Họ tên : Nguyễn Quang Anh

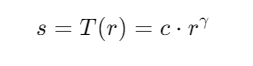
Mã sinh viên : B21DCDT036

CHƯƠNG 4 : CẢI THIỆN HÌNH ẢNH ( IMAGE ENHANCEMENT )

THUẬT TOÁN 1 : BIẾN ĐỔI GAMMA (Power-law transform)

1, Giới thiệu về thuật toán :

Power-law transform (hay còn gọi là phép biến đổi gamma) là một phép biến đổi ảnh rất phổ biến trong xử lý ảnh. Phép biến đổi này giúp điều chỉnh độ sáng của ảnh và có thể làm tối hoặc sáng các vùng của ảnh mà không thay đổi quá nhiều các vùng khác. Cụ thể, trong phép biến đổi gamma:



* r là giá trị pixel ban đầu của ảnh (trong khoảng [0, 1] đối với ảnh xám hoặc RGB).
* s là giá trị pixel sau khi áp dụng biến đổi.
* γ là một tham số điều khiển độ sáng của ảnh.
* c là một hằng số điều chỉnh để ảnh không bị vượt quá giá trị tối đa (1.0 đối với ảnh chuẩn).
* Nếu γ>1 , ảnh sẽ trở nên tối hơn, làm tăng độ tương phản ở các vùng sáng.
* Nếu γ<1 ảnh sẽ sáng hơn, làm tăng độ sáng của các vùng tối.
* Khi γ=1, không có thay đổi gì về độ sáng của ảnh.

2,Các bước thực hiện thuật toán

 B1: **Đọc ảnh RGB và chuyển đổi thành float**:

* Đầu tiên, ảnh được đọc và chuyển đổi thành ảnh kiểu float (nằm trong khoảng [0, 1]) để có thể thực hiện các phép toán chính xác.

 B2: **Áp dụng phép biến đổi gamma (Power-law transform)**:

* Phép biến đổi gamma được áp dụng với công thức im1=imγim1 = im^{\gamma}im1=imγ, trong đó γ\gammaγ được đặt là 5 trong đoạn mã. Điều này sẽ làm ảnh trở nên tối hơn, đặc biệt là ở các vùng sáng của ảnh.

 B3: **Vẽ ảnh gốc và ảnh đã biến đổi**:

* Sử dụng imshow để hiển thị ảnh gốc và ảnh sau khi đã áp dụng phép biến đổi gamma.

 B4: **Vẽ histogram cho ảnh gốc và ảnh đã biến đổi**:

* Các histogram của các kênh màu (đỏ, xanh lá và xanh dương) được vẽ cho cả ảnh gốc và ảnh sau khi biến đổi. Điều này giúp quan sát được sự thay đổi trong phân bố tần suất của các giá trị màu sau khi áp dụng phép biến đổi gamma.

3,Giải thích đoạn mã nguồn :

import numpy as np  
from skimage import img\_as\_float  
import matplotlib.pyplot as pylab  
from skimage.io import imread  
  
# Định nghĩa hàm plot\_hist để vẽ histogram cho các kênh RGB  
def plot\_hist(r, g, b, title):  
 pylab.hist(r.ravel(), bins=256, color='red', alpha=0.5, label='Red')  
 pylab.hist(g.ravel(), bins=256, color='green', alpha=0.5, label='Green')  
 pylab.hist(b.ravel(), bins=256, color='blue', alpha=0.5, label='Blue')  
 pylab.legend(loc='upper right')  
 pylab.title(title)  
  
# Đọc ảnh và chuyển sang kiểu float  
im = img\_as\_float(imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\horses.jpg"))  
  
# Áp dụng phép biến đổi gamma (gamma correction)  
gamma = 5  
im1 = im \*\* gamma  
  
# Cấu hình hiển thị  
pylab.style.use('ggplot')  
pylab.figure(figsize=(18, 6))  
  
# Vẽ ảnh gốc và ảnh đã biến đổi gamma  
pylab.subplot(141)  
pylab.imshow(im)  
pylab.axis('off')  
pylab.title('Original Image')  
  
pylab.subplot(142)  
pylab.imshow(im1)  
pylab.axis('off')  
pylab.title('Gamma Corrected Image')  
  
# Vẽ histogram cho ảnh gốc (RGB)  
pylab.subplot(143)  
plot\_hist(im[..., 0], im[..., 1], im[..., 2], ' (Input)')  
pylab.subplots\_adjust(wspace=0.3, hspace=0.3) # Tăng khoảng cách giữa các subplot  
# Vẽ histogram cho ảnh đã biến đổi gamma (RGB)  
pylab.subplot(144)  
plot\_hist(im1[..., 0], im1[..., 1], im1[..., 2], ' (Output)')  
  
# Hiển thị kết quả  
pylab.show()

**Import thư viện**:

 **numpy**: Sử dụng để làm việc với các mảng dữ liệu.

 **img\_as\_float từ skimage**: Chuyển ảnh về định dạng float với giá trị các pixel trong khoảng [0, 1]. Đây là bước quan trọng để các phép toán trên ảnh (như gamma correction) được thực hiện chính xác.

 **pylab**: Là thư viện của matplotlib dùng để vẽ đồ thị và hiển thị ảnh. Trong đoạn mã này, pylab được sử dụng để vẽ các histogram và hiển thị ảnh.

 **imread từ skimage.io**: Dùng để đọc ảnh từ đường dẫn và trả về ảnh dưới dạng mảng numpy.

**Định nghĩa hàm plot\_hist:**

Hàm plot\_hist vẽ histogram cho ba kênh màu RGB của ảnh. Các bước cụ thể:

* **r.ravel()**: Chuyển ảnh từ ma trận 2D thành một mảng 1D để dễ dàng tính histogram.
* **pylab.hist()**: Hàm vẽ histogram cho các kênh màu đỏ (r), xanh lá (g) và xanh dương (b), sử dụng 256 bins để thể hiện phân phối tần suất của các giá trị pixel.
  + **alpha=0.5**: Điều chỉnh độ trong suốt của histogram để dễ dàng chồng các histogram lên nhau.
  + **label**: Thêm nhãn cho từng histogram màu.
* **pylab.legend(loc='upper right')**: Hiển thị legend cho các kênh màu ở góc trên bên phải.
* **pylab.title(title)**: Thêm tiêu đề cho biểu đồ histogram.

**Đọc ảnh và chuyển sang kiểu float:**

* Đoạn mã này đọc ảnh từ đường dẫn và chuyển ảnh từ định dạng nguyên gốc (RGB với giá trị pixel từ 0 đến 255) sang định dạng float (với giá trị pixel trong khoảng [0, 1]) bằng cách sử dụng img\_as\_float của thư viện skimage.

**Áp dụng phép biến đổi gamma (Gamma Correction):**

* **gamma = 5**: Tham số gamma được thiết lập là 5, có nghĩa là ảnh sẽ bị tối đi, đặc biệt là các vùng sáng. Nếu gamma nhỏ hơn 1, ảnh sẽ sáng hơn.
* **im1 = im \*\* gamma**: Áp dụng phép biến đổi gamma cho ảnh. Phép biến đổi này sẽ nâng các giá trị pixel của ảnh lên lũy thừa γ\gammaγ. Việc này làm tăng độ tương phản của ảnh, đặc biệt là ở các vùng sáng.

**Cấu hình hiển thị với pylab:**

* **pylab.style.use('ggplot')**: Đặt phong cách hiển thị đồ thị theo kiểu ggplot, một kiểu phong cách rất phổ biến trong việc trực quan hóa dữ liệu.
* **pylab.figure(figsize=(18, 6))**: Tạo một cửa sổ vẽ đồ thị với kích thước chiều rộng là 18 inch và chiều cao là 6 inch.

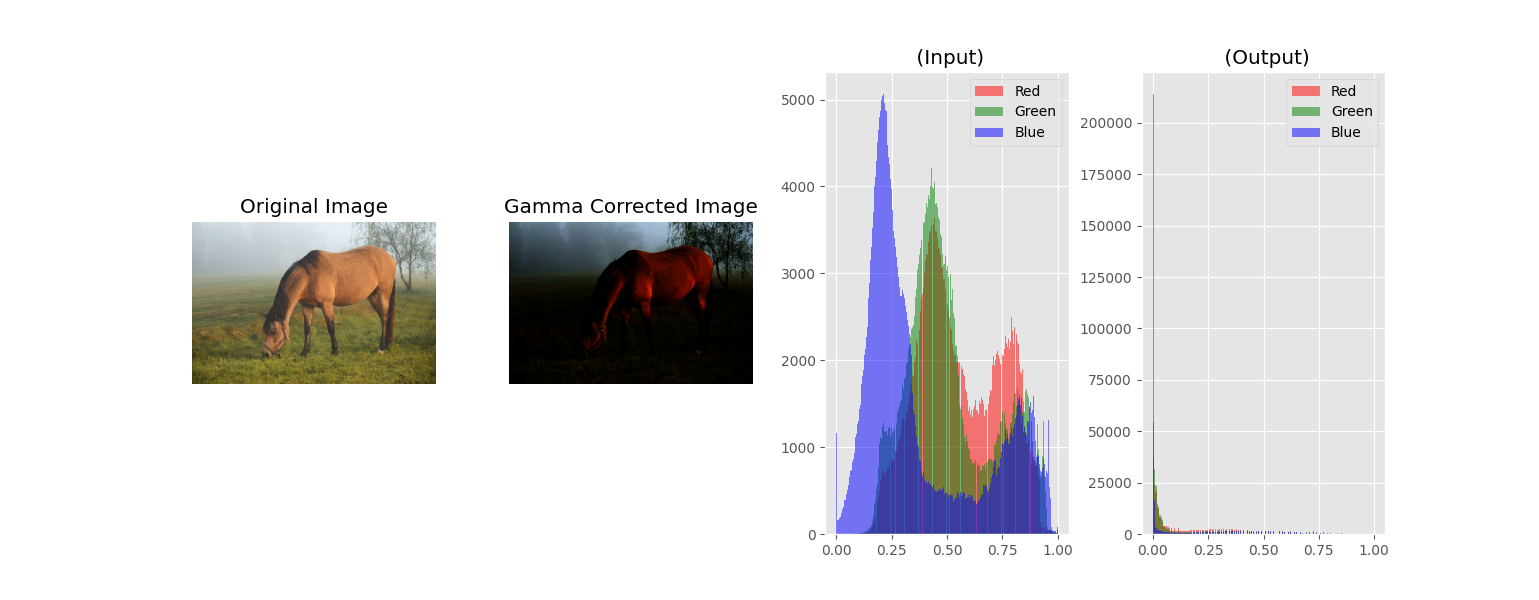
**Hiển thị ảnh gốc và ảnh đã biến đổi gamma:**

* **pylab.subplot(141)**: Tạo một vùng con để hiển thị ảnh gốc. 141 có nghĩa là 1 hàng, 4 cột, và đây là vùng con thứ 1.
* **pylab.imshow(im)**: Hiển thị ảnh gốc.
* **pylab.axis('off')**: Tắt các trục để chỉ hiển thị ảnh mà không có các chỉ số trục.
* **pylab.title('Original Image')**: Thêm tiêu đề "Original Image" cho ảnh.
* Các bước tương tự được thực hiện cho ảnh đã biến đổi gamma (với im1).

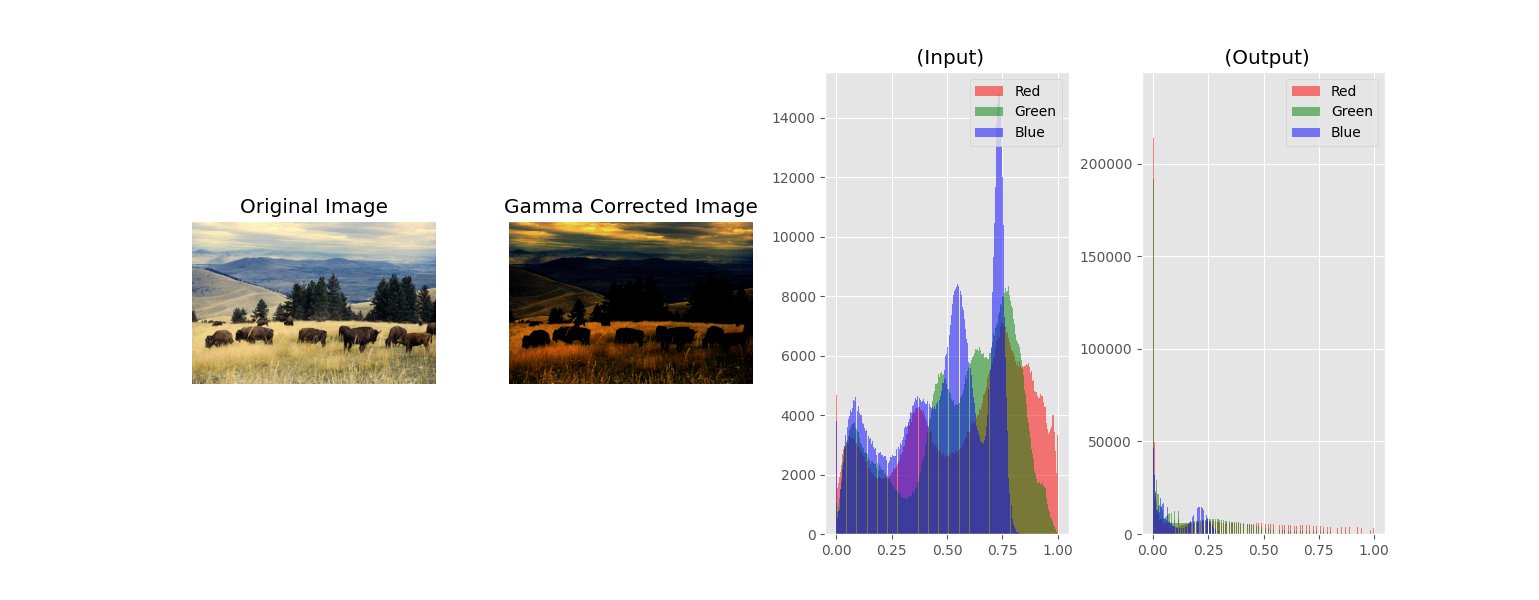
**Hiển thị histogram cho ảnh gốc và ảnh đã biến đổi gamma:**

* **pylab.subplot(143)**: Tạo vùng con thứ 3 để hiển thị histogram của ảnh gốc.
* **plot\_hist(im[..., 0], im[..., 1], im[..., 2], ' (Input)')**: Gọi hàm plot\_hist để vẽ histogram cho các kênh màu của ảnh gốc.
* **pylab.subplot(144)**: Tạo vùng con thứ 4 để hiển thị histogram của ảnh đã biến đổi gamma.
* **plot\_hist(im1[..., 0], im1[..., 1], im1[..., 2], ' (Output)')**: Gọi hàm plot\_hist để vẽ histogram cho các kênh màu của ảnh đã biến đổi gamma.

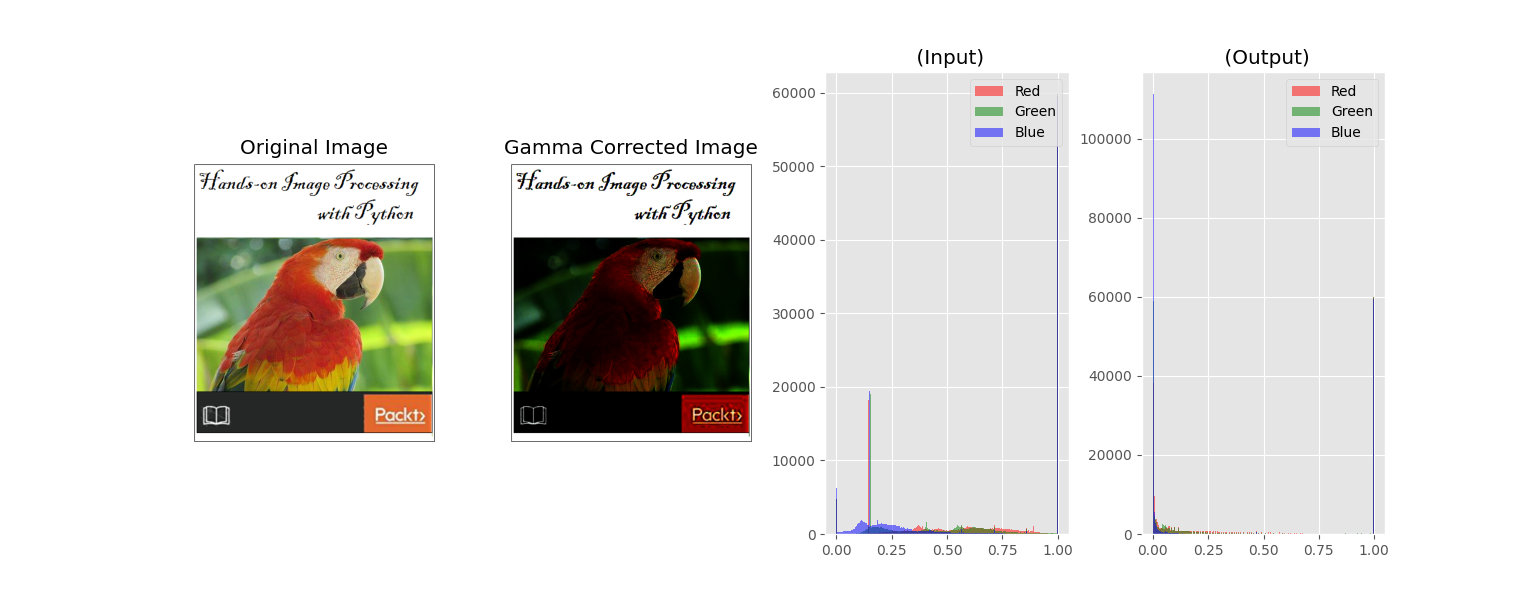
4, Kết quả sau khi thực hiện mô phỏng thuật toán :

Ví dụ 1 : 

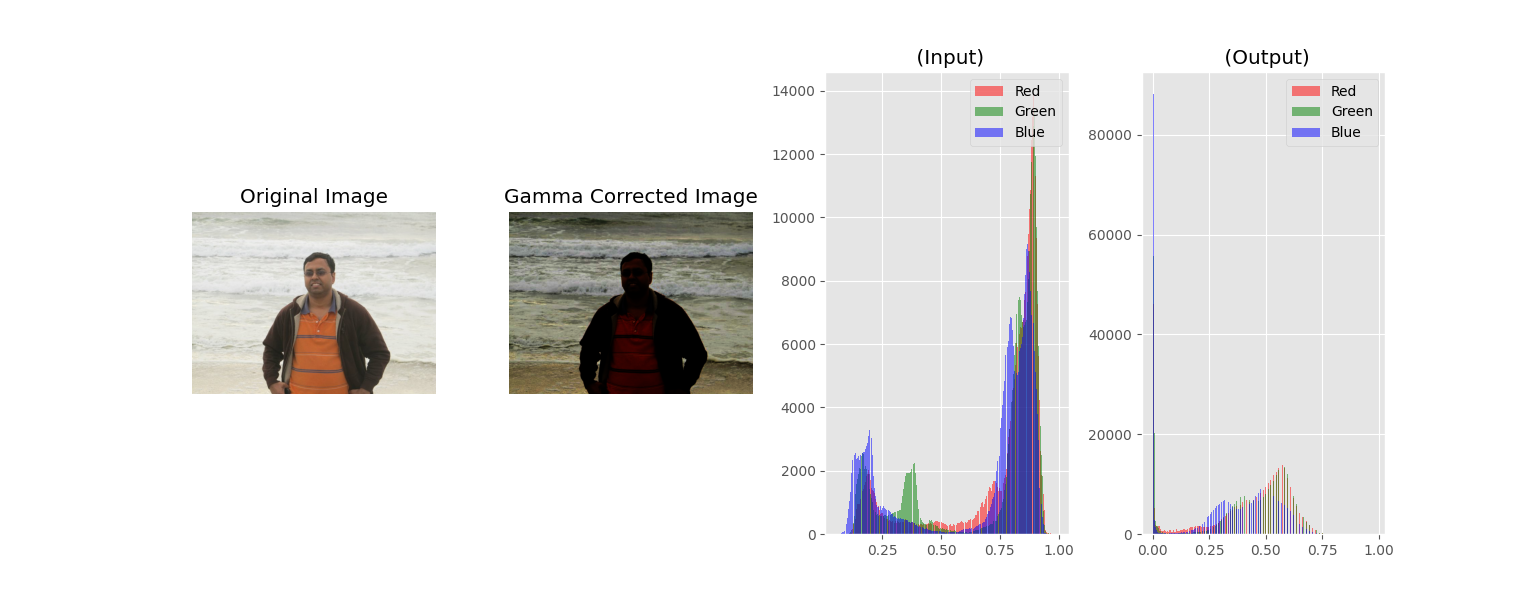
Ví dụ 2 :



Ví dụ 3 :



Ví dụ 4 :



Nhận xét về ảnh trước và sau khi biến đổi :

**Nhận xét ảnh trước và sau khi biến đổi gamma:**

1. **Ảnh gốc (Original Image):**
   * Đây là ảnh ban đầu với các giá trị pixel trong dải chuẩn (0-1 sau khi chuyển đổi bằng img\_as\_float).
   * Mức độ sáng và tương phản của các vùng trong ảnh cân đối, không có sự điều chỉnh rõ rệt về độ sáng hay độ tối.
2. **Ảnh sau biến đổi gamma (Gamma Corrected Image):**
   * Khi áp dụng phép biến đổi gamma với γ=5 , các giá trị pixel được nâng lên lũy thừa 5.
   * Điều này làm ảnh trở nên **rất tối** ở hầu hết các vùng, đặc biệt là những vùng sáng trong ảnh gốc bị giảm độ sáng mạnh.
   * Các vùng đã tối trong ảnh gốc gần như không còn nhìn thấy rõ ràng trong ảnh sau biến đổi.

**Nhận xét về 2 biểu đồ histogram:**

1. **Biểu đồ ảnh gốc (Input Histogram):**
   * Các kênh màu đỏ, xanh lá, và xanh dương có phân bố rộng, trải dài trên toàn bộ dải giá trị (0-1).
   * Điều này phản ánh rằng ảnh gốc có sự pha trộn đều giữa các màu, tạo nên ảnh cân bằng và tự nhiên.
2. **Biểu đồ ảnh sau biến đổi gamma (Output Histogram):**
   * Sau khi áp dụng gamma correction (γ=5), hầu hết các giá trị pixel của các kênh màu bị nén về gần 0.
   * Histogram cho thấy các kênh màu đều tập trung ở đầu dải giá trị (rất gần 0), thể hiện rằng ảnh đã bị tối đi đáng kể.
   * Sự mất cân bằng màu sắc sau biến đổi cũng rõ ràng hơn, do các kênh màu không còn phân phối đồng đều.

THUẬT TOÁN 2 : SỬ DỤNG BỘ LỌC TRUNG VỊ

**1, Giới thiệu về thuật toán :**

**Ý tưởng thuật toán:**

Bộ lọc trung vị là một kỹ thuật xử lý ảnh phi tuyến, sử dụng để giảm nhiễu trong ảnh. Nó đặc biệt hiệu quả đối với nhiễu salt-and-pepper, là loại nhiễu mà một số pixel trong ảnh bị biến đổi thành giá trị cực đại (255, màu trắng) hoặc cực tiểu (0, màu đen).

Bộ lọc trung vị là một bộ lọc phi tuyến. Thay vì lấy giá trị trung bình (như bộ lọc tuyến tính), bộ lọc trung vị chọn **giá trị trung vị** từ các giá trị trong vùng lân cận. Điều này giúp giảm thiểu ảnh hưởng của nhiễu "salt-and-pepper", do nhiễu này thường biểu hiện dưới dạng các giá trị cực đại hoặc cực tiểu.

**2 , Cách thức hoạt động :**

Bộ lọc trung vị hoạt động bằng cách:

1. **Chọn một cửa sổ (kernel)**:
   * Kernel là một vùng lân cận hình vuông, ví dụ: 3×33 \times 33×3, 5×55 \times 55×5, hoặc 7×77 \times 77×7.
   * Kích thước kernel được chọn tùy thuộc vào mức độ nhiễu trong ảnh. Kernel nhỏ giúp bảo toàn chi tiết tốt hơn nhưng không loại bỏ hoàn toàn nhiễu nặng. Kernel lớn loại bỏ nhiễu tốt hơn nhưng làm mất chi tiết.
2. **Xét từng vị trí trong ảnh**:
   * Kernel quét từng pixel trong ảnh, bắt đầu từ góc trên bên trái và di chuyển theo từng hàng cho đến góc dưới bên phải.
3. **Thu thập các giá trị pixel trong kernel**:
   * Tại mỗi vị trí, lấy giá trị pixel của tất cả các điểm trong vùng kernel.
4. **Tính giá trị trung vị**:
   * Sắp xếp các giá trị vừa thu thập được theo thứ tự tăng dần.
   * Lấy giá trị ở giữa danh sách (nếu danh sách có số lượng chẵn, lấy trung bình của hai giá trị giữa).
5. **Thay thế giá trị trung tâm của kernel**:
   * Pixel ở giữa vùng kernel được thay bằng giá trị trung vị vừa tính được.
6. **Lặp lại**:
   * Kernel di chuyển qua toàn bộ ảnh, lặp lại quá trình trên cho đến khi tất cả pixel trong ảnh được xử lý.

**3,Giải thích đoạn mã nguồn :**

import numpy as np  
from PIL import Image, ImageFilter  
import matplotlib.pyplot as pylab  
  
  
def plot\_image(im, title):  
 *"""  
 Helper function to plot an image with a title.  
 """* pylab.imshow(np.array(im))  
 pylab.title(title)  
  
  
# Main code  
i = 1  
pylab.figure(figsize=(25, 35))  
  
# Loop over noise proportions  
for prop\_noise in np.linspace(0.05, 0.3, 3):  
 im = Image.open(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\mandrill.jpg")  
 # Add salt-and-pepper noise to the image  
 n = int(im.width \* im.height \* prop\_noise)  
 x, y = np.random.randint(0, im.width, n), np.random.randint(0, im.height, n)  
 for (px, py) in zip(x, y):  
 im.putpixel((px, py), ((0, 0, 0) if np.random.rand() < 0.5 else (255, 255, 255))) # Add noise  
  
 im.save(f'C:/Users/Dell/Desktop/Sandipan\_Dey\_2018\_Sample\_Images/images/mandrill\_spnoise\_{prop\_noise:.2f}.jpg')  
  
 # Plot noisy image  
 pylab.subplot(6, 4, i)  
 plot\_image(im, f'Original Image with {int(100 \* prop\_noise)}% added noise')  
 i += 1  
  
 # Apply median filter with different sizes  
 for sz in [3, 7, 11]:  
 im1 = im.filter(ImageFilter.MedianFilter(size=sz))  
 pylab.subplot(6, 4, i)  
 plot\_image(im1, f'Output (Median Filter size={sz})')  
 i += 1  
pylab.subplots\_adjust(wspace=0.5, hspace=0.8) # Tăng khoảng cách giữa các subplot  
pylab.show()

**Import các thư viện**

* **numpy**: Thư viện hỗ trợ xử lý dữ liệu số, như tạo ngẫu nhiên tọa độ pixel bị nhiễu.
* **Pillow (PIL)**: Thư viện xử lý ảnh, cung cấp các công cụ mở, chỉnh sửa và áp dụng bộ lọc cho ảnh.
* **matplotlib.pyplot (pylab)**: Công cụ để hiển thị hình ảnh và tạo đồ thị.

**Hàm plot\_image**

* Hàm nhận đầu vào là một ảnh (im) và tiêu đề (title) để hiển thị bằng matplotlib.

**Chuẩn bị hiển thị các ảnh**

* **i**: Biến đếm, theo dõi vị trí ảnh trên đồ thị.
* **pylab.figure**: Thiết lập kích thước lớn để hiển thị nhiều ảnh trên một màn hình.

**Chuẩn bị hiển thị các ảnh**

* **i**: Biến đếm, theo dõi vị trí ảnh trên đồ thị.
* **pylab.figure**: Thiết lập kích thước lớn để hiển thị nhiều ảnh trên một màn hình.

**Lưu ảnh và hiển thị ảnh bị nhiễu**

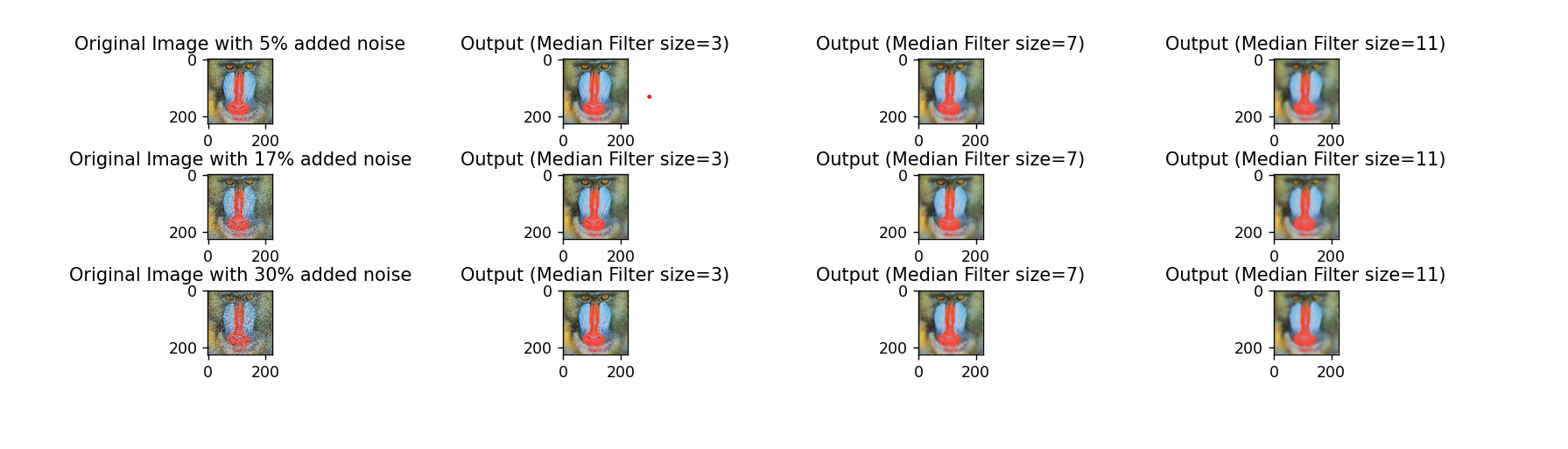
* **im.save**: Lưu ảnh bị nhiễu vào thư mục để sử dụng lại nếu cần.
* **pylab.subplot(6, 4, i)**: Tạo một ô trên lưới đồ thị để hiển thị ảnh bị nhiễu.
* **plot\_image**: Gọi hàm vẽ ảnh đã thêm nhiễu lên lưới đồ thị.

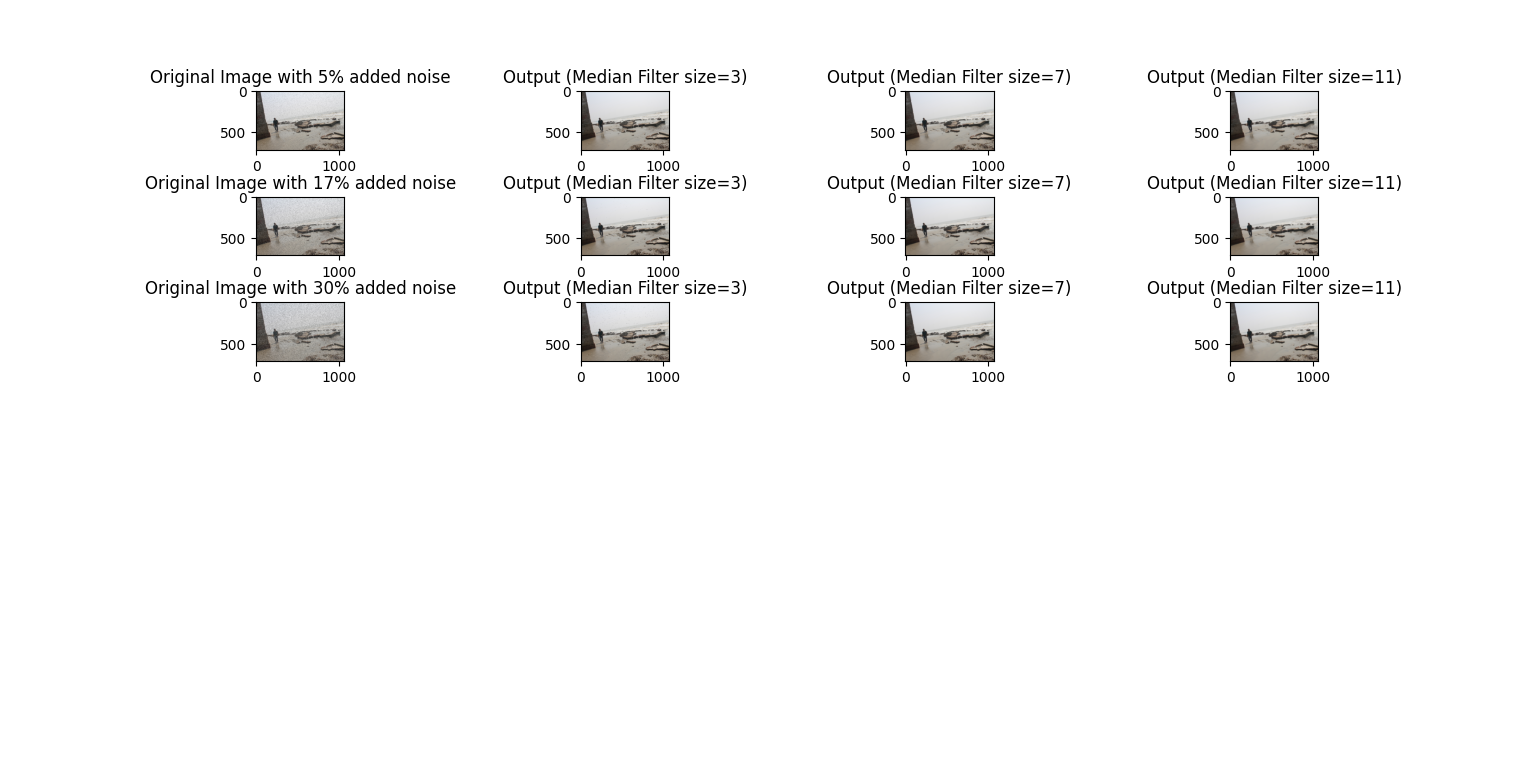
**Áp dụng lọc trung vị**

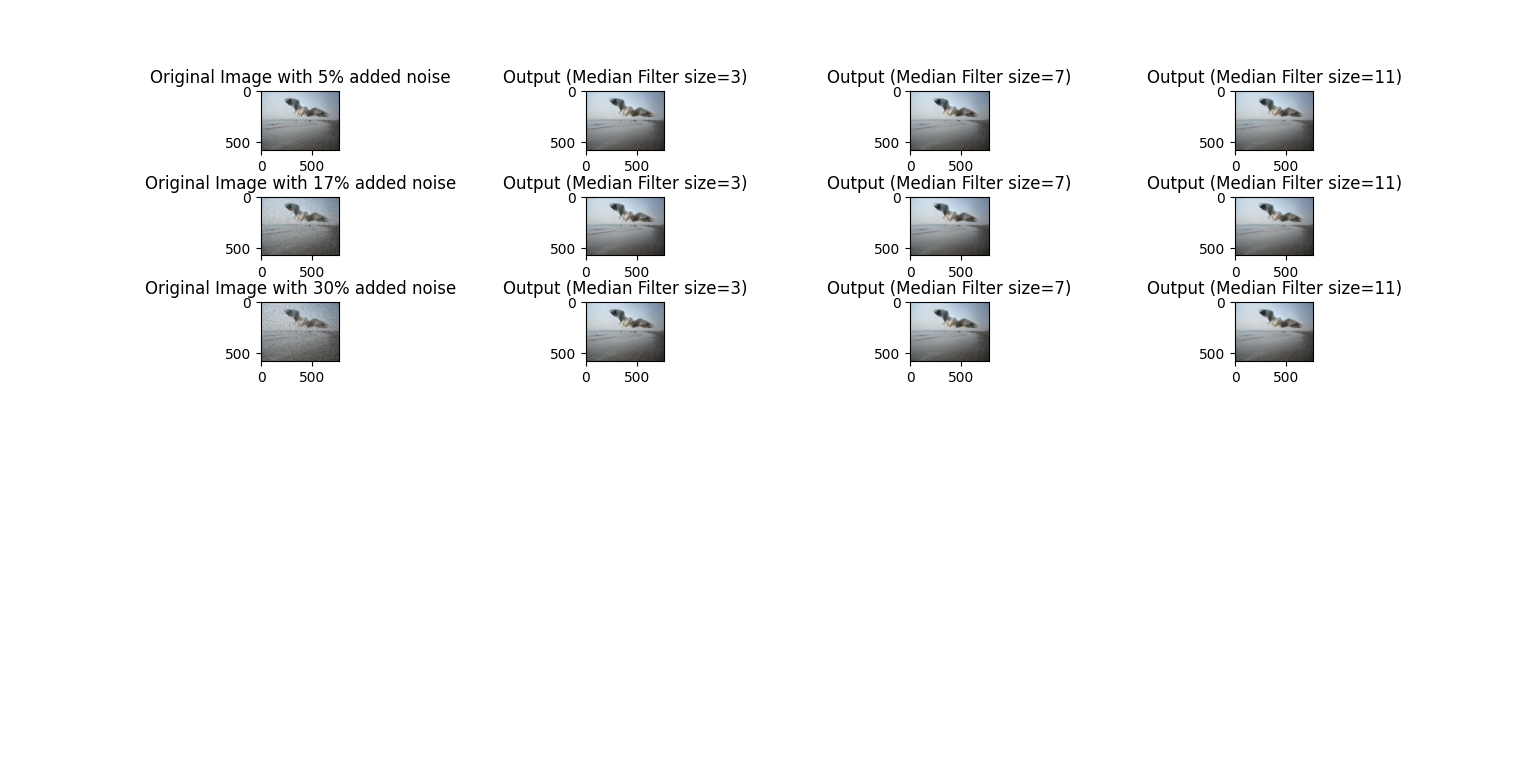
**Giải thích:**

* **ImageFilter.MedianFilter(size=sz)**: Lọc trung vị:
  + Lấy vùng lân cận sz×szsz \times szsz×sz của mỗi pixel.
  + Tính trung vị các giá trị trong vùng này và thay thế pixel trung tâm.
* **Kích thước kernel**:
  + **3x3**: Loại bỏ nhiễu nhẹ nhưng giữ lại chi tiết.
  + **7x7**: Loại bỏ nhiều nhiễu hơn nhưng làm mất một số chi tiết.
  + **11x11**: Loại bỏ nhiễu mạnh nhưng có thể làm mờ ảnh.

**4,Kết quả sau khi thực hiện thuật toán :**

Ví dụ 1 : 

Ví dụ 2 : 

Ví dụ 3 : 

Nhận xét ảnh :

**Nhận xét kết quả đầu ra :**

1. **Ảnh bị nhiễu**:
   * Với mức nhiễu thấp (5%): Ảnh vẫn rõ ràng, chỉ một số pixel bị nhiễu (đen/trắng).
   * Với mức nhiễu cao hơn (17.5%, 30%): Nhiễu tăng đáng kể, ảnh bị lốm đốm nhiều đốm đen/trắng.
2. **Ảnh sau lọc trung vị**:
   * **Kernel 3x3**:
     + Loại bỏ một số nhiễu nhẹ.
     + Chi tiết ảnh vẫn giữ được, không bị mờ đáng kể.
   * **Kernel 7x7**:
     + Loại bỏ nhiễu tốt hơn, ngay cả mức nhiễu cao (30%).
     + Một số chi tiết nhỏ (kết cấu, đường nét) bị làm mờ nhẹ.
   * **Kernel 11x11**:
     + Loại bỏ gần như toàn bộ nhiễu, nhưng ảnh bị làm mờ đáng kể, mất nhiều chi tiết.

THUẬT TOÁN 3 : LÀM MƯỢT ẢNH VỚI scipy ndimage

1,Giới thiệu về thuật toán sử dụng trong bài :

**Thuật toán percentile\_filter:**

percentile\_filter là một bộ lọc không gian dùng để thay đổi giá trị của mỗi pixel trong ảnh bằng một giá trị phần trăm (percentile) trong một cửa sổ lân cận (kernel). Đây là phiên bản tổng quát của bộ lọc trung vị. Cụ thể:

* **Bộ lọc Trung vị (Median Filter)** là trường hợp đặc biệt khi sử dụng phần trăm là 50.
* Bộ lọc này có thể được sử dụng để làm mượt ảnh và loại bỏ nhiễu, bao gồm nhiễu muối và hạt tiêu, mà không làm mờ các chi tiết quá nhiều.

Thuật toán hoạt động bằng cách tính toán phần trăm (percentile) của các giá trị pixel trong một cửa sổ nhỏ, sau đó thay thế giá trị pixel trung tâm bằng phần trăm đã tính được. Cửa sổ này di chuyển qua toàn bộ ảnh.

2,Các bước thực hiện thuật toán :

 B1 : **Chuẩn bị dữ liệu đầu vào:**

* Tải ảnh cần xử lý (trong trường hợp này là ảnh sea\_bird.jpg).
* Thêm nhiễu muối và hạt tiêu vào ảnh để mô phỏng tình huống xử lý nhiễu.

 B2 : **Thêm nhiễu vào ảnh:**

* Sử dụng hàm np.random.random() để tạo ra một ma trận ngẫu nhiên.
* Các giá trị lớn hơn 0.9 được gán là 255 (nhiễu muối), và các giá trị nhỏ hơn 0.1 được gán là 0 (nhiễu hạt tiêu).

 B3 : **Sử dụng percentile\_filter để làm mượt ảnh:**

* Hàm ndimage.percentile\_filter() sẽ được sử dụng để áp dụng bộ lọc phần trăm lên ảnh.
* Các tham số quan trọng bao gồm:
  + **percentile**: Xác định phần trăm mà bạn muốn (ví dụ: 50 cho bộ lọc trung vị, 90 cho một bộ lọc làm mượt mạnh hơn).
  + **size**: Kích thước của cửa sổ lân cận, quyết định mức độ mượt mà của bộ lọc (ví dụ: size=(5, 5) cho cửa sổ 5x5).

 B4 : **Hiển thị ảnh đã lọc:**

* Vẽ ảnh gốc và ảnh sau khi lọc bằng matplotlib.
* Tạo một loạt các bộ lọc với các phần trăm khác nhau (25%, 50%, 75%, 100%) và các kích thước cửa sổ khác nhau (5x5, 10x10, 15x15).
* So sánh các kết quả để tìm ra bộ lọc tối ưu cho việc loại bỏ nhiễu mà không làm mất quá nhiều chi tiết của ảnh.

3,Giải thích đoạn mã nguồn :

import numpy as np  
import matplotlib.pyplot as plt  
from scipy import ndimage  
import imageio  
  
# Load image using imageio instead of misc  
lena = imageio.imread(r"C:\Users\Dell\Desktop\Sandipan\_Dey\_2018\_Sample\_Images\images\sea\_bird.jpg")  
  
# Add salt-and-pepper noise to the input image  
noise = np.random.random(lena.shape)  
lena[noise > 0.9] = 255  
lena[noise < 0.1] = 0  
  
# Function to plot the image (assuming plot\_image is defined somewhere else)  
def plot\_image(img, title):  
 plt.imshow(img)  
 plt.title(title)  
 plt.axis('off') # Hide axes  
  
plot\_image(lena, 'Noisy image')  
plt.show()  
  
# Create the figure for displaying filtered images  
fig = plt.figure(figsize=(20, 15))  
i = 1  
for p in range(25, 100, 25):  
 for k in range(5, 25, 5):  
 plt.subplot(3, 4, i)  
 filtered = ndimage.percentile\_filter(lena, percentile=p, size=(k, k, 1))  
 plot\_image(filtered, f'{p} percentile, {k}x{k} kernel')  
 i += 1  
  
plt.show()

**Nhập các thư viện cần thiết:**

* **import numpy as np**: Thư viện NumPy được sử dụng để xử lý mảng (array) và thực hiện các phép toán số học.
* **import matplotlib.pyplot as plt**: Thư viện Matplotlib được sử dụng để vẽ biểu đồ và hiển thị hình ảnh.
* **from scipy import ndimage**: Thư viện SciPy cung cấp nhiều công cụ cho xử lý ảnh, bao gồm các bộ lọc, trong đó có percentile\_filter.
* **import imageio**: Thư viện ImageIO giúp tải ảnh từ các file hình ảnh.

**Tải ảnh và thêm nhiễu muối và hạt tiêu:**

* **imageio.imread()**: Tải ảnh từ đường dẫn được chỉ định (r"C:\...sea\_bird.jpg"). Hàm này đọc file hình ảnh và trả về nó dưới dạng mảng NumPy.
* **np.random.random(lena.shape)**: Tạo một mảng ngẫu nhiên có cùng kích thước với ảnh (lena.shape), với các giá trị nằm trong khoảng từ 0 đến 1.
* **lena[noise > 0.9] = 255**: Thay thế những pixel có giá trị ngẫu nhiên trong noise lớn hơn 0.9 (tức là những phần tử trong mảng ngẫu nhiên có giá trị gần 1) thành giá trị 255 (tạo nhiễu muối, biểu thị là pixel sáng).
* **lena[noise < 0.1] = 0**: Thay thế những pixel có giá trị ngẫu nhiên trong noise nhỏ hơn 0.1 (tạo nhiễu hạt tiêu, biểu thị là pixel tối).

**Hàm vẽ ảnh:**

* **plt.imshow(img)**: Hàm này hiển thị ảnh img dưới dạng hình ảnh trong Matplotlib.
* **plt.title(title)**: Đặt tiêu đề cho ảnh, tiêu đề này được truyền vào dưới dạng tham số title.
* **plt.axis('off')**: Tắt các trục (axes) của đồ thị, giúp hiển thị ảnh mà không có các chỉ số trục.

**Hiển thị ảnh gốc có nhiễu:**

* **plot\_image(lena, 'Noisy image')**: Gọi hàm plot\_image để hiển thị ảnh lena (ảnh gốc với nhiễu muối và hạt tiêu) và đặt tiêu đề là 'Noisy image'.
* **plt.show()**: Hiển thị ảnh trong cửa sổ đồ thị của Matplotlib.

**Lọc ảnh với bộ lọc phần trăm (percentile\_filter) và hiển thị các kết quả:**

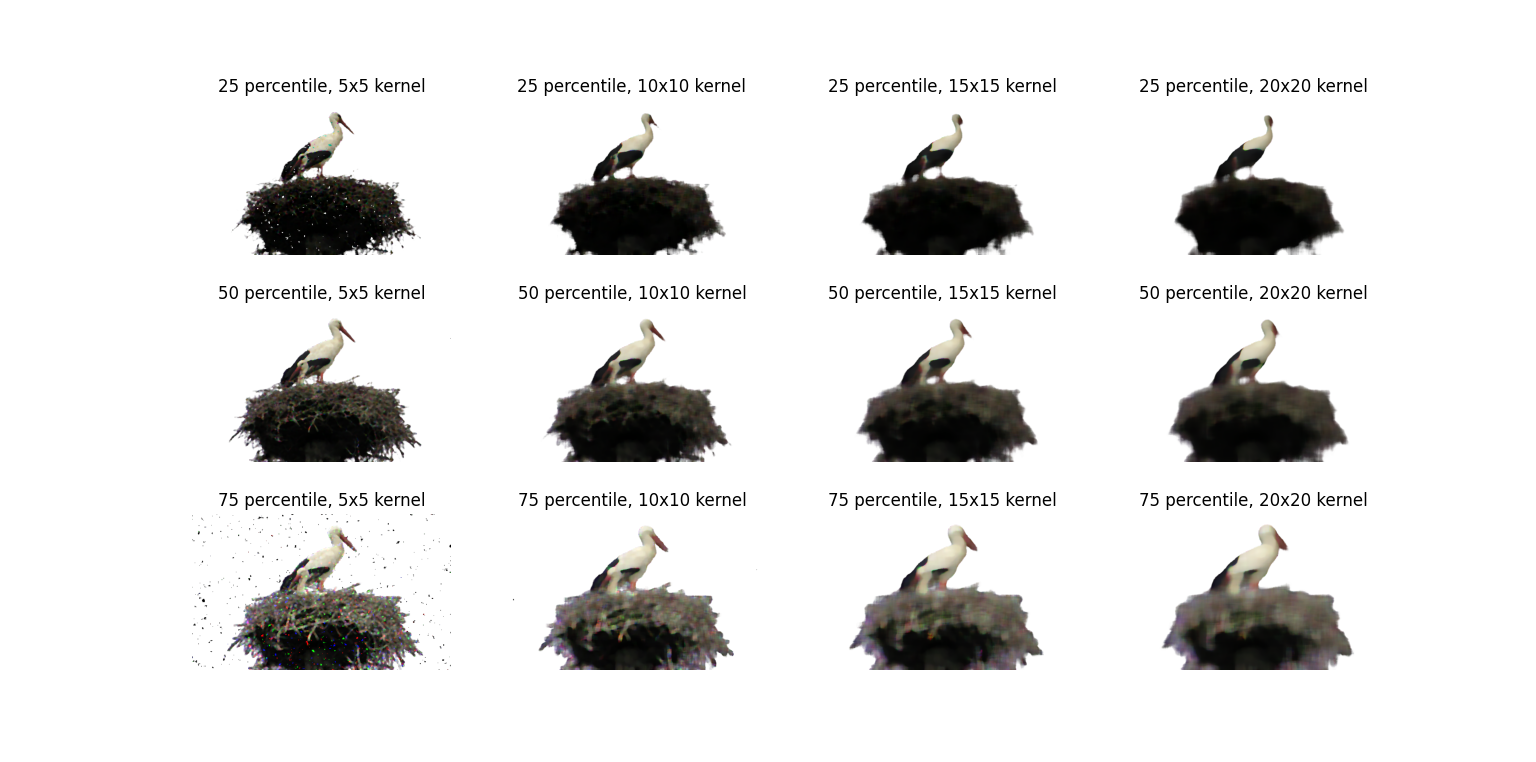
* **fig = plt.figure(figsize=(20, 15))**: Tạo một đối tượng figure mới với kích thước 20x15 inch để hiển thị nhiều ảnh cùng một lúc.
* **plt.subplot(3, 4, i)**: Chia cửa sổ đồ thị thành lưới 3 hàng và 4 cột, và vẽ ảnh vào vị trí thứ i (từ 1 đến 12, vì có 3x4 = 12 ảnh). Điều này cho phép hiển thị nhiều ảnh trong cùng một cửa sổ.
* **ndimage.percentile\_filter(lena, percentile=p, size=(k, k, 1))**: Áp dụng bộ lọc phần trăm lên ảnh lena:
  + **percentile=p**: Chỉ định phần trăm sử dụng trong bộ lọc, ví dụ: 25, 50, 75 cho các bộ lọc khác nhau.
  + **size=(k, k, 1)**: Kích thước của cửa sổ lân cận (kernel), ví dụ: 5x5, 10x10, 15x15 cho các kích thước khác nhau.
  + Bộ lọc phần trăm thay thế mỗi pixel trong ảnh bằng phần trăm (percentile) của các pixel trong cửa sổ lân cận.
* **plot\_image(filtered, f'{p} percentile, {k}x{k} kernel')**: Vẽ ảnh đã được lọc và thêm tiêu đề vào ảnh, ví dụ: '25 percentile, 5x5 kernel'.
* **i += 1**: Tăng biến i để vẽ ảnh vào ô tiếp theo trong lưới.
* **plt.show()**: Hiển thị tất cả các ảnh đã lọc trong cửa sổ đồ thị.

4, Kết quả sau khi thực hiện thuật toán :

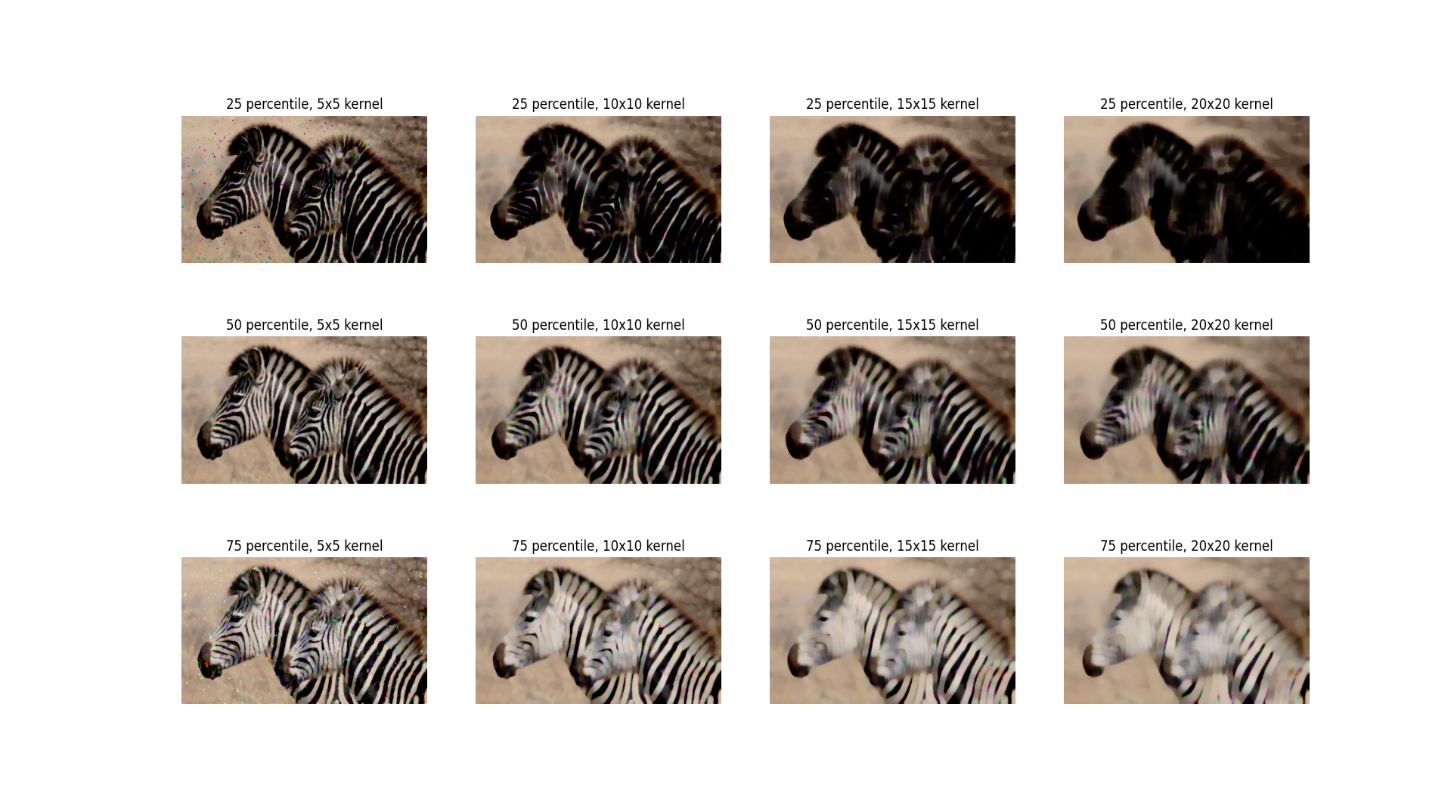
Ví dụ 1 : 



Ví dụ 2 : 



Ví dụ 3 : 



Nhận xét về kết quả sau khi thực hiện :

Khi ta chạy đoạn mã trên, ta sẽ nhận được một loạt các ảnh đã lọc, mỗi ảnh được tạo ra bằng cách áp dụng bộ lọc phần trăm (percentile\_filter) với các giá trị phần trăm khác nhau và các kích thước cửa sổ khác nhau. Dưới đây là nhận xét về các ảnh đầu ra:

**1. Ảnh gốc với nhiễu muối và hạt tiêu (Noisy Image):**

* **Nhiễu muối và hạt tiêu**: Ảnh gốc sẽ chứa những điểm sáng (muối) và điểm tối (hạt tiêu) phân tán ngẫu nhiên. Các pixel có giá trị gần 255 được thay thế bằng màu trắng (muối), trong khi các pixel có giá trị gần 0 được thay thế bằng màu đen (hạt tiêu).
* **Nhận xét**: Ảnh gốc bị nhiễu nghiêm trọng và khó nhận dạng các chi tiết, vì vậy cần sử dụng các phương pháp lọc để cải thiện chất lượng ảnh.

**2. Ảnh sau khi lọc với các giá trị percentile và kích thước cửa sổ khác nhau:**

Dưới đây là nhận xét chi tiết về ảnh đã lọc:

**a. Ảnh lọc với percentile = 25%:**

* **Nhận xét**: Khi sử dụng bộ lọc phần trăm với giá trị percentile thấp (25%), ảnh có xu hướng vẫn giữ lại phần lớn các nhiễu (vì phần trăm này tương đối thấp). Tuy nhiên, một số nhiễu vẫn được loại bỏ, nhưng hình ảnh vẫn còn khá nhiễu và chi tiết không rõ nét.
* **Kích thước cửa sổ nhỏ (ví dụ 5x5)**: Các chi tiết trong ảnh vẫn có thể bị mờ đi chút ít, vì cửa sổ nhỏ không thể lọc hiệu quả khi xử lý với percentile thấp.

**b. Ảnh lọc với percentile = 50% (Bộ lọc Trung vị):**

* **Nhận xét**: Bộ lọc với percentile = 50% chính là **bộ lọc trung vị**, đây là phương pháp lọc phổ biến và hiệu quả nhất để loại bỏ nhiễu muối và hạt tiêu. Bộ lọc trung vị sẽ thay thế mỗi pixel trong ảnh bằng giá trị trung vị của các pixel xung quanh nó trong cửa sổ lân cận. Khi sử dụng giá trị này:
  + **Kích thước cửa sổ nhỏ (5x5)**: Ảnh có vẻ sạch sẽ hơn nhiều so với ảnh gốc, với hầu hết nhiễu muối và hạt tiêu đã được loại bỏ mà vẫn giữ được khá tốt các chi tiết.
  + **Kích thước cửa sổ lớn (10x10 hoặc 15x15)**: Mặc dù bộ lọc trung vị vẫn làm sạch nhiễu tốt, nhưng một cửa sổ quá lớn có thể làm mờ đi chi tiết ảnh, ví dụ như các đường nét hoặc các đối tượng nhỏ trong ảnh.

**c. Ảnh lọc với percentile = 75%:**

* **Nhận xét**: Khi sử dụng percentile cao hơn (ví dụ, 75%), bộ lọc sẽ thay thế các giá trị của pixel bằng một giá trị cao hơn trong cửa sổ lân cận, làm cho ảnh trở nên sáng hơn.
  + **Kích thước cửa sổ nhỏ (5x5)**: Hình ảnh có thể có ít nhiễu hơn, nhưng các chi tiết sẽ không được giữ lại tốt như với percentile thấp hoặc trung vị.
  + **Kích thước cửa sổ lớn (10x10 hoặc 15x15)**: Khi cửa sổ lớn và percentile cao, các chi tiết trong ảnh có thể bị mất nhiều hơn. Việc áp dụng percentile cao có thể làm các vùng sáng trong ảnh bị làm mờ hoặc mất đi.

**d. Ảnh lọc với percentile = 100%:**

* **Nhận xét**: Với percentile = 100%, bộ lọc thay thế giá trị của pixel bằng giá trị lớn nhất trong cửa sổ lân cận, khiến ảnh trở nên sáng và có thể làm mất các chi tiết quan trọng.
  + **Kích thước cửa sổ nhỏ (5x5)**: Mặc dù giảm nhiễu, ảnh có thể trở nên quá sáng và thiếu chi tiết.
  + **Kích thước cửa sổ lớn (10x10 hoặc 15x15)**: Khi cửa sổ lớn và percentile cao, ảnh có thể mất đi quá nhiều chi tiết và trở nên quá sáng, khiến ảnh mất tính tự nhiên.

**3. Ảnh sau khi lọc với các cửa sổ lớn (10x10, 15x15):**

* **Nhận xét**: Khi sử dụng cửa sổ lớn (ví dụ 15x15), bộ lọc có thể làm mờ chi tiết ảnh một cách rõ rệt. Mặc dù nhiễu được giảm đáng kể, nhưng các chi tiết như các đường nét sắc nét, các đối tượng nhỏ trong ảnh sẽ trở nên mờ nhạt hơn.
* **Ảnh sau khi lọc với cửa sổ lớn** sẽ ít nhiễu hơn nhưng mất chi tiết nhiều hơn so với cửa sổ nhỏ.

**4. Tổng kết:**

* **Bộ lọc Trung vị (Percentile = 50%)**: Là lựa chọn tốt nhất khi muốn loại bỏ nhiễu muối và hạt tiêu mà không làm mất quá nhiều chi tiết ảnh. Khi kết hợp với cửa sổ nhỏ (5x5), bộ lọc trung vị hoạt động rất hiệu quả, vì nó giữ lại các chi tiết mà vẫn loại bỏ nhiễu.
* **Cửa sổ nhỏ vs Cửa sổ lớn**: Sử dụng cửa sổ nhỏ (ví dụ 5x5) giúp giữ lại chi tiết ảnh tốt hơn, trong khi cửa sổ lớn hơn (ví dụ 15x15) sẽ làm mờ các chi tiết nhưng giảm nhiễu hiệu quả hơn.
* **Percentile cao**: Bộ lọc với percentile cao (ví dụ 75% hoặc 100%) có thể làm sáng ảnh và mất đi nhiều chi tiết, vì vậy cần sử dụng một cách cẩn thận.