Họ tên : Nguyễn Quang Anh

Mã sinh viên : B21DCDT036

CHƯƠNG 3 : TÍCH CHẬP VÀ LỌC TRONG MIỀN TẦN SỐ (Tích chập và Lọc trong Miền Tần số )

THUẬT TOÁN 1 : BỘ LỌC GAUSSIAN TRONG MIỀN TẦN SỐ VỚI numpy fft

**1,Giới thiệu thuật toán lọc Gaussian trong miền tần số**

Thuật toán lọc Gaussian trong miền tần số dựa trên **định lý tích chập**, trong đó tích chập trong miền không gian tương đương với phép nhân trong miền tần số. Điều này giúp thực hiện phép lọc nhanh hơn và hiệu quả hơn, đặc biệt với ảnh kích thước lớn.

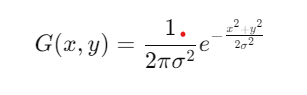
Quy trình cơ bản:

1. Chuyển ảnh gốc và kernel lọc (Gaussian) từ miền không gian sang miền tần số bằng **Biến đổi Fourier 2D**.
2. Nhân phổ của ảnh với phổ của kernel trong miền tần số.
3. Chuyển kết quả ngược về miền không gian bằng **Biến đổi Fourier ngược** để nhận được ảnh đã được lọc.

**2,Các bước thực hiện thuật toán** :

**Bước 1: Chuẩn bị ảnh và kernel Gaussian**

* **Ảnh gốc**: Chuyển ảnh sang dạng grayscale (nếu là ảnh màu) để xử lý đơn giản hơn.
* **Kernel Gaussian**: Là một ma trận đại diện cho hàm Gaussian 2D, có tác dụng làm mờ (blur) ảnh. Hàm Gaussian được định nghĩa:



* x,y : tọa độ của pixel.
* Σ (sigma) : độ lệch chuẩn, điều khiển độ rộng của kernel Gaussian.

**Bước 2: Biến đổi Fourier 2D**

* Sử dụng **Biến đổi Fourier 2D** (numpy.fft.fft2) để chuyển ảnh và kernel từ miền không gian sang miền tần số.

**Bước 3: Nhân phổ trong miền tần số**

* Theo định lý tích chập: FFT(f∗g)=FFT(f)⋅FFT(g) . Trong đó:
  + f: ảnh gốc.
  + g: kernel Gaussian.
  + ∗ : phép tích chập.
* Nhân phổ của ảnh và kernel trong miền tần số.

**Bước 4: Biến đổi Fourier ngược**

* Áp dụng **Biến đổi Fourier ngược** (numpy.fft.ifft2) để chuyển kết quả từ miền tần số về miền không gian, thu được ảnh đã lọc.

**Bước 5: Hiển thị kết quả**

* Hiển thị ảnh gốc, kernel, và ảnh đã lọc để quan sát kết quả.
* Biểu diễn phổ tần số của ảnh và kernel để minh họa quá trình xử lý trong miền tần số.

**3,Giải thích đoạn mã nguồn (hàm hay API) :**

import numpy as np  
import pylab  
from scipy import signal  
from scipy.fft import fft2, ifft2, fftshift, ifftshift  
from imageio import imread  
  
pylab.figure(figsize=(15, 10)) # Kích thước figure lớn hơn  
pylab.gray() # Hiển thị ảnh dạng grayscale  
  
# Load and process the image  
im = np.mean(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\swans.jpg"), axis=2)  
  
# Create Gaussian kernel  
gauss\_kernel = np.outer(signal.windows.gaussian(im.shape[0], 5), signal.windows.gaussian(im.shape[1], 5))  
  
# Perform FFT on the image and kernel  
freq = fft2(im)  
assert freq.shape == gauss\_kernel.shape  
freq\_kernel = fft2(ifftshift(gauss\_kernel))  
  
# Convolve in the frequency domain  
convolved = freq \* freq\_kernel  
im1 = ifft2(convolved).real  
  
# Plot results  
pylab.subplot(2, 3, 1), pylab.imshow(im), pylab.title('Original Image', size=16), pylab.axis('off')  
pylab.subplot(2, 3, 2), pylab.imshow(gauss\_kernel), pylab.title('Gaussian Kernel', size=16), pylab.axis('off')  
pylab.subplot(2, 3, 3), pylab.imshow(im1), pylab.title('Output Image', size=16), pylab.axis('off')  
pylab.subplot(2, 3, 4), pylab.imshow((20 \* np.log10(0.1 + fftshift(freq))).astype(int))  
pylab.title('Original Image Spectrum', size=16), pylab.axis('off')  
pylab.subplot(2, 3, 5), pylab.imshow((20 \* np.log10(0.1 + fftshift(freq\_kernel))).astype(int))  
pylab.title('Gaussian Kernel Spectrum', size=16), pylab.axis('off')  
pylab.subplot(2, 3, 6), pylab.imshow((20 \* np.log10(0.1 + fftshift(convolved))).astype(int))  
pylab.title('Output Image Spectrum', size=16), pylab.axis('off')  
  
# Adjust spacing  
pylab.subplots\_adjust(wspace=0.3, hspace=0.3) # Tăng khoảng cách giữa các subplot  
pylab.show()

**Import các thư viện cần thiết** :

* **numpy**: Dùng để xử lý ma trận và các phép toán số học.
* **pylab**: Dùng để tạo các biểu đồ và hiển thị hình ảnh.
* **scipy.signal**: Cung cấp hàm để tạo kernel Gaussian.
* **scipy.fft**: Thực hiện phép biến đổi Fourier 2D (FFT) và các phép toán liên quan.
* **imageio.imread**: Đọc ảnh từ tệp.

**Tạo figure và thiết lập ảnh grayscale**:

* **pylab.figure(figsize=(15, 10))**: Tạo canvas hiển thị có kích thước 15x10 inch.
* **pylab.gray()**: Thiết lập tất cả hình ảnh được hiển thị ở dạng grayscale (ảnh xám).

**Đọc ảnh và chuyển sang grayscale**:

* **imread**: Đọc tệp ảnh từ đường dẫn cung cấp.
* **np.mean(..., axis=2)**: Chuyển ảnh màu (RGB) sang grayscale bằng cách lấy trung bình giá trị các kênh màu.

**Tạo Gaussian kernel**:

* **signal.windows.gaussian**: Tạo một mảng Gaussian 1D với độ rộng chuẩn (standard deviation) là 5.
* **np.outer**: Tính tích ngoài (outer product) để tạo Gaussian kernel 2D từ hai mảng Gaussian 1D.

**Thực hiện biến đổi Fourier (FFT)**:

* **fft2(im)**: Tính biến đổi Fourier 2D của ảnh gốc.
* **ifftshift(gauss\_kernel)**: Dịch chuyển Gaussian kernel về trung tâm (vì FFT yêu cầu tâm của kernel phải ở vị trí (0,0)).
* **fft2(ifftshift(...))**: Tính FFT của Gaussian kernel sau khi dịch chuyển.

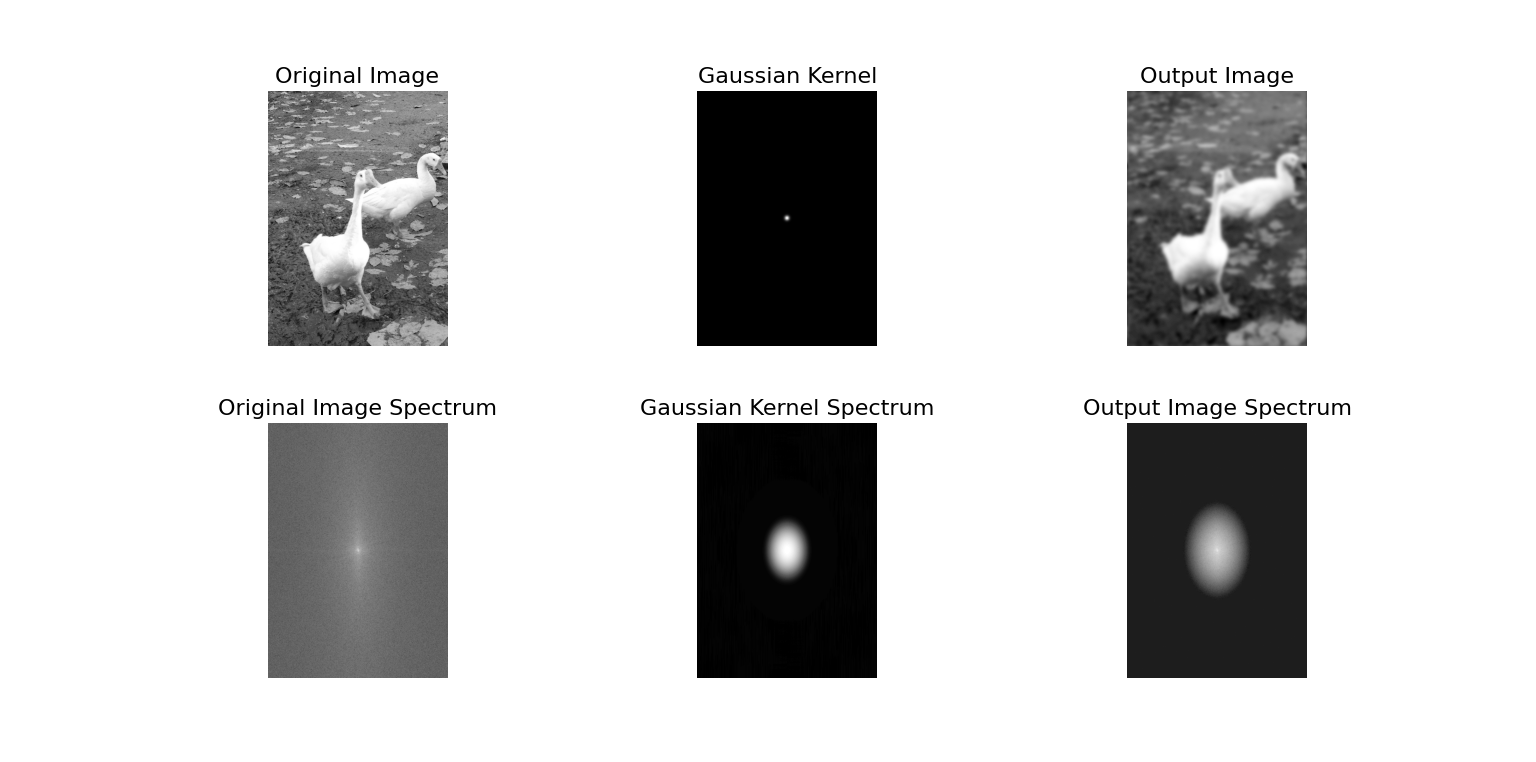
**Nhân chập trong miền tần số**:

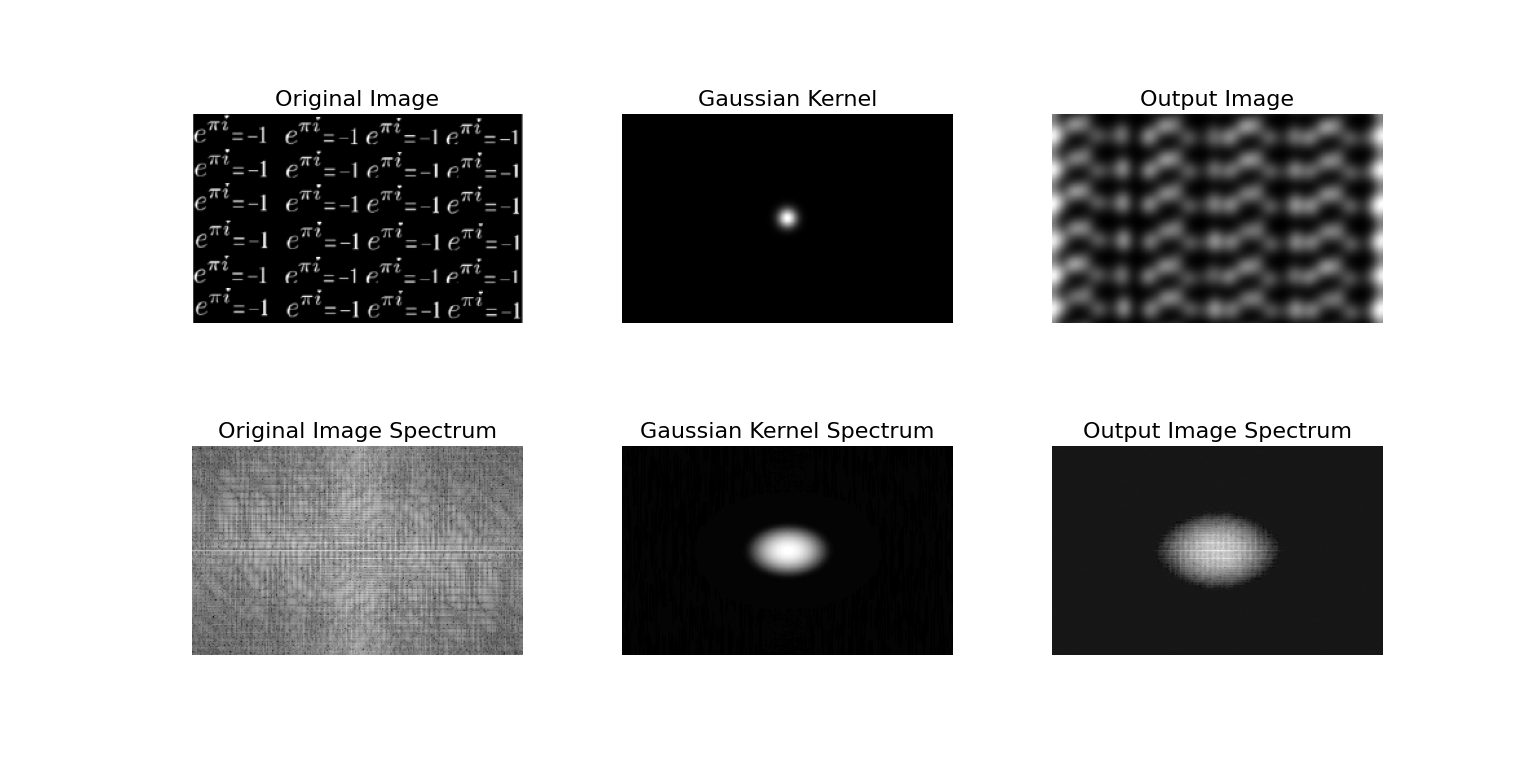
* **freq \* freq\_kernel**: Nhân tín hiệu ảnh gốc và Gaussian kernel trong miền tần số (tương đương phép nhân chập trong miền không gian).
* **ifft2(convolved)**: Áp dụng phép biến đổi Fourier ngược (IFFT) để đưa ảnh trở lại miền không gian.
* **.real**: Lấy phần thực của ảnh kết quả (bỏ qua phần ảo do sai số tính toán).

**Hiển thị kết quả**:

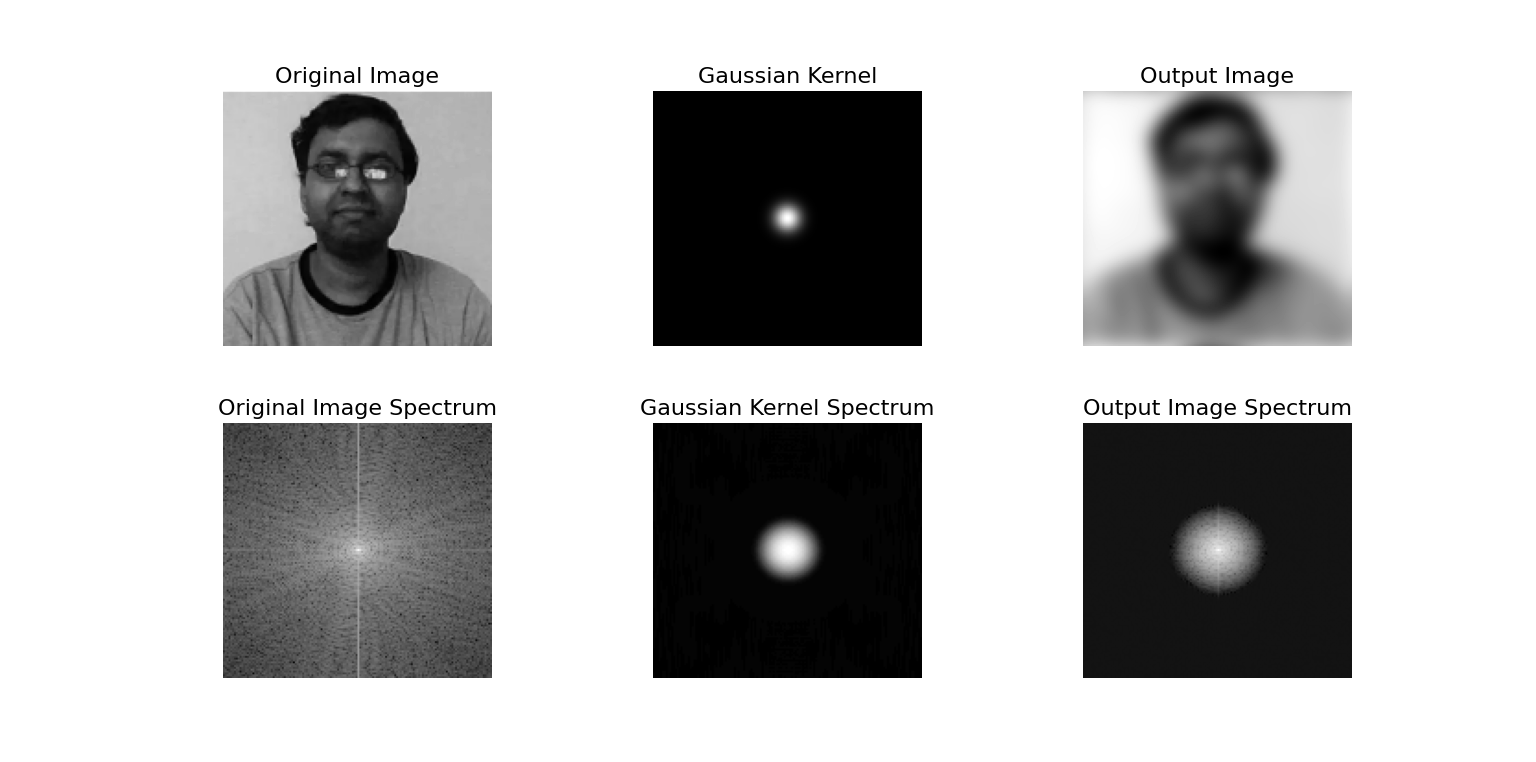
* **subplot(2, 3, i)**: Chia vùng hiển thị thành lưới 2x3 và đặt mỗi hình ở vị trí tương ứng (1 đến 6).
* **imshow(...)**: Hiển thị hình ảnh trong subplot.
* **title(...)**: Gắn tiêu đề cho mỗi hình.
* **axis('off')**: Tắt hiển thị trục tọa độ.

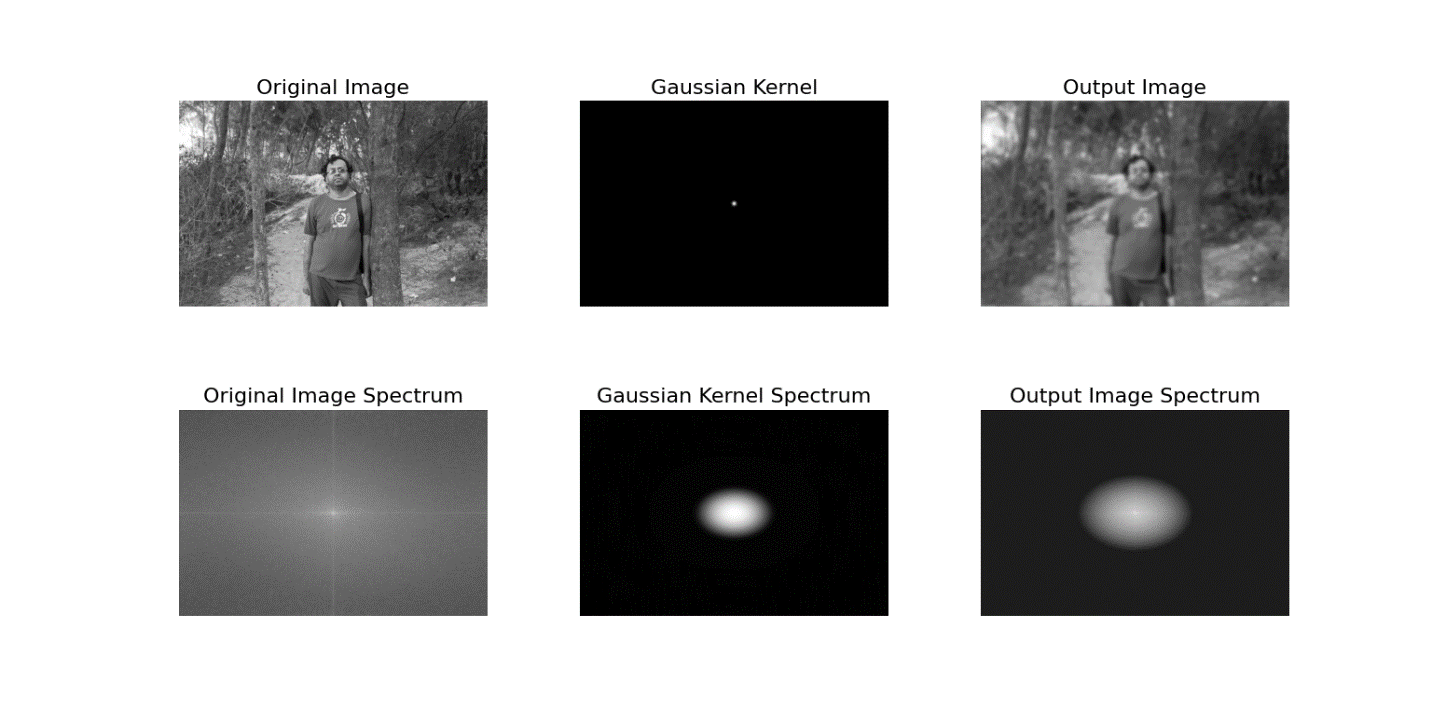
4,Kết quả sau khi thực hiện thuật toán :

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 :



Ví dụ 4 : 

Nhận xét kết quả :

**1. Original Image (Ảnh gốc)**

* **Nhận xét**:  
  Ảnh gốc (trái trên cùng) được chuyển sang thang độ xám (grayscale) và hiển thị rõ ràng với đầy đủ chi tiết. Điều này chứng minh rằng bước xử lý ban đầu (chuyển ảnh RGB sang grayscale bằng cách lấy trung bình các kênh màu) đã được thực hiện chính xác.

**2. Gaussian Kernel**

* **Nhận xét**:  
  Gaussian kernel (giữa trên) được tạo với giá trị lớn nhất ở trung tâm và giảm dần theo hình dạng đối xứng, giống như một "đỉnh chuông". Kernel này có cường độ rất thấp ở các vùng biên, cho thấy độ rộng chuẩn (sigma) của kernel khá nhỏ, làm cho Gaussian kernel tập trung nhiều hơn vào các vùng lân cận trung tâm.  
  Đây là bộ lọc phù hợp để làm mờ ảnh.

### 3. ****Output Image (Ảnh sau khi làm mờ)****

* **Nhận xét**:  
  Ảnh đầu ra (phải trên cùng) hiển thị ảnh gốc sau khi được làm mờ bởi Gaussian kernel.
  + Các chi tiết nhỏ trong ảnh, như các sợi tóc và kết cấu nhỏ trên bề mặt, đã bị làm mờ đi, trong khi các đặc điểm lớn (như khuôn mặt và nón) vẫn giữ được hình dạng chính.
  + Hiệu ứng làm mờ này là kết quả mong muốn khi sử dụng Gaussian filter.

### 4. ****Original Image Spectrum (Phổ tần số của ảnh gốc)****

* **Nhận xét**:
  + Phổ tần số (trái dưới cùng) của ảnh gốc hiển thị rõ các thành phần tần số thấp tập trung ở trung tâm, trong khi tần số cao phân bố xung quanh viền.
  + Điều này là bình thường vì ảnh tự nhiên chủ yếu chứa thông tin ở tần số thấp (các chi tiết lớn).
  + Dải sáng ở trung tâm cho thấy ảnh có nhiều thông tin quan trọng nằm ở tần số thấp.

### 5. ****Gaussian Kernel Spectrum (Phổ tần số của Gaussian Kernel)****

* **Nhận xét**:
  + Phổ tần số của Gaussian kernel (giữa dưới) hiển thị dưới dạng một vùng sáng tập trung mạnh ở trung tâm và giảm dần ra ngoài biên, tương ứng với đặc điểm làm mờ của Gaussian filter.
  + Kernel chủ yếu ảnh hưởng đến tần số thấp và gần như không tác động nhiều đến tần số cao, điều này phù hợp với đặc tính làm mờ ảnh của Gaussian.

### 6. ****Output Image Spectrum (Phổ tần số của ảnh sau khi làm mờ)****

* **Nhận xét**:
  + Phổ tần số của ảnh sau khi làm mờ (phải dưới) cho thấy rõ ràng rằng các thành phần tần số cao đã bị triệt tiêu (mờ đi), chỉ còn lại các thành phần tần số thấp nổi bật.
  + Điều này giải thích tại sao ảnh sau khi làm mờ bị mất chi tiết nhưng các đặc điểm lớn vẫn giữ được.

THUẬT TOÁN 2 : BỘ LỌC BĂNG THÔNG (BPF) VỚI Dog

1,Giới thiệu về thuật toán :

**Bộ lọc băng thông (Band-Pass Filter - BPF)** là một công cụ trong xử lý tín hiệu, dùng để cho phép các tần số trong một dải cụ thể và loại bỏ các tần số thấp hơn hoặc cao hơn dải này. Một cách tiếp cận phổ biến để xây dựng bộ lọc băng thông là sử dụng phương pháp **Difference of Gaussians (DoG)**.

Thuật toán **Difference of Gaussians (DoG)** là sự kết hợp của hai bộ lọc Gaussian với độ rộng khác nhau. Sự khác biệt giữa hai bộ lọc Gaussian có thể được sử dụng để mô phỏng một bộ lọc băng thông, cho phép các tần số trung bình (tức là trong dải tần số giữa) và loại bỏ các tần số thấp và cao hơn.

2,Các bước thực hiện thuật toán :

 B1: **Tải và hiển thị ảnh ban đầu**:

* Đầu tiên, ảnh được tải vào dưới dạng float với img\_as\_float() từ thư viện skimage. Sau đó, ảnh này được hiển thị sử dụng pylab.imshow().

 B2 : **Xây dựng kernel Gaussian**:

* Tạo ra một dải x với các giá trị từ -10 đến 10, sau đó tạo ra một bộ lọc Gaussian (kernel 1D) bằng cách sử dụng công thức np.exp(-a \* x\*\*2), với a là một hệ số điều chỉnh.
* Sau khi tạo kernel 1D, ta chuẩn hóa kernel này sao cho tổng của nó bằng 1, bằng cách chia nó cho giá trị tổng tích phân của nó.

 B3: **Tạo kernel Gaussian 2D**:

* Sử dụng phép nhân chéo giữa kernel 1D với chính nó để tạo kernel Gaussian 2D. Đây là bước chuẩn bị cho bộ lọc DoG.

 B4: **Tạo bộ lọc DoG**:

* Tạo hai kernel Gaussian 2D với các độ rộng khác nhau. Sau đó, tính sự chênh lệch giữa hai kernel này để tạo ra bộ lọc DoG.
* Bộ lọc DoG này sẽ hoạt động như một bộ lọc băng thông, loại bỏ các tần số thấp và cao hơn, chỉ giữ lại tần số nằm trong dải.

 B5: **Áp dụng bộ lọc DoG vào ảnh**:

* Sử dụng fftconvolve() từ thư viện scipy.signal để áp dụng bộ lọc DoG vào ảnh bằng cách thực hiện phép chập (convolution) trong miền tần số.
* Sau khi xử lý, ảnh đầu ra được chuẩn hóa và hiển thị.

3, Giải thích đoạn mã nguồn :

import numpy as np  
from skimage import img\_as\_float  
from scipy import signal  
import matplotlib.pyplot as plt # Thay vì pylab  
  
# Đọc và hiển thị ảnh  
im = img\_as\_float(plt.imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\tigers.jpeg"))  
plt.figure(), plt.imshow(im), plt.axis('off'), plt.show()  
  
# Tạo kernel Gaussian 1D  
x = np.linspace(-10, 10, 15)  
kernel\_1d = np.exp(-0.005 \* x \*\* 2)  
kernel\_1d /= np.trapz(kernel\_1d) # chuẩn hóa tổng = 1  
gauss\_kernel1 = kernel\_1d[:, np.newaxis] \* kernel\_1d[np.newaxis, :]  
  
# Tạo kernel Gaussian 2  
kernel\_1d = np.exp(-5 \* x \*\* 2)  
kernel\_1d /= np.trapz(kernel\_1d) # chuẩn hóa tổng = 1  
gauss\_kernel2 = kernel\_1d[:, np.newaxis] \* kernel\_1d[np.newaxis, :]  
  
# Tạo kernel DoG  
DoGKernel = gauss\_kernel1[:, :, np.newaxis] - gauss\_kernel2[:, :, np.newaxis]  
  
# Áp dụng DoG kernel vào ảnh  
im = signal.fftconvolve(im, DoGKernel, mode='same')  
  
# Hiển thị ảnh sau khi lọc  
plt.figure(), plt.imshow(np.clip(im, 0, 1)), print(np.max(im)), plt.show()

**Import thư viện**

* **numpy**: Thư viện toán học của Python, giúp thực hiện các phép toán số học, mảng và ma trận.
* **skimage.img\_as\_float**: Hàm này chuyển đổi ảnh đầu vào thành dạng float trong phạm vi [0, 1]. Điều này có ích để đảm bảo rằng ảnh được xử lý dưới dạng số thực (floating-point).
* **scipy.signal**: Thư viện này cung cấp các công cụ để xử lý tín hiệu, bao gồm phép toán **convolution** (chập) mà chúng ta sẽ sử dụng.
* **matplotlib.pyplot**: Dùng để hiển thị ảnh hoặc đồ thị. Thay vì pylab, đây là cách thường được sử dụng trong thực tế để vẽ ảnh.

### ****Đọc và hiển thị ảnh ban đầu****

* **plt.imread()**: Đọc ảnh từ đường dẫn tệp. imread trả về ảnh dưới dạng mảng NumPy.
* **img\_as\_float()**: Chuyển đổi giá trị pixel của ảnh thành giá trị float trong phạm vi [0, 1] (nếu ảnh gốc có giá trị pixel trong phạm vi [0, 255] thì được chuẩn hóa lại).
* **plt.imshow()**: Hiển thị ảnh lên đồ thị.
* **plt.axis('off')**: Tắt các trục để chỉ hiển thị ảnh mà không có thang đo.
* **plt.show()**: Hiển thị cửa sổ đồ họa.

### ****Tạo kernel Gaussian 1D (Để tạo bộ lọc Gaussian đầu tiên)****

* **np.linspace(-10, 10, 15)**: Tạo một mảng gồm 15 giá trị từ -10 đến 10. Đây là giá trị của các điểm x trên trục tọa độ.
* **np.exp(-0.005 \* x \*\* 2)**: Tạo một hàm Gaussian 1D. Hàm này có dạng exp⁡(−αx2)\exp(-\alpha x^2)exp(−αx2) trong đó α=0.005\alpha = 0.005α=0.005, và các giá trị Gaussian này sẽ giảm dần khi x tăng về phía ngoài.
* **np.trapz(kernel\_1d)**: Tính tích phân (sử dụng phương pháp Simpson) của hàm Gaussian để chuẩn hóa hàm sao cho tổng của các giá trị trong kernel là 1.
* **kernel\_1d[:, np.newaxis] \* kernel\_1d[np.newaxis, :]**: Chuyển từ kernel 1D sang kernel 2D bằng cách lấy tích của vector kernel 1D theo hai chiều. Điều này tạo ra một ma trận Gaussian 2D.

### ****Tạo kernel Gaussian 2 (Để tạo bộ lọc Gaussian thứ hai với độ rộng khác)****

* Đây là bước tạo một kernel Gaussian thứ hai với độ rộng (spread) khác. Hệ số α=5\alpha = 5α=5 sẽ làm cho bộ lọc này "hẹp" hơn so với bộ lọc đầu tiên. Việc tạo kernel này cũng tương tự như bộ lọc đầu tiên, nhưng với độ sắc nét cao hơn.

### ****Tạo kernel DoG (Difference of Gaussians)****

* **Difference of Gaussians (DoG)**: Đây là sự khác biệt giữa hai bộ lọc Gaussian. DoG có thể được coi là một phương pháp lọc tần số dải thông (band-pass filter), nghĩa là nó sẽ cho phép các tần số trong một dải cụ thể đi qua và loại bỏ các tần số thấp và cao ngoài dải này.
* Ở đây, **gauss\_kernel1** là Gaussian với độ rộng lớn (low-pass filter), và **gauss\_kernel2** là Gaussian với độ rộng nhỏ hơn (high-pass filter). Khi trừ chúng đi, chúng ta nhận được một bộ lọc dải thông, có khả năng lọc và làm nổi bật các chi tiết trong một dải tần số nhất định.

**Áp dụng DoG kernel vào ảnh**

* **signal.fftconvolve(im, DoGKernel, mode='same')**: Thực hiện phép toán chập (convolution) giữa ảnh và kernel DoG. Đây là một bước quan trọng trong việc áp dụng bộ lọc vào ảnh.
  + **fftconvolve** là một phương pháp tối ưu để thực hiện phép chập, sử dụng biến đổi Fourier nhanh (FFT). Điều này giúp việc chập trở nên hiệu quả hơn, đặc biệt đối với ảnh lớn.
  + **mode='same'** đảm bảo rằng kết quả trả về có cùng kích thước với ảnh đầu vào.

**Hiển thị ảnh sau khi áp dụng bộ lọc**

* **np.clip(im, 0, 1)**: Đảm bảo rằng giá trị pixel của ảnh sau khi chập không vượt quá phạm vi [0, 1], vì ảnh được chuẩn hóa ở bước đầu.
* **print(np.max(im))**: In ra giá trị lớn nhất của ảnh sau khi chập, để kiểm tra xem có sự thay đổi gì đáng kể trong giá trị pixel của ảnh.
* **plt.imshow()**: Hiển thị ảnh đã qua xử lý.
* **plt.show()**: Hiển thị cửa sổ đồ họa.

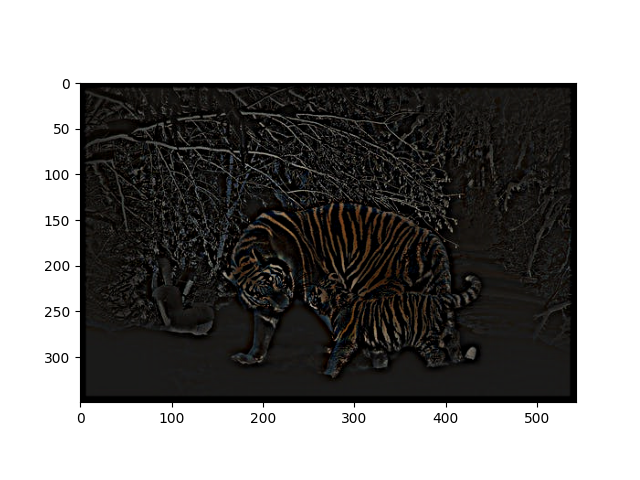
4,Kết quả sau khi thực hiện thuật toán :

Ví dụ 1 :

**Original Image**

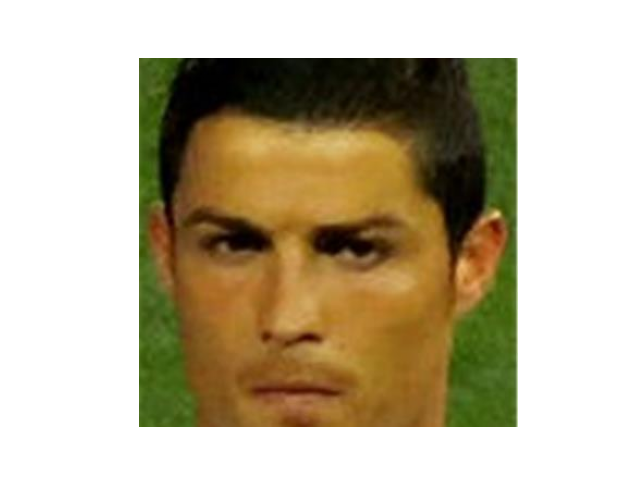


**Output Image**

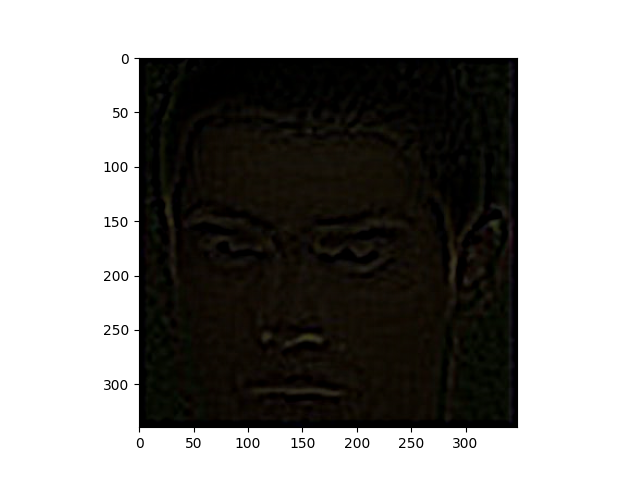
****

Ví dụ 2 :

**Original Image**



**Output Image**

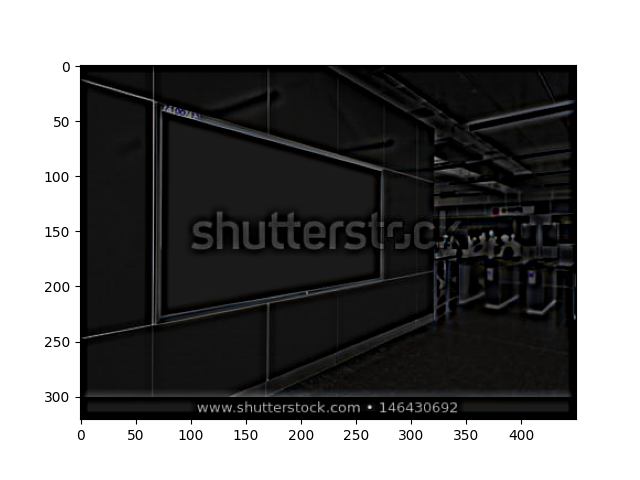


Ví dụ 3 :

**Original Image**

****

**Output Image**



4, Nhận xét ảnh sau khi thực hiện :

Ảnh sau khi thực hiện **bộ lọc DoG** sẽ có các đặc điểm sau:

* **Mờ hơn** ở các chi tiết nhỏ và tần số thấp.
* **Cải thiện biên và các chi tiết trung bình**, giúp làm nổi bật các chuyển tiếp rõ ràng trong ảnh.
* **Tăng độ sắc nét của các chi tiết biên** nhưng giảm độ sắc nét ở các phần khác của ảnh.

Điều này rất hữu ích trong các ứng dụng như **phát hiện biên**, **lọc tần số** và các phương pháp **phân tích ảnh** yêu cầu làm nổi bật các chi tiết ở mức độ vừa phải mà không làm mất quá nhiều thông tin.

THUẬT TOÁN 3 : THỰC HIỆN PHÉP GIẢI TÍCH CHẬP VỚI BỘ LỌC WIENER

1, Giới thiệu về thuật toán :

Thuật toán deconvolution với bộ lọc Wiener được sử dụng để phục hồi ảnh bị mờ và nhiễu. Bộ lọc Wiener là một bộ lọc tối ưu được thiết kế để giảm nhiễu trong khi cố gắng khôi phục thông tin gốc của ảnh càng nhiều càng tốt. Bộ lọc này hoạt động trên nguyên lý giảm nhiễu mà không làm mất đi quá nhiều thông tin của ảnh gốc, đặc biệt trong trường hợp nhiễu gauss.

2, Các bước thực hiện thuật toán :

**B1 : Đọc ảnh và chuyển đổi sang ảnh xám**: Đầu tiên, ảnh gốc được đọc vào và chuyển đổi thành ảnh xám để đơn giản hóa xử lý. Hàm color.rgb2gray trong scikit-image được sử dụng để chuyển đổi ảnh màu (RGB) thành ảnh xám.

**B2 : Làm mờ ảnh với kernel (PSF - Point Spread Function)**: Một ảnh bị mờ được tạo ra bằng cách áp dụng một kernel mờ (PSF) lên ảnh gốc. Kernel này là một ma trận 2D với các giá trị bình thường hóa. PSF có thể được mô phỏng bằng một bộ lọc trung bình, ví dụ: một kernel có kích thước 7×77 \times 77×7 với các phần tử bằng nhau. Sau đó, ảnh được làm mờ bằng cách thực hiện phép nhân chập (convolution) với PSF này.

**B3 : Thêm nhiễu vào ảnh mờ**: Sau khi làm mờ, một lượng nhiễu gauss được thêm vào ảnh để mô phỏng việc nhiễu trong ảnh thực tế. Nhiễu được tạo ra với một độ lệch chuẩn nhất định, và ảnh nhiễu này sẽ được phục hồi sau đó.

**B4: Áp dụng bộ lọc Wiener không giám sát**: Bộ lọc Wiener được áp dụng để phục hồi ảnh từ ảnh mờ và nhiễu. restoration.unsupervised\_wiener là một hàm trong scikit-image giúp tự động điều chỉnh tham số của bộ lọc Wiener để tối ưu hóa kết quả phục hồi. Bộ lọc này sử dụng mô hình thống kê của nhiễu và mờ để ước lượng lại ảnh gốc.

**B5: Hiển thị kết quả**: Cuối cùng, ba ảnh được hiển thị:

* **Ảnh gốc** (không mờ và không nhiễu).
* **Ảnh bị mờ và nhiễu**.
* **Ảnh phục hồi bằng bộ lọc Wiener**.

3,Giải thích đoạn mã nguồn :

import numpy as np  
from skimage import img\_as\_float, color, data, restoration  
from skimage.io import imread # Import imread từ skimage  
  
from scipy.signal import convolve2d as conv2  
import matplotlib.pyplot as pylab  
  
# Đọc ảnh và chuyển đổi thành ảnh xám  
im = color.rgb2gray(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\cycling.png"))  
  
# Tạo kernel mờ (PSF)  
n = 7  
psf = np.ones((n, n)) / n\*\*2  
im1 = conv2(im, psf, 'same')  
  
# Thêm nhiễu vào ảnh mờ  
im1 += 0.1 \* np.random.standard\_normal(im.shape) # astro.std() không được định nghĩa  
  
# Áp dụng bộ lọc Wiener không giám sát  
im2, \_ = restoration.unsupervised\_wiener(im1, psf)  
  
# Hiển thị kết quả  
fig, axes = pylab.subplots(nrows=1, ncols=3, figsize=(20, 4), sharex=True, sharey=True)  
pylab.gray()  
axes[0].imshow(im), axes[0].axis('off'), axes[0].set\_title('Original image', size=20)  
axes[1].imshow(im1), axes[1].axis('off'), axes[1].set\_title('Noisy blurred image', size=20)  
axes[2].imshow(im2), axes[2].axis('off'), axes[2].set\_title('Self tuned restoration', size=20)  
fig.tight\_layout()  
pylab.show()

**Đọc và chuyển đổi ảnh màu sang ảnh xám:**

* Đoạn mã này sử dụng imread() từ thư viện skimage.io để đọc ảnh từ file và sau đó chuyển ảnh màu RGB sang ảnh xám (rgb2gray) bằng hàm từ thư viện skimage.color.
* im bây giờ là một ma trận 2D đại diện cho ảnh xám.

**Tạo PSF (Point Spread Function) cho mờ ảnh:**

* psf là **kernel mờ** (Point Spread Function), ở đây được tạo ra bằng cách sử dụng một ma trận 7x7 với các phần tử có giá trị bằng nhau (được chia cho n2n^2n2 để chuẩn hóa).
* Kernel này mô phỏng một hiện tượng mờ ảnh đơn giản, có tác dụng làm mờ ảnh khi áp dụng phép tích chập (convolve2d).

**Áp dụng kernel PSF để làm mờ ảnh:**

* Sử dụng hàm convolve2d từ scipy.signal để thực hiện phép tích chập 2D giữa ảnh gốc im và kernel psf. Kết quả là một ảnh mờ, được lưu vào biến im1.
* Cờ 'same' đảm bảo rằng kích thước ảnh đầu ra im1 sẽ giữ nguyên kích thước của ảnh đầu vào.

**Thêm nhiễu vào ảnh mờ:**

* Đoạn mã này tạo ra một nhiễu ngẫu nhiên với phân phối chuẩn và độ lệch chuẩn là 0.1, sau đó cộng vào ảnh mờ im1.
* Nhiễu này mô phỏng tình trạng ảnh bị nhiễu (ví dụ như nhiễu Gaussian), điều này giúp kiểm tra khả năng khôi phục của bộ lọc Wiener.

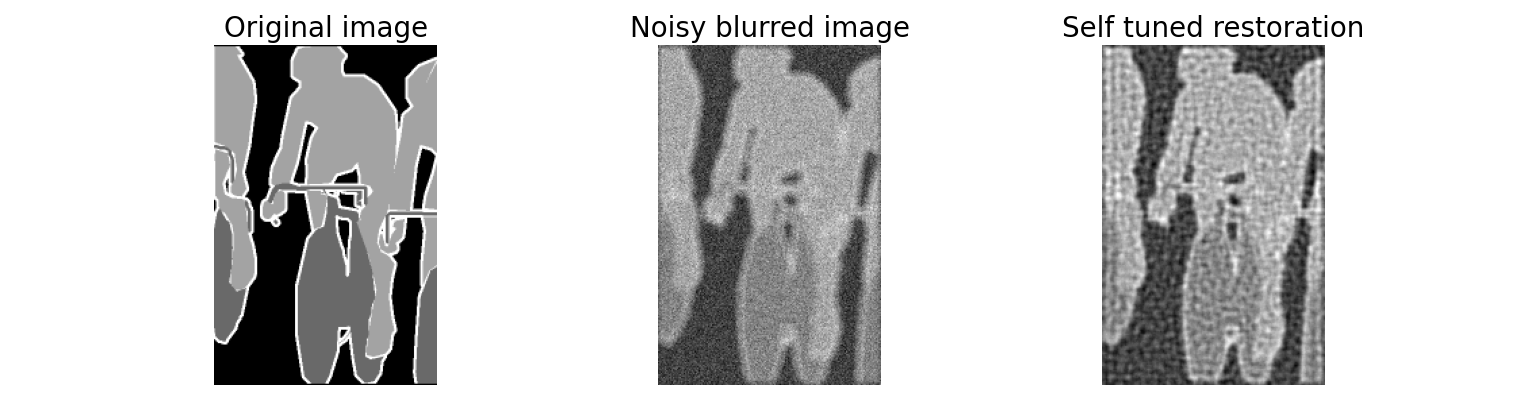
**Sử dụng bộ lọc Wiener không giám sát để phục hồi ảnh:**

* restoration.unsupervised\_wiener() áp dụng bộ lọc Wiener không giám sát để phục hồi ảnh từ ảnh mờ và nhiễu im1.
* Hàm này tự động điều chỉnh các tham số của bộ lọc Wiener để phục hồi ảnh, sử dụng kernel psf là thông tin mô phỏng mờ ảnh (Point Spread Function).
* Kết quả là ảnh được phục hồi im2.

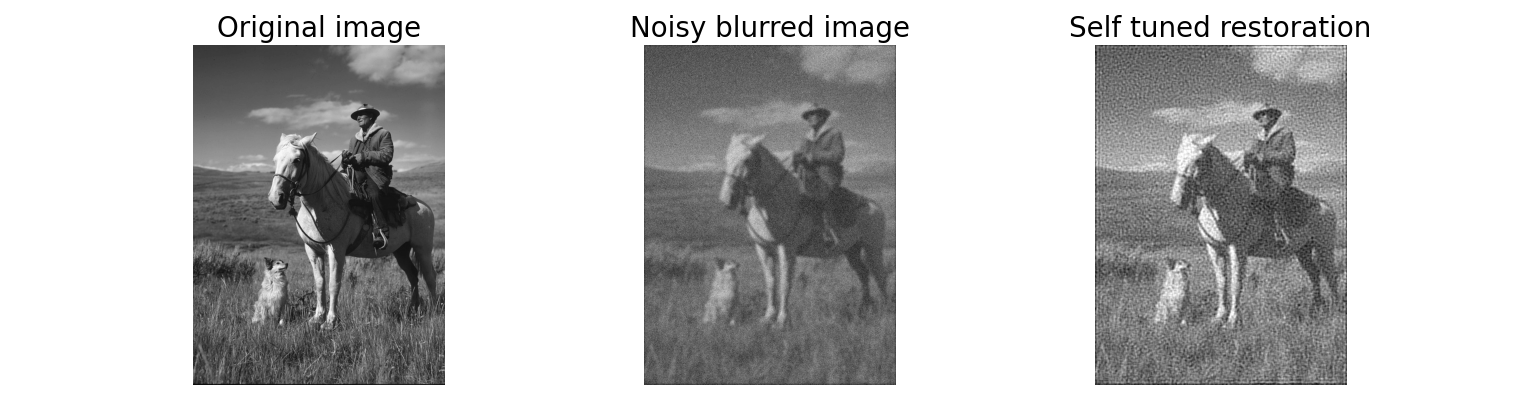
**Hiển thị các kết quả:**

* Đoạn mã này hiển thị ba ảnh trên một figure:
  + **Ảnh gốc**: Ảnh ban đầu (im).
  + **Ảnh mờ và nhiễu**: Ảnh sau khi đã bị mờ và thêm nhiễu (im1).
  + **Ảnh phục hồi**: Ảnh đã được phục hồi nhờ bộ lọc Wiener không giám sát (im2).
* imshow() từ pylab được sử dụng để hiển thị từng ảnh, và các tiêu đề được đặt cho từng ảnh để dễ phân biệt.

4, Kết quả sau khi thực hiện thuật toán :

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 : 

Nhận xét ảnh sau khi thực hiện thuật toán :

 **Ảnh gốc (Original image):**

* Đây là ảnh gốc trước khi bị mờ và nhiễu.
* Ảnh này sẽ sắc nét, rõ ràng với các chi tiết tốt, không bị ảnh hưởng bởi bất kỳ quá trình mờ hay nhiễu nào.

 **Ảnh mờ và nhiễu (Noisy blurred image):**

* Ảnh này là kết quả của việc áp dụng kernel mờ (PSF) lên ảnh gốc, sau đó thêm nhiễu Gaussian.
* Do ảnh đã bị làm mờ và nhiễu, nên các chi tiết trở nên mờ dần và có thêm các đốm nhiễu ngẫu nhiên. Bạn có thể thấy các vùng của ảnh bị nhòe, và có thể xuất hiện các đốm sáng/tối ngẫu nhiên gây mất chi tiết.

 **Ảnh phục hồi (Self-tuned restoration):**

* Đây là ảnh sau khi áp dụng bộ lọc Wiener không giám sát để phục hồi ảnh.
* Bộ lọc Wiener giúp loại bỏ phần nào nhiễu và làm sắc nét ảnh bị mờ. Sau khi phục hồi, ảnh sẽ có chất lượng tốt hơn, chi tiết rõ hơn so với ảnh mờ và nhiễu, mặc dù có thể vẫn còn một số hiện tượng mờ nhẹ hoặc nhiễu không hoàn toàn được loại bỏ (tùy vào mức độ nhiễu và mờ của ảnh ban đầu).
* Tuy nhiên, bộ lọc Wiener không thể khôi phục hoàn toàn ảnh gốc nếu như ảnh bị nhiễu và mờ quá mức.

### Ý nghĩa của bộ lọc Wiener:

* Bộ lọc Wiener là một phương pháp phục hồi ảnh trong điều kiện ảnh bị mờ và nhiễu. Nó hoạt động dựa trên ước lượng thống kê để tối ưu hóa sự phục hồi của ảnh, tìm kiếm một ảnh khôi phục sao cho giảm thiểu lỗi giữa ảnh gốc và ảnh khôi phục, đặc biệt là trong các tình huống có nhiễu.