In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
```

XỬ LÝ HISTOGRAM ẢNH

1.1. Xuất Histogram ảnh đơn sắc (grayscale images) Yêu cầu: Nghiên cứu bài hướng dẫn thông qua link thao khảo https://docs.opencv.org/4.x/d1/db7/tutorial_py_histogram_begins.html và thực hiện các yêu cầu sau:

1. Trình bày và giải thích cú pháp hàm cho phương pháp tính Histogram với OpenCV, với NumPy. Nêu ví dụ minh họa từng bước cụ thể.

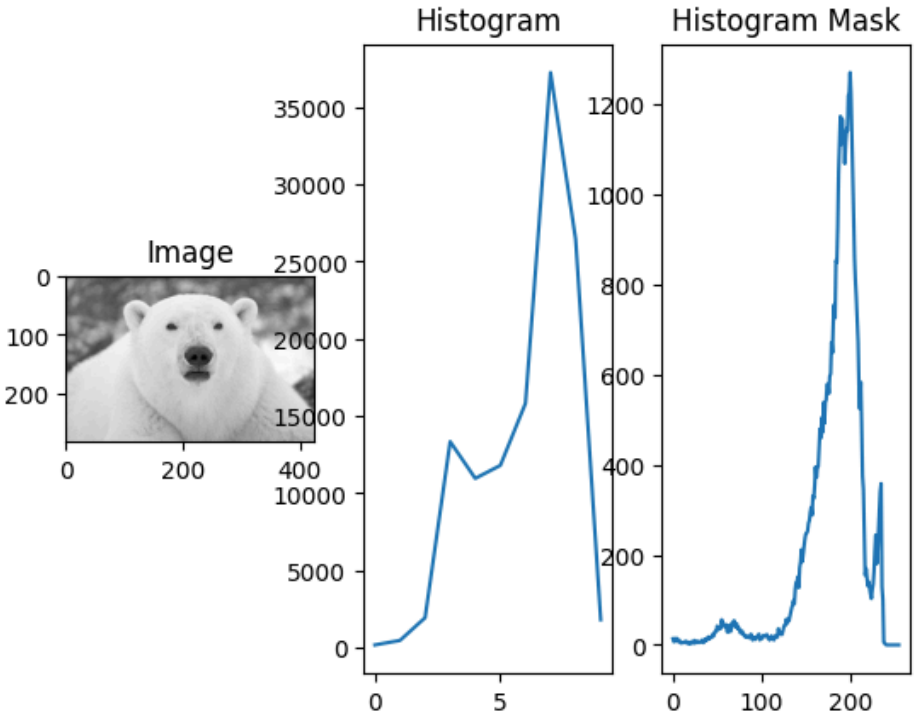
Histogram Calculation in OpenCV- Tính toán biểu đồ trong OpenCV

Hàm cv.calcHist() cú pháp cv.calcHist(images, channels, mask, histSize, ranges[, hist[, accumulate]])

Trong đó
images: ảnh đầu vào và được đặt trong dấu []
channel: đây là chỉ số kênh mà ta muốn tính Histogram, đặt trong dấu [], nếu là ảnh xám thì thường sẽ là [0], ảnh màu có thể là [0],[1] hoặc [2].
mask: Đây là ảnh mặt nạ. Để tính histogram cho toàn bộ ảnh mặc định là none, muốn tính một vùng cụ thể thì phải tạo một mặt nạ cho khu vực đó
hisSize: Số lượng bin của histogram
range: phạm vi giá trị của histogram

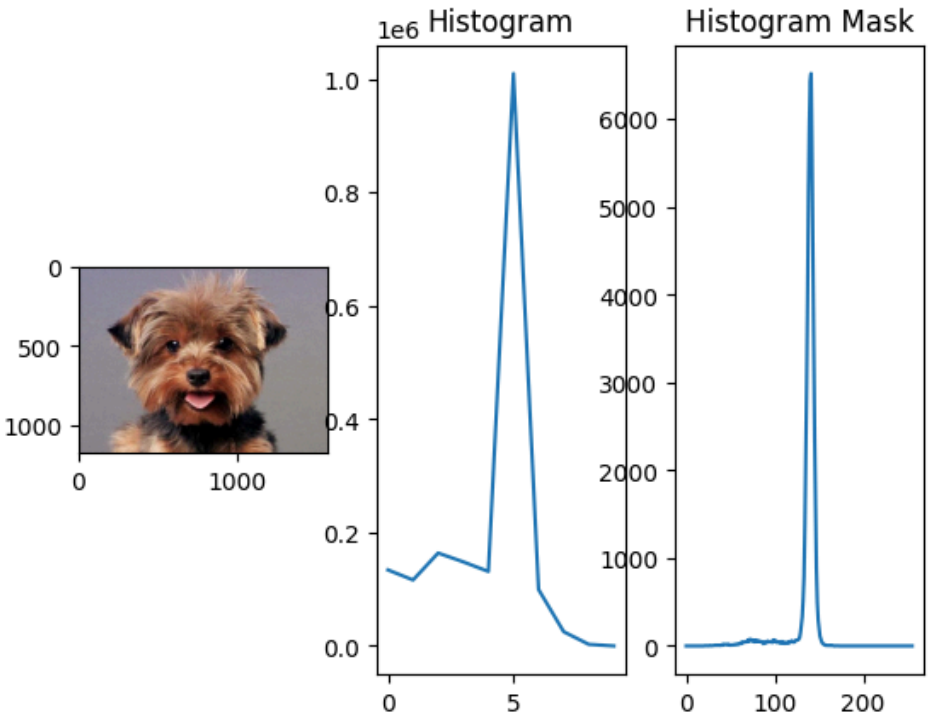
In [ ]:
```
#đọc ảnh đầu vào vào img
img = cv2.imread('Bear.jpg', cv2.IMREAD_GRAYSCALE)
#ảnh đầu vào là img, vì ảnh là ảnh xám nên channel sẽ là  [0], và không sài mặt nạ,
#  hisSize cho là 256, range cho chạy từ [0,256] vì hisSize tối đa là 256
hist = cv2.calcHist([img],[0],None,[10],[0,256])

# Tạo một mảng có kích thước giống với ảnh (img)
mask = np.zeros(img.shape[:2], np.uint8)
#Đặt giá trị 255 cho tất cả các điểm ảnh trong phạm vi [100:300, 100:400] của mặt nạ
mask[100:300, 100:400] = 255
#ảnh đầu vào là img, vì ảnh là ảnh xám nên channel sẽ là  [0], và có mặt nạ,
#  hisSize cho là 256, range cho chạy từ [0,256] vì hisSize tối đa là 256
hist2 = cv2.calcHist([img],[0],mask,[256],[0,256])

# Hiển thị ảnh gốc
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Image')
```

```
# Hiển thị histogram không có mặt nạ
plt.subplot(1, 3, 2)
plt.plot(hist)
plt.title('Histogram')
# Hiển thị histogram  có mặt nạ

plt.subplot(1, 3, 3)
plt.plot(hist2)
plt.title('Histogram Mask')

plt.show()
```



In [ ]:
```
#đọc ảnh đầu vào vào img
img2 = cv2.imread('Dog.jpg')
#ảnh đầu vào là img, đọc ảnh màu và cho giá trị là [2] tương ứng màu đỏ , và không sài mặt nạ,
#  hisSize cho là 256, range cho chạy từ [0,256] vì hisSize tối đa là 256
hist3 = cv2.calcHist([img2],[1],None,[10],[0,256])

# Tạo một mảng có kích thước giống với ảnh (img)
mask = np.zeros(img2.shape[:2], np.uint8)
#Đặt giá trị 255 cho tất cả các điểm ảnh trong phạm vi [100:300, 100:400] của mặt nạ
```

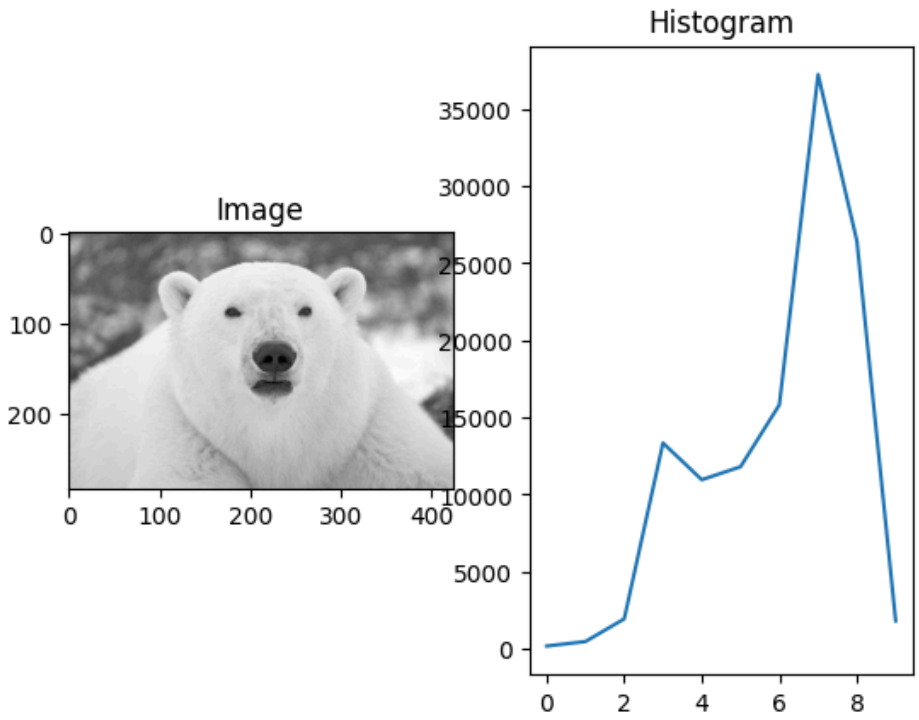
```python
mask[100:300, 100:400] = 255
#ảnh đầu vào là img, đọc ảnh màu và cho giá trị là [2] tương ứng màu đỏ , và có mặt nạ,
#  hisSize cho là 256, range cho chạy từ [0,256] vì hisSize tối đa là 256
hist4 = cv2.calcHist([img2],[1],mask,[256],[0,256])

# Hiển thị ảnh gốc
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))  # Chuyển đổi màu không gian từ BGR sang RGB

# Hiển thị histogram không có mặt nạ
plt.subplot(1, 3, 2)
plt.plot(hist3)
plt.title('Histogram')
# Hiển thị histogram  có mặt nạ
plt.subplot(1, 3, 3)
plt.plot(hist4)
plt.title('Histogram Mask')

plt.show()
```



Histogram Calculation in Numpy - Tính toán biểu đồ trong Numpy Hàm np.histogram cú pháp: np.histogram(img.ravel(),256,[0,256]) trong đó: img.ravel(): Đây là ảnh đã được làm phẳng thành một mảng một chiều. Điều này có nghĩa là tất cả các giá trị pixel của ảnh sẽ được đưa vào một mảng một chiều duy nhất để tính toán histogram 256 là số lượng bin histogram [0,255] phạm vi giá trị của histogram

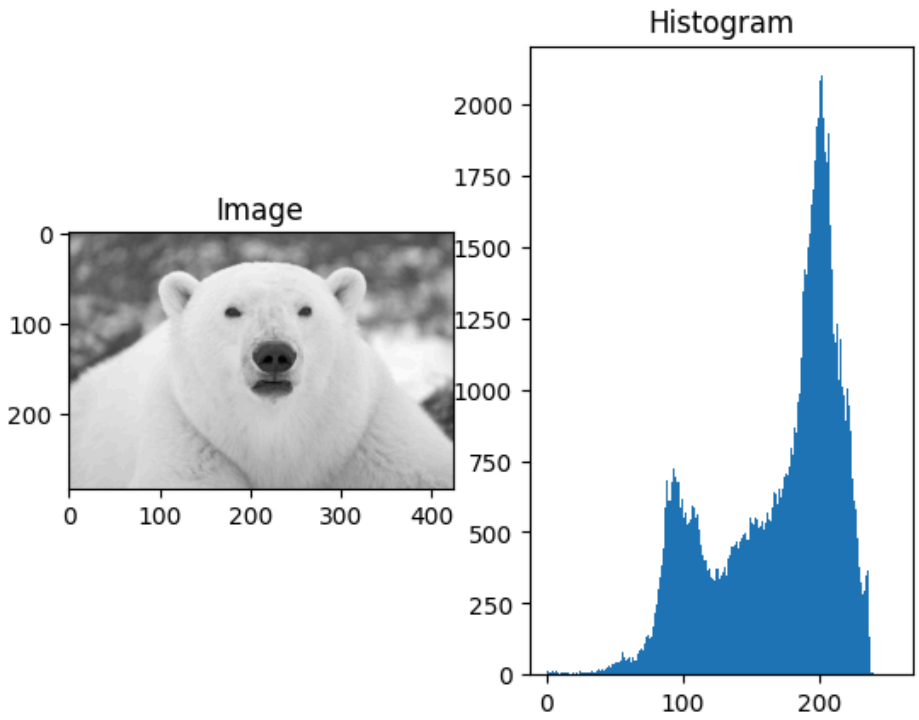
In [ ]:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Đọc ảnh đầu vào
img = cv2.imread('Bear.jpg', cv2.IMREAD_GRAYSCALE)

# Tính toán histogram
histnp, bins = np.histogram(img.ravel(), 256, [0, 256])

# Hiển thị ảnh gốc
plt.subplot(1, 2, 1)
plt.imshow(img, cmap='gray')
plt.title('Image')

# Hiển thị histogram
plt.subplot(1, 2, 2)
plt.plot(hist)
plt.title('Histogram')

plt.show()
```



2. Trình bày và giải thích cú pháp hàm cho phương pháp vẽ Histogram với Matplotlib, với OpenCV. Nêu ví dụ minh họa từng bước cụ thể.

In [ ]:
```
#Đọc ảnh
imgBear = cv2.imread('Bear.jpg', cv2.IMREAD_GRAYSCALE)
#Thông báo lỗi nếu đọc ảnh không thành công
assert imgBear is not None, "file could not be read, check with os.path.exists()"
#

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

# Hiển thị histogram
plt.subplot(1, 2, 2)
plt.hist(imgBear.ravel(),256,[0,256]);
plt.title('Histogram')

plt.show()
```

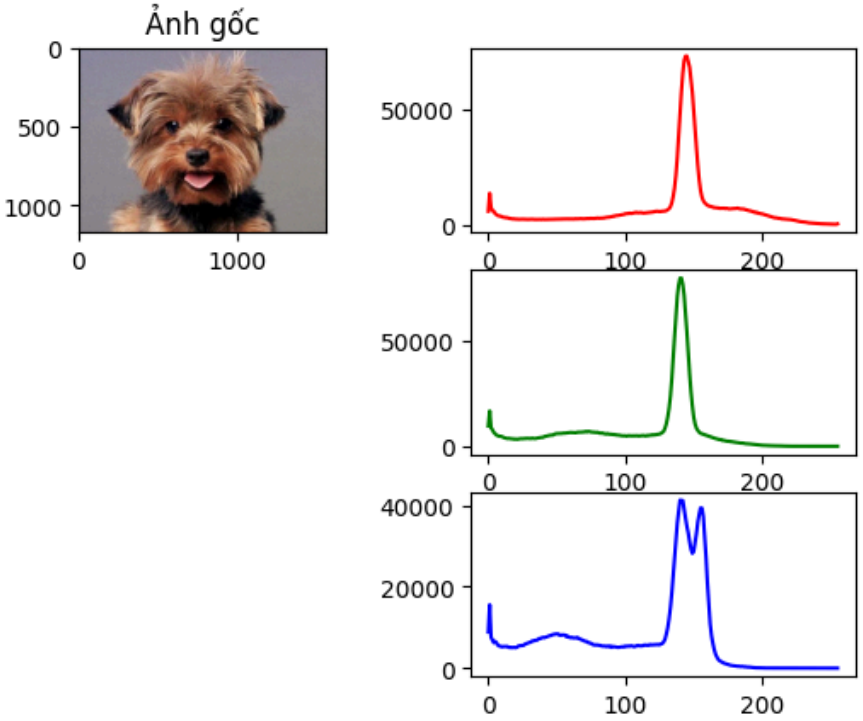


In [ ]:
```python
#dọc ảnh đầu vào là ảnh
imgBearCoLor = cv2.imread('Bear.jpg')
#Thông báo lỗi nếu đọc ảnh không thành công
assert imgBearCoLor is not None, "file could not be read, check with os.path.exists()"

# Dùng vòng lặp để tính toán và vẽ histogram cho mỗi kênh màu
for i, col in enumerate(color):
    # Tính toán histogram cho kênh màu thứ i của ảnh màu imgBearCoLor
    histr = cv2.calcHist([imgBearCoLor], [i], None, [256], [0, 256])

    # Vẽ histogram của kênh màu thứ i bằng màu tương ứng
    plt.plot(histr, color=col)

    # Đặt giới hạn trục x của biểu đồ từ 0 đến 256 (phạm vi giá trị của mỗi kênh màu)
    plt.xlim([0, 256])

# Hiển thị biểu đồ histogram của ảnh màu
plt.show()
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[6], line 7
      4 assert imgBearCoLor is not None, "file could not be read, check with os.path.exists()"
      6 # Dùng vòng lặp để tính toán và vẽ histogram cho mỗi kênh màu
----> 7 for i, col in enumerate(color):
      8     # Tính toán histogram cho kênh màu thứ i của ảnh màu imgBearCoLor
      9     histr = cv2.calcHist([imgBearCoLor], [i], None, [256], [0, 256])
     11     # Vẽ histogram của kênh màu thứ i bằng màu tương ứng

NameError: name 'color' is not defined
```

3. Vẽ Histogram trên các kênh màu của ảnh màu RGB theo ví dụ minh họa sau:

In [ ]:
```python
# Đọc ảnh màu từ file "Dog.jpg"
imgDogCoLor = cv2.imread('Dog.jpg')

# Kiểm tra xem việc đọc ảnh có thành công không. Nếu không thành công, thông báo lỗi sẽ được hiển thị.
assert imgDogCoLor is not None, "file could not be read, check with os.path.exists()"
imgDogRGB=cv2.cvtColor(imgDogCoLor, cv2.COLOR_BGR2RGB)
# Hiển thị ảnh gốc
plt.subplot(3, 2, 1)
plt.imshow(imgDogRGB,cmap='gray')  # Chuyển đổi màu không gian từ BGR sang RGB
plt.title('Ảnh gốc')
#thứ tự kênh màu
color = ('r','g','b')
index=0

# Dùng vòng lặp để tính toán và vẽ histogram cho mỗi kênh màu
for i, col in enumerate(color):
    index = index+2
    # Tính toán histogram cho kênh màu thứ i của ảnh màu imgDogRGB
    histr1 = cv2.calcHist([imgDogRGB], [i], None, [256], [0, 256])
    plt.subplot(3,2,index)

    # Vẽ histogram của kênh màu thứ i bằng màu tương ứng
    plt.plot(histr1, color=col)


# Hiển thị biểu đồ histogram của ảnh màu
plt.show()
```



Xử lý Histogram

In [ ]:
```python
#Trượt histogram: Đây là hàm để thêm một hằng số c vào mỗi giá trị của histogram.
def slide_histogram(image, c):
    output = image + c
    return output
```

In [ ]:
```python
#Căng histogram: Đây là hàm để nhân một hằng số c với mỗi giá trị của histogram.
def stretch_histogram(image, c):
    output = image * c
    return output
```

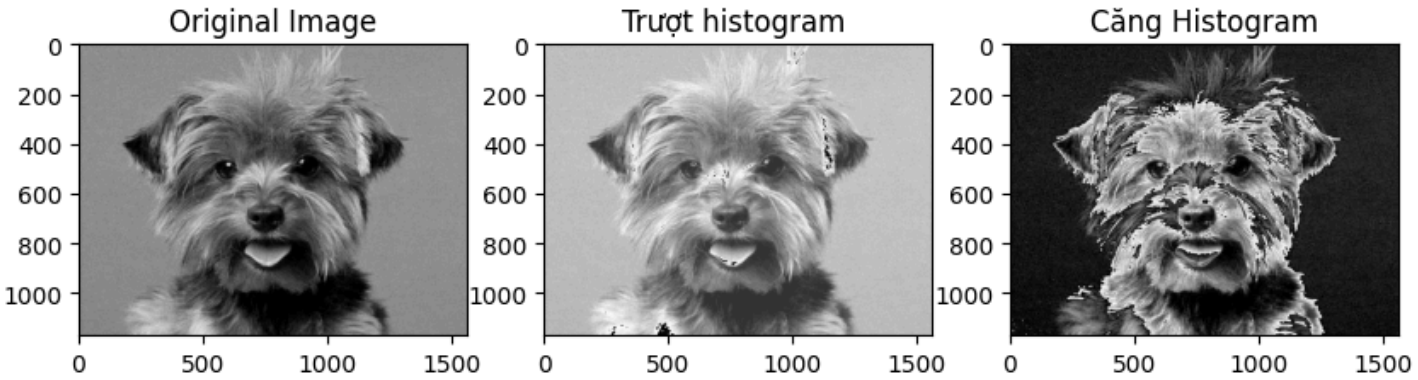
In [ ]:
```python
#Biến đổi tuyến tính histogram: Đây là hàm để biến đổi histogram theo công thức đã cho.
def linear_transform_histogram(image):
    min_val = np.min(image)
    max_val = np.max(image)
    output = (image - min_val) / (max_val - min_val) * 255
    return output
```

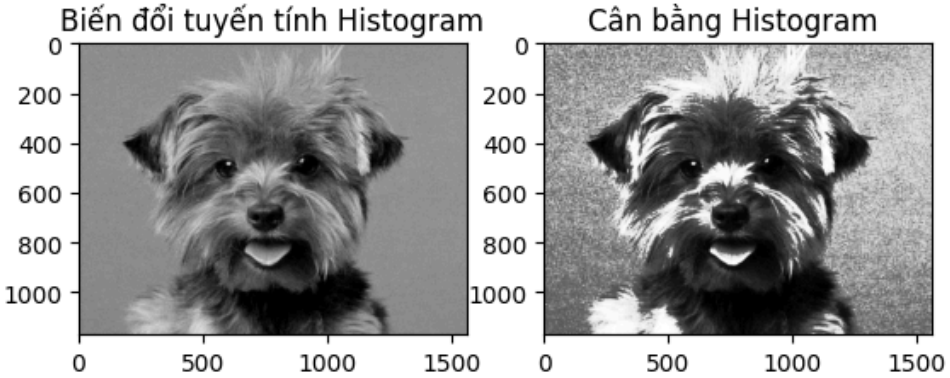
In [ ]:
```python
#Cân bằng histogram: Đây là hàm để cân bằng histogram của ảnh.
def histogram_equalization(image):
    hist, _ = np.histogram(image.flatten(), bins=256, range=[0,256])
    cdf = hist.cumsum()
    cdf_normalized = cdf * hist.max() / cdf.max()
    output = np.interp(image.flatten(), range(256), cdf_normalized).reshape(image.shape)
    return output
```

In [ ]:
```python
# Lấy một ảnh thử nghiệm
test_image = cv2.imread('Dog.jpg', cv2.IMREAD_GRAYSCALE)

# Chọn các hằng số cho các hàm
c_slide = 50
c_stretch = 2

# Trượt histogram
result_slide = slide_histogram(test_image, c_slide)

# Căng histogram
result_stretch = stretch_histogram(test_image, c_stretch)

# Biến đổi tuyến tính histogram
result_linear_transform = linear_transform_histogram(test_image)

# Cân bằng histogram
result_histogram_equalization = histogram_equalization(test_image)
```

In [ ]:
```python
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 8))

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(test_image, cmap='gray')
plt.title('Original Image')

# Kết quả sau khi trượt histogram histogram
plt.subplot(2, 3, 2)
plt.imshow(result_slide, cmap='gray')
plt.title('Trượt histogram ')

# Kết quả sau khi căng histogram
plt.subplot(2, 3, 3)
plt.imshow(result_stretch, cmap='gray')
plt.title('Căng Histogram')

# Kết quả sau khi biến đổi tuyến tính histogram
```

```
plt.subplot(2, 3, 4)
plt.imshow(result_linear_transform, cmap='gray')
plt.title('Biến đổi tuyến tính Histogram')

# Kết quả sau khi cân bằng histogram
plt.subplot(2, 3, 5)
plt.imshow(result_histogram_equalization, cmap='gray')
plt.title('Cân bằng Histogram ')

plt.show()
```

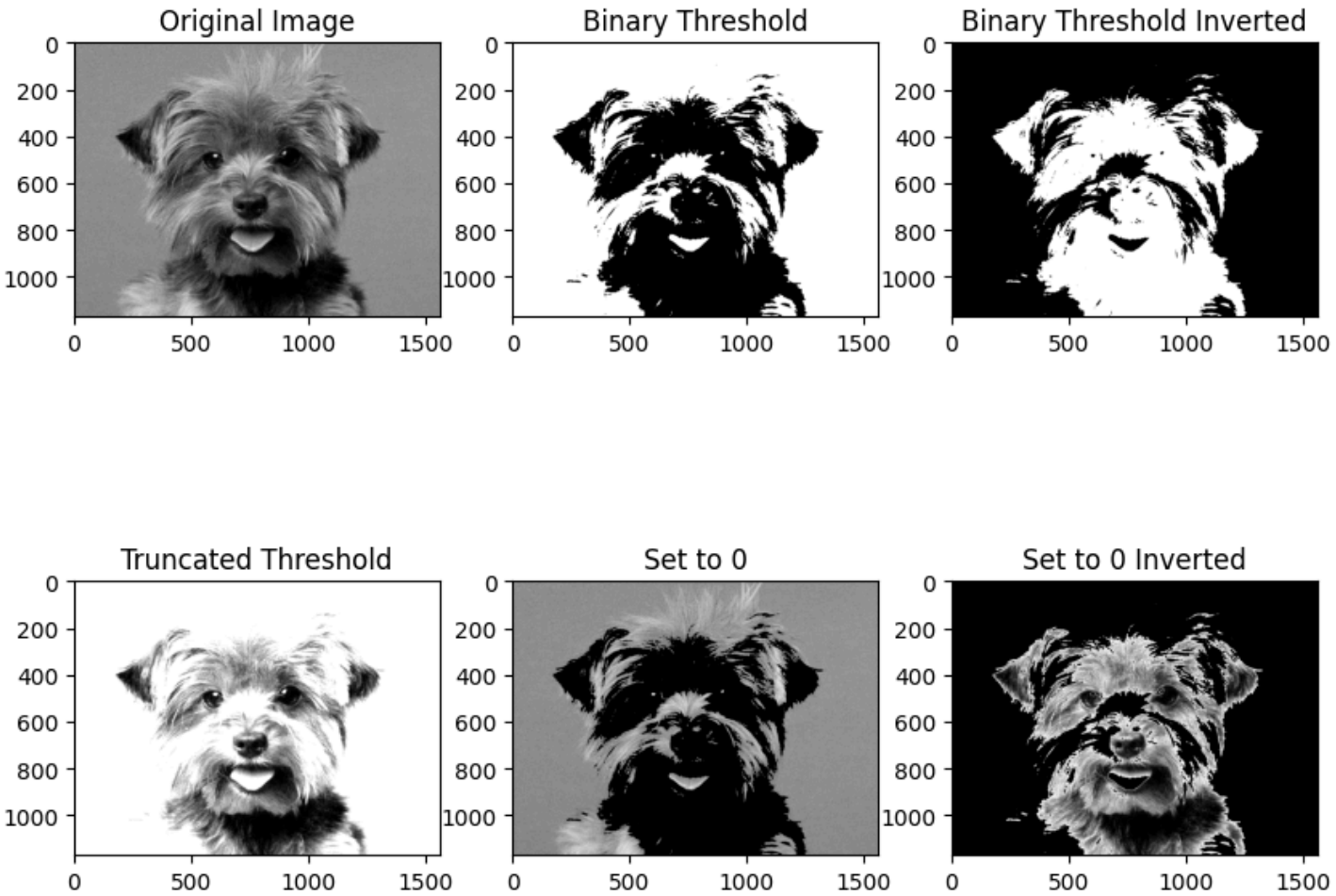




PHÂN NGƯỠNG ẢNH

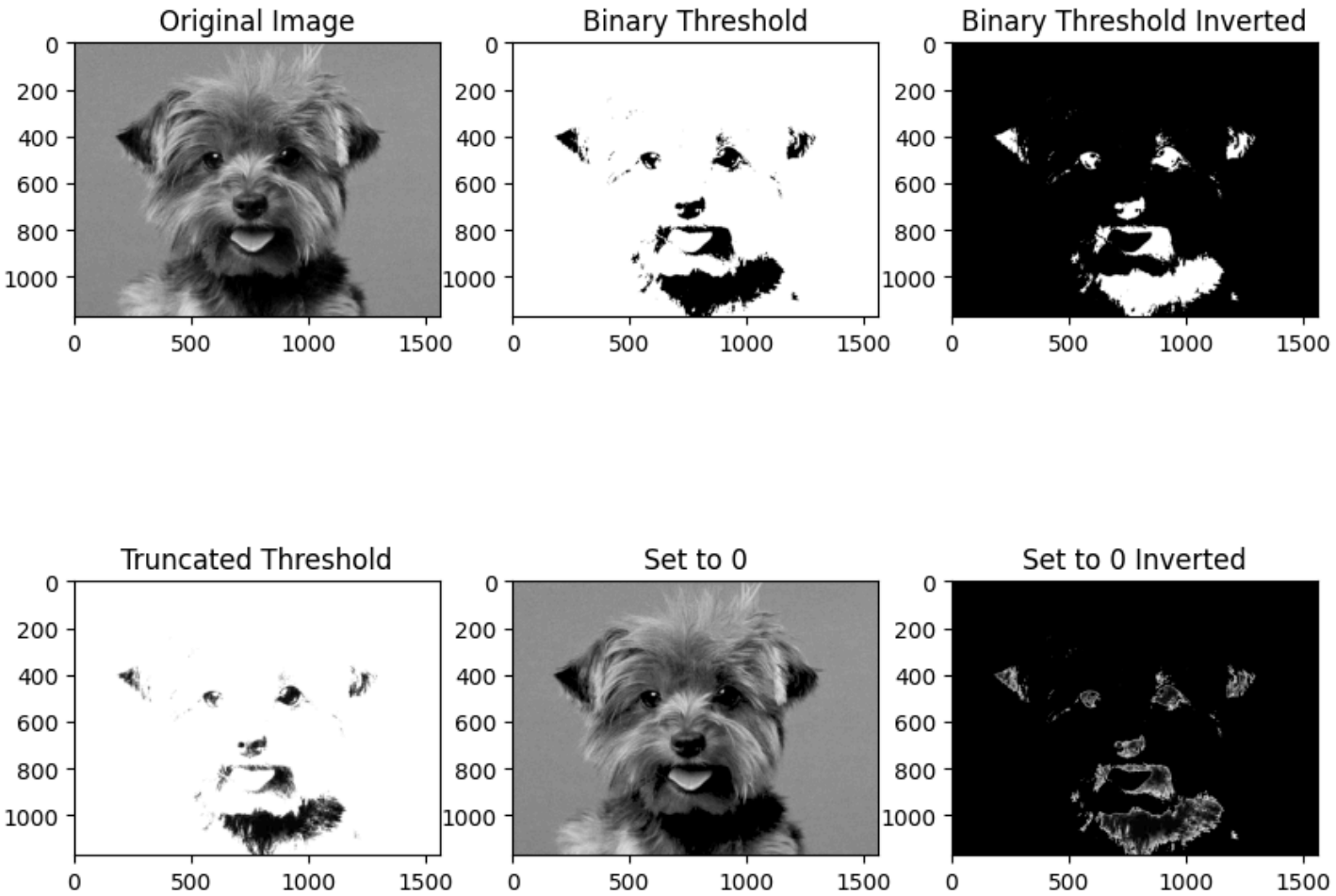
1. Phân ngưỡng đơn Phân ngưỡng đơn là gì? --> Phân ngưỡng đơn là so sánh các pixel với một ngưỡng giá trị. Nếu giá trị pixel nhỏ hơn ngưỡng thì sẽ được đặt thành 0, và ngược lại nếu sẽ thành giá trị tối đa là 255.
   -->Cú pháp hàm: cv2.threshold(source, ThresholdValue, maxVal, ThresringTechnique)
   -->> Trong đó:
   source : hình ảnh đầu vào (phải ở dạng Grayscale).
   ThresholdValue : Giá trị của Threshold bên dưới và bên trên giá trị pixel nào sẽ thay đổi tương ứng.
   maxVal : Giá trị tối đa có thể gán cho một pixel.
   ThresringTechnique : Loại ngưỡng được áp dụng.

In [ ]:
```python
#Đọc ảnh và chuyển đổi ảnh thành ảnh xám.
img = cv2.imread('Dog.jpg', cv2.IMREAD_GRAYSCALE)

# Các kỹ thuật phân ngưỡng

ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 120, 255, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 120, 255, cv2.THRESH_TOZERO_INV)


# Tạo một figure mới với các subplot
plt.figure(figsize=(10, 8))

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Các kỹ thuật ngưỡng hóa
titles = ['Binary Threshold', 'Binary Threshold Inverted', 'Truncated Threshold', 'Set to 0', 'Set to 0 Inverted']
thresh_images = [thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(5):
    plt.subplot(2, 3, i+2)  # subplot index bắt đầu từ 1, nên phải cộng thêm 1
    plt.imshow(thresh_images[i], cmap='gray')
    plt.title(titles[i])

# Hiển thị figure
plt.show()
```



In [ ]:
```python
#Đọc ảnh và chuyển đổi ảnh thành ảnh xám.
img = cv2.imread('Dog.jpg', cv2.IMREAD_GRAYSCALE)


# Các kỹ thuật phân ngưỡng

ret, thresh1 = cv2.threshold(img, 30, 155, cv2.THRESH_BINARY)
ret, thresh2 = cv2.threshold(img, 30, 155, cv2.THRESH_BINARY_INV)
ret, thresh3 = cv2.threshold(img, 30, 155, cv2.THRESH_TRUNC)
ret, thresh4 = cv2.threshold(img, 30, 155, cv2.THRESH_TOZERO)
ret, thresh5 = cv2.threshold(img, 30, 155, cv2.THRESH_TOZERO_INV)
```

```python
# Tạo một figure mới với các subplot
plt.figure(figsize=(10, 8))

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Các kỹ thuật ngưỡng hóa
titles = ['Binary Threshold', 'Binary Threshold Inverted', 'Truncated Threshold', 'Set to 0', 'Set to 0 Inverted']
thresh_images = [thresh1, thresh2, thresh3, thresh4, thresh5]

for i in range(5):
    plt.subplot(2, 3, i+2)  # subplot index bắt đầu từ 1, nên phải cộng thêm 1
    plt.imshow(thresh_images[i], cmap='gray')
    plt.title(titles[i])

# Hiển thị figure
plt.show()
```



2. Phân ngưỡng thích nghi Phân ngưỡng thích nghi là gì? -->Trong phân ngưỡng thích nghi, ngưỡng được tính toán không dựa trên một giá trị cố định cho toàn bộ ảnh, mà thay vào đó nó được tính toán dựa trên một vùng cụ thể của ảnh

   -->Cú pháp hàm: cv2.adaptiveThreshold(source, maxVal, adaptiveMethod, thresholdType, blocksize, constant)

   -->Trong đó

   source : hình ảnh đầu vào

   maxVal : Giá trị tối đa có thể được gán cho một pixel.

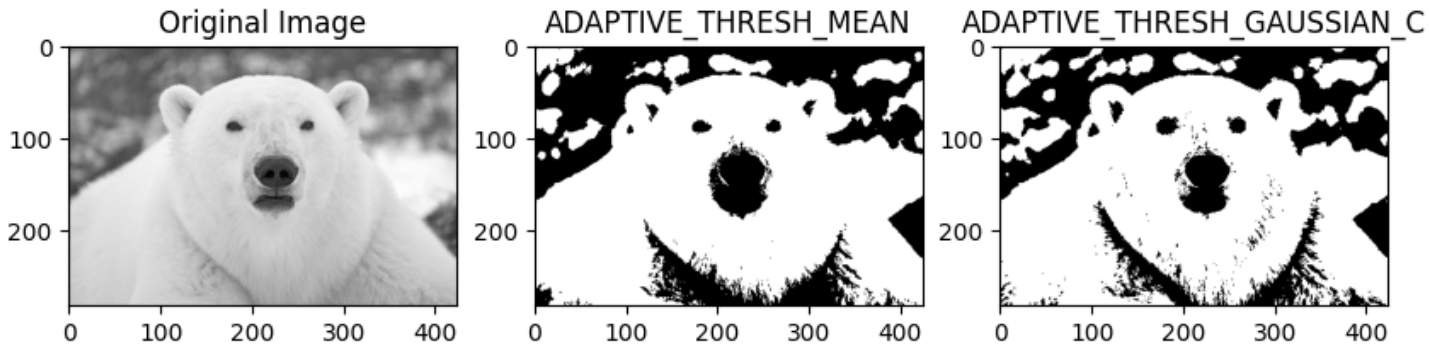
   AdaptMethod : Phương thức tính giá trị ngưỡng.

   1/cv2.ADAPTIVE_THRESH_MEAN_C : Giá trị ngưỡng = (Giá trị trung bình của các giá trị khu vực lân cận – giá trị không đổi).

   Nói cách khác, nó là giá trị trung bình của vùng lân cận blockSize×blockSize của một điểm trừ hằng số.

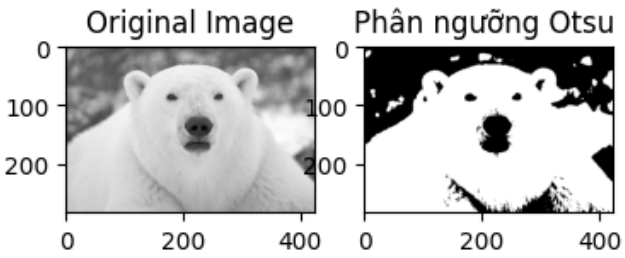
2/cv2.ADAPTIVE_THRESH_GAUSSIAN_C : Giá trị ngưỡng = (Tổng trọng số Gaussian của các giá trị lân cận – giá trị không đổi).
Nói cách khác, nó là tổng có trọng số của vùng lân cận blockSize×blockSize của một điểm trừ hằng số.
thresholdType : Loại ngưỡng được áp dụng.
blocksize : Kích thước của vùng lân cận pixel được sử dụng để tính giá trị ngưỡng.
constant: Một giá trị không đổi được trừ khỏi tổng trung bình hoặc trọng số của các pixel lân cận.

In [ ]:
```
img = cv2.imread('Bear.jpg', cv2.IMREAD_GRAYSCALE)

# Các kỹ thuật phân ngưỡng thích ứng

thresh1 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_MEAN_C,
                                                                    cv2.THRESH_BINARY, 199, 5)

thresh2 = cv2.adaptiveThreshold(img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                                                    cv2.THRESH_BINARY, 199, 5)

# Tạo một figure mới với các subplot
plt.figure(figsize=(10, 8))

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Các kỹ thuật ngưỡng hóa
titles = ['ADAPTIVE_THRESH_MEAN', 'ADAPTIVE_THRESH_GAUSSIAN_C']
thresh_images = [thresh1, thresh2]

for i in range(2):
    plt.subplot(2, 3, i+2)  # subplot index bắt đầu từ 1, nên phải cộng thêm 1
    plt.imshow(thresh_images[i], cmap='gray')
    plt.title(titles[i])

# Hiển thị figure
plt.show()
```



3. Phân ngưỡng Otsu Phân Ngưỡng Otsu -->Phân ngưỡng Otsu là một kỹ thuật tự động để phân loại ảnh thành hai lớp (nhị phân), được sử dụng rộng rãi trong xử lý ảnh.
   Trong phân ngưỡng Otsu, thay vì chọn một ngưỡng cụ thể, ngưỡng được xác định tự động dựa trên phân bố của các giá trị pixel trong ảnh
   -->Cú pháp hàm: cv2.threshold(source, thresholdValue, maxVal, thresholdingTechnique)
   --> Trong đó
   source: Mảng Ảnh đầu vào (phải ở dạng Grayscale)
   thresholdValue: Giá trị của Threshold bên dưới và bên trên giá trị pixel nào sẽ thay đổi tương ứng
   maxVal: Giá trị tối đa có thể gán cho một pixel.
   thresholdingTechnique: Giá trị tối đa có thể gán cho một pixel.

In [ ]:
```
img = cv2.imread('Bear.jpg', cv2.IMREAD_GRAYSCALE)
ret, thresh1 = cv2.threshold(img, 120, 255, cv2.THRESH_BINARY +
                                                                    cv2.THRESH_OTSU)

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original Image')

# Kết quả sau khi phân ngưỡng Otsu
plt.subplot(2, 3, 2)
plt.imshow(thresh1, cmap='gray')
plt.title('Phân ngưỡng Otsu ')
# Hiển thị figure
plt.show()
```



LỌC ẢNH

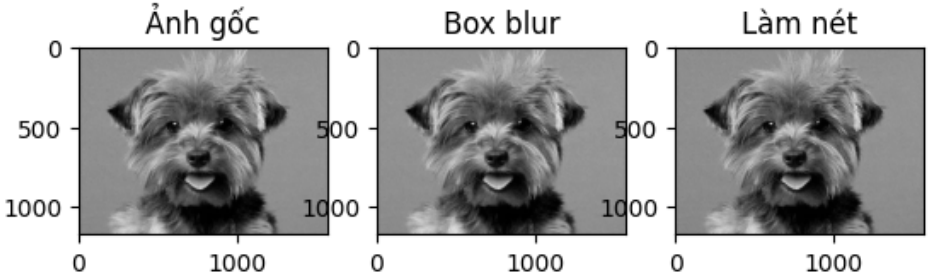
1. Hãy trình bày cú pháp hàm và giải thích các đối số tương ứng cho phương pháp
   lọc tuyến tính (sử dụng tích chập) trên ảnh
   Cú pháp hàm và các đối số tương ứng cho phương pháp lọc tuyến tính (sử dụng tích chập) trên ảnh:
   filtered_image = cv2.filter2D(image, ddepth, kernel[, dst[, anchor[, delta[, borderType]]]])

Trong đó
image: Ảnh đầu vào cần được lọc.
ddepth: Kiểu dữ liệu của ảnh đầu ra. Nếu đặt thành -1, thì kiểu dữ liệu của ảnh đầu ra sẽ giống với ảnh đầu vào.
kernel: Ma trận kernel hoặc bộ lọc được áp dụng lên ảnh. Đây là ma trận các trọng số được sử dụng để tính toán giá trị mới cho mỗi pixel trong ảnh đầu vào.
dst: Ảnh đầu ra (không bắt buộc).
anchor: Vị trí của trung tâm của kernel. Mặc định là (-1, -1), tức là trung tâm của kernel là trung tâm của kernel.
delta: Giá trị được thêm vào sau khi tính toán tích chập (không bắt buộc).
borderType: Kiểu biên của ảnh. Mặc định là cv2.BORDER_DEFAULT.

2. Thực hiện lọc làm mịn (box blur) và lọc làm nét ảnh các bước sau:

In [ ]:
```python
# Bước 1: Đọc ảnh và chuyển ảnh đa cấp xám
image = cv2.imread('Dog.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Bước 2: Tạo bộ lọc trung bình (box blur) và bộ lọc box blur
box_blur_kernel = np.ones((3, 3), np.float32) / 9  # Bộ lọc trung bình
sharpen_kernel = np.array([[0, -1, 0], [-1, 5, -1], [0, -1, 0]], np.float32)  # Bộ lọc làm nét

# Bước 3: Thực hiện lọc ảnh với các bộ lọc trên
smoothed_image = cv2.filter2D(gray_image, -1, box_blur_kernel)
sharpened_image = cv2.filter2D(gray_image, -1, sharpen_kernel)
```

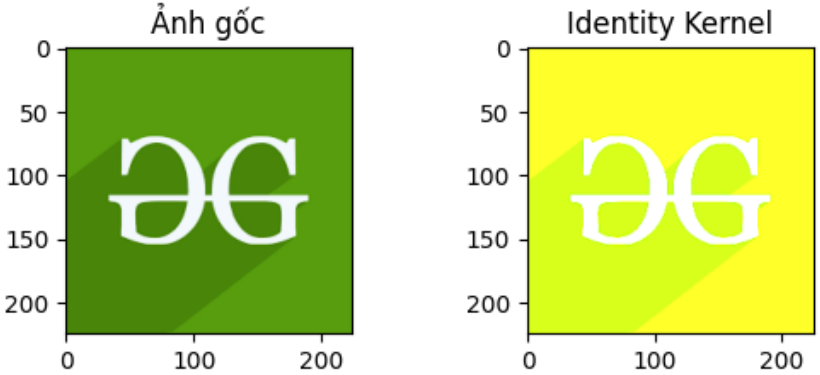
```
#Bước 4: Hiển thị kết quả

# Ảnh gốc
plt.subplot(2, 3, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Ảnh gốc')

# Kết quả Box blur
plt.subplot(2, 3, 2)
plt.imshow(smoothed_image, cmap='gray')
plt.title('Box blur')
plt.subplot(2, 3, 3)
# Kết quả làm nét

plt.imshow(sharpened_image, cmap='gray')
plt.title('Làm nét')
# Hiển thị figure
plt.show()
```



3. Tìm hiểu Gaussian Blur và Median Blur thông qua link tham khảo sau đây:

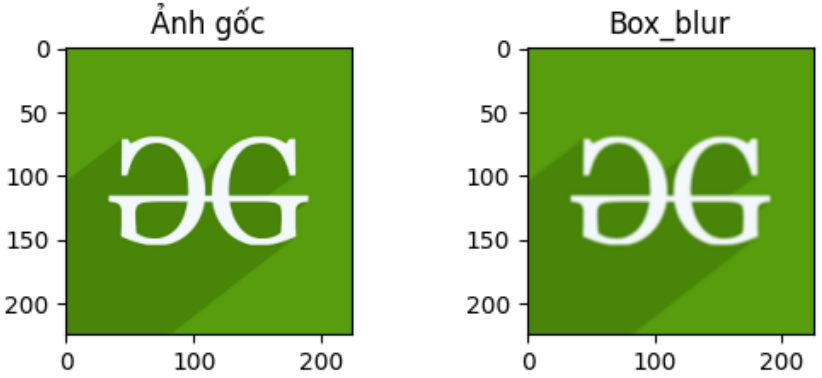
In [ ]:
```
#Identity Kernel
# Importing OpenCV and Numpy Libraries

#Đọc ảnh
img = cv2.imread('geeksforgeeks.png')

#Tạo ma trận
id_kernel = np.array([[0, 0, 0],
                                        [0, 3, 0],
                                        [0, 0, 0]])

#Sử dụng hàm cv2.filter2D() để áp dụng bộ lọc đã tạo lên hình ảnh đầu vào. Kết quả được lưu trong biến flt_img
```

```
flt_img = cv2.filter2D(src=img, ddepth=-1, kernel=id_kernel)

# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')

# Kết quả sau Identity Kernel
plt.subplot(2, 2, 2)
plt.imshow(flt_img)
plt.title('Identity Kernel')



# Hiển thị figure
plt.show()
```



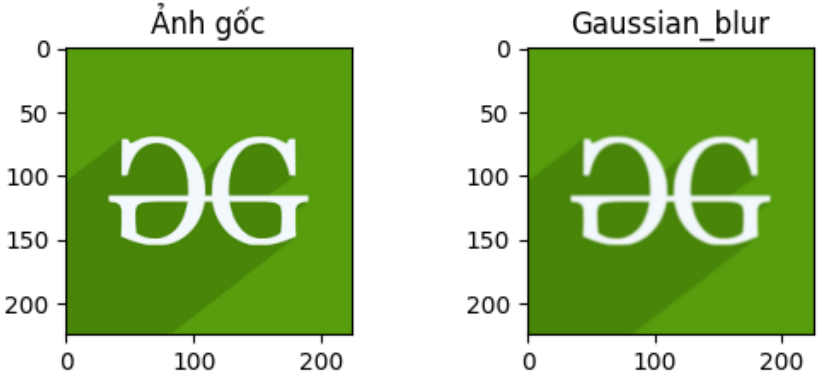
In [ ]:
```
#Blurring

img = cv2.imread('geeksforgeeks.png')


#Tạo ma trận bộ lọc cho hiệu ứng làm mờ hộp (box blur filter):
box_blur_ker = np.array([[0.1111111, 0.1111111, 0.1111111],
                         [0.1111111, 0.1111111, 0.1111111],
                         [0.1111111, 0.1111111, 0.1111111]])
#Áp dụng bộ lọc Blurring
Box_blur = cv2.filter2D(src=img, ddepth=-1, kernel=box_blur_ker)

# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')
```

```
# Kết quả sau Box_blur
plt.subplot(2, 2, 2)
plt.imshow(Box_blur)
plt.title('Box_blur')



# Hiển thị figure
plt.show()
```

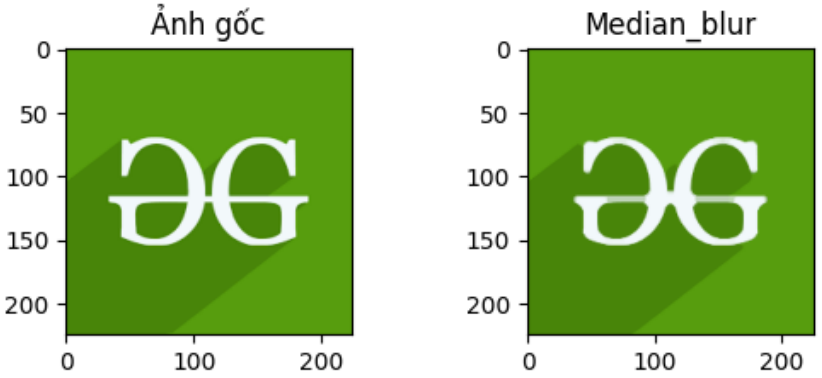


In [ ]:
```
#Gaussian Blur
img = cv2.imread('geeksforgeeks.png')


#Áp dụng bộ lọc GaussianBlur
#trong đó:
#src: Đây là hình ảnh nguồn mà bạn muốn áp dụng hiệu ứng làm mờ lên
#ksize: Kích thước của kernel Gaussian
#sigmaX: Độ lệch chuẩn của trục X trong hàm Gaussian
#sigmaY: Độ lệch chuẩn của trục Y trong hàm Gaussian
gaussian_blur = cv2.GaussianBlur(src=img, ksize=(3,3),sigmaX=1, sigmaY=0)

# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')

# Kết quả sau Gaussian_blur
plt.subplot(2, 2, 2)
plt.imshow(gaussian_blur)
plt.title('Gaussian_blur')
```

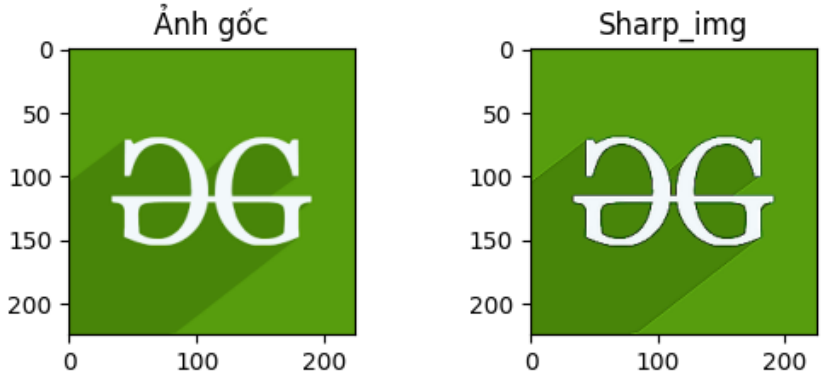
```
# Hiển thị figure
plt.show()
```



In [ ]:
```
#Median Blur:
img = cv2.imread('geeksforgeeks.png')


#Áp dụng bộ lọc medianBlur
#trong đó:
#src: Đây là hình ảnh nguồn mà bạn muốn áp dụng hiệu ứng làm mờ lên
#ksize: Kích thước của kernel Gaussian () (9 là ma trận 9x9)
median_blur = cv2.medianBlur(src=img, ksize=9)
# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')

# Kết quả sau Median_blur
plt.subplot(2, 2, 2)
plt.imshow(median_blur)
plt.title('Median_blur')



# Hiển thị figure
plt.show()
```

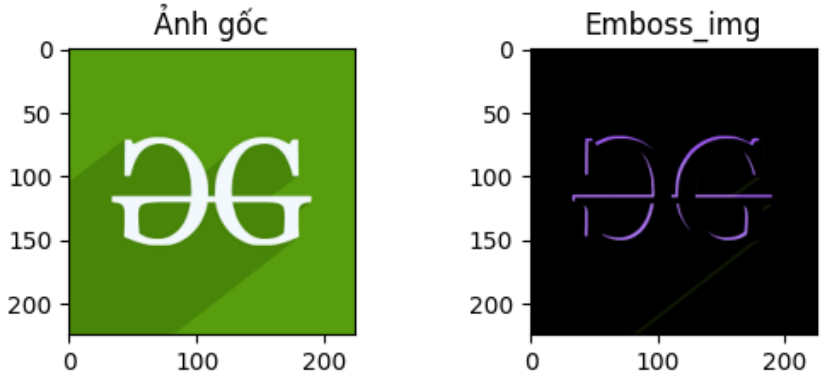


In [ ]:
```python
#Sharpening

img = cv2.imread('geeksforgeeks.png')

# Tạo ma trận sharp_kernel
sharp_kernel = np.array([[0, -1, 0],
                         [-1, 5, -1],
                         [0, -1, 0]])

#Áp dụng bộ lọc Sharpening

sharp_img = cv2.filter2D(src=img, ddepth=-1, kernel=sharp_kernel)

# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')

# Kết quả sau Sharpening
plt.subplot(2, 2, 2)
plt.imshow(sharp_img)
plt.title('Sharp_img')



# Hiển thị figure
plt.show()
```



In [ ]:

```python
#Emboss
img = cv2.imread('geeksforgeeks.png')


# # Tạo ma trận embossing
emboss_kernel = np.array([[-1, 0, 0],
                    [0, 0, 0],
                    [0, 0, 1]])

emboss_img = cv2.filter2D(src=img, ddepth=-1, kernel=emboss_kernel)

# Ảnh gốc
plt.subplot(2, 2, 1)
plt.imshow(img)
plt.title('Ảnh gốc')

# Kết quả sau Emboss
plt.subplot(2, 2, 2)
plt.imshow(emboss_img)
plt.title('Emboss_img')



# Hiển thị figure
plt.show()
```



TẠO ẢNH LAI (HYBRID IMAGE) – XỬ LÝ ẢNH TRONG MIỀN TẦN SỐ

In [ ]:
```python
#Custom Function for Image Plotting
import cv2
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from math import sqrt, exp


def plot_figure(images: list, titles: list, rows: int, columns: int, fig_width=15, fig_height=7):
    fig = plt.figure(figsize=(fig_width, fig_height))
    count = 1
    for image, title in zip(images, titles):
        fig.add_subplot(rows, columns, count)
        count += 1
        plt.imshow(image, 'gray')
        plt.axis('off')
        plt.title(title)


#Custom Function to get Euclidean Distance
def distance(point1, point2):
    return sqrt((point1[0] - point2[0]) ** 2 + (point1[1] - point2[1]) ** 2)
```

In [ ]:
```python
#Bước 1: Lấy thành phần tần số thấp của ảnh Dog
#Function to get low frequency component
#D0 is cutoff frequency
def gaussianLP(D0, imgShape):
    base = np.zeros(imgShape[:2])
```

```
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for i in range(rows):
        for j in range(cols):
            base[i, j] = np.exp(-distance((i, j), center)**2 / (2 * D0**2))
    return base
```

In [ ]:
```
#Bước 2: Lấy thành phần tần số cao của ảnh Bear
#Function to get high frequency component
#D0 is cutoff frequency
def gaussianHP(D0, imgShape):
    base = np.zeros(imgShape[:2])
    rows, cols = imgShape[:2]
    center = (rows/2, cols/2)
    for i in range(rows):
        for j in range(cols):
            base[i, j] = 1 - np.exp(-distance((i, j), center)**2 / (2 * D0**2))
    return base
```

In [ ]:
```
#Function to generate hybrid image
#D0 is cutoff frequency
def hybrid_images(image1, image2, D0 = 50):
    original1 = np.fft.fft2(image1)                          #Get the fourier of image1
    center1 = np.fft.fftshift(original1)                     #Apply Centre shifting
    LowPassCenter = center1 * gaussianLP(D0, image1.shape)   #Extract low frequency component
    LowPass = np.fft.ifftshift(LowPassCenter)
    inv_LowPass = np.fft.ifft2(LowPass)                         #Get image using Inverse FFT

    original2 = np.fft.fft2(image2)
    center2 = np.fft.fftshift(original2)
    HighPassCenter = center2 * gaussianHP(D0, image2.shape)  #Extract high frequency component
    HighPass = np.fft.ifftshift(HighPassCenter)
    inv_HighPass = np.fft.ifft2(HighPass)
    hybrid = np.abs(inv_LowPass) + np.abs(inv_HighPass)      #Generate the hybrid image
    return hybrid
```

In [ ]:
```
# Load images
# Make sure to choose the same image format for both images (Ex- .png)
A = cv2.imread('Dog.jpg') # high picture
B = cv2.imread('Bear.jpg') # low picture

# Convert both images to Grayscale to avoid any Color Channel Issue
A_grayscale = cv2.cvtColor(A, cv2.COLOR_BGR2GRAY)
B_grayscale = cv2.cvtColor(B, cv2.COLOR_BGR2GRAY)

# Resize both images to 128x128 to avoid different image size issue
A_resized = cv2.resize(A_grayscale, (128, 128))
```

```
B_resized = cv2.resize(B_grayscale, (128, 128))

result = hybrid_images(A_resized,B_resized,1)

plot_figure([A,B,result], ['A','B','Hybrid Image'],1,3)
```