Image Processing (CSE281)

Fall 2025/2026

Dr. Essam Abdellatef

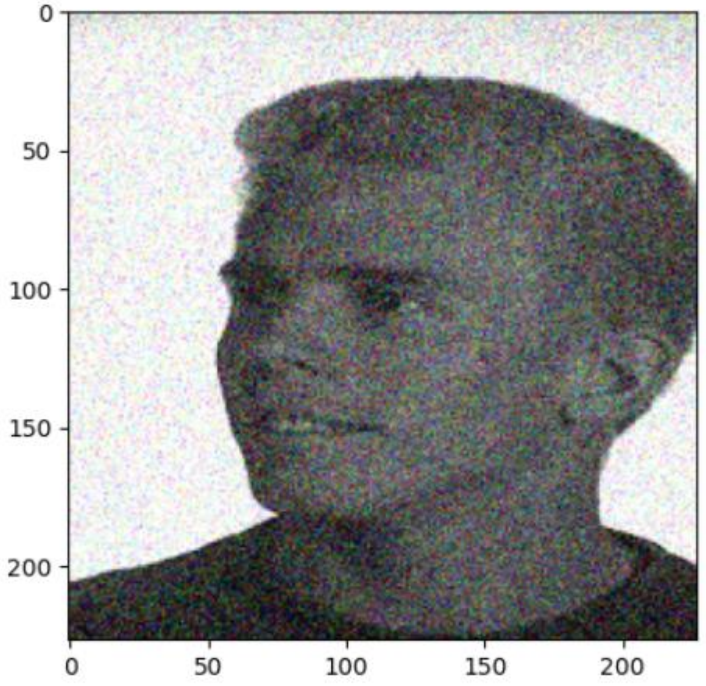
Contact Number: 012 8 192 55 90

Email: eabdellatef@Aiu.edu.eg

Gaussian Noise

```
import numpy as np
import cv2
import matplotlib.pyplot as plt
def add_gaussian_noise(image, mean = 0, sigma = 25):
    row, col, ch = image.shape
    gaussian = np.random.normal(mean, sigma, (row, col, ch))
    noisy_image = image + gaussian
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_gaussian_noise(image, sigma = 30)
plt.figure()
plt.imshow(noisy_image)
```



Gaussian Noise

row, col, ch = image.shape → Get image dimensions: rows, columns, and channels

gaussian = np.random.normal(mean, sigma, (row, col, ch))

- Generate Gaussian (normal) distribution noise with specified mean and standard deviation
- ***np.random.normal*** creates random numbers from a normal distribution.
- The shape (row, col, ch) matches the image dimensions

noisy_image = image + Gaussian

- Add the generated noise to the original image

noisy_image = np.clip(noisy_image, 0, 255)

- Clip values to ensure they stay within valid pixel range [0, 255]
- Values below 0 become 0, values above 255 become 255

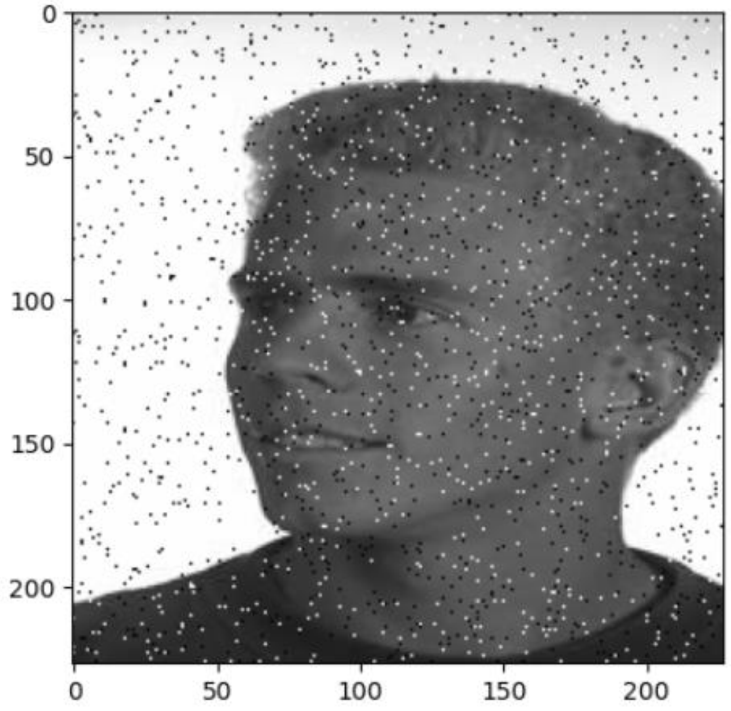
return noisy_image.astype(np.uint8) → Convert back to unsigned 8-bit integer (standard image format)

Salt & Pepper Noise

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

def add_salt_pepper_noise(image, salt_prob = 0.01, pepper_prob = 0.01):
    noisy_image = np.copy(image)
    salt_mask = np.random.random(image.shape[:2]) < salt_prob
    noisy_image[salt_mask] = 255
    pepper_mask = np.random.random(image.shape[:2]) < pepper_prob
    noisy_image[pepper_mask] = 0
    return noisy_image

image = cv2.imread('F:\\13.jpg')
noisy_image = add_salt_pepper_noise(image, salt_prob = 0.02, pepper_prob = 0.02)
plt.figure()
plt.imshow(noisy_image)
```



Salt & Pepper Noise

salt_mask = np.random.random(image.shape[:2]) < salt_prob

- Salt noise (white pixels)
- Generate random numbers between 0-1 for each pixel position
- Create a boolean mask where values < salt_prob become True

noisy_image[salt_mask] = 255

- Set all channels of selected pixels to maximum value (255 = white)

pepper_mask = np.random.random(image.shape[:2]) < pepper_prob

- Pepper noise (black pixels)

noisy_image[pepper_mask] = 0

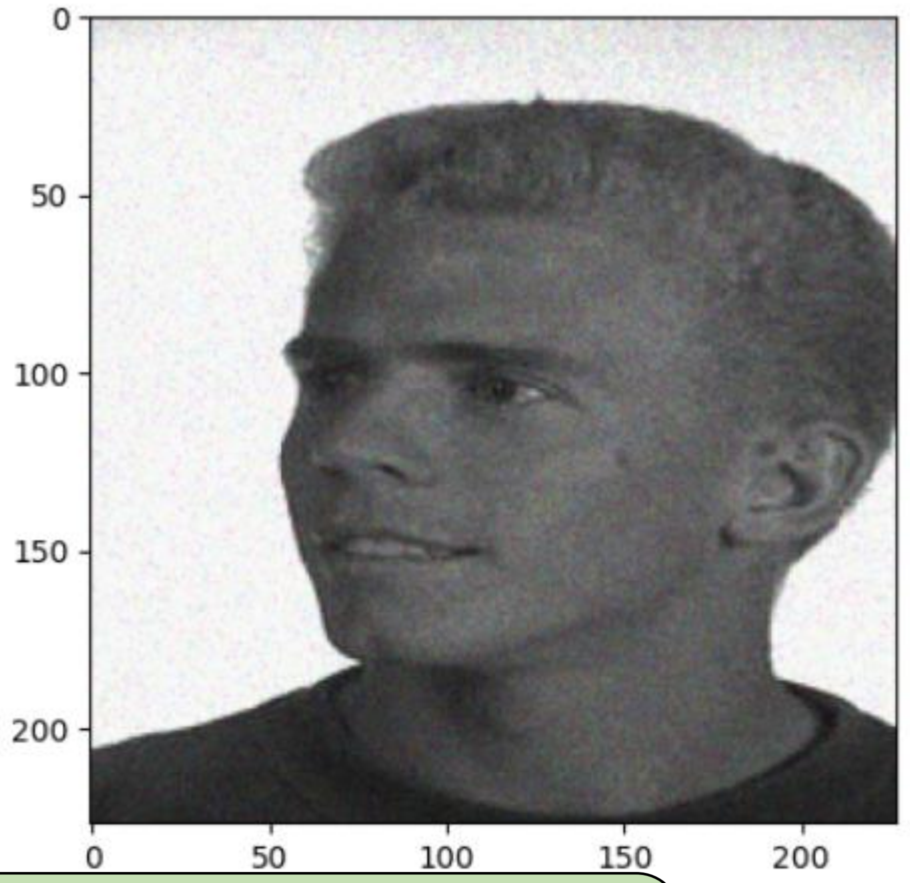
- Set all channels of selected pixels to minimum value (0 = black)

Poisson Noise

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

def add_poisson_noise(image):
    noise = np.random.poisson(image)
    noisy_image = np.clip(noise, 0, 250)
    return noisy_image.astype(np.uint8)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_poisson_noise(image)
plt.figure()
plt.imshow(noisy_image)
```



Generate *Poisson-distributed* random numbers

The parameter for Poisson is the pixel intensity value itself

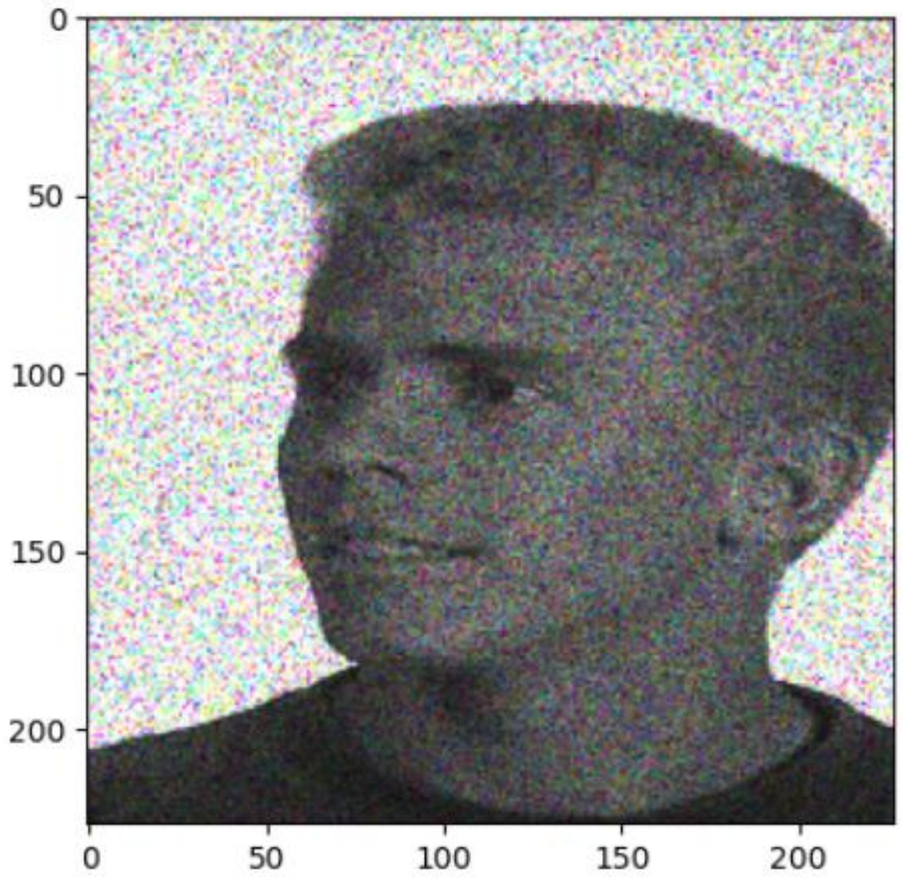
Brighter pixels get more noise, darker pixels get less

Speckle Noise

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

def add_speckle_noise(image, sigma = 0.1):
    row, col, ch = image.shape
    speckle = np.random.randn(row, col, ch) * sigma
    noisy_image = image + image * speckle
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_speckle_noise(image, sigma = 0.3)
plt.figure()
plt.imshow(noisy_image)
```



Generate random noise from *standard normal distribution (mean = 0, std = 1)*
Then scale it by sigma parameter to control noise intensity

np.random.random → **Uniform distribution [0, 1)**

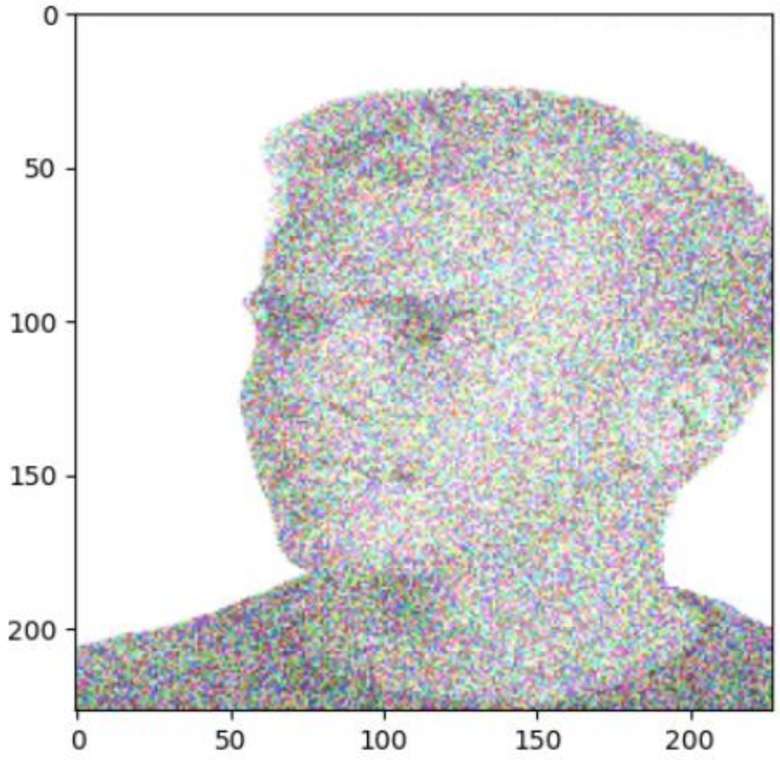
np.random.randn → **Standard normal distribution (mean = 0, std = 1)**

Uniform Noise

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

def add_uniform_noise(image, low = 0.01, high = 25):
    row, col, ch = image.shape
    uniform_noise = np.random.uniform(low, high, (row, col, ch))
    noisy_image = image + uniform_noise
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_uniform_noise(image, low = 20, high = 200)
plt.figure()
plt.imshow(noisy_image)
```



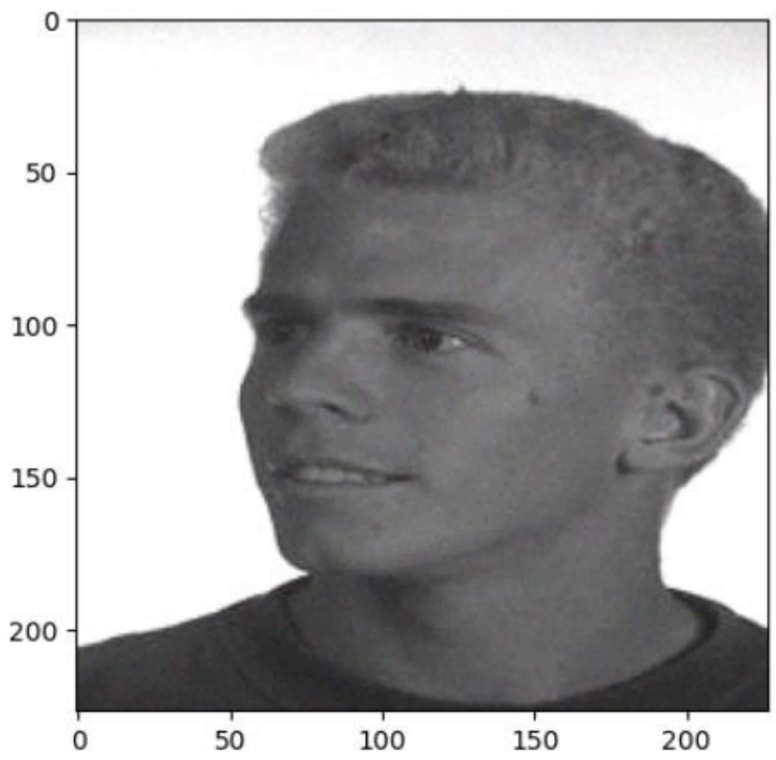
Generate noise from *uniform distribution*
Every value between *'low' and 'high'* has equal probability
Shape matches the image dimensions

Uniform Noise

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

def add_uniform_noise(image, low = 0.01, high = 25):
    row, col, ch = image.shape
    uniform_noise = np.random.uniform(low, high, (row, col, ch))
    noisy_image = image + uniform_noise
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_uniform_noise(image, low = 2, high = 20)
plt.figure()
plt.imshow(noisy_image)
```



Gaussian Filter

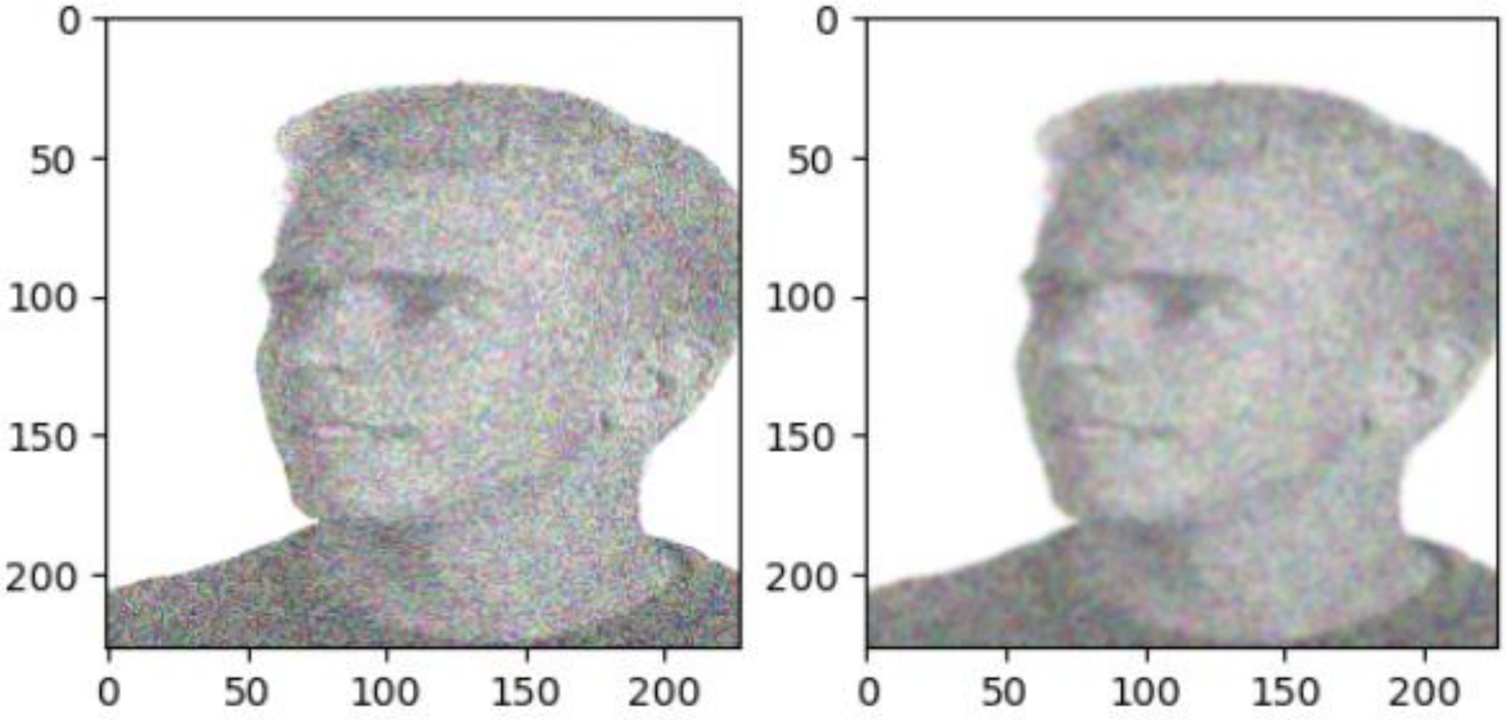
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def add_uniform_noise(image, low = 25, high = 150):
    row, col, ch = image.shape
    uniform_noise = np.random.uniform(low, high, (row, col, ch))
    noisy_image = image + uniform_noise
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

def gaussian_filter(image, kernel_size = 5, sigma = 1.0):
    return cv2.GaussianBlur(image, (kernel_size, kernel_size), sigma)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_uniform_noise(image, low = 25, high = 150)
denoised_image = gaussian_filter(noisy_image, kernel_size = 5, sigma = 1)
plt.figure()
plt.subplot(1, 2, 1)
plt.imshow(noisy_image)
plt.subplot(1, 2, 2)
plt.imshow(denoised_image)
plt.show()
```

Gaussian Filter



kernel_size = 5 **→ The size of the filter window (5×5 pixels)**
Sigma = 1 **→ The standard deviation of the Gaussian distribution**

Median Filter

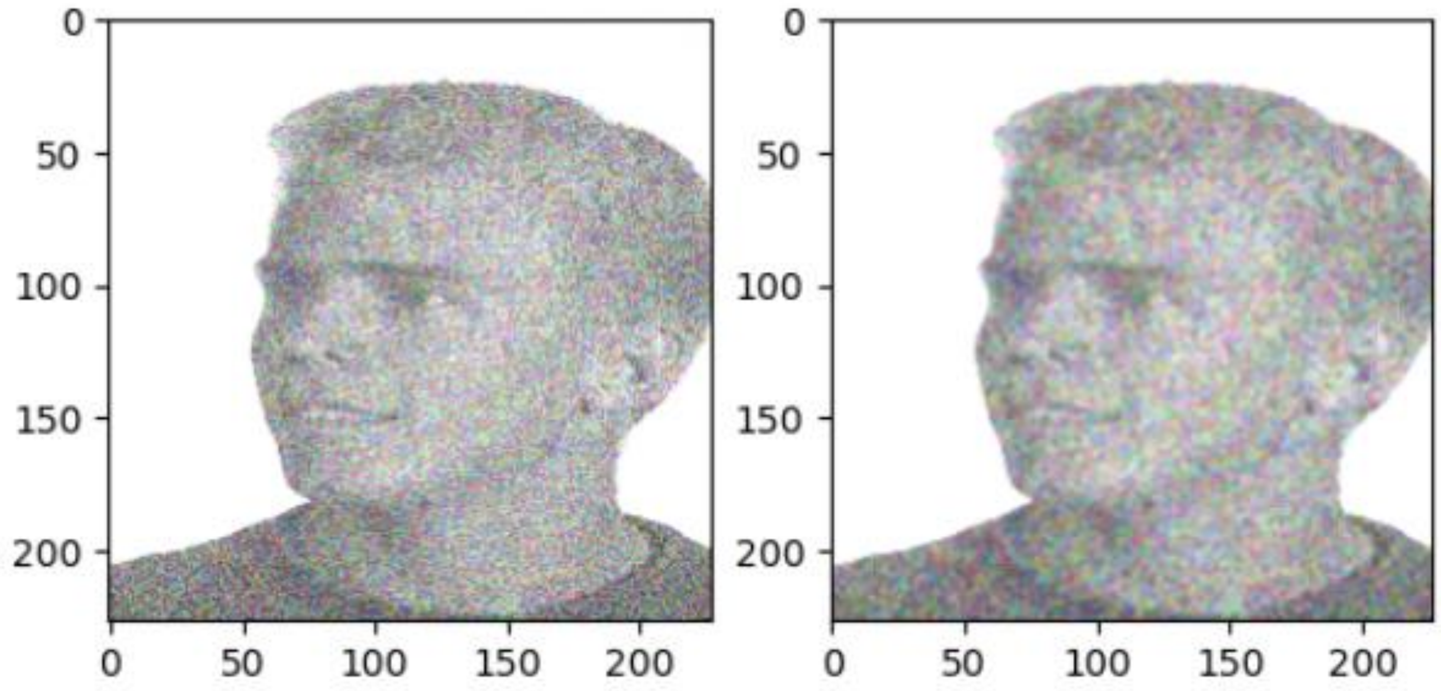
```
import numpy as np
import cv2
import matplotlib.pyplot as plt

def add_uniform_noise(image, low = 25, high = 150):
    row, col, ch = image.shape
    uniform_noise = np.random.uniform(low, high, (row, col, ch))
    noisy_image = image + uniform_noise
    noisy_image = np.clip(noisy_image, 0, 255)
    return noisy_image.astype(np.uint8)

def median_filter(image, kernel_size = 5):
    return cv2.medianBlur(image, kernel_size)

image = cv2.imread('F:\\13.jpg')
noisy_image = add_uniform_noise(image, low = 25, high = 150)
denoised_image = median_filter(noisy_image, kernel_size = 3)
plt.figure()
plt.subplot(1, 2, 1)
plt.imshow(noisy_image)
plt.subplot(1, 2, 2)
plt.imshow(denoised_image)
plt.show()
```

Median Filter



Fundamentals of Spatial Filtering

Spatial Filter:

Use the neighborhood (of each pixel) instead of a single pixel

- Low-pass
- Filter Smoothing
- High-pass
- Sharpening

Fundamentals of Spatial Filtering

- Think of a digital image as a grid of pixels, each with a value (like its brightness).
- Simple point operations change a pixel's value based only on that pixel's original value.
- Spatial filtering is more sophisticated.
- Instead of looking at a single pixel in isolation, it looks at a neighborhood of pixels surrounding it to decide what the new value for that pixel should be.

Fundamentals of Spatial Filtering

Spatial filters (also called spatial masks, kernels, templates, and windows)

The spatial filter consists of:

- Neighborhood, (typically a small rectangle).
- Predefined operation that is performed on the image pixels by the neighborhood.

Filtering creates a new pixel with a new value (the result of filtering operation).

Spatial Filtering:

- Linear spatial filtering.
- Nonlinear spatial filtering.

Fundamentals of Spatial Filtering

Linear Spatial Filtering (Convolution)

- This is the most common type.
- The operation is a linear mathematical operation.
- The kernel is placed over a neighborhood.
- Each pixel in the neighborhood is multiplied by the corresponding coefficient in the kernel.
- All these products are then summed up to produce the new value.

Fundamentals of Spatial Filtering

Nonlinear Spatial Filtering

- The operation is nonlinear.
- ***Median Filter:*** The new pixel value is the median (the middle value) of all the pixels in the neighborhood. This is excellent for removing "salt-and-pepper" noise.
- ***Maximum Filter:*** The new pixel value is the maximum value in the neighborhood.
- ***Minimum Filter:*** The new pixel value is the minimum value in the neighborhood.

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

?	?	?
?	?	?
?	?	?

Product = 1*1 + 1*0 + 1*1 + 0*0 + 1*1 + 1*0 + 0*1 + 0*1 + 0*0 + 0*0 + 1*1

Product = 4

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	?	?
?	?	?
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	?
?	?	?
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
?	?	?
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

Feature Map

4	3	4
2	?	?
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
2	4	?
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
2	4	3
?	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
2	4	3
2	?	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
2	4	3
2	3	?

Fundamentals of Spatial Filtering

Input

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

Filter

1	0	1
0	1	0
1	0	1

4	3	4
2	4	3
2	3	4

Fundamentals of Spatial Filtering

Image

10	10	200	200
10	10	200	200
10	10	200	200
10	10	200	200

Filter

1	2	1
2	4	2
1	2	1

Result = ?

Fundamentals of Spatial Filtering

Zero Padding

0	0	0	0	0	0
0	10	10	200	200	0
0	10	10	200	200	0
0	10	10	200	200	0
0	10	10	200	200	0
0	0	0	0	0	0

Fundamentals of Spatial Filtering

Border Padding

10	10	10	200	200	200
10	10	10	200	200	200
10	10	10	200	200	200
10	10	10	200	200	200
10	10	10	200	200	200
10	10	10	200	200	200

Fundamentals of Spatial Filtering

150	151	155	150	151
152	256	153	150	152
153	154	155	0	153
157	158	159	157	155

After Applying 3×3 Median Filter, Result = ?

Fundamentals of Spatial Filtering

After Border Padding, Result = ?