

## ✓ Pattern Recognition: Noise Reduction

```
1 import numpy as np
2 import cv2
3 from matplotlib import pyplot as plt
```

```
1 def apply_gaussian_filter(image: np.ndarray, kernel: np.ndarray) -> np.ndarray:
2     """
3     Apply a Gaussian filter to reduce noise in an image.
4
5     Args:
6         image (np.ndarray): The input image to be filtered.
7         kernel (np.ndarray): The Gaussian kernel to be applied for filtering.
8
9     Returns:
10        np.ndarray: The filtered image.
11    """
12    return cv2.filter2D(image, -1, kernel)
```

```
1 def generate_gaussian_kernel(size: int = 3, sigma: float = 1.0) -> np.ndarray:
2     """
3     Generate a normalized 2D Gaussian kernel.
4
5     Args:
6         size (int): Size of the kernel (e.g., 3x3, 5x5).
7         sigma (float): Standard deviation of the Gaussian distribution.
8
9     Returns:
10        np.ndarray: A 2D Gaussian kernel.
11    """
12    kernel_1d = cv2.getGaussianKernel(size, sigma)
13    kernel_2d = kernel_1d @ kernel_1d.T
14    return kernel_2d / np.sum(kernel_2d)
```

```
1 def add_salt_and_pepper_noise(image: np.ndarray, amount: float = 0.05) -> np.ndarray:
2     """
3     Add salt-and-pepper noise to an image.
4
5     Args:
6         image (np.ndarray): The input image to add noise to.
7         amount (float): The proportion of pixels to be affected by noise (0 to 1).
8
9     Returns:
10        np.ndarray: The noisy image.
11    """
12    noisy_image = image.copy()
13    total_pixels = noisy_image.size
14    num_salt = int(amount * total_pixels * 0.5)
15    num_pepper = int(amount * total_pixels * 0.5)
16
17    # Salt noise (white pixels)
18    salt_coords = [np.random.randint(0, dim - 1, num_salt) for dim in noisy_image.shape]
19    noisy_image[salt_coords[0], salt_coords[1]] = 255
20
21    # Pepper noise (black pixels)
22    pepper_coords = [np.random.randint(0, dim - 1, num_pepper) for dim in noisy_image.shape]
23    noisy_image[pepper_coords[0], pepper_coords[1]] = 0
24
25    return noisy_image
```

```
1 def plot_images(original: np.ndarray, noisy: np.ndarray, filtered: np.ndarray) -> None:
2     """
3     Plot the original, noisy, and filtered images side by side.
4
5     Args:
6         original (np.ndarray): The original image.
7         noisy (np.ndarray): The noisy image.
8         filtered (np.ndarray): The filtered image.
```

```

9
10 Returns:
11     None
12     """
13 plt.figure(figsize=(15, 5))
14
15 # Original Image
16 plt.subplot(1, 3, 1)
17 plt.title("Original Image")
18 plt.imshow(original, cmap='gray')
19 plt.axis('off')
20
21 # Noisy Image
22 plt.subplot(1, 3, 2)
23 plt.title("Noisy Image")
24 plt.imshow(noisy, cmap='gray')
25 plt.axis('off')
26
27 # Filtered Image
28 plt.subplot(1, 3, 3)
29 plt.title("Filtered Image")
30 plt.imshow(filtered, cmap='gray')
31 plt.axis('off')
32
33 plt.show()

```

```

1 def main() -> None:
2     """
3     Main function to demonstrate noise reduction using a Gaussian filter.
4     Loads an image, adds salt-and-pepper noise, applies a Gaussian filter,
5     and plots the original, noisy, and filtered images.
6
7     Returns:
8         None
9     """
10    # Load the image in grayscale
11    image = cv2.imread('/content/original.jpg', cv2.IMREAD_GRAYSCALE)
12
13    if image is None:
14        raise FileNotFoundError("Image not found. Please check the file path.")
15
16    # Add salt-and-pepper noise to the image
17    noisy_image = add_salt_and_pepper_noise(image, amount=0.05)
18
19    # Generate a 3x3 Gaussian kernel
20    gaussian_kernel = generate_gaussian_kernel(size=3, sigma=1)
21
22    # Apply the Gaussian filter to the noisy image
23    filtered_image = apply_gaussian_filter(noisy_image, gaussian_kernel)
24
25    # Plot the images
26    plot_images(original=image, noisy=noisy_image, filtered=filtered_image)
27
28
29 if __name__ == "__main__":
30     main()
31

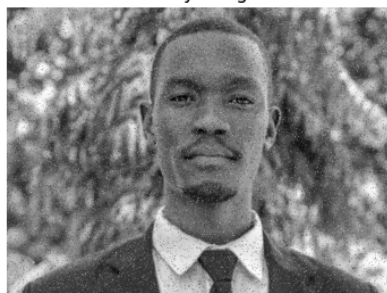
```



Original Image



Noisy Image



Filtered Image

