**EX.NO:5 DATE: 13.02.2025**

**EDGE DETECTION, NOISE ADDITION AND REMOVAL**

**Aim:**

To apply edge detection, noise addition, and noise removal techniques to analyze image processing methods.

**Algorithm:**

1. Convert the image to grayscale.  
2. Apply different edge detection techniques such as Sobel, Prewitt, Roberts, and Laplacian of Gaussian.  
3. Implement noise addition methods including Gaussian, Salt & Pepper, and Impulse noise.  
4. Use various noise removal techniques such as Gaussian Blur, Median Filtering, and Bilateral Filtering.  
5. Display the processed images and analyze their histograms.

Code

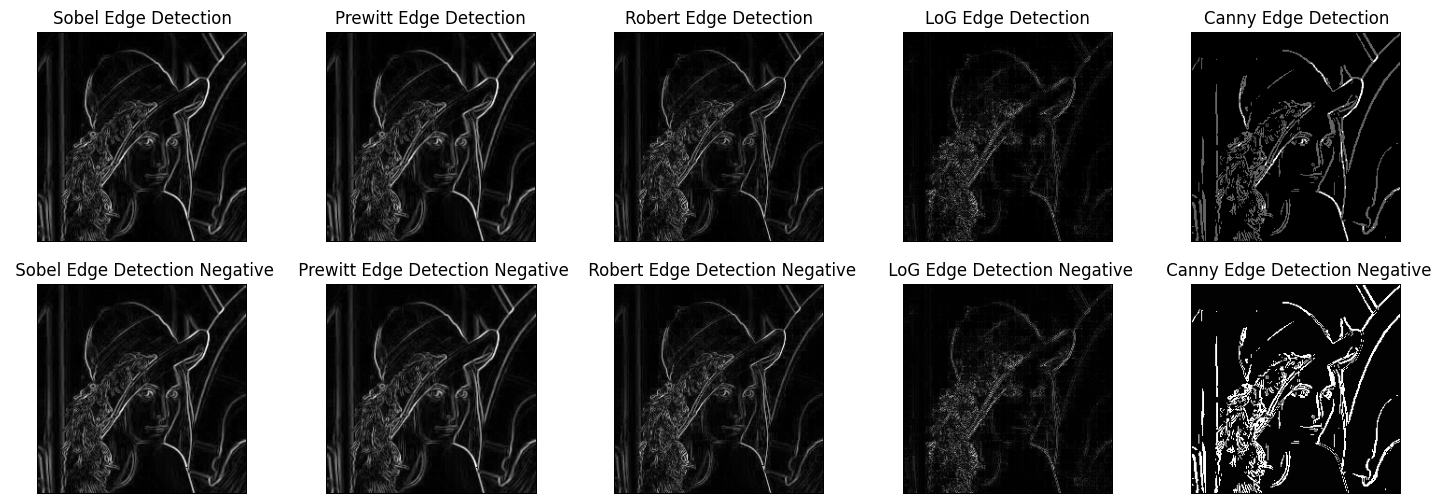
import cv2  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns

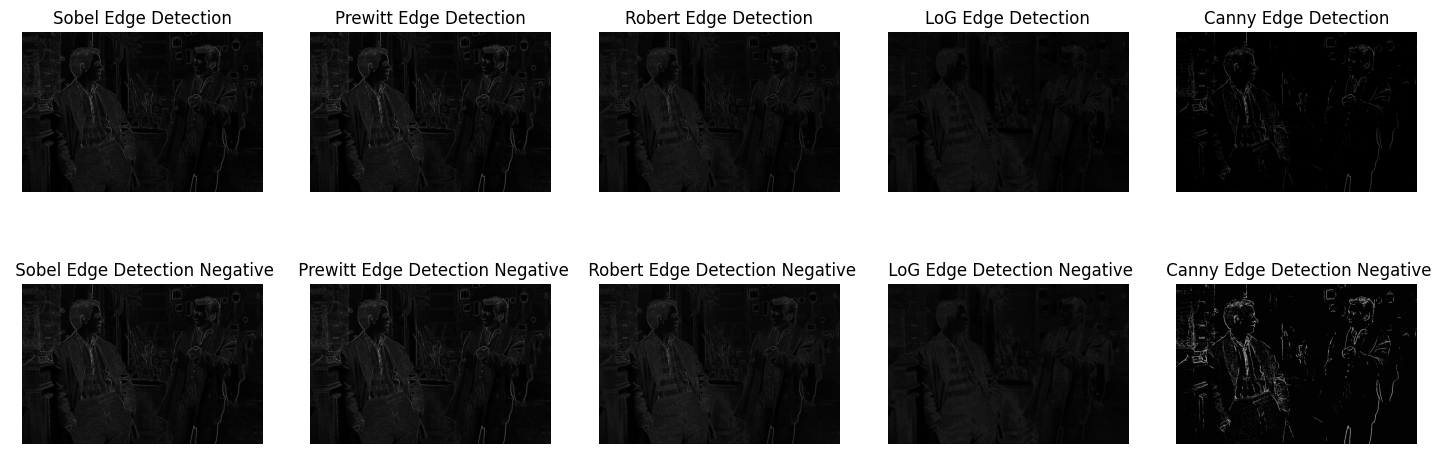
image = cv2.imread("lena.jpeg", 0)  
image2 = cv2.imread("edge.jpg", 0)

# Edge Detection

def edge\_detection(image):  
   
 rows, cols = image.shape  
   
 sobel\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])  
 sobel\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])  
   
 prewitt\_x = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])  
 prewitt\_y = np.array([[-1, -1, -1], [0, 0, 0], [1, 1, 1]])  
   
 robert\_x = np.array([[1, 0], [0, -1]])  
 robert\_y = np.array([[0, 1], [-1, 0]])  
   
 laplacian = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])  
   
 methods = {  
 'Sobel': (sobel\_x, sobel\_y),  
 'Prewitt': (prewitt\_x, prewitt\_y),  
 'Robert': (robert\_x, robert\_y),  
 'LoG': (laplacian, None)  
 }  
   
 results = {}  
   
 for method, (kernel\_x, kernel\_y) in methods.items():  
 gradient\_x = np.zeros\_like(image, dtype=np.float32)  
 gradient\_y = np.zeros\_like(image, dtype=np.float32) if kernel\_y is not None else None  
   
 for i in range(1, rows-1):  
 for j in range(1, cols-1):  
 if method in ['Sobel', 'Prewitt', 'LoG']:  
 gradient\_x[i, j] = np.sum(kernel\_x \* image[i-1:i+2, j-1:j+2])  
 if kernel\_y is not None:  
 gradient\_y[i, j] = np.sum(kernel\_y \* image[i-1:i+2, j-1:j+2])  
 elif method == 'Robert':  
 gradient\_x[i, j] = np.sum(kernel\_x \* image[i:i+2, j:j+2])  
 gradient\_y[i, j] = np.sum(kernel\_y \* image[i:i+2, j:j+2])  
   
 if kernel\_y is not None:  
 gradient\_magnitude = np.sqrt(gradient\_x\*\*2 + gradient\_y\*\*2)  
 else:  
 gradient\_magnitude = np.abs(gradient\_x)  
   
 gradient\_magnitude = (gradient\_magnitude - gradient\_magnitude.min()) / (gradient\_magnitude.max() - gradient\_magnitude.min()) \* 255  
 results[method] = gradient\_magnitude.astype(np.uint8)  
   
 weak, strong = 50, 150  
 canny\_edges = np.zeros\_like(image)  
 strong\_edges = results['Sobel'] > strong  
 weak\_edges = (results['Sobel'] >= weak) & (results['Sobel'] <= strong)  
 canny\_edges[strong\_edges] = 255  
 canny\_edges[weak\_edges] = 100  
 results['Canny'] = canny\_edges  
  
 plt.figure(figsize=(18, 6))  
 for i, (method, result) in enumerate(results.items(), 1):  
 plt.subplot(2, 5, i)  
 plt.imshow(result, cmap='gray')  
 plt.title(f'{method} Edge Detection')  
 plt.axis('off')  
 plt.subplot(2,5,i+5)  
 plt.imshow(result-255, cmap='gray')  
 plt.title(f' {method} Edge Detection Negative')  
 plt.axis('off')  
   
 plt.show()

edge\_detection(image)  
edge\_detection(image2)





# Noise Addition

import numpy as np  
import matplotlib.pyplot as plt  
import cv2  
  
def noise\_addition(image, nv=25, sp\_ratio=0.02, quant\_levels=10, impulse\_ratio=0.02):  
 if len(image.shape) == 3:  
 gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)  
 else:  
 gray\_image = image  
   
 images = {"Original Color": image, "Original Gray": gray\_image}  
 noises = {}  
   
 gaussian\_noise = np.random.normal(0, nv, image.shape)  
 noises["Gaussian Color"] = np.clip(image + gaussian\_noise, 0, 255).astype(np.uint8)  
 noises["Gaussian Gray"] = np.clip(gray\_image + gaussian\_noise[..., 0], 0, 255).astype(np.uint8)  
   
 salt\_pepper = np.random.rand(\*gray\_image.shape)  
 sp\_color = np.copy(image)  
 sp\_gray = np.copy(gray\_image)  
 sp\_color[salt\_pepper < sp\_ratio/2] = 0  
 sp\_color[salt\_pepper > 1 - sp\_ratio/2] = 255  
 sp\_gray[salt\_pepper < sp\_ratio/2] = 0  
 sp\_gray[salt\_pepper > 1 - sp\_ratio/2] = 255  
 noises["Salt & Pepper Color"] = sp\_color  
 noises["Salt & Pepper Gray"] = sp\_gray  
   
 noises["Quantization Color"] = (image // (256 // quant\_levels)) \* (256 // quant\_levels)  
 noises["Quantization Gray"] = (gray\_image // (256 // quant\_levels)) \* (256 // quant\_levels)  
   
 impulse\_noise = np.random.rand(\*gray\_image.shape)  
 impulse\_color = np.copy(image)  
 impulse\_gray = np.copy(gray\_image)  
 impulse\_color[impulse\_noise < impulse\_ratio/2] = 0  
 impulse\_color[impulse\_noise > 1 - impulse\_ratio/2] = 255  
 impulse\_gray[impulse\_noise < impulse\_ratio/2] = 0  
 impulse\_gray[impulse\_noise > 1 - impulse\_ratio/2] = 255  
 noises["Impulse Color"] = impulse\_color  
 noises["Impulse Gray"] = impulse\_gray  
   
 images.update(noises)  
   
 fig, axs = plt.subplots(len(images), 2, figsize=(12, len(images) \* 3))  
   
 for i, (noise\_type, img) in enumerate(images.items()):  
 cmap = None if len(img.shape) == 3 else 'gray'  
 axs[i, 0].imshow(img, cmap=cmap)  
 axs[i, 0].set\_title(f"{noise\_type} Image")  
 axs[i, 0].axis("off")  
 axs[i, 1].hist(img.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7)  
 axs[i, 1].set\_title(f"Histogram of {noise\_type} Image")  
   
 plt.tight\_layout()  
 plt.show()  
  
 return noises["Gaussian Color"]

image = cv2.imread('lena.jpeg', 1)  
image = cv2.cvtColor(image,cv2.COLOR\_BGR2RGB)  
noisy\_image = noise\_addition(image,70)

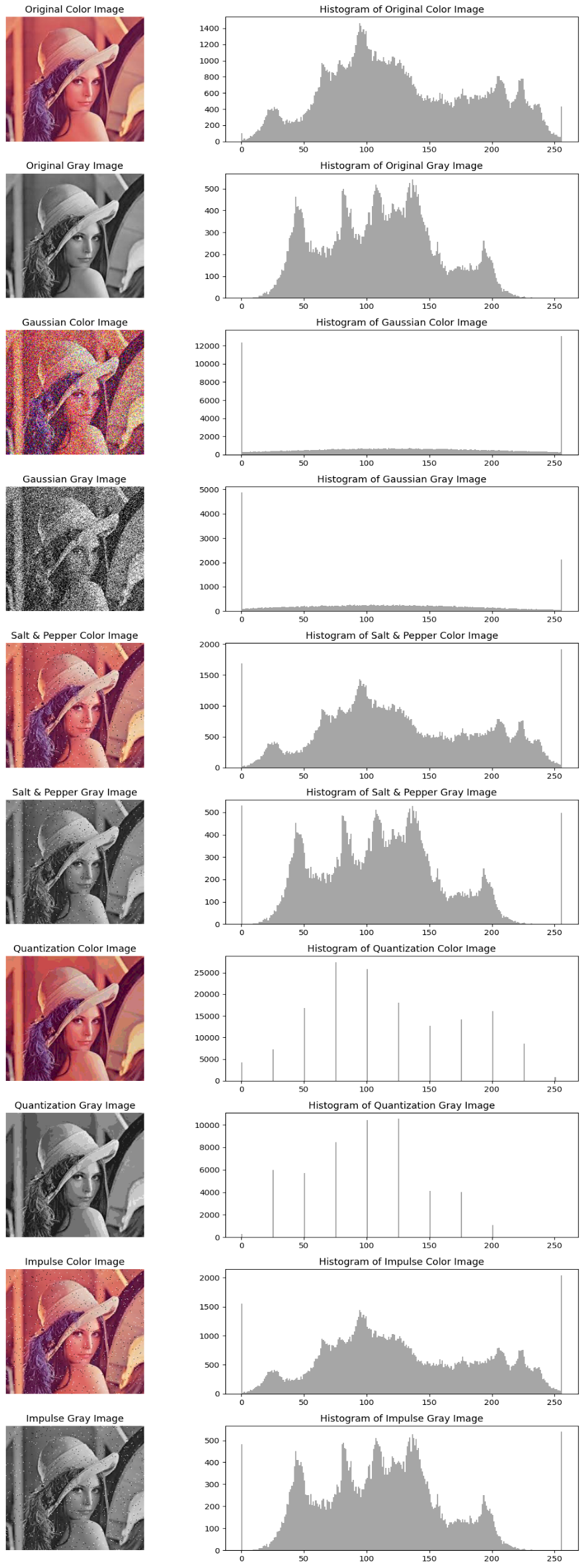
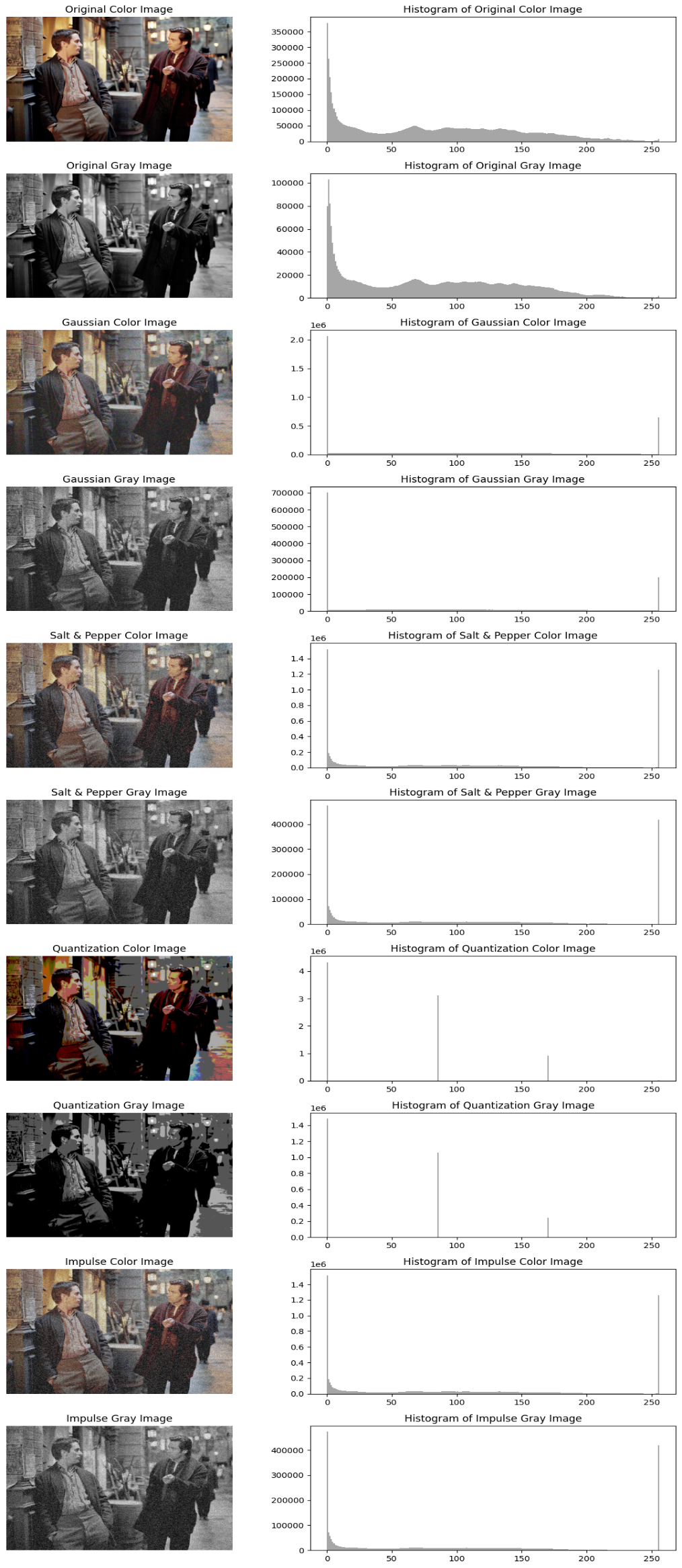


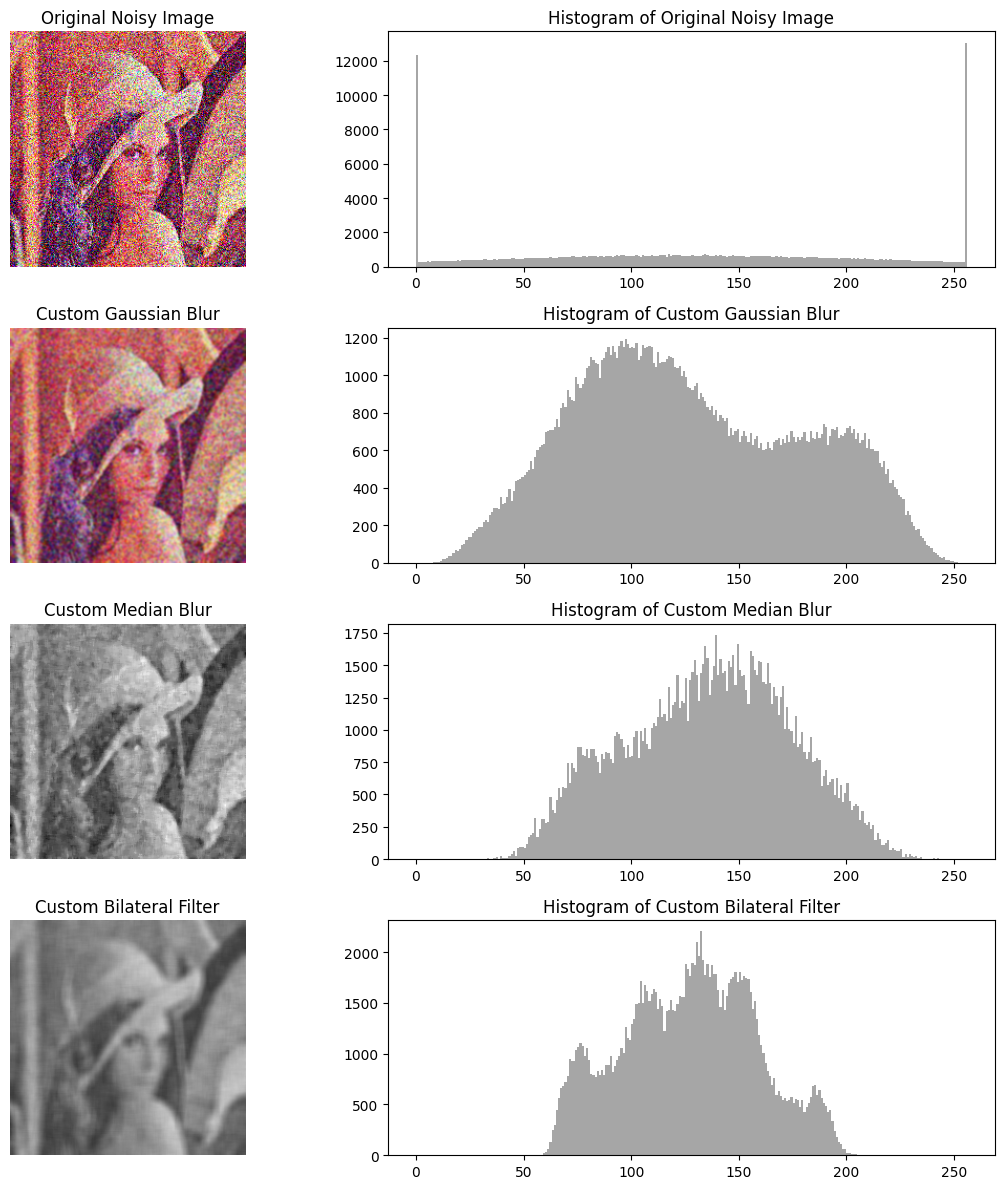
image2 = cv2.imread("edge.jpg", 1)  
image2 = cv2.cvtColor(image2, cv2.COLOR\_BGR2RGB)  
nimage = noise\_addition(image2, nv=100, sp\_ratio=0.3, quant\_levels=3, impulse\_ratio=0.3)



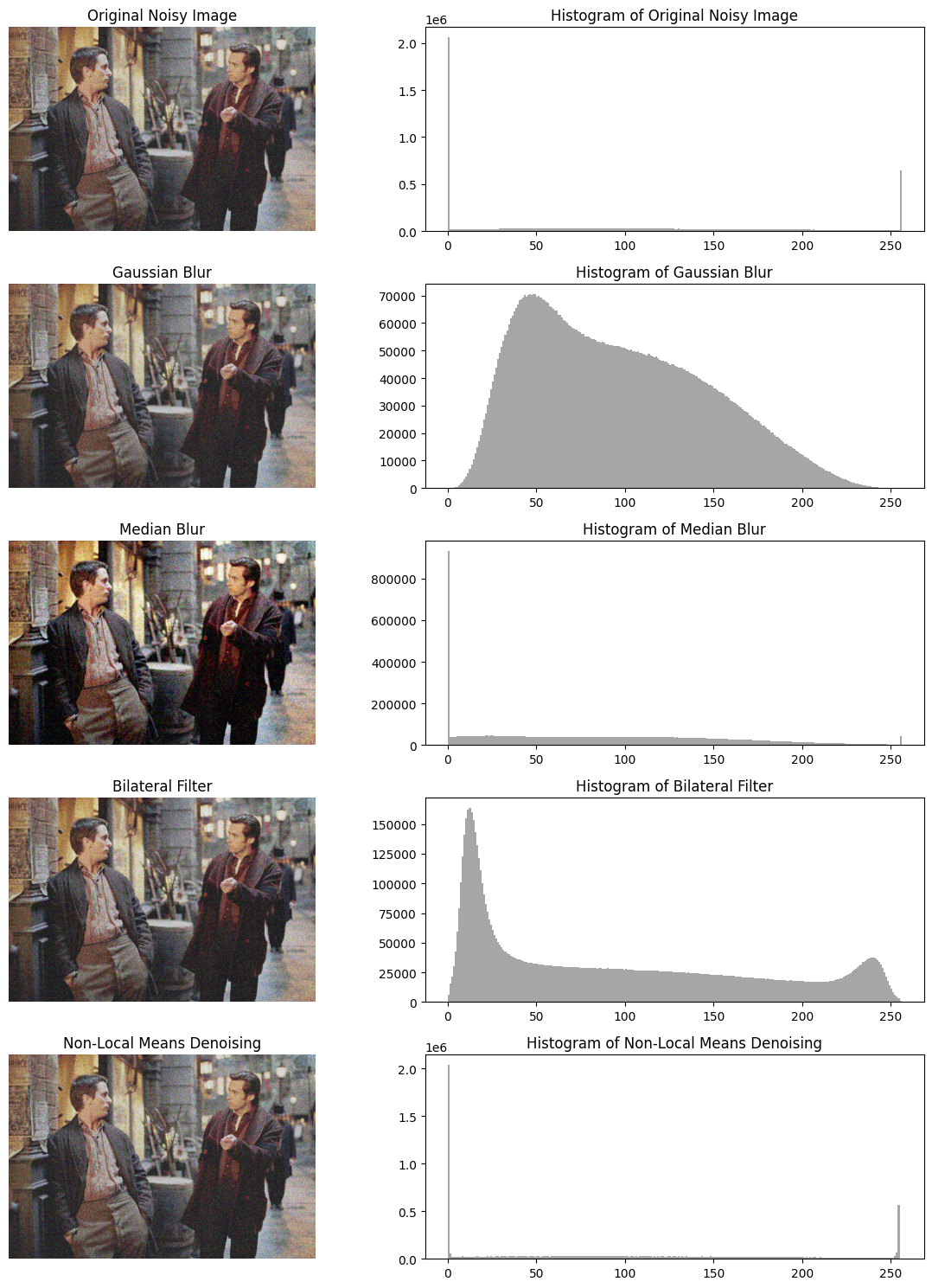
# Noise Removal

def custom\_gaussian\_blur(image, kernel\_size=5, sigma=1.0):  
 kernel = cv2.getGaussianKernel(kernel\_size, sigma)  
 kernel = np.outer(kernel, kernel)  
 return cv2.filter2D(image, -1, kernel)  
  
def custom\_median\_blur(image, kernel\_size=5):  
 padded\_image = np.pad(image, kernel\_size // 2, mode='edge')  
 output = np.zeros\_like(image)  
 for i in range(image.shape[0]):  
 for j in range(image.shape[1]):  
 output[i, j] = np.median(padded\_image[i:i+kernel\_size, j:j+kernel\_size])  
 return output  
  
def custom\_bilateral\_filter(image, d=9, sigma\_color=75, sigma\_space=75):  
 filtered\_image = np.copy(image)  
 for i in range(image.shape[0]):  
 for j in range(image.shape[1]):  
 neighborhood = image[max(0, i-d//2):min(image.shape[0], i+d//2+1),  
 max(0, j-d//2):min(image.shape[1], j+d//2+1)]  
 color\_weights = np.exp(-((neighborhood - image[i, j])\*\*2) / (2 \* sigma\_color\*\*2))  
 space\_weights = np.exp(-((np.arange(neighborhood.shape[0])[:, None] - d//2)\*\*2 +   
 (np.arange(neighborhood.shape[1])[None, :] - d//2)\*\*2) / (2 \* sigma\_space\*\*2))  
 weights = color\_weights \* space\_weights[..., np.newaxis]  
 weights /= np.sum(weights)  
 filtered\_image[i, j] = np.sum(neighborhood \* weights)  
 return filtered\_image  
  
def remove\_all\_noises(image):  
   
 denoised\_methods = {  
 "Custom Gaussian Blur": custom\_gaussian\_blur(image),  
 "Custom Median Blur": custom\_median\_blur(image),  
 "Custom Bilateral Filter": custom\_bilateral\_filter(image)  
 }  
   
 fig, axs = plt.subplots(len(denoised\_methods) + 1, 2, figsize=(12, (len(denoised\_methods) + 1) \* 3))  
   
 axs[0, 0].imshow(image, cmap='gray' if len(image.shape) == 2 else None)  
 axs[0, 0].set\_title("Original Noisy Image")  
 axs[0, 0].axis("off")  
 axs[0, 1].hist(image.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7)  
 axs[0, 1].set\_title("Histogram of Original Noisy Image")  
   
 for i, (method, denoised) in enumerate(denoised\_methods.items(), 1):  
 cmap = 'gray' if len(denoised.shape) == 2 else None  
 axs[i, 0].imshow(denoised, cmap=cmap)  
 axs[i, 0].set\_title(f"{method}")  
 axs[i, 0].axis("off")  
 axs[i, 1].hist(denoised.ravel(), bins=256, range=[0, 256], color='gray', alpha=0.7)  
 axs[i, 1].set\_title(f"Histogram of {method}")  
   
 plt.tight\_layout()  
 plt.show()

remove\_all\_noises(noisy\_image)



remove\_all\_noises(nimage)



**Inference**

- Edge detection highlights object boundaries using different gradient-based methods.  
- Adding noise affects image quality, making analysis difficult.  
- Noise removal techniques help restore image clarity by reducing unwanted distortions.

**Result:**The applied edge detection, noise addition, and removal techniques successfully processed images, demonstrating the impact of various filters and noise types on image quality.