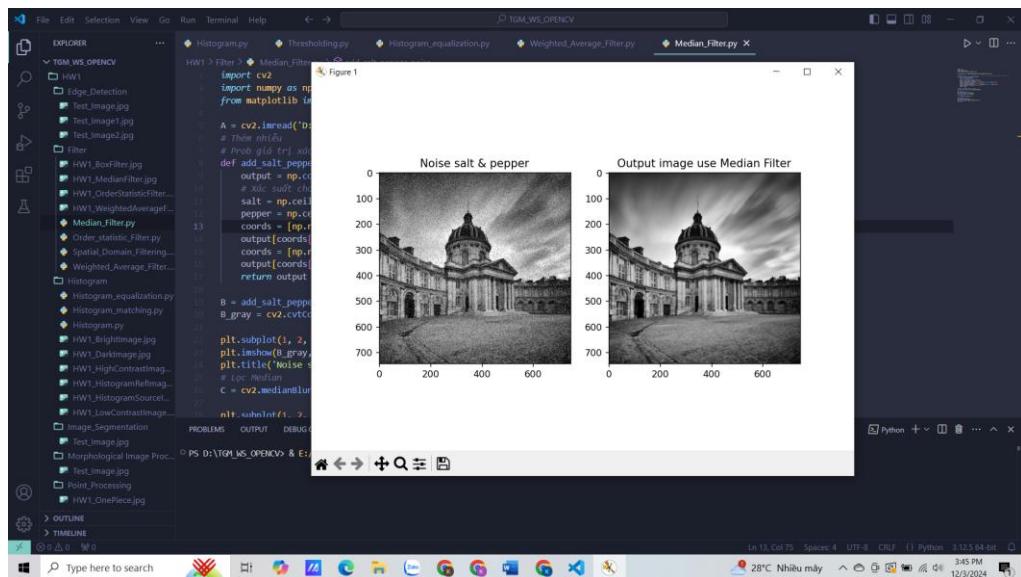
→ Kết quả:



- Dùng Median Filter để xử lý ảnh bị nhiễu muối tiêu (salt and pepper).



→ Kết quả:



✓ Morphological Image Processing:

- Erosion:

- + Syntax: `cv2.erode(src, kernel, dst, anchor, iterations, borderType, borderValue)`
- + Kernel có thể được tạo ra bằng hàm: `cv2.getStructuringElement(shape, ksize, anchor)`
- + Sử dụng kernel 5x5 với tất cả phần tử là 1, erode 2 lần.

```
Morphological Image Processing > Erosion.py > ...
import cv2
import numpy as np
from matplotlib import pyplot as plt

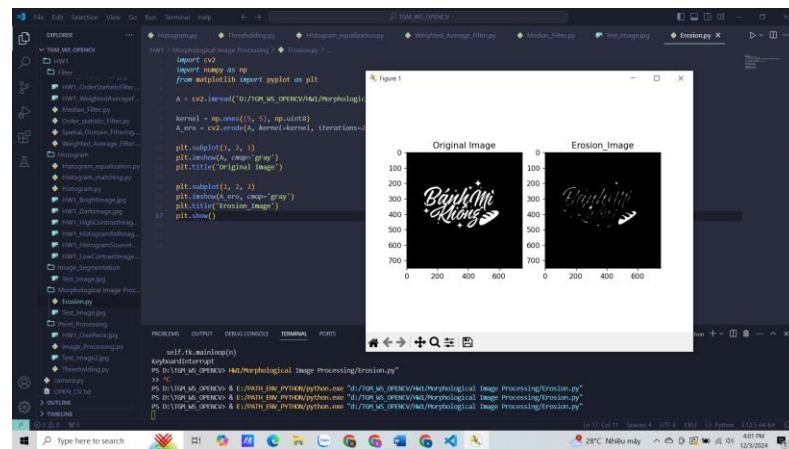
A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Morphological Image Processing/Test_Image.jpg')

kernel = np.ones((5, 5), np.uint8)
A_ero = cv2.erode(A, kernel, iterations=2)

plt.subplot(1, 2, 1)
plt.imshow(A, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(A_ero, cmap='gray')
plt.title('Erosion_Image')
plt.show()
```

→ Kết quả:



- Dilatation:

- + Syntax: **cv2.dilate(src, kernel, dst, anchor, iterations, borderType, borderValue)**
- + Dilate tám ảnh vừa được erode ở ví dụ trước:

```
> Morphological Image Processing > Dilation.py > ...
import cv2
import numpy as np
from matplotlib import pyplot as plt

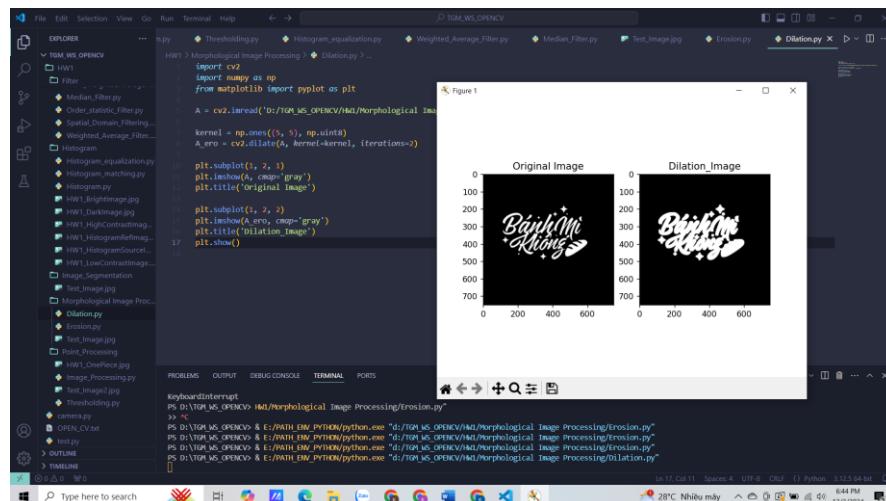
A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Morphological Image Processing/Test_Image.jpg')

kernel = np.ones((5, 5), np.uint8)
A_ero = cv2.dilate(A, kernel=kernel, iterations=2)

plt.subplot(1, 2, 1)
plt.imshow(A, cmap='gray')
plt.title('original Image')

plt.subplot(1, 2, 2)
plt.imshow(A_ero, cmap='gray')
plt.title('dilation_Image')
plt.show()
```

→ Kết quả:



- **Opening:** Opening là một phép erode rồi sau đó dilate. Nó rất hữu ích trong việc loại bỏ nhiễu. Ở đây chúng ta sử dụng hàm **cv2.morphologyEx()**

```
Morphological_Image_Processing > Opening.py > ...
import cv2
import numpy as np
from matplotlib import pyplot as plt

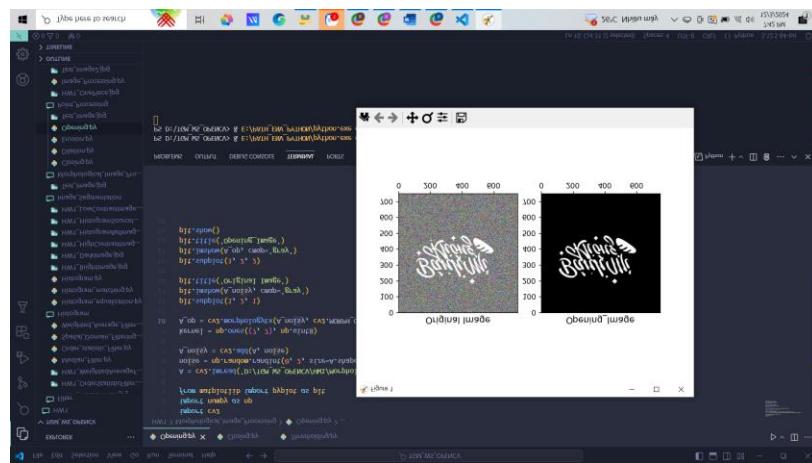
A = cv2.imread('D:/TGM WS OPENCV/HW1/Morphological_Image_Processing/Test_Image.jpg')
noise = np.random.randint(0, 2, size=A.shape, dtype=np.uint8) * 255
A_noisy = cv2.add(A, noise)

kernel = np.ones((7, 7), np.uint8)
A_op = cv2.morphologyEx(A_noisy, cv2.MORPH_OPEN, kernel)

plt.subplot(1, 2, 1)
plt.imshow(A_noisy, cmap='gray')
plt.title('Original Image')

plt.subplot(1, 2, 2)
plt.imshow(A_op, cmap='gray')
plt.title('Opening Image')
plt.show()
```

→ Kết quả:



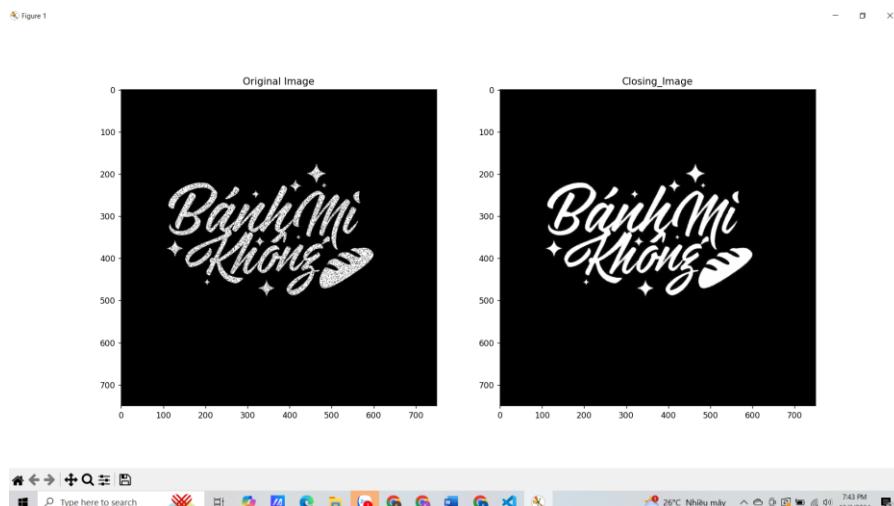
- **Closing:** Ngược lại với opening, closing gồm dilate trước và erode sau. Nó rất hữu ích trong việc đóng hay triệt tiêu đi các lỗ nhỏ bên trong vật thể.

```

Closing.py x
HW1 > Morphological_Image_Processing > Closing.py > ...
1 import cv2
2 import numpy as np
3 import random
4 from matplotlib import pyplot as plt
5
6 A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Morphological_Image_Processing/Test_Image.jpg')
7 if len(A.shape) == 3:
8     A = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
9 _, binary_image = cv2.threshold(A, 128, 255, cv2.THRESH_BINARY)
10
11 def add_black_noise(img, noise_value):
12     noisy_img = img.copy()
13     h, w = noisy_img.shape
14     for _ in range(noise_value):
15         x = random.randint(0, w - 1)
16         y = random.randint(0, h - 1)
17         noisy_img[y, x] = 0
18     return noisy_img
19
20 noisy_image = add_black_noise(A, noise_value=100000)
21 kernel = np.ones((3, 3), np.uint8)
22 closed_image = cv2.morphologyEx(noisy_image, cv2.MORPH_CLOSE, kernel)
23
24 plt.subplot(1, 2, 1)
25 plt.imshow(noisy_image, cmap='gray')
26 plt.title('Original Image')
27
28 plt.subplot(1, 2, 2)
29 plt.imshow(closed_image, cmap='gray')
30 plt.title('Closing Image')
31 plt.show()
32

```

→ Kết quả:



✓ Image Segmentation:

- Global thresholding (basic):

+ Syntax: cv2.threshold(src, thresh, maxval, type)

- Src: ảnh cần threshold, nên là ảnh xám.
- Thresh: mức threshold.

- Maxval: max value của pixel sau khi threshold.
- Type gồm các hàm sau: **cv2.THRESH_BINARY** (thường dùng), **cv2.THRESH_BINARY_INV**, **cv2.THRESH_TRUNC**, **cv2.THRESH_TOZERO**, **cv2.THRESH_TOZERO_INV**.

+ Phương thức này trả về 2 kết quả, đầu tiên là mức threshold đã dùng và hai là ảnh sau khi thresh hold.

+ Global Threshold với 5 kiểu khác nhau:

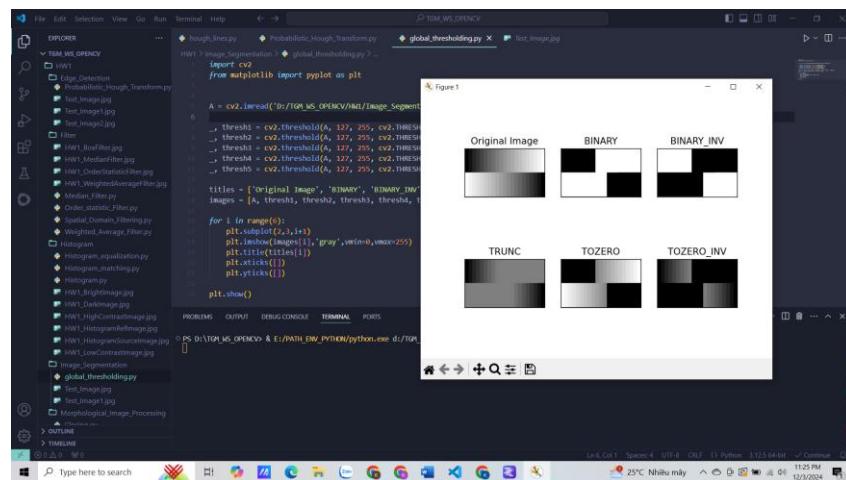


```

1  import cv2
2  from matplotlib import pyplot as plt
3
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Image_Segmentation/Test_Image1.jpg')
6
7  thresh1 = cv2.threshold(A, 127, 255, cv2.THRESH_BINARY)
8  thresh2 = cv2.threshold(A, 127, 255, cv2.THRESH_BINARY_INV)
9  thresh3 = cv2.threshold(A, 127, 255, cv2.THRESH_TRUNC)
10 thresh4 = cv2.threshold(A, 127, 255, cv2.THRESH_TOZERO)
11 thresh5 = cv2.threshold(A, 127, 255, cv2.THRESH_TOZERO_INV)
12
13 titles = ['Original Image', 'BINARY', 'BINARY_INV', 'TRUNC', 'TOZERO', 'TOZERO_INV']
14 images = [A, thresh1, thresh2, thresh3, thresh4, thresh5]
15
16 for i in range(6):
17     plt.subplot(2,3,i+1)
18     plt.imshow(images[i], 'gray', vmin=0, vmax=255)
19     plt.title(titles[i])
20     plt.xticks([])
21     plt.yticks([])
22
23 plt.show()

```

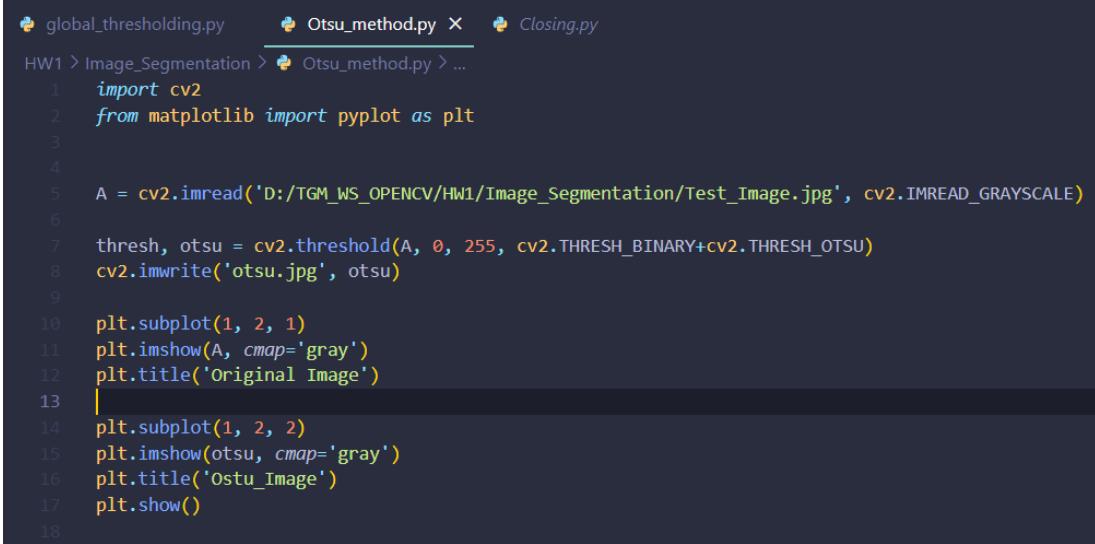
→ Kết quả:



- Otsu method:

+ Phương pháp Otsu sẽ tự động tìm mức Threshold phù hợp khi sử dụng global threshold từ histogram của tấm ảnh.

+ Để sử dụng phương pháp Otsu, chúng ta pass extra flag **cv2.THRESH_OTSU** vào phương thức **cv2.threshold()**. Threshold value có thể chọn bất kì, sau đó thuật toán sẽ tự động tìm ra threshold value phù hợp và trả về kết quả đầu tiên.

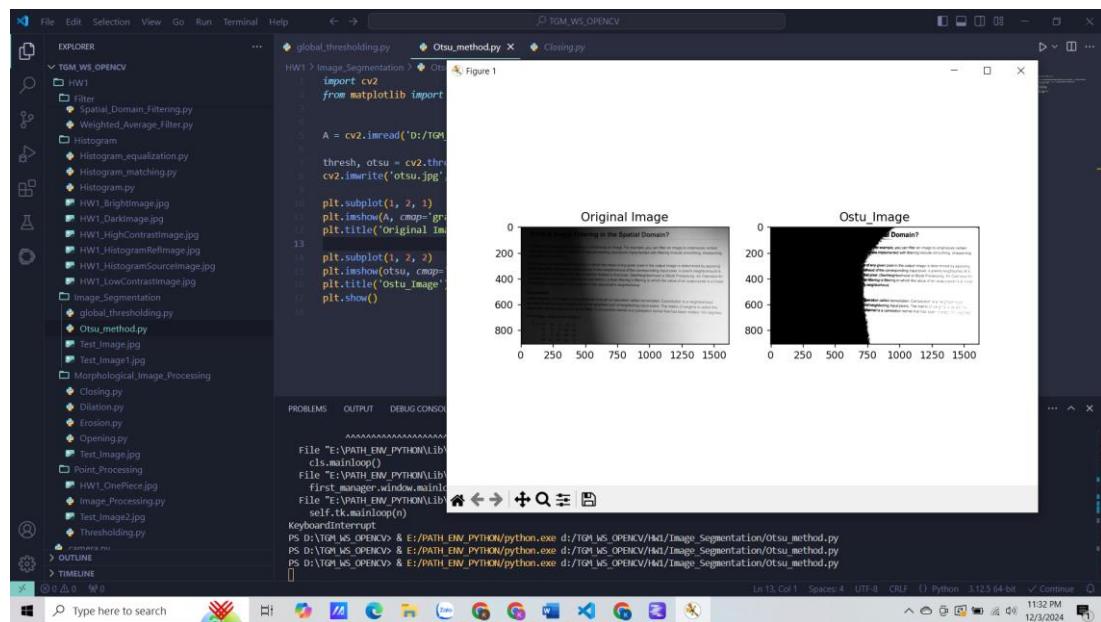


```

global_thresholding.py          Otsu_method.py ×      Closing.py
HW1 > Image_Segmentation > Otsu_method.py > ...
1  import cv2
2  from matplotlib import pyplot as plt
3
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Image_Segmentation/Test_Image.jpg', cv2.IMREAD_GRAYSCALE)
6
7  thresh, otsu = cv2.threshold(A, 0, 255, cv2.THRESH_BINARY+cv2.THRESH_OTSU)
8  cv2.imwrite('otsu.jpg', otsu)
9
10 plt.subplot(1, 2, 1)
11 plt.imshow(A, cmap='gray')
12 plt.title('Original Image')
13 |
14 plt.subplot(1, 2, 2)
15 plt.imshow(otsu, cmap='gray')
16 plt.title('Ostu_Image')
17 plt.show()
18

```

→ Kết quả:



- Adaptive Threshold:

+ Trong phần 1 và 2 chúng ta đã sử dụng phương pháp global thresholding. Nhưng phương pháp này không tốt cho mọi trường hợp, ví dụ như tấm ảnh có điều kiện ánh sáng khác nhau ở những vùng khác nhau. Trong trường hợp này chúng ta có thể sử dụng adaptive thresholding để xử lý. Thuật toán này tính giá

trị threshold cho từng vùng ảnh nhỏ, do đó chúng ta có được những ngưỡng threshold khác nhau cho từng vùng ảnh khác nhau, từ đó cho ra kết quả tốt hơn.

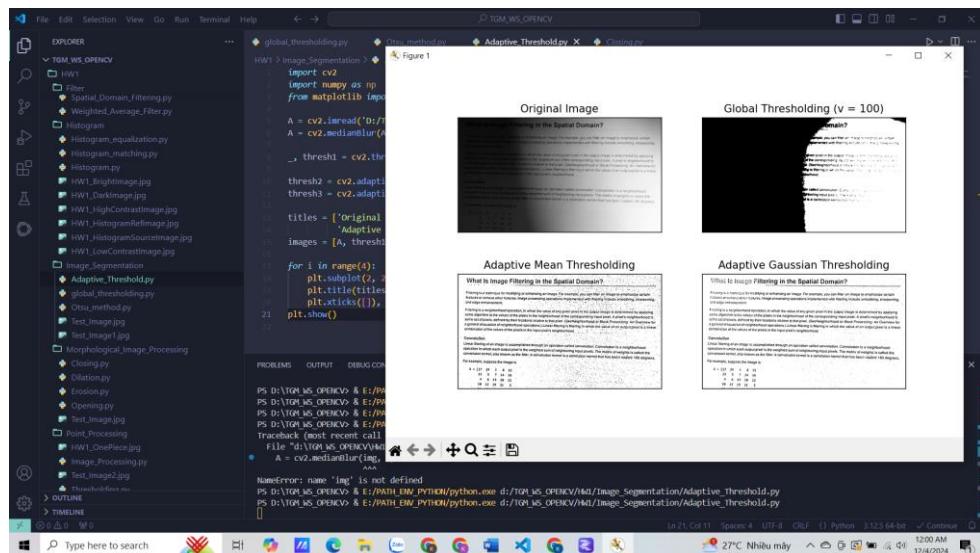
+ Ngoài những tham số như hàm global threshold, method **cv.adaptiveThreshold()** có thêm 3 tham số:

- **adaptiveMethod:** **cv.ADAPTIVE_THRESH_MEAN_C** threshold value được tính bằng trung bình cộng của vùng ảnh trừ đi hằng số **C**, **cv.ADAPTIVE_THRESH_GAUSSIAN_C** threshold value bằng tổng gaussian có trọng số trừ đi hằng số **C**.
- **blockSize:** kích thước của vùng ảnh tính threshold value.
- **C:** hằng số.

+ So sánh kết quả của 2 phương pháp basic và adaptive threshold.

```
global_thresholding.py      Otsu_method.py      Adaptive_Threshold.py  Closing.py
W1 > Image_Segmentation > Adaptive_Threshold.py > ...
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Image_Segmentation/Test_Image.jpg', cv2.IMREAD_GRAYSCALE)
6  A = cv2.medianBlur(A, 5)
7
8  _, thresh1 = cv2.threshold(A, 127, 255, cv2.THRESH_BINARY)
9
10 thresh2 = cv2.adaptiveThreshold(A, 255, cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)
11 thresh3 = cv2.adaptiveThreshold(A, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)
12
13 titles = ['Original Image', 'Global Thresholding (v = 100)', 'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
14 images = [A, thresh1, thresh2, thresh3]
15
16 for i in range(4):
17     plt.subplot(2, 2, i+1), plt.imshow(images[i], 'gray')
18     plt.title(titles[i])
19     plt.xticks([]), plt.yticks([])
20
21 plt.show()
```

→ Kết quả:



✓ Edge detection:

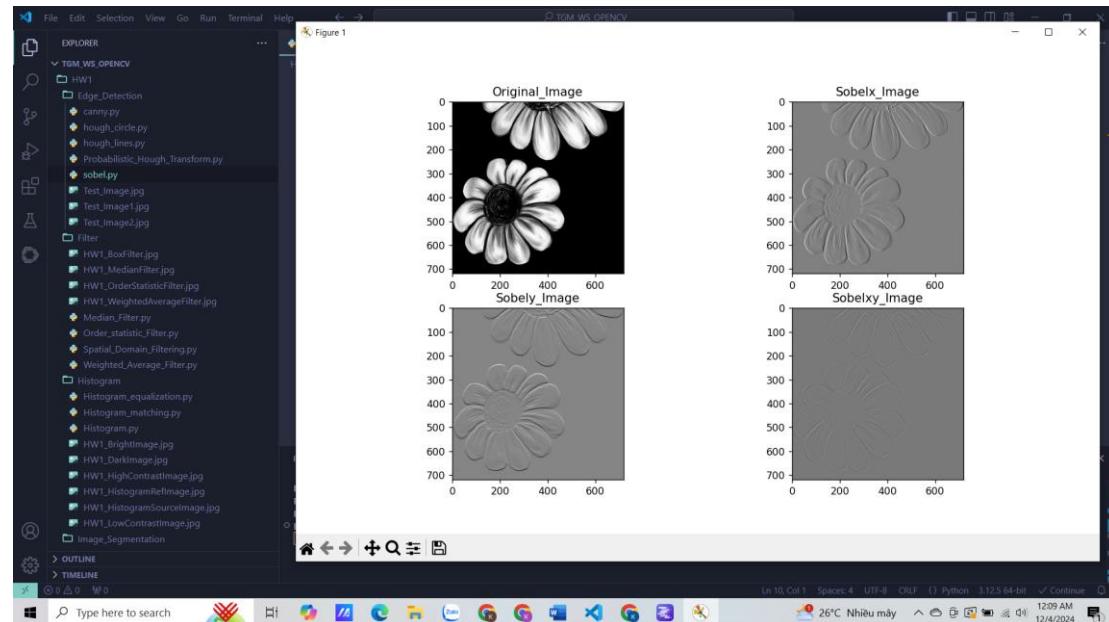
- Sobel: là sự kết hợp giữa Gaussian smoothing cộng với phép đạo hàm rời rạc. Nó tính toán xấp xỉ hàm gradient của một bức ảnh.
 - + Syntax: **cv2.Sobel(src, dst, ddepth, dx, dy, ksize)**
 - + ddepth: số nguyên đại diện cho chiều sâu bức ảnh (mặc định – 1).
 - + (dx,dy) = (1,0): đạo hàm theo chiều x.
 - + (dx,dy) = (0,1): đạo hàm theo chiều y.
 - + (dx,dy) = (1,1): đạo hàm theo cả 2 chiều x và y.
 - + ksize: kích thước kernel.
 - + Sử dụng kernel 5x5 và depth = -1 để có kết quả là kiểu np.uint8.

```

sobel.py  X  hough_circle.py  canny.py
HW1 > Edge_Detection > sobel.py > ...
1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
4
5 A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Edge_Detection/Test_Image.jpg', cv2.IMREAD_GRAYSCALE)
6
7 sobelx = cv2.Sobel(A, cv2.CV_64F, 1, 0, ksize=5)
8 sobely = cv2.Sobel(A, cv2.CV_64F, 0, 1, ksize=5)
9 sobelxy = cv2.Sobel(A, cv2.CV_64F, 1, 1, ksize=5)
10
11 plt.subplot(2, 2, 1)
12 plt.imshow(A, cmap='gray')
13 plt.title('Original_Image')
14
15 plt.subplot(2, 2, 2)
16 plt.imshow(sobelx, cmap='gray')
17 plt.title('Sobelx_Image')
18
19 plt.subplot(2, 2, 3)
20 plt.imshow(sobely, cmap='gray')
21 plt.title('Sobely_Image')
22
23 plt.subplot(2, 2, 4)
24 plt.imshow(sobelxy, cmap='gray')
25 plt.title('Sobelxy_Image')
26 plt.show()

```

→ Kết quả:



- **Canny:** Thuật toán Canny bao gồm các bước:

- + Khử nhiễu: sử dụng 5x5 Gaussian filter.
- + Dùng **cv2.Canny()** với 2 mức threshold là 50 và 150.

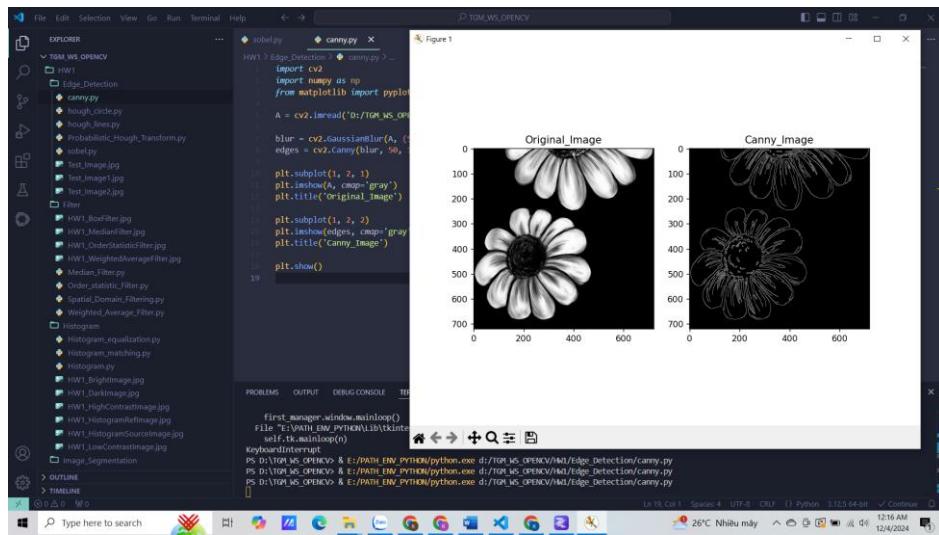


```

sobel.py          canny.py
HW1 > Edge_Detection > canny.py > ...
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Edge_Detection/Test_Image.jpg', cv2.IMREAD_GRAYSCALE)
6
7  blur = cv2.GaussianBlur(A, (5, 5), 1.4) # Áp dụng bộ lọc Gaussian để giảm nhiễu
8  edges = cv2.Canny(blur, 50, 150)
9
10 plt.subplot(1, 2, 1)
11 plt.imshow(A, cmap='gray')
12 plt.title('Original_Image')
13
14 plt.subplot(1, 2, 2)
15 plt.imshow(edges, cmap='gray')
16 plt.title('Canny_Image')
17
18 plt.show()
19

```

→ Kết quả:



- Hough Line:

Hough Transform:

Syntax: `lines = cv2.HoughLines(src, rho, theta, threshold)`

+ `lines`: vector chứa các tham số (`rho, theta`) của những đường thẳng được phát hiện.

+ `src`: kết quả của edge detector, nên là ảnh xám (thực tế là ảnh binary).

+ `rho`: độ phân giải của `rho` theo pixel.

+ `theta`: độ phân giải của `theta` theo radian.

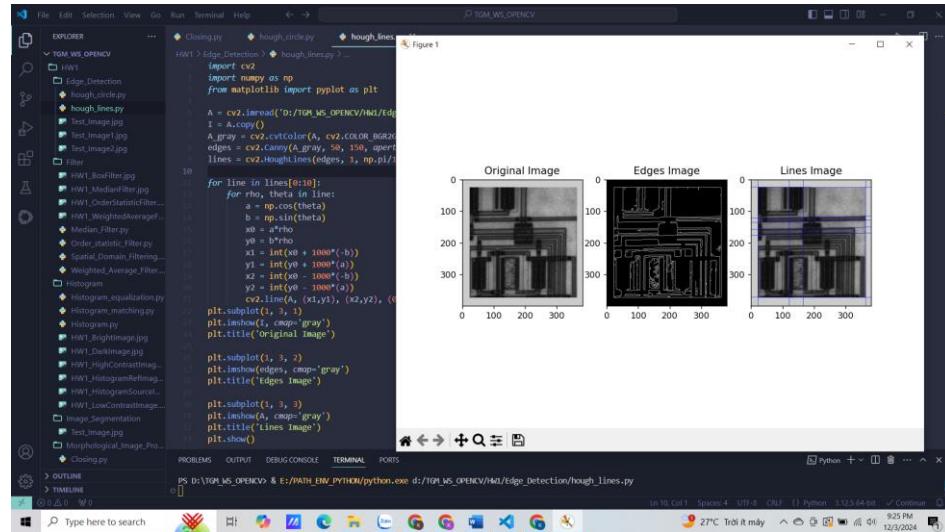
+ `threshold`: số giao điểm nhỏ nhất để được xem là một đường thẳng.

```

Closing.py          hough_circle.py      hough_lines.py ×
HW1 > Edge_Detection > hough_lines.py > ...
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Edge_Detection/Test_Image1.jpg')
6  I = A.copy()
7  A_gray = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
8  edges = cv2.Canny(A_gray, 50, 150, apertureSize=3)
9  lines = cv2.HoughLines(edges, 1, np.pi/180, 100)
10 |
11 for line in lines[0:10]:
12     for rho, theta in line:
13         a = np.cos(theta)
14         b = np.sin(theta)
15         x0 = a*rho
16         y0 = b*rho
17         x1 = int(x0 + 1000*(-b))
18         y1 = int(y0 + 1000*(a))
19         x2 = int(x0 - 1000*(-b))
20         y2 = int(y0 - 1000*(a))
21         cv2.line(A, (x1,y1), (x2,y2), (0,0,255), 1)
22 plt.subplot(1, 3, 1)
23 plt.imshow(I, cmap='gray')
24 plt.title('Original Image')
25
26 plt.subplot(1, 3, 2)
27 plt.imshow(edges, cmap='gray')
28 plt.title('Edges Image')
29
30 plt.subplot(1, 3, 3)
31 plt.imshow(A, cmap='gray')
32 plt.title('Lines Image')
33 plt.show()

```

→ Kết quả:



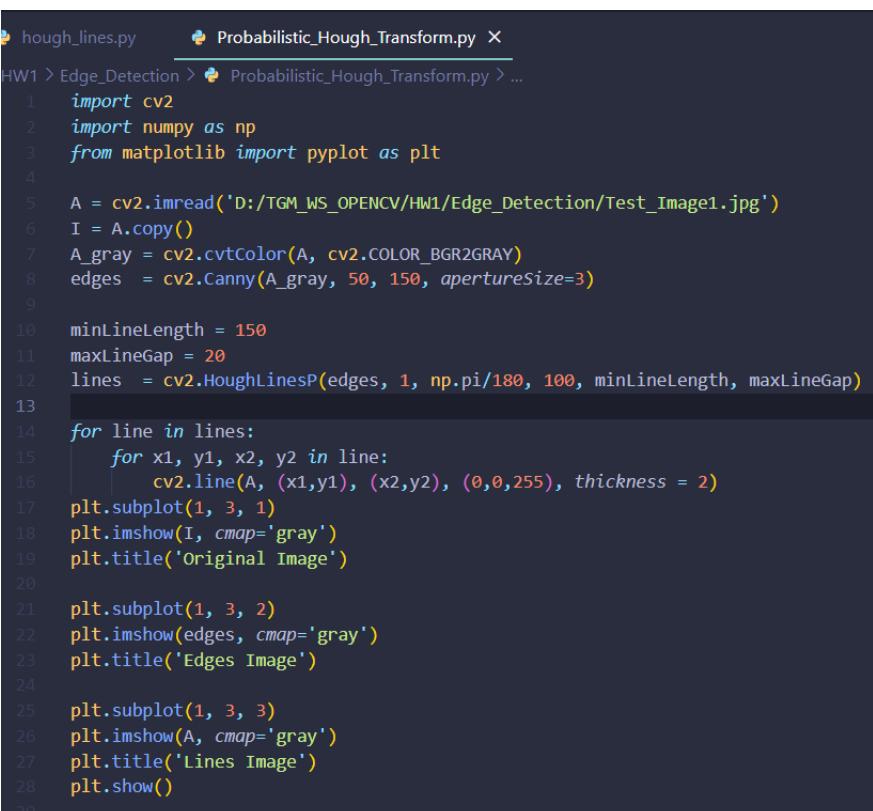
- Probabilistic Hough Transform:

+ Với Hough Transform, với mỗi đường thẳng có 2 thông số cũng cần rất nhiều tính toán. Probabilistic Hough Transform là dạng tối ưu của Hough Transform.

Nó không lấy tất cả mọi điểm để tính toán mà lấy một tập ngẫu nhiên các điểm đủ để tạo thành một đường thẳng.

Syntax: **cv2.HoughLinesP()**. Có 2 tham số:

- + minLineLength: chiều dài ngắn nhất của đường thẳng. Đoạn thẳng nào ngắn hơn thì bị loại bỏ.
- + maxLineGap: khoảng trống cho phép lớn nhất giữa 2 đoạn thẳng để vẫn được xem là một đường thẳng và hàm này trả về 2 điểm của 2 đầu đường thẳng.

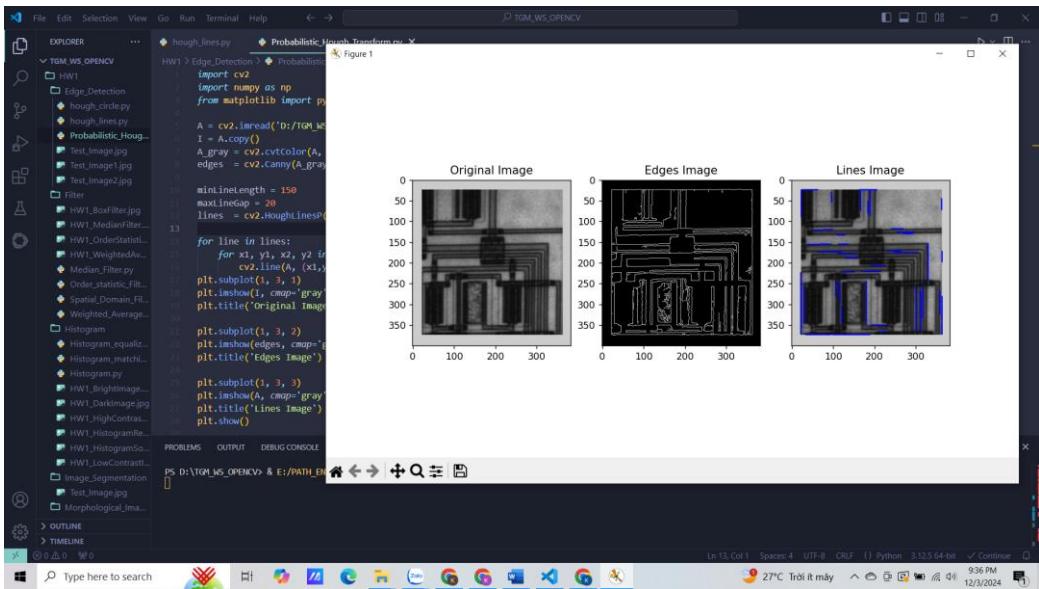


```

hough_lines.py  Probabilistic_Hough_Transform.py X
HW1 > Edge_Detection > Probabilistic_Hough_Transform.py > ...
1  import cv2
2  import numpy as np
3  from matplotlib import pyplot as plt
4
5  A = cv2.imread('D:/TGM_WS_OPENCV/HW1/Edge_Detection/Test_Image1.jpg')
6  I = A.copy()
7  A_gray = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
8  edges = cv2.Canny(A_gray, 50, 150, apertureSize=3)
9
10 minLineLength = 150
11 maxLineGap = 20
12 lines = cv2.HoughLinesP(edges, 1, np.pi/180, 100, minLineLength, maxLineGap)
13
14 for line in lines:
15     for x1, y1, x2, y2 in line:
16         cv2.line(A, (x1,y1), (x2,y2), (0,0,255), thickness = 2)
17 plt.subplot(1, 3, 1)
18 plt.imshow(I, cmap='gray')
19 plt.title('Original Image')
20
21 plt.subplot(1, 3, 2)
22 plt.imshow(edges, cmap='gray')
23 plt.title('Edges Image')
24
25 plt.subplot(1, 3, 3)
26 plt.imshow(A, cmap='gray')
27 plt.title('Lines Image')
28 plt.show()

```

→ Kết quả:



- Hough Circle:

Syntax: **cv2.HoughCircles(image, circles, method, dp, minDist, param1, param2, minRadius, maxRadius)**

+ method: **HOUGH_GRADIENT** hoặc **HOUGH_GRADIENT_ALT**.

+ dp: tỉ số của độ phân giải bức ảnh với độ phân giải của accumulator. Ví dụ nếu dp = 1 thì cả 2 cùng độ phân giải. Nếu dp = 2 thì chiều dài và chiều rộng của bức ảnh gấp đôi của accumulator. Với **HOUGH_GRADIENT_ALT**, dp phù hợp là 1.5 trừ khi cần phát hiện một số vòng tròn nhỏ

+ minDist: khoảng cách ngắn nhất giữa các tâm của các đường tròn. Nếu tham số quá nhỏ, nhiều vòng tròn lân cận có thể bị phát hiện sai ngoài một vòng tròn đúng. Nếu nó quá lớn, một số vòng tròn có thể bị bỏ sót.

+ Param1: trong trường hợp **HOUGH_GRADIENT** hoặc **HOUGH_GRADIENT_ALT** thì param1 là ngưỡng threshold cao đưa vào Canny detector (ngưỡng thấp hơn thì bằng một nửa).

+ Param2: là ngưỡng threshold của accumulator cho tâm những đường tròn. Càng nhỏ thì càng phát hiện nhiều đường tròn sai. Giá trị này càng cao thì yêu cầu độ chính xác của đường tròn càng cao, giảm số lượng vòng tròn được phát hiện

+ minRadius: bán kính nhỏ nhất của đường tròn

+ maxRadius: bán kính lớn nhất của đường tròn

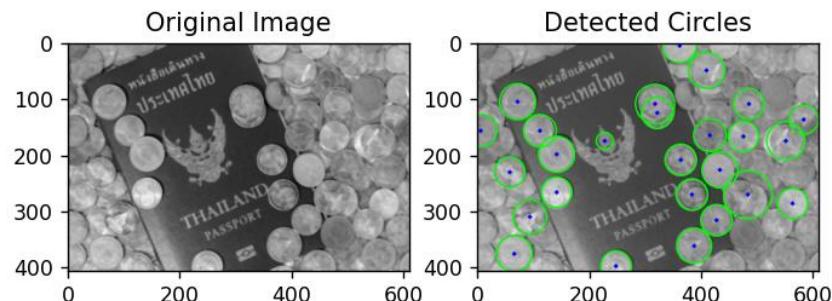
```

Closing.py      hough_circle.py x
HW1 > Edge_Detection > hough_circle.py > ...
1   import cv2
2   import numpy as np
3   from matplotlib import pyplot as plt
4
5   img = cv2.imread('D:/TGM_WS_OPENCV/HW1/Edge_Detection/Test_Image2.jpg', cv2.IMREAD_GRAYSCALE)
6   img = cv2.medianBlur(img, 5)
7   cimg = cv2.cvtColor(img, cv2.COLOR_GRAY2BGR)
8
9   circles = cv2.HoughCircles(
10     img,
11     cv2.HOUGH_GRADIENT,
12     dp=1,
13     minDist=15,
14     param1=100, #Giá trị càng lớn thì giảm bớt các cạnh yếu, từ đó giảm số lượng vòng tròn được phát hiện
15     param2=30, #Giá trị này càng cao thì yêu cầu độ chính xác của đường tròn càng cao, giảm số lượng vòng tròn được phát hiện
16     minRadius=10,
17     maxRadius=50
18 )
19
20 if circles is not None:
21   circles = np.uint16(np.around(circles))
22   for i in circles[0, :]:
23     cv2.circle(cimg, (i[0], i[1]), i[2], (0, 255, 0), 2) # Vẽ vòng tròn bên ngoài
24     cv2.circle(cimg, (i[0], i[1]), 2, (0, 0, 255), 3) # Vẽ tâm của đường tròn
25
26 plt.subplot(1, 2, 1)
27 plt.imshow(img, cmap='gray')
28 plt.title('Original Image')
29
30 plt.subplot(1, 2, 2)
31 plt.imshow(cimg, cmap='gray')
32 plt.title('Detected Circles')
33 plt.show()

```

→ Kết quả:

Figure 1

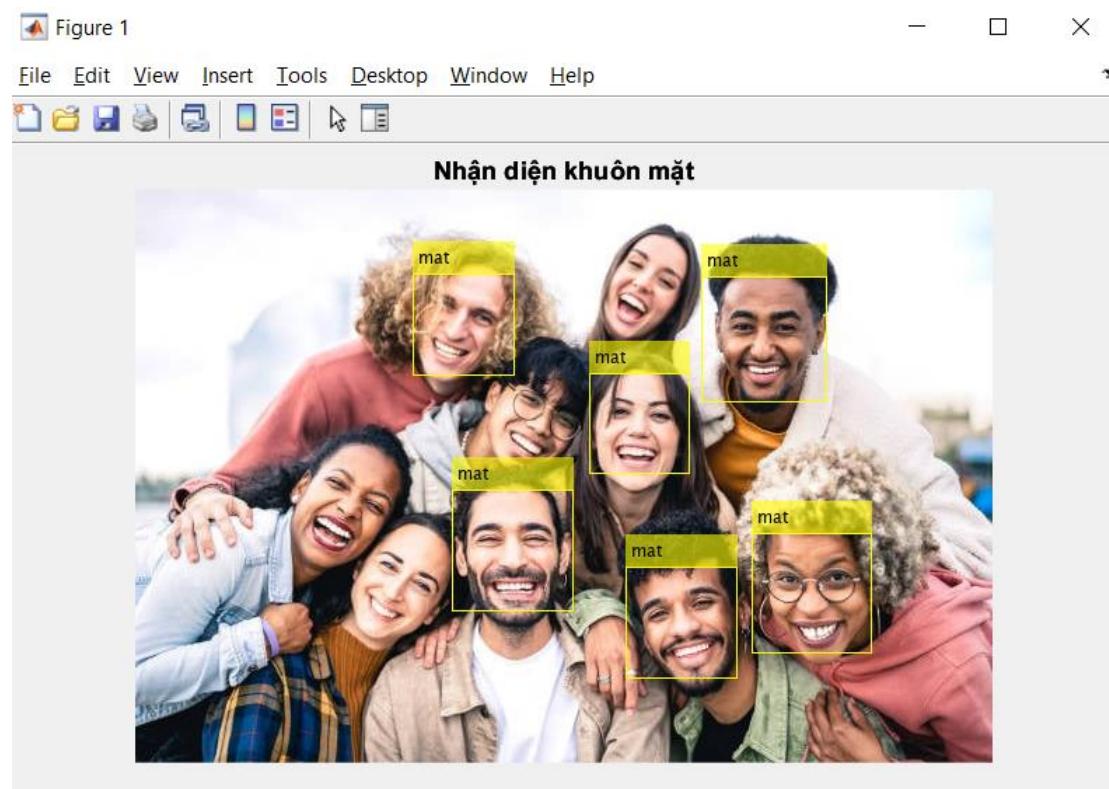


2. Homework 2: Use built-in functions in MATLAB such as `vision.CascadeObjectDetector` to detect several objects like face, nose, mouth, and eyes, then derive metrics like precision, recall, and accuracy.

2.1 Thử với model nhận diện khuôn mặt

```
FaceDetector = vision.CascadeObjectDetector;
I = imread('D:\TGM_WS_MATLAB\TestImages\Test_Image21.jpg');
bboxes = FaceDetector(I);
IFaces = insertObjectAnnotation(I,'rectangle',bboxes,'mat');
figure
imshow(IFaces)
title('Nhận diện khuôn mặt');
```

Kết quả:



Nhận xét:

Thuật toán còn yếu khi nhận diện thiết khuôn mặt bởi những lí do như đeo kính, bị che khuất, đội nón, góc nghiêng của mặt ...

2.2 Thủ với model nhận diện miệng

```

MouthDetector = vision.CascadeObjectDetector('Mouth');

A = imread('D:\TGM_WS_MATLAB\TestImages\Test_Image22.jpg');

bboxes = MouthDetector(A);

AMouths= insertObjectAnnotation(A,'rectangle',bboxes,'mieng');

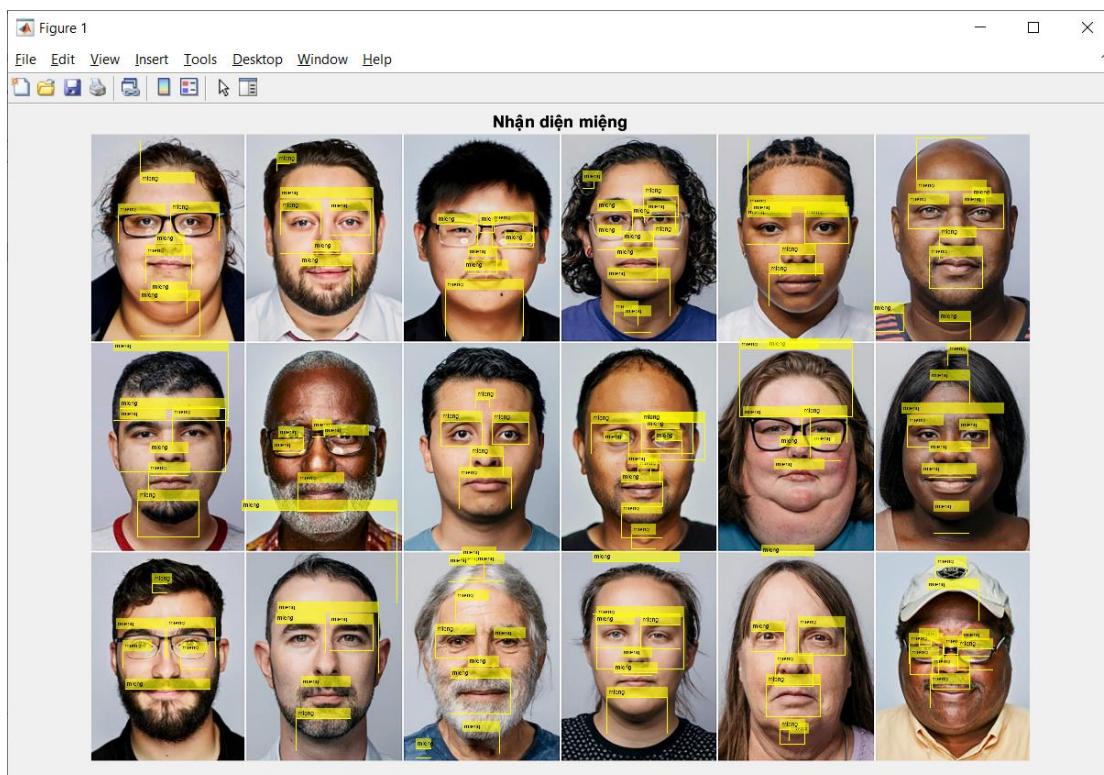
figure

imshow(AMouths)

title('Nhận diện miệng');

```

Kết quả:



Nhận xét:

Model còn yếu và không phân biệt được mắt và miệng cũng như các nơi có số frame và số pixel hơi giống so với miệng.

2.3 Thử với model nhận diện mũi

```

NoseDetector = vision.CascadeObjectDetector('Nose');

B = imread('D:\TGM_WS_MATLAB\TestImages\Test_Image22.jpg');

bodyDetector.MinSize = [0.005 0.005];

bboxes = NoseDetector(B);

BNoses = insertObjectAnnotation(B,'rectangle',bboxes,'mui');

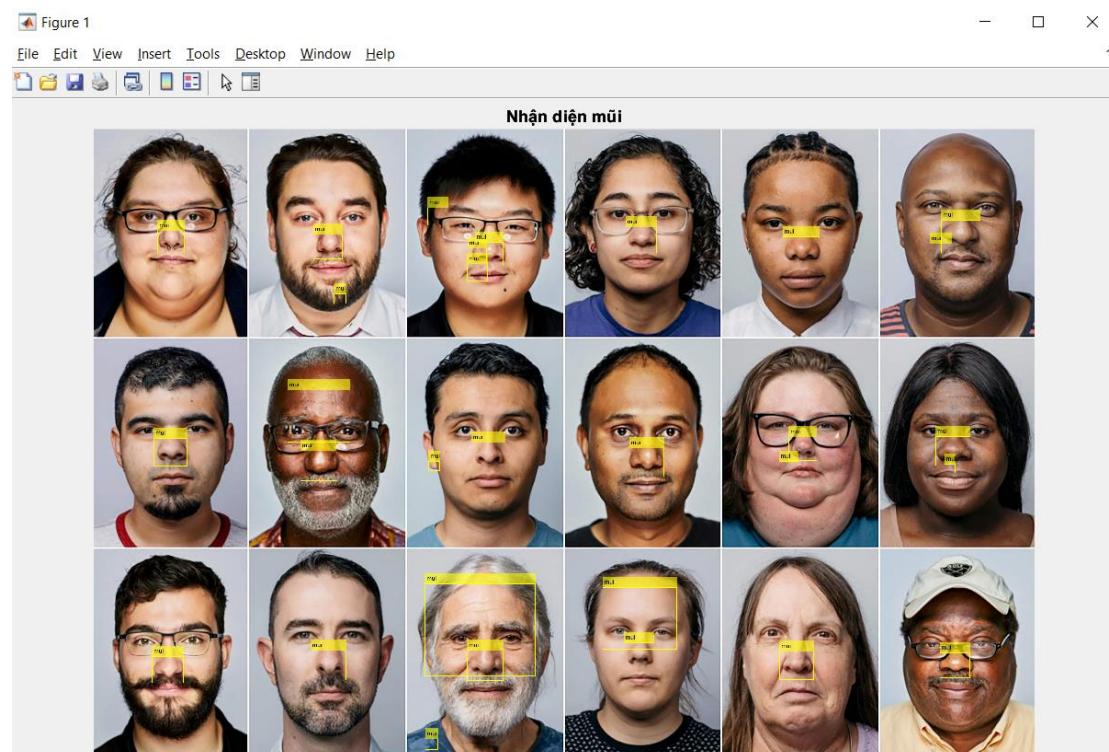
figure

imshow(BNoses)

title('Nhận diện mũi');

```

Kết quả:



Nhận xét:

Model còn yếu do nhận diện có các lỗi sai như nhận diện trán ra mũi. Bounding box không bao đầy đủ vật thể.

2.4 Thử với model nhận diện mắt

```

EyeDetector = vision.CascadeObjectDetector('EyePairBig');

C = imread('D:\TGM_WS_MATLAB\TestImages\Test_Image22.jpg');

bodyDetector.MinSize = [0.005 0.005];

bboxes = EyeDetector(C);

CEyes = insertObjectAnnotation(C,'rectangle',bboxes,'mat');

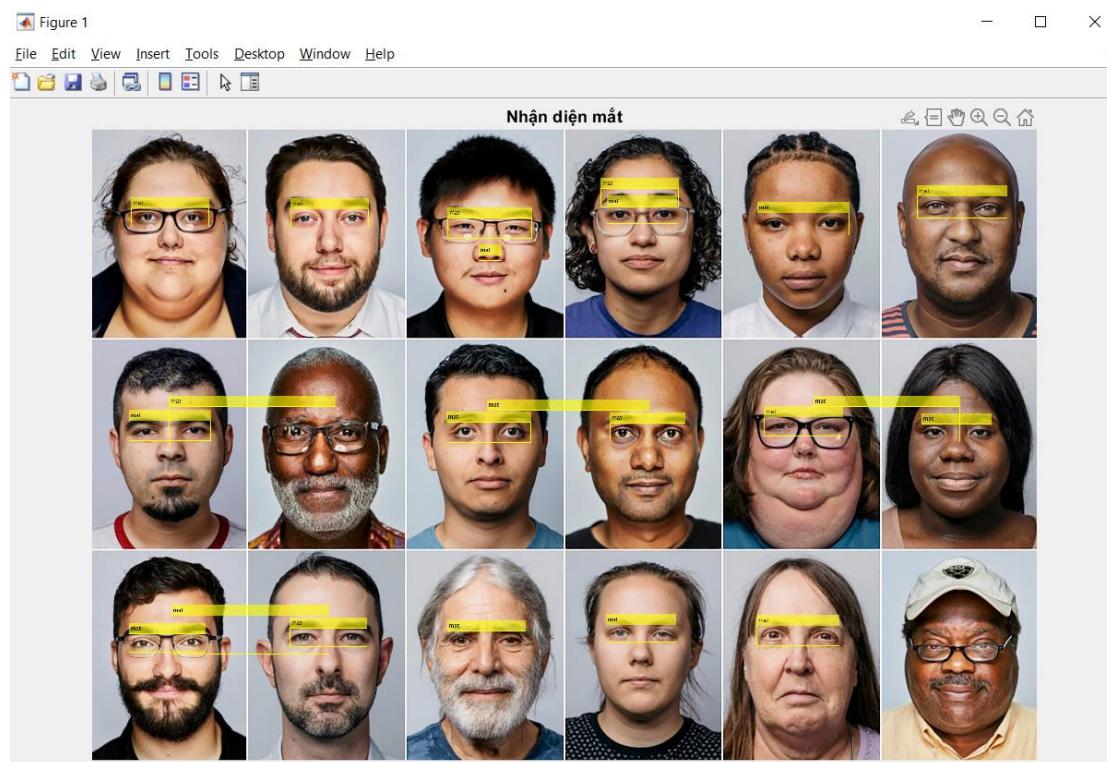
figure

imshow(CEyes)

title('Nhận diện mắt');

```

Kết quả:



Nhận xét:

Model còn yếu gắp một số lỗi như nhận diện mũi và trán là mắt. Đôi lúc không nhận diện được mắt bị che bởi kính.

Tính các chỉ số precision, recall, and accuracy với model nhận diện “khuôn mặt”

Thông tin đầu vào:

1. Ground truth (thực tế): 10 khuôn mặt trong ảnh.
2. Predictions (dự đoán): 6 khung (tất cả đều đúng).
3. True Positives (TP): 6 khung nhận diện chính xác.
4. True Negatives (TN): Là những vùng trong ảnh không chứa đối tượng (khuôn mặt) và cũng không bị thuật toán nhầm lẫn là đối tượng. Vì thế chỉ số TN bị bỏ qua vì nó không ảnh hưởng trực tiếp đến hiệu quả phát hiện các đối tượng.
5. False Positives (FP): 0 (vì không có khung nào sai).
6. False Negatives (FN): 4 (4 khuôn mặt bị bỏ sót).

Công thức và tính toán:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{6}{6 + 0} = 1$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{6}{6 + 4} = 0.6$$

$$\text{Accuracy} = \frac{TP}{TP + FP + FN} = \frac{6}{6 + 0 + 4} = 0.6$$

Kết quả:

- Precision: 100%
- Recall: 60%
- Accuracy: 60%

Kết luận:

Thuật toán hoạt động chính xác với những gì nó phát hiện (Precision = 100%), nhưng tỷ lệ phát hiện so với số khuôn mặt thực tế (Recall) và tổng độ chính xác (Accuracy) vẫn cần cải thiện.

Tính các chỉ số precision, recall, and accuracy với model nhận diện “mũi”

Thông tin đầu vào:

7. Ground truth (thực tế): 18 khuôn mặt trong ảnh.
8. Predictions (dự đoán): 30 khung.
9. True Positives (TP): 18 khung nhận diện chính xác.
10. True Negatives (TN): Là những vùng trong ảnh không chứa đối tượng (mũi) và cũng không bị thuật toán nhầm lẫn là đối tượng. Vì thế chỉ số TN bị bỏ qua vì nó không ảnh hưởng trực tiếp đến hiệu quả phát hiện các đối tượng.
11. False Positives (FP): 12 (vì có khung nhận diện không phải mũi).
12. False Negatives (FN): 0 (không có khuôn mặt nào bị bỏ sót).

Công thức và tính toán:

$$\text{Precision} = \frac{TP}{TP + FP} = \frac{18}{18 + 12} = 0.6$$

$$\text{Recall} = \frac{TP}{TP + FN} = \frac{18}{18 + 0} = 1$$

$$\text{Accuracy} = \frac{TP}{TP + FP + FN} = \frac{18}{18 + 12 + 0} = 0.6$$

Kết quả:

- Precision: 100%
- Recall: 60%
- Accuracy: 60%

Kết luận:

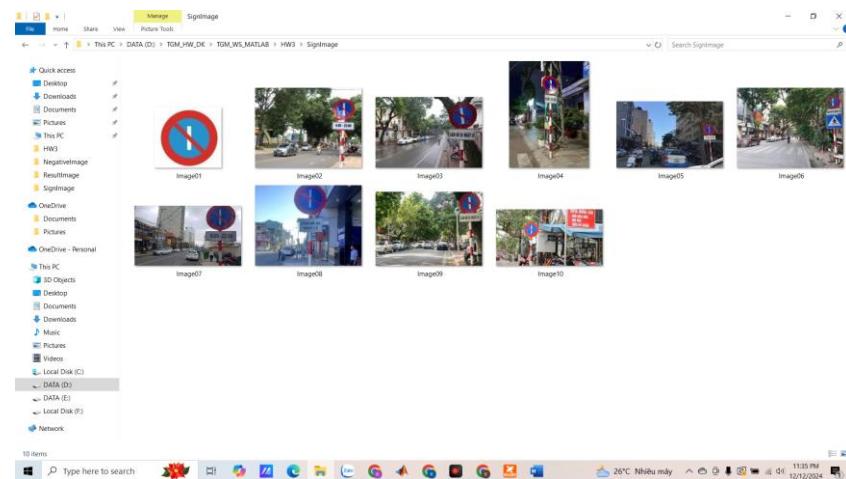
Precision thấp do mô hình tạo ra nhiều phát hiện sai (12 FP). Recall cao vì tất cả mũi đều được phát hiện, không bị bỏ sót.

3. Homework 3: HW1: Viola-Jones algorithm: Training

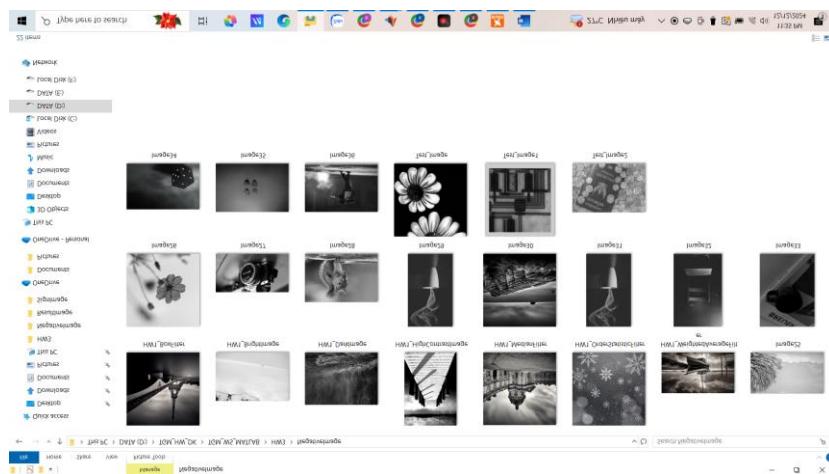
Ta huấn luyện mô hình phát hiện biến báo dùng giải thuật Viola-Jones.

Bước 1: Chuẩn bị dữ liệu

- Tập dữ liệu chứa biển báo:

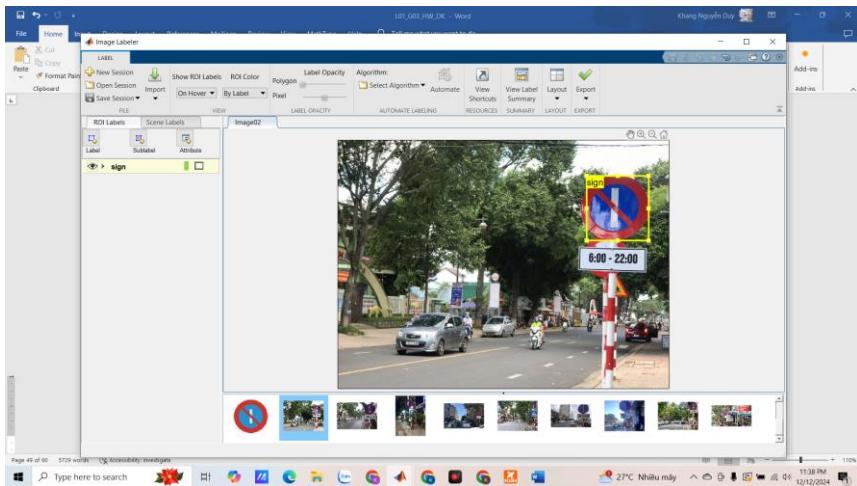


- Tập dữ liệu không có mặt biển báo:



Bước 2: Gán nhãn cho dữ liệu sử dụng Image Labeler trong Matlab

- Thực hiện Import dữ liệu và chọn toàn bộ ảnh trong folder SignImage để gán nhãn Label.
- Chọn Export -> To File để tạo file .mat chứa 4 thông số biểu diễn tọa độ, kích thước bounding box và tên label.



Bước 3: Huấn luyện mô hình detect biển báo

- Load đường dẫn lưu dữ liệu và thông tin về bounding box của các ảnh sau khi gán nhãn

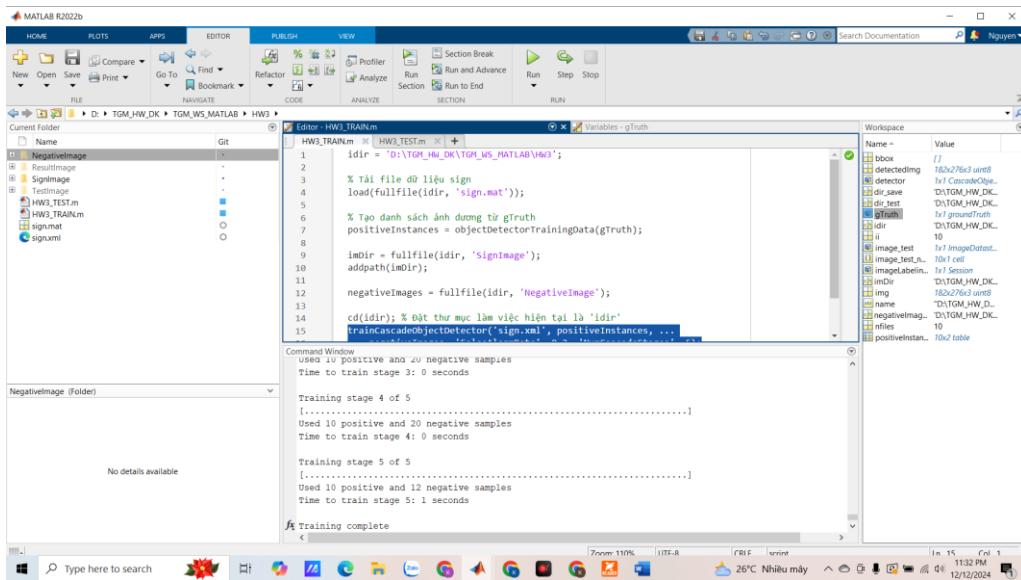
```
idir = 'D:\TGM_HW_DK\TGM_WS_MATLAB\HW3';
load(fullfile(idir, 'sign.mat'));
positiveInstances = objectDetectorTrainingData(gTruth);
```

- Thêm đường dẫn chứa ảnh vào Matlab

```
imDir = fullfile(idir, 'SignImage');
addpath(imDir);
```

- Huấn luyện mô hình với giải thuật Viola-Jones. Thông số FalseAlarmRate là giá trị False Positive trong Confusion Matrix. Thông số NumCascadeStages là số tầng của bộ phân loại Cascade (dữ liệu càng lớn thì số tầng càng nhiều).

```
trainCascadeObjectDetector('sign.xml', positiveInstances, ...
negativeImages, 'FalseAlarmRate', 0.2, 'NumCascadeStages', 5);
```



- Dùng mô hình đã huấn luyện để phát hiện biển báo:

```
idir = 'D:\TGM_HW_DK\TGM_WS_MATLAB\HW3';
```

```
detector = vision.CascadeObjectDetector('sign.xml');
```

```
dir_test = fullfile(idir, 'TestImage');
```

```
dir_save = fullfile(idir, 'ResultImage');
```

% Đọc tất cả các ảnh kiểm tra

```
image_test = image datastore(dir_test);
```

```
image_test_name = image_test.Files(:,1);
```

% Số lượng ảnh kiểm tra

```
nfiles = length(image_test_name);
```

```
for ii = 1:nfiles
```

```
img = imread(string(image_test_name(ii,1)));
```

% Phát hiện đối tượng

```
bbox = step(detector, img);
```

% Vẽ khung chữ nhật bao quanh

```
detectedImg = insertObjectAnnotation(img, 'rectangle', bbox, 'Sign');
```

```
% Tạo tên file kết quả
```

```
name = fullfile(dir_save, sprintf("test_%d.jpg", ii));
```

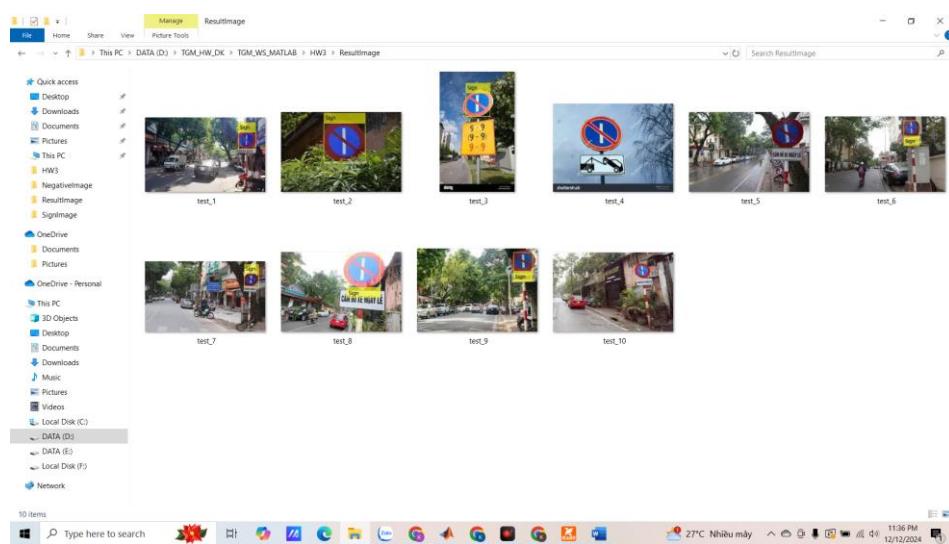
```
% Lưu ảnh
```

```
imwrite(detectedImg, name, "Quality", 100);
```

```
end
```

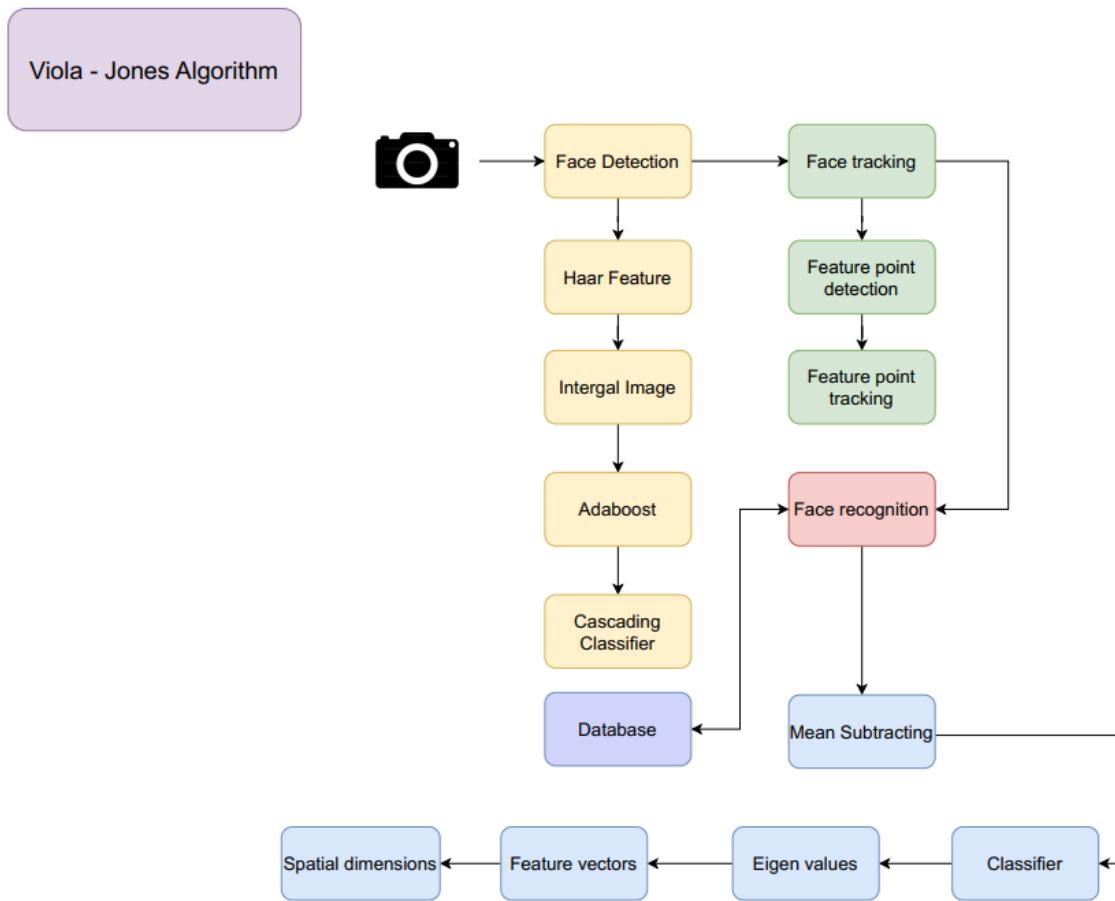
```
disp('Processing completed. Check the result folder for output images.');
```

- Kết quả thu được trong folder ResultImage:



=> Mô hình dự đoán được 7/10 ảnh có mặt biển báo với góc nhìn trực diện. Những ảnh còn lại không thể detect ra vì 2 lý do: ảnh có góc nhìn nghiêng và dữ liệu huấn luyện chưa đủ nhiều.

4. Homework 4: Create a mind map for the Viola-Jones Algorithm



Viola-Jones Algorithm

Thuật toán Viola-Jones là nền tảng cho việc phát hiện khuôn mặt trong ảnh. Được chia thành các khối chính: Face Detection, Haar Feature, Integral Image, Adaboost, và Cascade Classifier. Những khối này liên kết chặt chẽ với nhau, tạo thành một quy trình phát hiện nhanh và chính xác.

Face Detection (Phát hiện khuôn mặt)

- Mục tiêu: Xác định vị trí khuôn mặt trong ảnh hoặc video.
- Cơ chế: Sử dụng cửa sổ trượt trên ảnh với kích thước 24x24 pixel để kiểm tra từng vùng ảnh xem có khuôn mặt hay không.
- Phân loại nhị phân: Mỗi cửa sổ được phân loại là "face" hoặc "non-face" bằng bộ phân loại được huấn luyện trước.

Haar Feature

- Đặc trưng Haar: Là các mẫu hình chữ nhật, mô tả sự khác biệt về độ sáng giữa các vùng trong ảnh. Có ba loại đặc trưng chính:
 - Cạnh (Edge): Phân biệt giữa vùng sáng và tối.
 - Dải (Line): Xác định các chi tiết dải sáng-tối xen kẽ.
 - Hình chữ nhật: Phân biệt các vùng đối lập nhau.

Integral Image

- Khái niệm: Một kỹ thuật để tính tổng pixel trong một vùng chữ nhật bất kỳ trên ảnh một cách nhanh chóng.
- Lợi ích: Tăng tốc độ tính toán các đặc trưng Haar, vì chỉ cần cộng/trừ các giá trị tích phân thay vì tính toán trực tiếp trên từng pixel.
- Ứng dụng: Tạo điều kiện để xử lý hàng trăm nghìn đặc trưng Haar trong thời gian thực.

Adaboost (Adaptive Boosting)

- Mục đích: Tăng cường độ mạnh của bộ phân loại bằng cách kết hợp nhiều bộ phân loại yếu.
- Cơ chế:
 - Huấn luyện nhiều bộ phân loại yếu (Weak Classifiers), mỗi bộ dựa trên một đặc trưng Haar.
 - Kết hợp các bộ phân loại yếu thành một bộ phân loại mạnh (Strong Classifier).
- Hiệu quả: Chỉ cần 200 bộ phân loại yếu để tạo thành bộ phân loại cuối cùng với độ chính xác cao, giảm thiểu False positive.

Cascade Classifier (Bộ phân tầng)

- Ý tưởng: Tổ chức bộ phân loại theo nhiều tầng, mỗi tầng lọc các cửa sổ không chứa khuôn mặt.
 - Tầng đầu tiên: Đơn giản, loại bỏ nhanh các vùng chắc chắn không phải khuôn mặt.

- Tầng sau: Phức tạp hơn, kiểm tra kỹ càng các vùng nghi ngờ.
- Lợi ích: Giảm đáng kể thời gian xử lý vì chỉ các vùng tiềm năng mới được phân tích sâu.
- Cấu trúc:
 - Mỗi tầng sử dụng một số lượng đặc trưng nhất định (ví dụ: tầng đầu dùng 1 đặc trưng, tầng sau dùng 25 đặc trưng).
 - Trung bình chỉ cần kiểm tra 10 đặc trưng trên mỗi cửa sổ.

Face Tracking

- Kỹ thuật mở rộng: Sau khi phát hiện khuôn mặt, có thể theo dõi các điểm đặc trưng (Feature Points) để nhận diện chuyển động hoặc biểu cảm.
- Ứng dụng: Nhận diện khuôn mặt trong video hoặc theo dõi đối tượng trong thời gian thực.

Face Recognition

- Bước tiếp theo sau phát hiện khuôn mặt: So sánh khuôn mặt phát hiện được với dữ liệu.
- Kỹ thuật:
 - Mean Subtracting: Loại bỏ các giá trị trung bình khỏi dữ liệu đầu vào.
 - Eigen Values: Giảm chiều dữ liệu, trích xuất các đặc trưng quan trọng.
 - Feature Vectors: So sánh sự khớp nhau của vector đặc trưng của khuôn mặt với dữ liệu để xác định danh tính.

5. Homework 5: 2 questions: gradient vector, exploration - exploitation

A. How are exploration and exploitation related to Mini-batch Gradient Descent, Batch Gradient Descent, and Stochastic Gradient Descent?

- Exploration:

Exploration là khả năng của mô hình để khám phá các giải pháp khác nhau và thoát khỏi các điểm cực tiểu cục bộ.

- **Stochastic Gradient Descent (SGD):**

Ưu điểm: SGD cập nhật trọng số sau mỗi mẫu dữ liệu, do đó nó có khả năng khám phá rất cao. Mô hình có thể thoát khỏi các điểm cực tiểu cục bộ nhờ tính ngẫu nhiên và sự biến đổi lớn trong hướng gradient.

Nhược điểm: Tuy nhiên, sự dao động lớn trong quá trình cập nhật có thể làm cho quá trình hội tụ trở nên không ổn định.

- **Batch Gradient Descent:**

Ưu điểm: Batch Gradient Descent sử dụng toàn bộ tập dữ liệu để tính toán gradient, do đó hướng đi của nó rất chính xác và ít bị ảnh hưởng bởi sự ngẫu nhiên.

Nhược điểm: Khả năng exploration kém do nó ít dao động và dễ bị mắc kẹt ở các điểm cực tiểu cục bộ.

- **Minibatch Gradient Descent:**

Ưu điểm: Mini-batch Gradient Descent lấy mẫu dữ liệu nhỏ để tính toán gradient, kết hợp tính ngẫu nhiên của SGD và tính ổn định của Batch Gradient Descent. Điều này giúp mô hình có khả năng khám phá tốt hơn so với Batch Gradient Descent và ít dao động hơn so với SGD, do đó cân bằng giữa việc khám phá và sự ổn định trong quá trình học.

Nhược điểm: Mặc dù có sự ngẫu nhiên nhất định, nhưng không mạnh mẽ như SGD. Tuy nhiên, điều này thường đủ để tránh các điểm cực tiểu cục bộ trong hầu hết các tình huống thực tế.

- **Exploitation:**

Exploitation là khả năng của mô hình để khai thác tốt nhất các giải pháp đã biết và tối ưu hóa chúng.

- **Stochastic Gradient Descent (SGD):**

Ưu điểm: Tính ngẫu nhiên giúp mô hình thoát khỏi các điểm cực tiểu cục bộ, giúp tăng khả năng tìm kiếm các giải pháp mới tiềm năng.

Nhược điểm: Dao động quá lớn, quá trình hội tụ chậm và không ổn định, làm giảm khả năng khai thác hiệu quả các giải pháp tốt nhất.

- **Batch Gradient Descent:**

Ưu điểm: Hội tụ rất ổn định và nhanh chóng khi đã gần điểm cực tiểu toàn cục, vì nó sử dụng toàn bộ tập dữ liệu để cập nhật gradient.

Nhược điểm: Tốn kém về tính toán và bộ nhớ, đặc biệt với các tập dữ liệu lớn. Điều này có thể làm giảm tốc độ khai thác các giải pháp tối ưu.

- **Minibatch Gradient Descent:**

Ưu điểm: Kết hợp được sự ổn định của Batch Gradient Descent và tính ngẫu nhiên của SGD, Mini-batch Gradient Descent có thể khai thác các giải pháp tốt hơn và hội tụ nhanh hơn so với SGD do sự dao động ít hơn. Nó cũng ít tốn kém hơn Batch Gradient Descent về mặt tính toán và bộ nhớ.

Nhược điểm: Hội tụ có thể không nhanh và ổn định như Batch Gradient Descent trong một số trường hợp, nhưng thường đủ nhanh và ổn định để khai thác các giải pháp tối ưu trong hầu hết các trường hợp.

Kết luận: Mini-batch Gradient Descent cung cấp một sự cân bằng tốt giữa exploration và exploitation, làm cho nó trở thành lựa chọn phổ biến trong nhiều ứng dụng học máy. Nó tận dụng được lợi thế của cả sự ngẫu nhiên để thoát khỏi các điểm cực tiểu cục bộ và tính ổn định để hội tụ nhanh hơn đến điểm cực tiểu toàn cục, đồng thời tối ưu hóa việc sử dụng tài nguyên tính toán và bộ nhớ.

B. $-\frac{dL}{dW}$ (the mean vector for batch and mini batch) is the direction of the steepest descent for batch gradient descent but that's not true for stochastic/mini batch gradient descent. Why not?

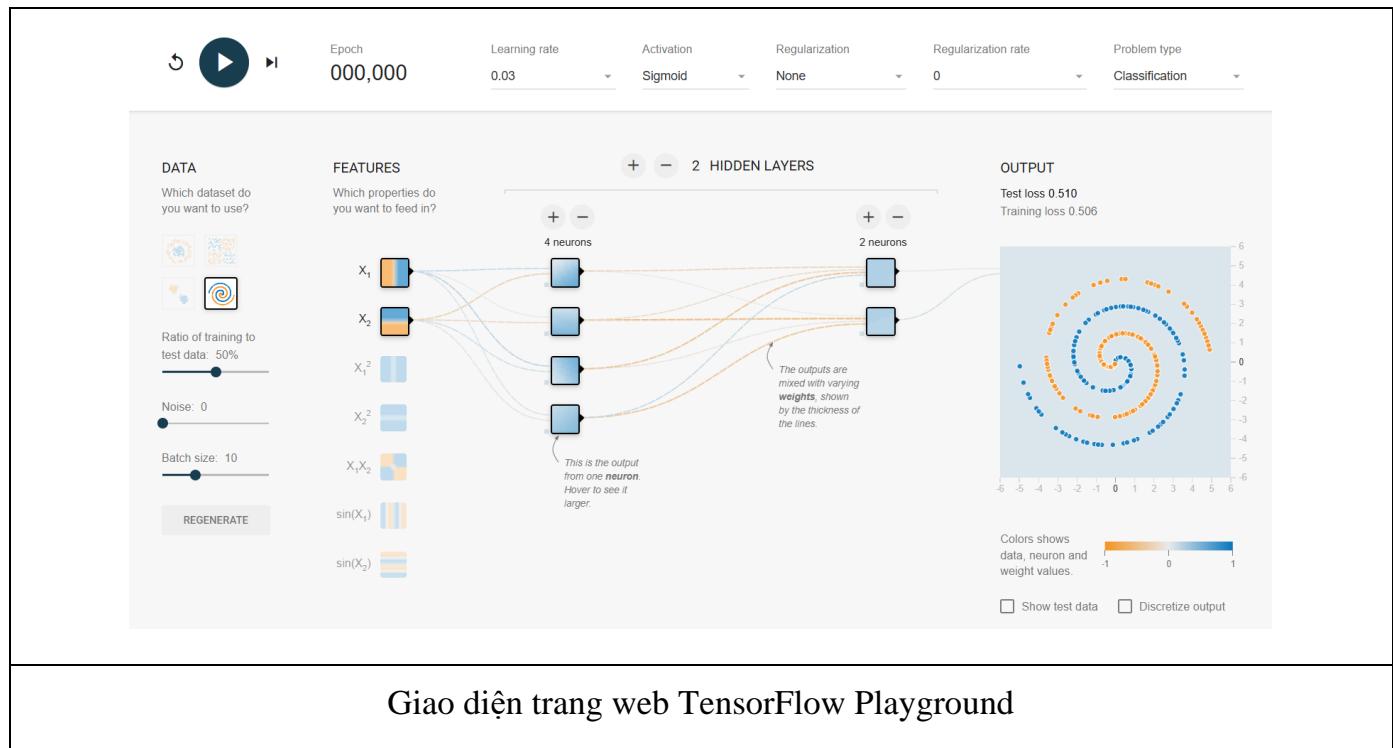
- Với batch gradient descent, việc tính đạo hàm của hàm mất mát theo trọng số W dựa trên toàn bộ tập dữ liệu đào tạo. Do đó, $-\frac{dL}{dW}$ trong batch

gradient descent thực sự biểu thị hướng dốc nhất cho toàn bộ dữ liệu huấn luyện.

- Tuy nhiên, trong stochastic gradient descent hoặc mini-batch gradient descent, chúng ta chỉ sử dụng một mẫu (đối với stochastic) hoặc một số mẫu (đối với mini-batch) để cập nhật trọng số. Do đó, $-dL/dW$ được tính dựa trên một lượng nhỏ dữ liệu, chỉ cung cấp xấp xỉ cho hướng dốc nhất thực sự dựa trên dữ liệu giới hạn được sử dụng.
- Trong stochastic gradient descent, sự biến động lớn trong hướng của đạo hàm $-dL/dW$ giữa các mẫu có thể dẫn đến nhảy vọt trong không gian trọng số và làm chậm quá trình hội tụ. Điều này làm cho hướng của đạo hàm không còn đại diện cho hướng giảm độ dốc chung cho toàn bộ dữ liệu.
- Do đó trong mini-batch/stochastic gradient descent, hướng cập nhật không còn chính xác là hướng giảm độ dốc của toàn bộ tập dữ liệu, và quá trình tối ưu có thể có sự dao động lớn hơn.

6. Homework 6: ANN playground

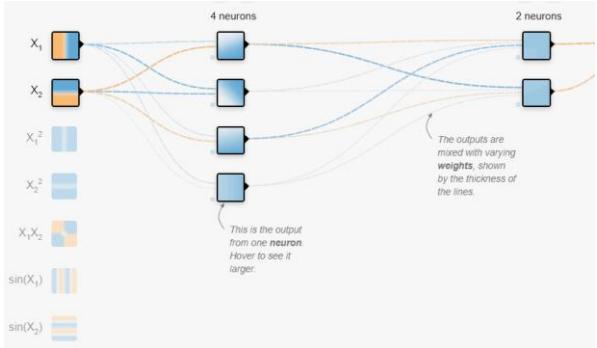
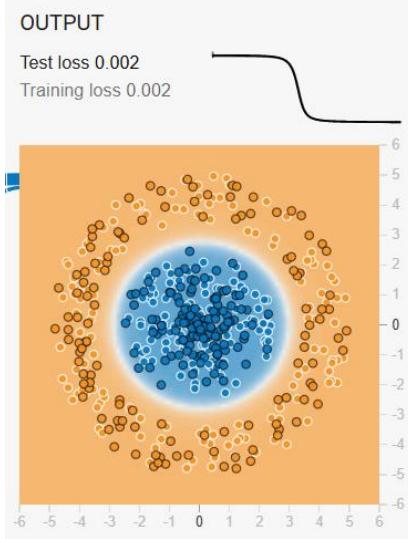
TensorFlow Playground là một công cụ trực quan cho phép chúng ta tương tác và hiểu cách hoạt động của mạng nơ-ron nhân tạo. Giao diện của trang web như sau



Các thành phần của trang web bao gồm

Thành phần	Hình ảnh		Ý nghĩa
Dữ liệu đầu vào (Data)			Dữ liệu huấn luyện mô hình, có 4 loại dữ liệu: Circle, Xor, Gauss, Spiral

Thông số của tập dữ liệu	<p>Ratio of training to test data: 50%</p> <p>Noise: 0</p> <p>Batch size: 10</p>	<p>Ratio of training to test data: Điều chỉnh tỷ lệ phần trăm dữ liệu được sử dụng cho huấn luyện so với kiểm thử.</p> <p>Noise: Thêm nhiễu vào dữ liệu để mô phỏng các biến động thực tế và kiểm tra độ bền của mô hình.</p> <p>Batch size: Xác định số lượng mẫu dữ liệu được sử dụng trong mỗi lần cập nhật trọng số của mô hình.</p>
Feature (đặc trưng)	<p>FEATURES</p> <p>Which properties do you want to feed in?</p> <p>x_1</p> <p>x_2</p> <p>x_1^2</p> <p>x_2^2</p> <p>x_1x_2</p> <p>$\sin(x_1)$</p> <p>$\sin(x_2)$</p>	<p>Chọn các đặc trưng đầu vào của mô hình, gồm có:</p> <p>x, y: Tọa độ x và y.</p> <p>x^2, y^2: Bình phương của tọa độ x và y.</p> <p>$x*y$: Tích của x và y.</p> <p>$\sin(x), \cos(x)$: Giá trị \sin và \cos của x.</p>

Cấu trúc mạng Nơ ron		Biểu diễn cấu trúc mạng nơ ron đang thực hiện, cho phép tăng/giảm số lớp ẩn, số nơ ron mỗi lớp ẩn để phục vụ nhu cầu người dùng.
Ngõ ra		Biểu diễn kết quả thực hiện bài toán của mạng Nơ ron, có các thông số sau: Test loss: mức độ sai lệch giữa đầu ra của mô hình với giá trị thực trên tập test. Training loss: mức độ sai lệch giữa đầu ra của mô hình với giá trị thực trên tập huấn luyện Test loss >>> training loss => Overfitting Cả 2 đều cao => Underfitting
Activation (Hàm kích hoạt)	<p>Activation</p> <p>Sigmoid</p>	Hàm kích hoạt quyết định đầu ra của một neuron. Có các loại sau: ReLU (Rectified Linear Unit): Trả về giá trị đầu vào nếu dương, ngược lại trả về 0. Tanh(Hyperbolic Tangent): Chuyển đổi giá trị đầu vào

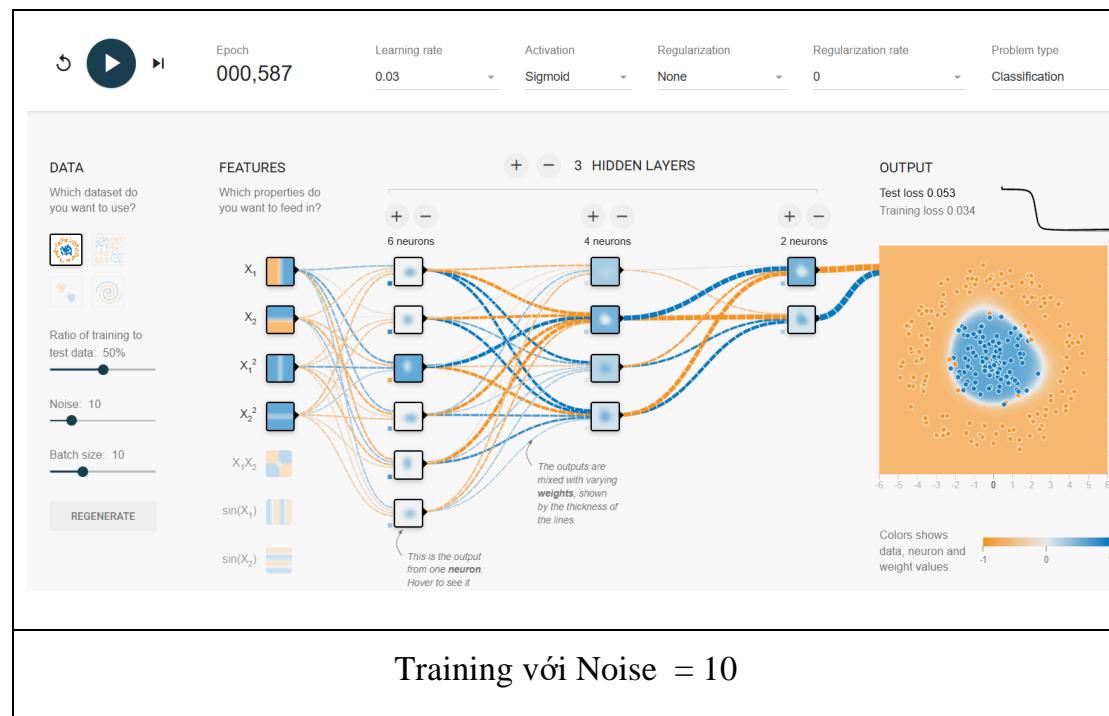
		<p>thành khoảng từ -1 đến 1.</p> <p>Sigmoid: Chuyển đổi giá trị đầu vào thành khoảng từ 0 đến 1.</p> <p>Linear: Giữ nguyên giá trị đầu vào.</p>
Learning Rate (Tốc độ học)	<p>Learning rate</p> <p>0.03</p>	<p>Xác định mức độ điều chỉnh trọng số trong mỗi lần cập nhật. Tốc độ học cao có thể dẫn đến việc bỏ qua điểm tối ưu, trong khi tốc độ học thấp có thể làm quá trình huấn luyện chậm.</p>
Regularization (Chuẩn hóa)	<p>Regularization</p> <p>None</p>	<p>Kỹ thuật ngăn chặn mô hình quá khớp bằng cách thêm một thuật ngữ phạt vào loss function. Có các loại sau:</p> <p>None: Không sử dụng chuẩn hóa.</p> <p>L1: Thêm tổng giá trị tuyệt đối của các trọng số vào hàm mất mát.</p> <p>L2: Thêm tổng bình phương của các trọng số vào hàm mất mát.</p> <p>L1 Regularization có tác dụng chọn lọc đặc trưng, L2 có tác dụng giảm overfitting.</p>

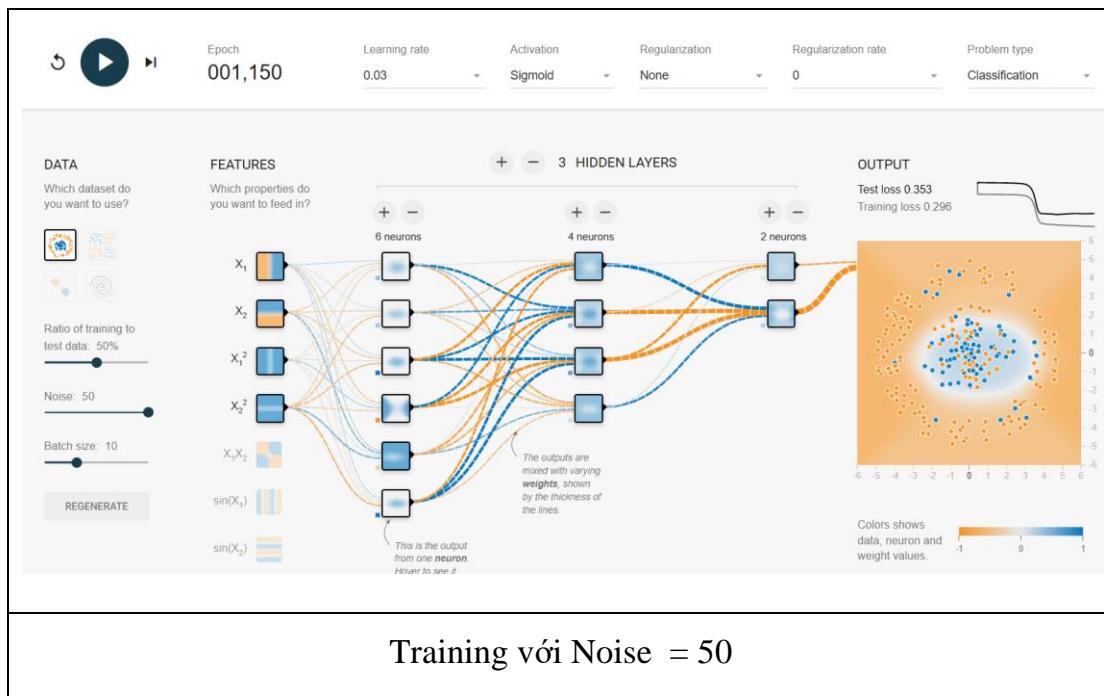
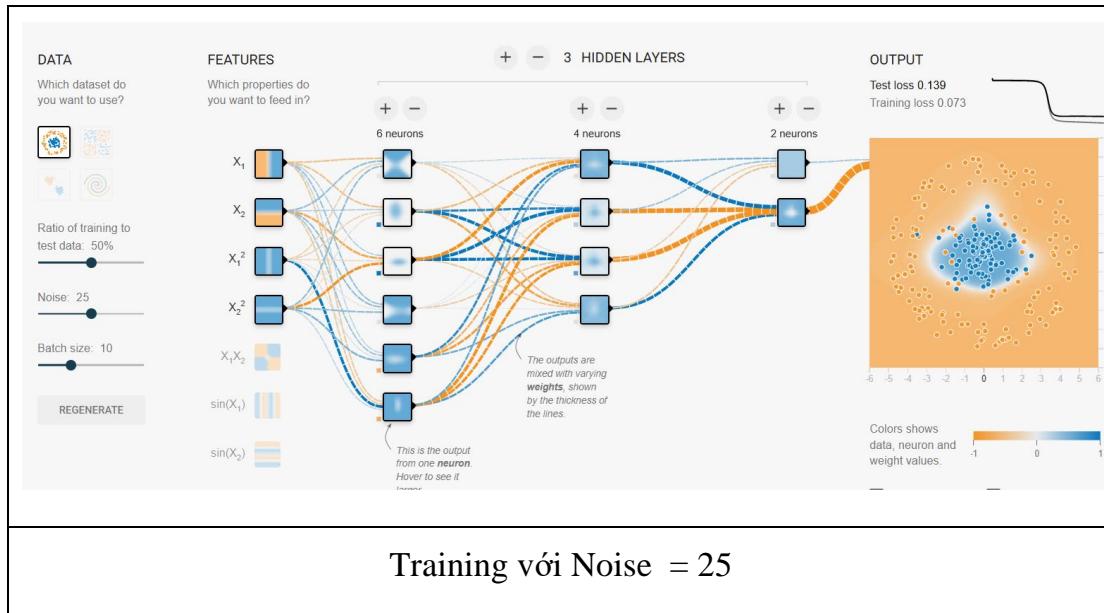
Regularization Rate (Tỷ lệ chuẩn hóa)	Regularization rate 0	Điều chỉnh mức độ ảnh hưởng regularization, cân bằng underfitting và overfitting
Problem (Loại bài toán)	Problem type Classification	Chọn loại bài toán mà mô hình sẽ giải quyết: Classification: Phân loại dữ liệu vào các nhóm. Regression: Dự đoán giá trị liên tục.

Nhận xét

Trong phần nhận xét này, nhóm chọn bài toán Classification để nhận xét.

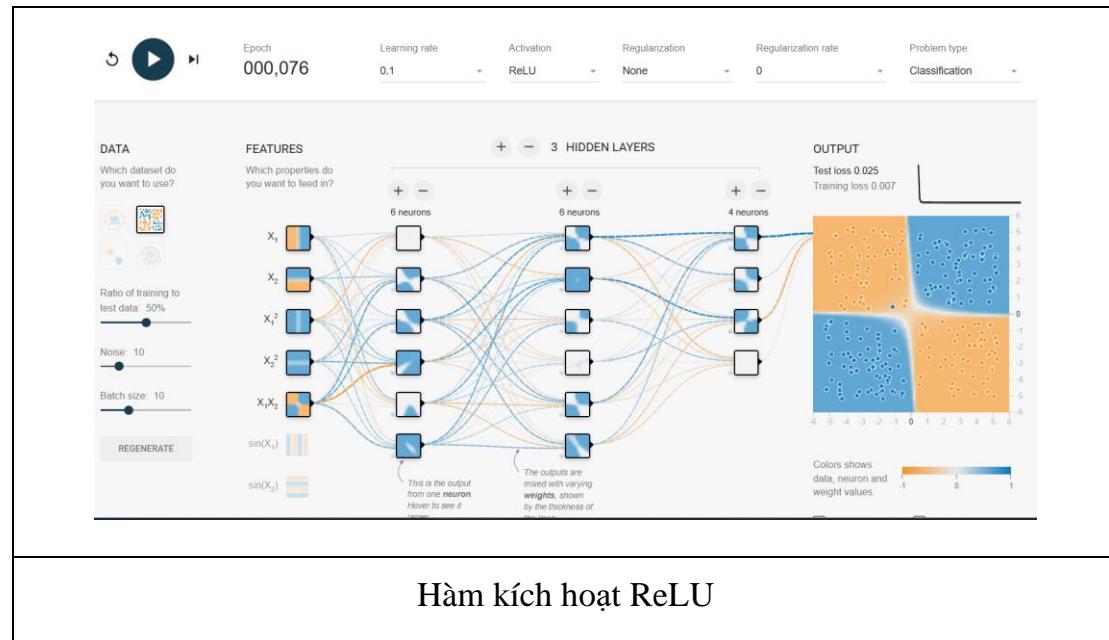
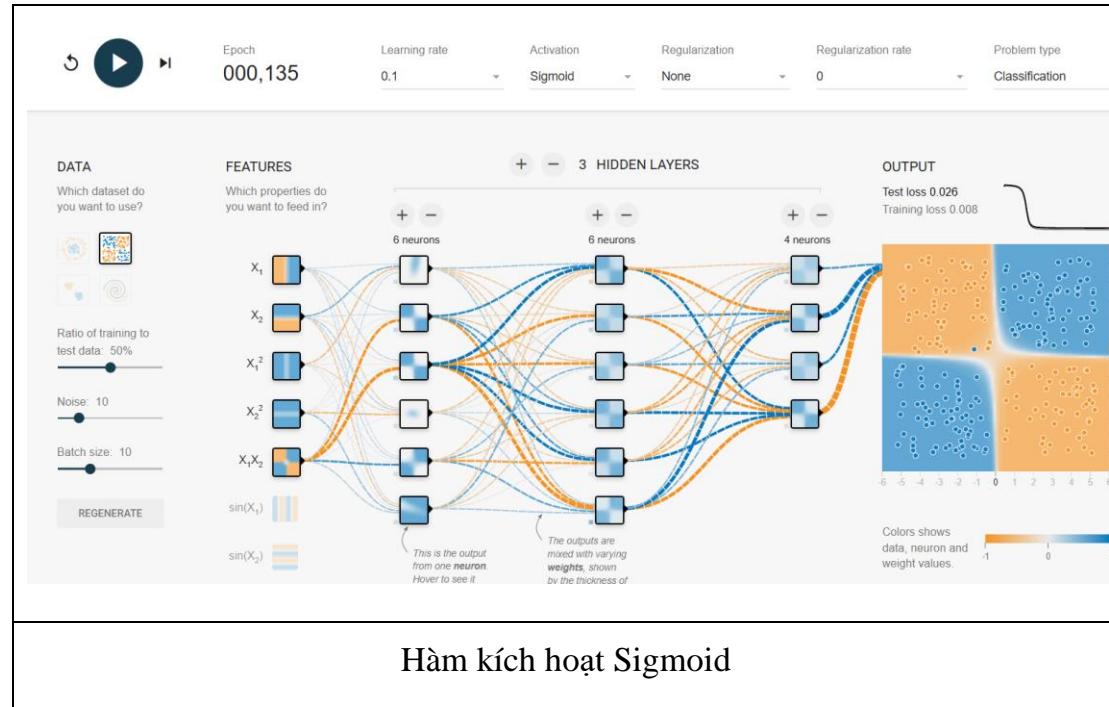
Data 1: Circle - Tăng nhiễu

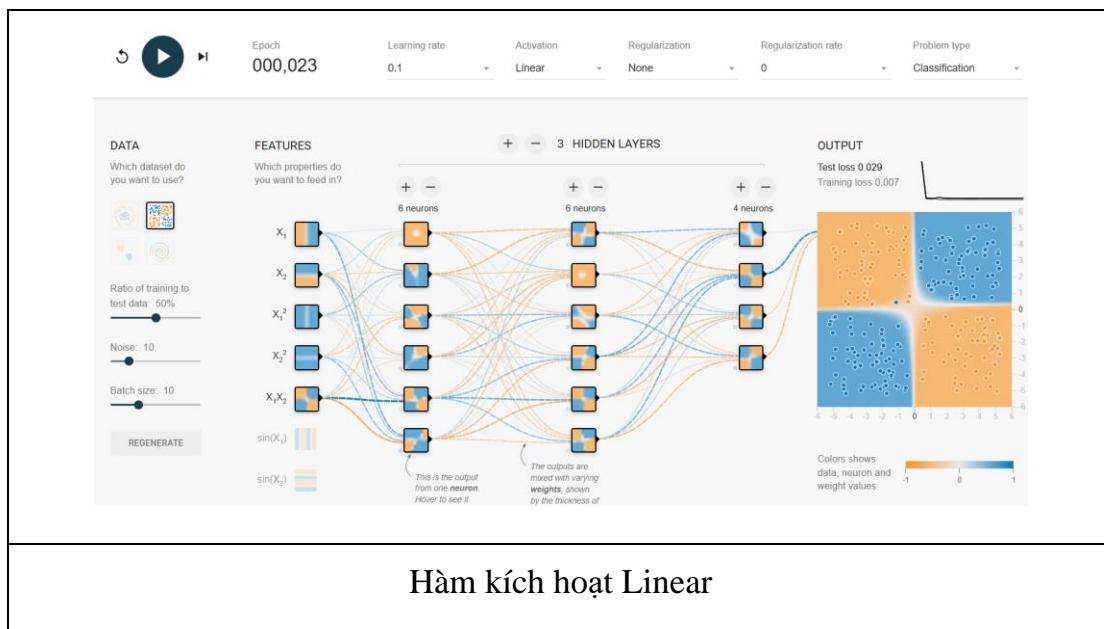
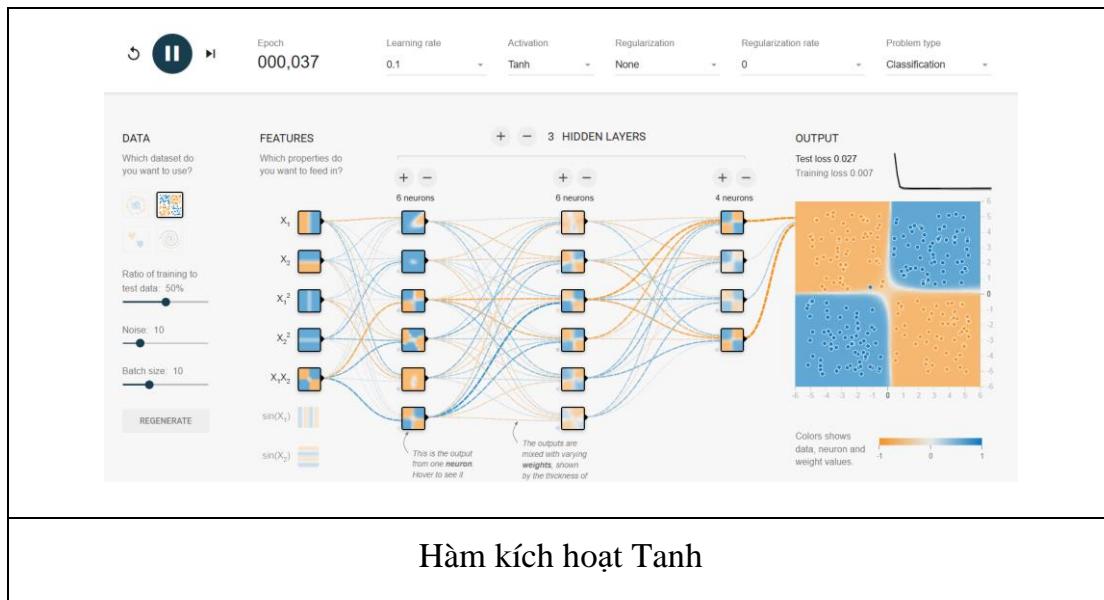




Nhận xét: Tỉ lệ nhiễu càng tăng, tỉ lệ Test lost và Training loss càng tăng => Mô hình ngày càng không học được cách phân loại.

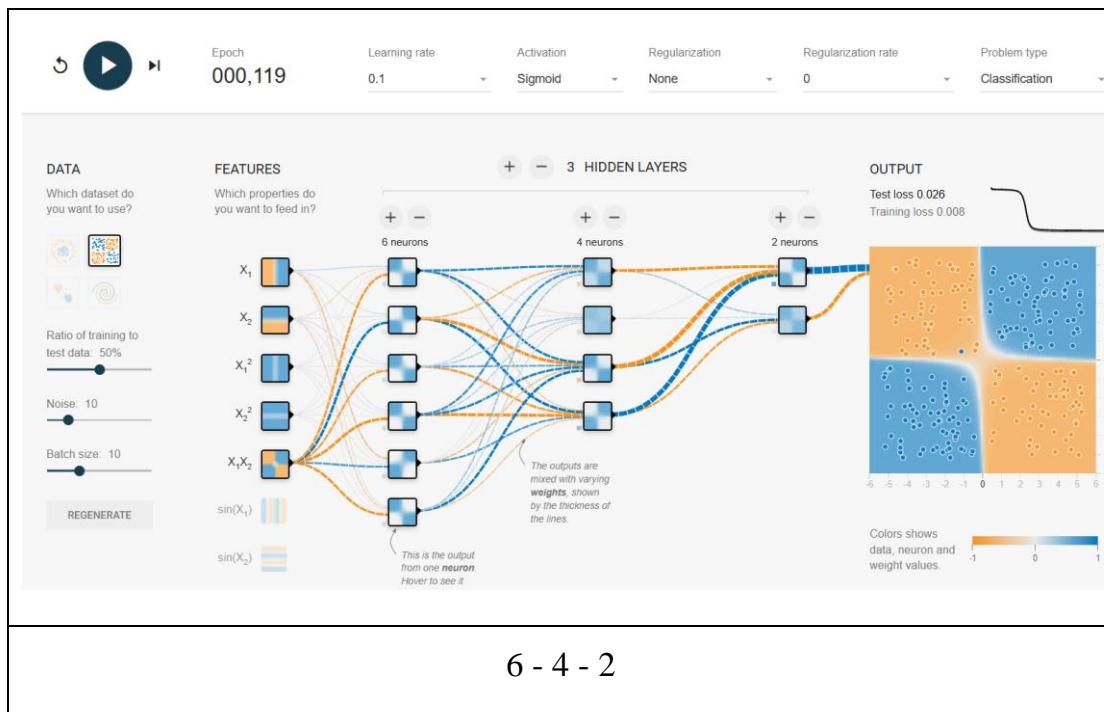
Data 2: Xor - Đổi hàm kích hoạt

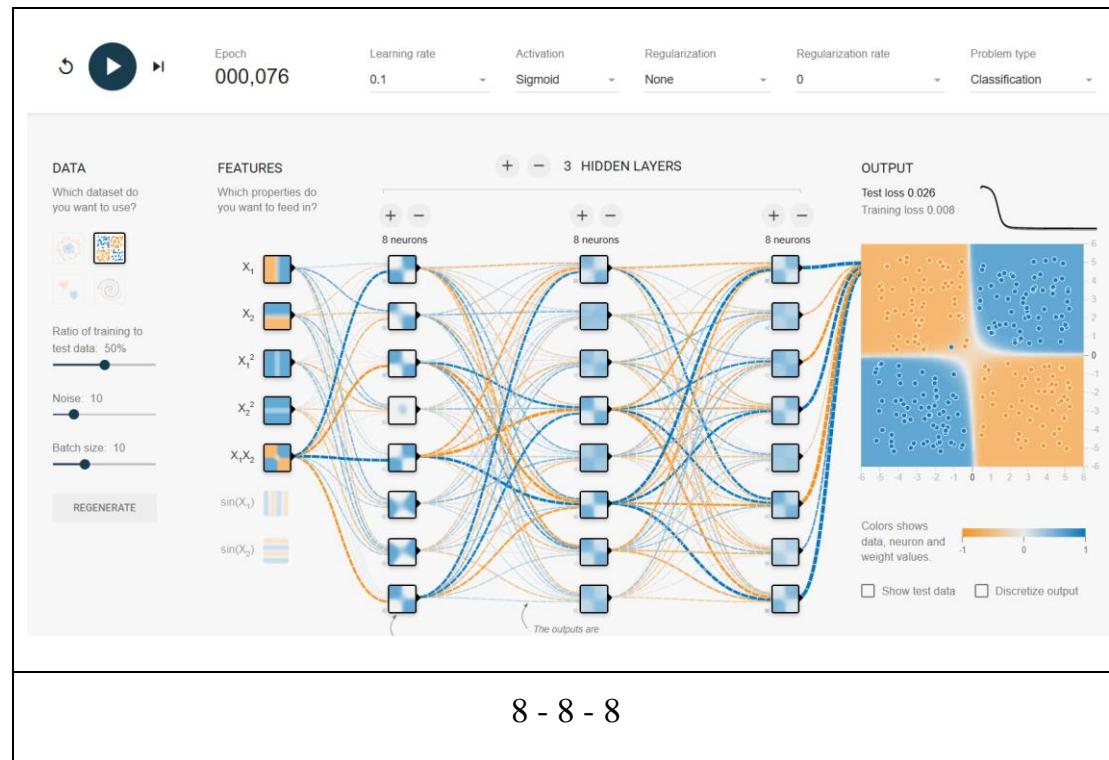




Nhận xét: Với cùng loại bài toán và thông số đầu vào, hàm kích hoạt đóng vai trò quyết định đến tốc độ giảm của sai số mô hình, thời gian huấn luyện.

Data 3: Gauss - Tăng giảm số lớp ẩn - số lớp neuron lớp ẩn





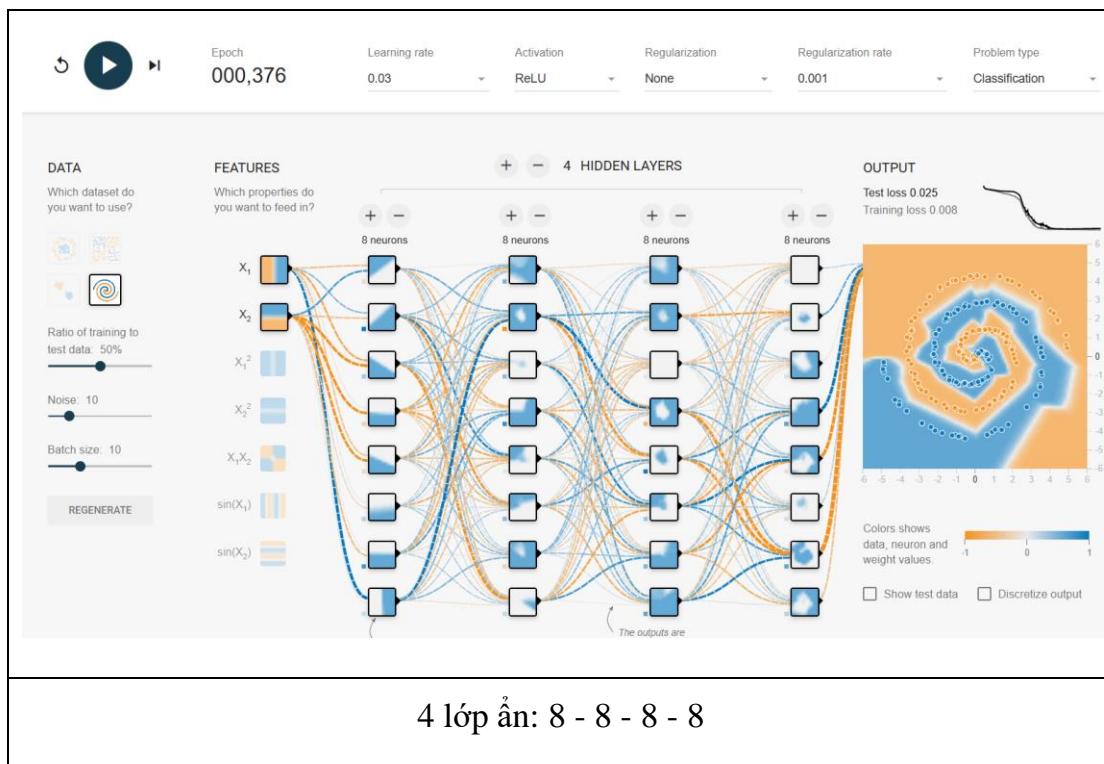
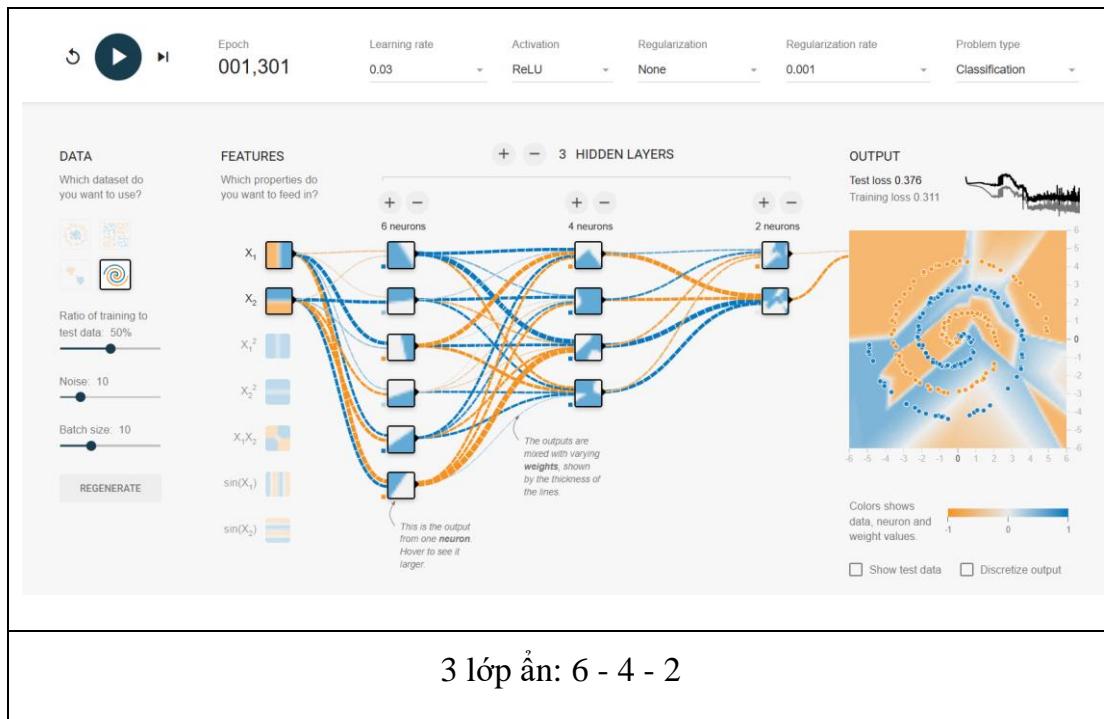
Nhận xét: Số nơ ron lớp ẩn tăng \Rightarrow Số chu kỳ huấn luyện giảm, mô hình nhận dạng chính xác hơn do mang theo nhiều đặc trưng

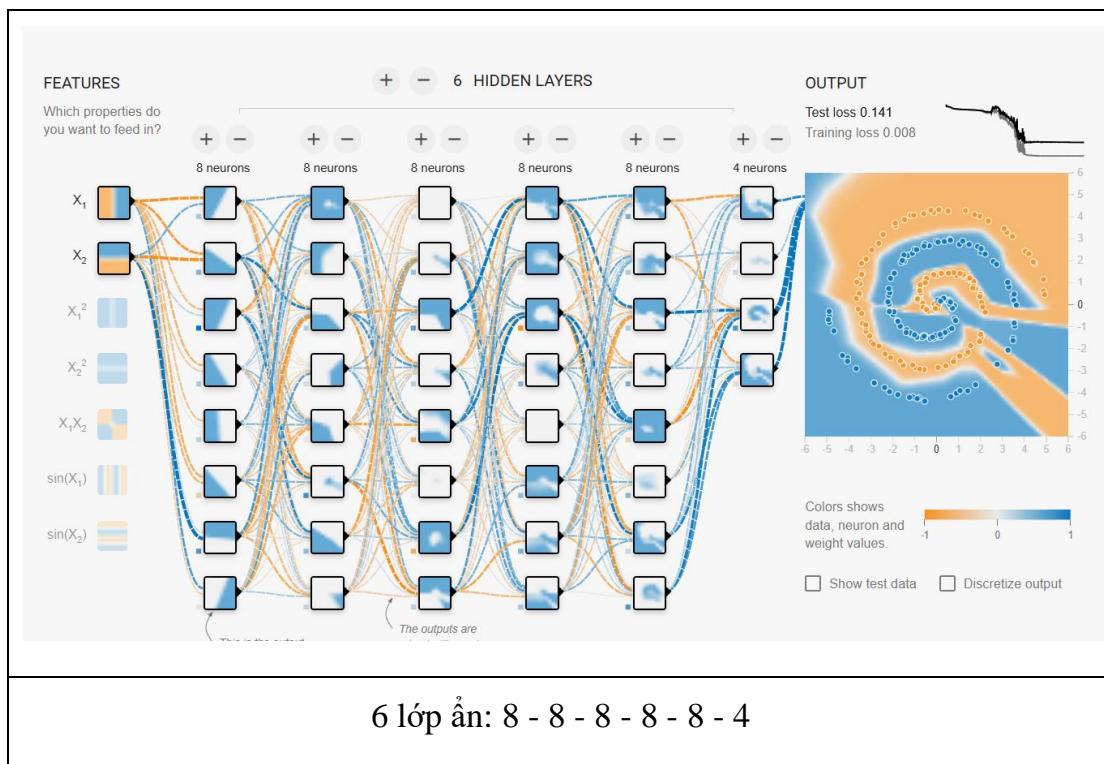




Nhận xét: Số lớp ẩn tăng \rightarrow Thời gian huấn luyện lâu hơn, chu kì huấn luyện nhiều hơn.

Data 4: Spiral - Tăng giảm số lớp ẩn và số nơ ron lớp ẩn





Nhận xét: Với quá ít số lớp ẩn và số nơ ron ở mỗi lớp ẩn \Rightarrow Mô hình mạng NN không thể phân biệt được các mẫu dữ liệu (underfitting)

Với số lớp ẩn và số nơ ron lớp ẩn phù hợp, mô hình có thể học được cách phân biệt các mẫu dữ liệu chính xác nhất trong thời gian nhanh nhất

Với quá nhiều lớp ẩn và số lượng nơ ron mỗi lớp ẩn \Rightarrow Mô hình học chính xác theo tập training \Rightarrow Tỉ lệ Training loss thấp nhưng tỉ lệ Test loss cao \Rightarrow Overfitting.

7. Homework 7

A. Intuitively, receiving information from multiple sources seems better, so why does CNN use local connectivity?

Local Connectivity (Kết nối cục bộ) trong CNN:

CNN (Convolutional Neural Network) sử dụng các kết nối cục bộ để phân tích hình ảnh hoặc dữ liệu có cấu trúc không gian (như hình ảnh). Mỗi nơ-ron chỉ kết nối với một vùng nhỏ của đầu vào, thường được gọi là *receptive field*.

Lý do CNN không kết nối toàn bộ đầu vào như ANN:

- **Giảm độ phức tạp tính toán:** Nếu mỗi nơ-ron trong một mạng phải kết nối với toàn bộ đầu vào (như ANN), số lượng tham số sẽ rất lớn, dẫn đến quá trình huấn luyện chậm hơn và dễ bị overfitting. Local connectivity giảm thiểu số lượng tham số cần thiết.
- **Tận dụng thông tin không gian:** Trong hình ảnh, các đặc trưng như cạnh, góc, hoặc kết cấu thường là các mẫu cục bộ. CNN khai thác các mối quan hệ cục bộ để phát hiện đặc trưng ở từng phần nhỏ, sau đó kết hợp chúng ở các lớp tiếp theo.
- **Khái niệm Translation Invariance:** Dùng local connectivity giúp CNN có khả năng nhận diện đặc trưng không phụ thuộc vào vị trí của chúng trong hình ảnh

So sánh với ANN:

ANN kết nối mỗi nơ-ron với toàn bộ đầu vào, phù hợp với các bài toán không gian không có cấu trúc như dữ liệu dạng bảng. Tuy nhiên, điều này không tận dụng được thông tin không gian của hình ảnh.

B. In an ANN, each neuron has its own set of parameters to adjust, but why does CNN share the same set of parameters?

Cấu trúc của ANN (Artificial Neural Network):

Trong ANN, mỗi nơ-ron ở lớp ẩn kết nối với tất cả các nơ-ron ở lớp đầu vào, và mỗi kết nối có trọng số (weight) riêng. Điều này cho phép mạng học các mối quan hệ phức tạp giữa tất cả các đầu vào.

CNN sử dụng shared parameters (chia sẻ tham số):

Trong CNN, một bộ lọc (filter/kernel) được sử dụng để trượt qua toàn bộ hình ảnh. Bộ lọc này áp dụng chung một bộ tham số cho mọi vị trí trên ảnh, thay vì có tham số riêng biệt cho từng vị trí.

Lý do CNN dùng chung bộ tham số:

- **Phát hiện đặc trưng lặp lại:** Các đặc trưng như cạnh, góc hoặc mẫu kết cấu có thể xuất hiện ở bất kỳ vị trí nào trong hình ảnh. Sử dụng cùng một

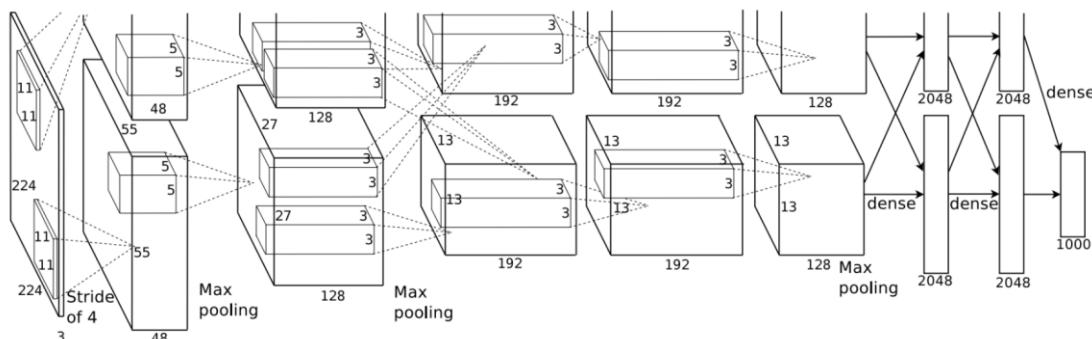
bộ lọc giúp mạng học cách phát hiện các đặc trưng này trên toàn bộ hình ảnh, bất kể vị trí.

- **Giảm số lượng tham số:** Chia sẻ tham số làm giảm đáng kể số lượng trọng số cần học, giúp CNN hoạt động hiệu quả hơn và tránh overfitting.
- **Tính dịch chuyển bất biến:** Khi cùng một bộ lọc được áp dụng ở nhiều vị trí, CNN có thể nhận diện một đặc trưng dù nó xuất hiện ở đâu trong hình ảnh.

Kết luận:

Trong khi ANN sử dụng các tham số riêng cho mỗi nơ-ron để học mối quan hệ giữa toàn bộ đầu vào và đầu ra, CNN sử dụng chung bộ tham số để khai thác các mẫu cục bộ lặp lại trên hình ảnh, giúp mô hình đơn giản hơn, hiệu quả hơn, và tận dụng tốt các đặc trưng không gian.

8. Homework 8 Alexnet: size & number of kernels, number of parameters



Stage	Kernel	Kernel size	Weights	Biases	Parameters
Conv 1	2 x 48	11 x 11 x 3	96 x (11 x 11 x 3)	96	34,944
Conv 2	2 x 128	5 x 5 x 48	256 x (5 x 5 x 48) x 2	256	614,656
Conv 3	4 x 192	3 x 3 x 128	384 x (3 x 3 x 128) x 2	384	885,120
Conv 4	2 x 192	3 x 3 x 192	384 x (3 x 3 x 192) x 2	384	1,327,488

Conv 5	2 x 128	3 x 3 x 192	256 x (3 x 3 x 192) x 2	256	884,992
FC 1	2048 x 2		2048 x 2 x (6 x 6 x 128)	4096	37,752,832
FC 2	2048 x 2		2048 x 2 x 4096	4096	16,781,312
FC 3	1000 x 1		1000 x 4096	1000	4,097,0
Total					62,378,344

9. Homework 9

Alexnet: evaluation (top-1, top-5 error rate) with 100 images, 5 classes

Chương trình

```
% Use the AlexNet model

net = alexnet;

% Get the input size and class names of the model

inputSize = net.Layers(1).InputSize;

classNames = net.Layers(end).ClassNames;

numClasses = numel(classNames);

% Specify the folder containing the images to classify

% imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/Dog';

% imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/cellphone';

% imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/airplanes';
```

```
imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/elephant';  
% imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/crayfish';  
% imageFolder = 'D:/TGM_HW_DK/TGM_WS_MATLAB/HW9/Cucumber';  
% Find all the files in the folder matching the pattern  
theFiles = dir(fullfile(imageFolder, '*.jpg'));  
% theFiles = dir(fullfile(imageFolder, '*.png'));  
% Loop over each image file and classify using the AlexNet model  
figure  
for k = 1 : length(theFiles)  
    baseFileName = theFiles(k).name;  
    fullFileName = fullfile(theFiles(k).folder, baseFileName);  
    fprintf(1, 'Now reading %s\n', fullFileName);  
    % Read and resize the image  
    image = imread(fullFileName);  
    image = imresize(image, inputSize(1:2));  
    % Classify the image  
    [label, scores] = classify(net, image);  
    [~, idx] = sort(scores, 'descend');  
    idx = idx(5:-1:1);  
    classNamesTop = net.Layers(end).ClassNames(idx);  
    scoresTop = scores(idx);  
    % Display the image and top 5 predictions  
    if k <= 10  
        subplot(4, 10, k);
```

```
imshow(image);

subplot(4, 10, k+10);

barh(scoresTop)

xlim([0 1])

title('Top 5 Predictions')

xlabel('Probability')

yticklabels(classNamesTop)

else

subplot(4, 10, k+10);

imshow(image)

subplot(4, 10, k+20);

barh(scoresTop)

xlim([0 1])

title('Top 5 Predictions')

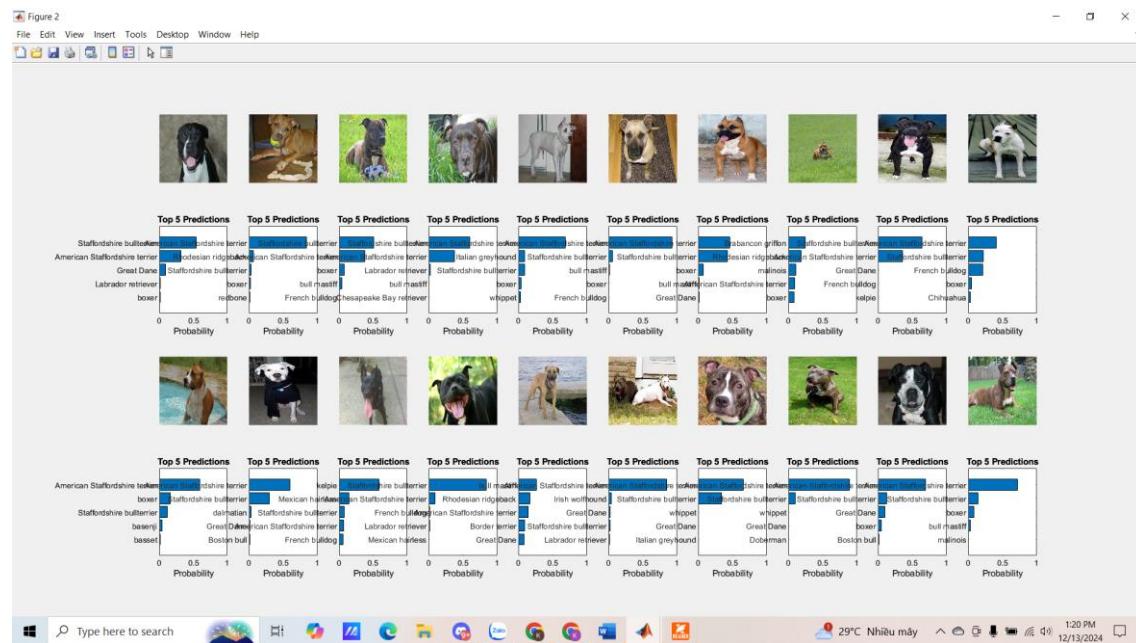
xlabel('Probability')

yticklabels(classNamesTop)

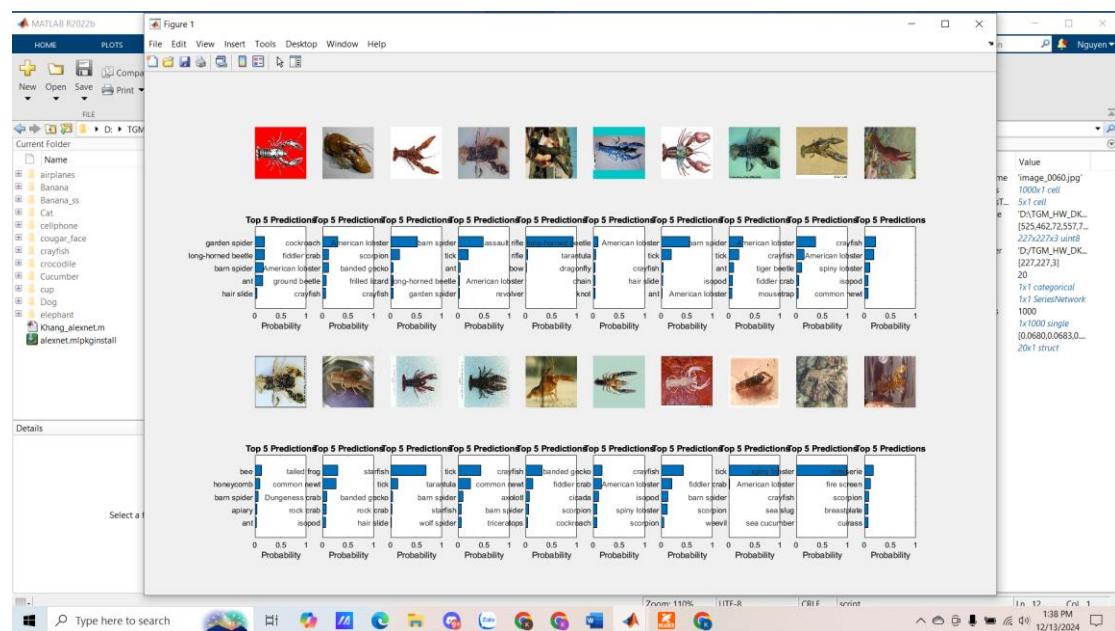
end

end
```

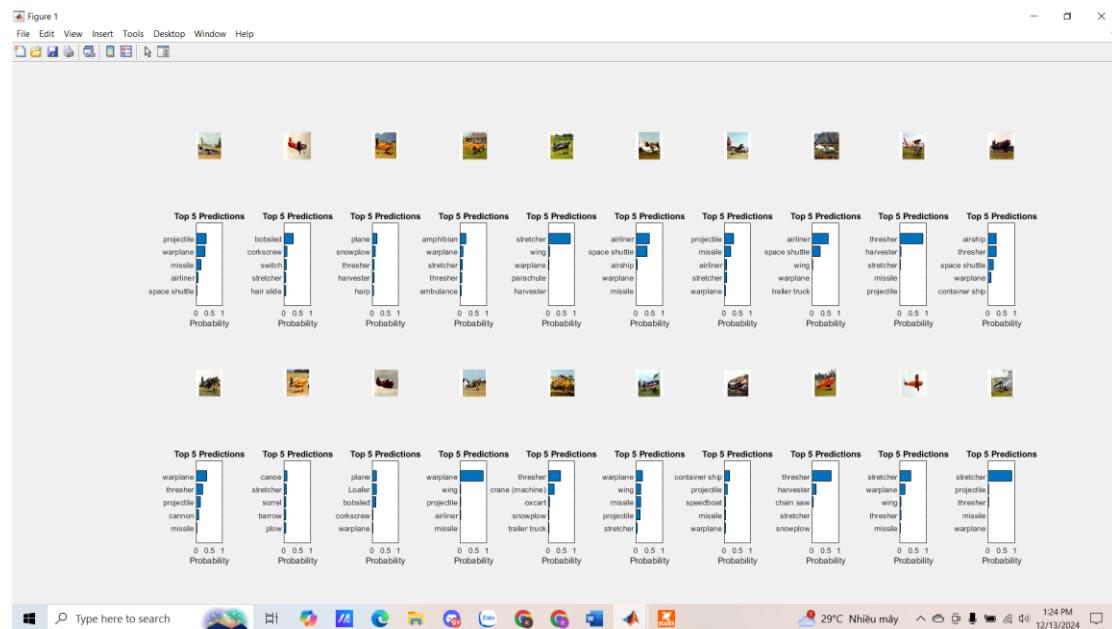
Kết quả với class ‘Dog’



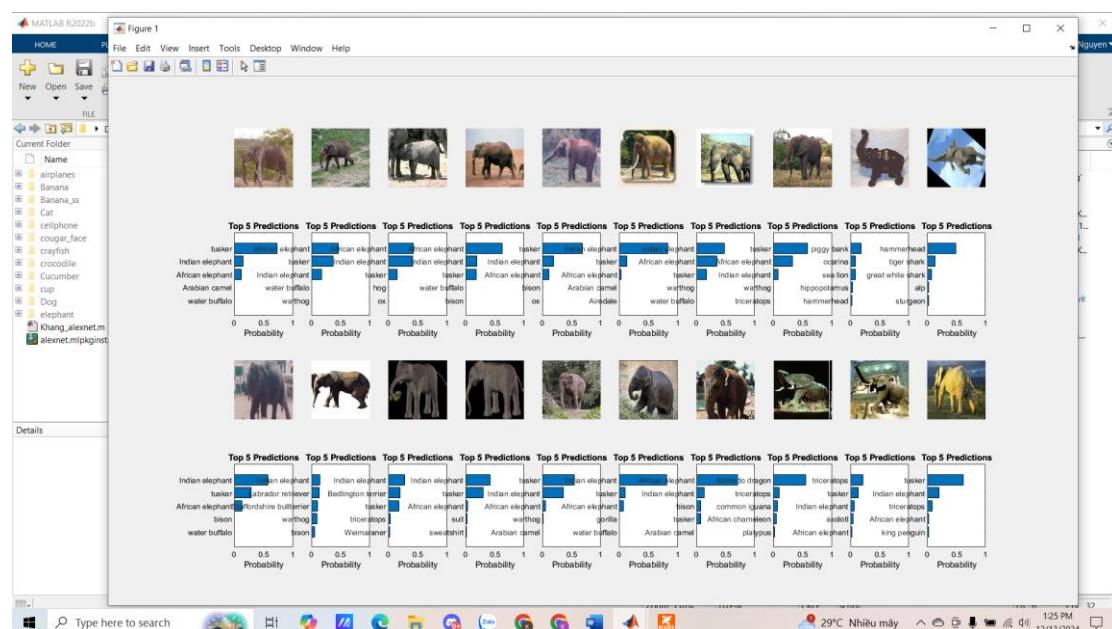
Kết quả với class ‘crayfish’



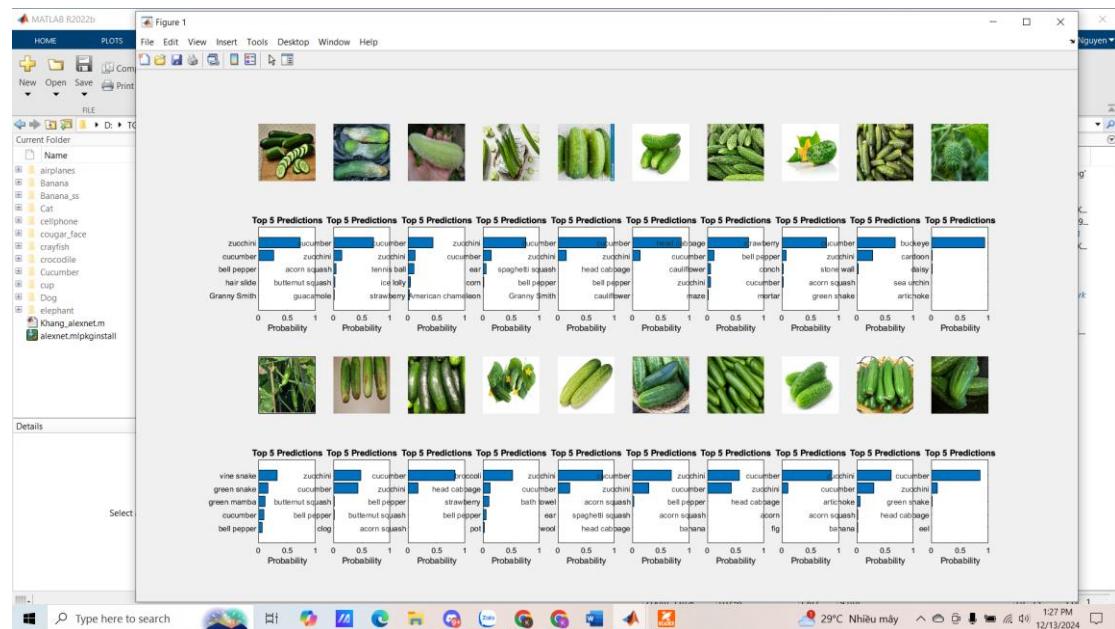
Kết quả với class ‘airplanes’



Kết quả với class ‘elephant’



Kết quả với class ‘Cucumber’



Nhận xét: Top-1 error rate là 27% và Top-5 error rate là 13%