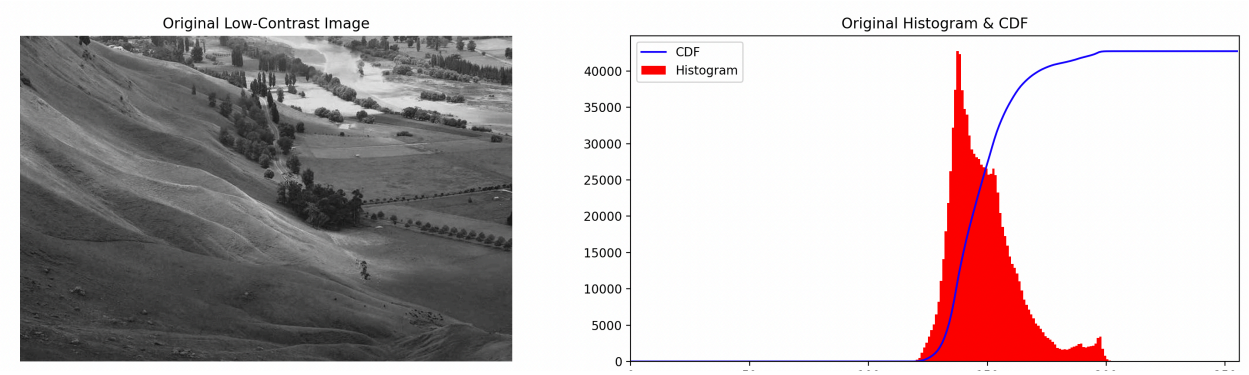
Lab 3: Image Processing

Name: Vedant Sinha Student ID: 100420615

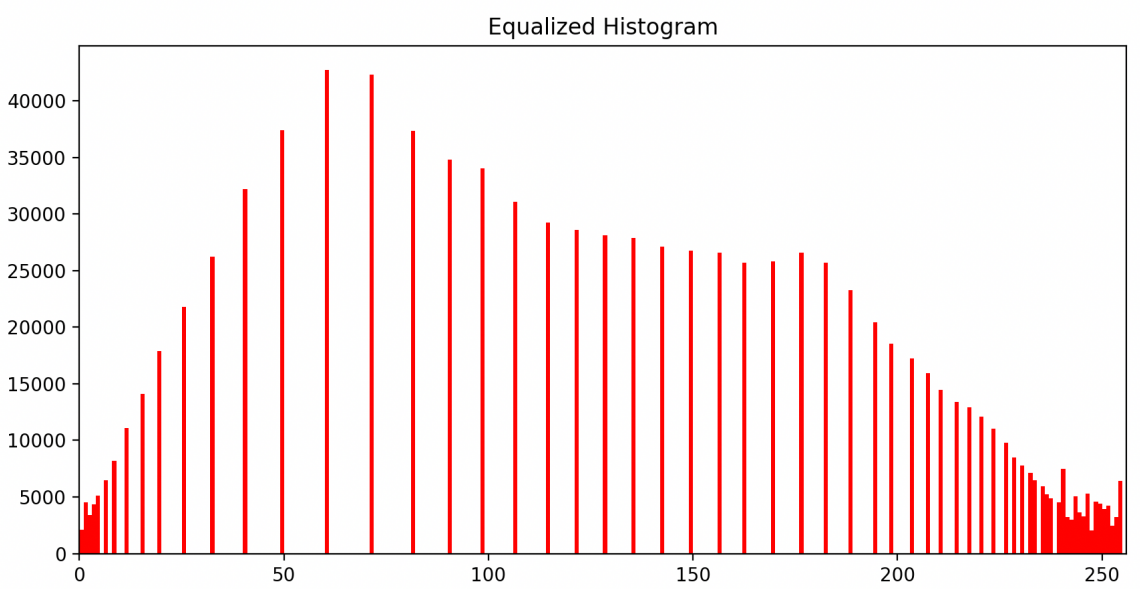
1. Histogram equalization

1. Compute and visualize histogram and cumulative distance function (CDF) of an input gray-scale image



2. Apply histogram equalization using obtained CDF on the input image

3. Compute and visualize histogram of output image



2. Image denoising

1. Read the input image and convert to a grayscale image

2. Add two types of noise including Gaussian noise and Salt/Pepper noise (Implement your own functions to add noise to an image)

3. Implement mean and median filtering in 5x5 windows

4. Check if mean or median filtering is able to completely remove Gaussian noise or Salt/Pepper noise. Compare original image and denoised image.

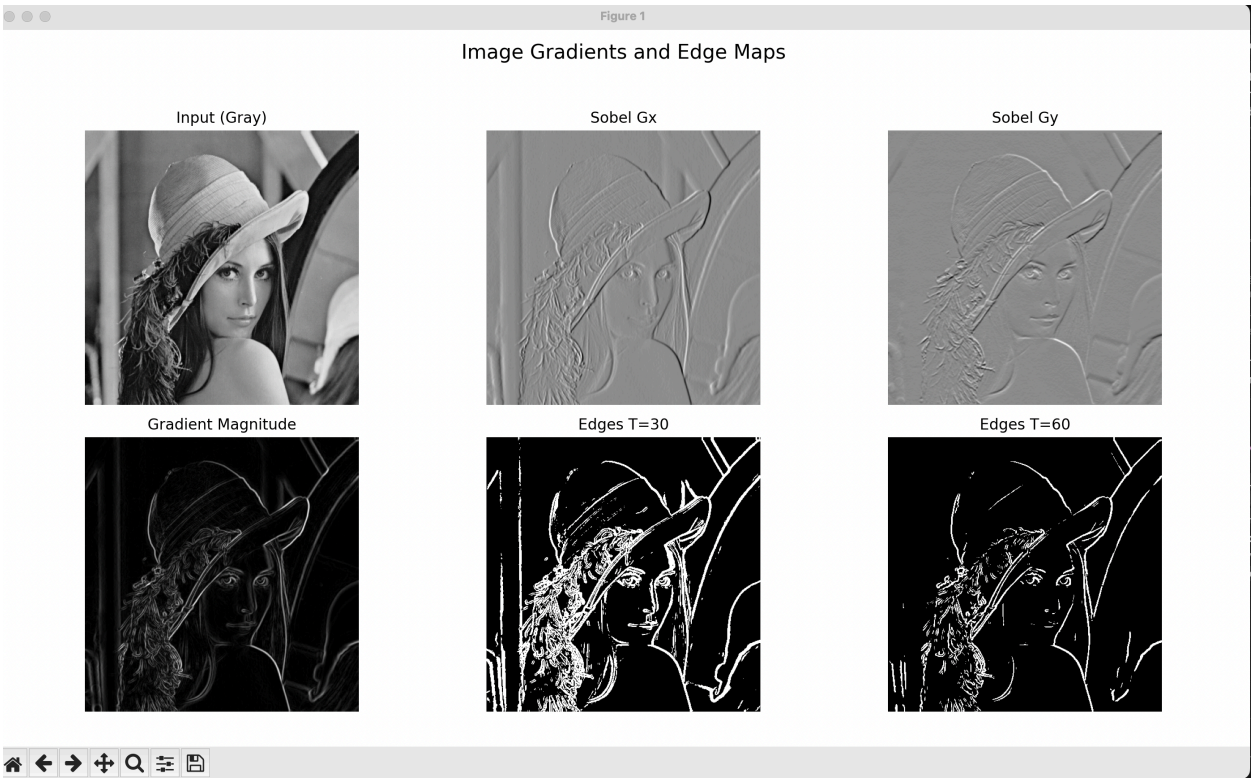
3. Image gradient

1. Compute image gradient in x and y direction respectively

2. Read the input image and convert to a grayscale image

3. Compute magnitude of image gradient for each pixel

4. Thresholding on magnitude to determine image edges, try various thresholds.



4. Code screenshots

Question 1 Code:

```python
import cv2

import numpy as np
from matplotlib import pyplot as plt

try:
    img = cv2.imread('bay.png', cv2.IMREAD_GRAYSCALE)

    if img is None:
        raise FileNotFoundError("The image 'bay.png' could not be found. Please check the file path.")

    hist, bins = np.histogram(img.flatten(), 256, [0, 256])
```

```
    cdf = hist.cumsum()
    cdf_normalized = cdf * float(hist.max()) / cdf.max()
    cdf_m = np.ma.masked_equal(cdf, 0)
    cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
    cdf_final = np.ma.filled(cdf_m, 0).astype('uint8')
    img_equalized = cdf_final[img]
    hist_equalized, bins_equalized = np.histogram(img_equalized.flatten(), 256, [0,
256])


    plt.figure(figsize=(15, 10))
    plt.suptitle('Histogram Equalization', fontsize=16)

    plt.subplot(2, 2, 1)
    plt.imshow(img, cmap='gray')
    plt.title('Original Low-Contrast Image')
    plt.axis('off')


    plt.subplot(2, 2, 2)
    plt.plot(cdf_normalized, color='b', label='CDF')
    plt.hist(img.flatten(), 256, [0, 256], color='r', label='Histogram')
    plt.xlim([0, 256])
    plt.legend(loc='upper left')
    plt.title('Original Histogram & CDF')

    plt.subplot(2, 2, 3)
    plt.imshow(img_equalized, cmap='gray')
    plt.title('High-Contrast Equalized Image')
    plt.axis('off')

    plt.subplot(2, 2, 4)
    plt.hist(img_equalized.flatten(), 256, [0, 256], color='r')
    plt.xlim([0, 256])
    plt.title('Equalized Histogram')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()

except FileNotFoundError as e:
    print(e)
except Exception as e:
    print(f"An error occurred: {e}"
```

Question 2 Screenshot:

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


def add_gaussian_noise(image: np.ndarray, mean: float = 0.0, sigma: float = 20.0) ->
np.ndarray:
    image_float = image.astype(np.float32)
    noise = np.random.normal(loc=mean, scale=sigma,
size=image.shape).astype(np.float32)
    noisy = image_float + noise
    noisy_clipped = np.clip(noisy, 0, 255).astype(np.uint8)
    return noisy_clipped


def add_salt_and_pepper_noise(image: np.ndarray, amount: float = 0.02, s_vs_p: float =
0.5) -> np.ndarray:

    noisy = image.copy()
    num_pixels = image.size

    num_salt = int(np.ceil(amount * num_pixels * s_vs_p))
    coords_salt_rows = np.random.randint(0, image.shape[0], num_salt)
    coords_salt_cols = np.random.randint(0, image.shape[1], num_salt)
    noisy[coords_salt_rows, coords_salt_cols] = 255

    num_pepper = int(np.ceil(amount * num_pixels * (1.0 - s_vs_p)))
    coords_pepper_rows = np.random.randint(0, image.shape[0], num_pepper)
    coords_pepper_cols = np.random.randint(0, image.shape[1], num_pepper)
    noisy[coords_pepper_rows, coords_pepper_cols] = 0

    return noisy


def apply_mean_filter(image: np.ndarray, kernel_size: int = 5) -> np.ndarray:
    kernel = np.ones((kernel_size, kernel_size), dtype=np.float32) / float(kernel_size
* kernel_size)
    return cv2.filter2D(src=image, ddepth=-1, kernel=kernel)


def apply_median_filter(image: np.ndarray, kernel_size: int = 5) -> np.ndarray:
    return cv2.medianBlur(src=image, ksize=kernel_size)
```

```python
def compute_psnr(reference: np.ndarray, target: np.ndarray) -> float:
    return cv2.PSNR(reference, target)


def main() -> None:
    image_path = 'lena.png'
    lena = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if lena is None:
        raise FileNotFoundError("The image 'lena.png' could not be found. Please place
it next to this script.")

    gaussian_sigma = 20.0  # adjust as desired
    sp_amount = 0.05       # percent of pixels to corrupt
    sp_ratio = 0.5         # 50% salt, 50% pepper

    lena_gauss = add_gaussian_noise(lena, mean=0.0, sigma=gaussian_sigma)
    lena_sp = add_salt_and_pepper_noise(lena, amount=sp_amount, s_vs_p=sp_ratio)

    kernel_size = 5
    lena_gauss_mean = apply_mean_filter(lena_gauss, kernel_size)
    lena_gauss_median = apply_median_filter(lena_gauss, kernel_size)

    lena_sp_mean = apply_mean_filter(lena_sp, kernel_size)
    lena_sp_median = apply_median_filter(lena_sp, kernel_size)

    print('PSNR vs Original (higher is better):')
    print(f'  Gaussian noisy:           {compute_psnr(lena, lena_gauss):.2f} dB')
    print(f'  Gaussian -> mean (5x5):   {compute_psnr(lena, lena_gauss_mean):.2f} dB')
    print(f'  Gaussian -> median (5x5): {compute_psnr(lena, lena_gauss_median):.2f}
dB')
    print(f'  S&P noisy:                {compute_psnr(lena, lena_sp):.2f} dB')
    print(f'  S&P -> mean (5x5):        {compute_psnr(lena, lena_sp_mean):.2f} dB')
    print(f'  S&P -> median (5x5):      {compute_psnr(lena, lena_sp_median):.2f} dB')

    cv2.imwrite('lena_gray.png', lena)
    cv2.imwrite('lena_gaussian.png', lena_gauss)
    cv2.imwrite('lena_gaussian_mean5.png', lena_gauss_mean)
    cv2.imwrite('lena_gaussian_median5.png', lena_gauss_median)
    cv2.imwrite('lena_sp.png', lena_sp)
    cv2.imwrite('lena_sp_mean5.png', lena_sp_mean)
    cv2.imwrite('lena_sp_median5.png', lena_sp_median)

    plt.figure(figsize=(16, 8))
    plt.suptitle('Image Denoising (5x5 Mean vs Median)', fontsize=16)

    plt.subplot(2, 4, 1)
    plt.imshow(lena, cmap='gray')
    plt.title('Original')
```

```
    plt.axis('off')

    plt.subplot(2, 4, 2)
    plt.imshow(lena_gauss, cmap='gray')
    plt.title('Gaussian noisy')
    plt.axis('off')

    plt.subplot(2, 4, 3)
    plt.imshow(lena_gauss_mean, cmap='gray')
    plt.title('Mean 5x5')
    plt.axis('off')

    plt.subplot(2, 4, 4)
    plt.imshow(lena_gauss_median, cmap='gray')
    plt.title('Median 5x5')
    plt.axis('off')

    plt.subplot(2, 4, 5)
    plt.imshow(lena, cmap='gray')
    plt.title('Original')
    plt.axis('off')

    plt.subplot(2, 4, 6)
    plt.imshow(lena_sp, cmap='gray')
    plt.title('S&P noisy')
    plt.axis('off')

    plt.subplot(2, 4, 7)
    plt.imshow(lena_sp_mean, cmap='gray')
    plt.title('Mean 5x5')
    plt.axis('off')

    plt.subplot(2, 4, 8)
    plt.imshow(lena_sp_median, cmap='gray')
    plt.title('Median 5x5')
    plt.axis('off')

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()


if __name__ == '__main__':
    main()
```

Question 3 screenshot:

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


def compute_sobel_gradients(image_gray: np.ndarray) -> tuple[np.ndarray, np.ndarray]:
    grad_x = cv2.Sobel(src=image_gray, ddepth=cv2.CV_32F, dx=1, dy=0, ksize=3)
    grad_y = cv2.Sobel(src=image_gray, ddepth=cv2.CV_32F, dx=0, dy=1, ksize=3)
    return grad_x, grad_y


def gradient_magnitude(grad_x: np.ndarray, grad_y: np.ndarray) -> np.ndarray:
    magnitude = cv2.magnitude(grad_x, grad_y)
    return magnitude


def normalize_to_uint8(image_float: np.ndarray) -> np.ndarray:
    min_val, max_val = float(np.min(image_float)), float(np.max(image_float))
    if max_val - min_val < 1e-6:
        return np.zeros_like(image_float, dtype=np.uint8)
    norm = (image_float - min_val) / (max_val - min_val)
    return (norm * 255.0).astype(np.uint8)


def threshold_edges(magnitude: np.ndarray, thresholds: list[int]) -> dict[int,
np.ndarray]:
    mag_uint8 = normalize_to_uint8(magnitude)
    edges = {}
    for t in thresholds:
        _, edge = cv2.threshold(mag_uint8, t, 255, cv2.THRESH_BINARY)
        edges[t] = edge
    return edges


def main() -> None:
    image_path = 'lena.png'
    image_gray = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    if image_gray is None:
        raise FileNotFoundError("The image 'lena.png' could not be found. Please place
it next to this script.")

    image_gray_blur = cv2.GaussianBlur(image_gray, ksize=(3, 3), sigmaX=0)
```

```
grad_x, grad_y = compute_sobel_gradients(image_gray_blur)

mag = gradient_magnitude(grad_x, grad_y)

cv2.imwrite('lena_gray.png', image_gray)
cv2.imwrite('lena_grad_x.png', normalize_to_uint8(grad_x))
cv2.imwrite('lena_grad_y.png', normalize_to_uint8(grad_y))
cv2.imwrite('lena_grad_mag.png', normalize_to_uint8(mag))

thresholds = [30, 60, 90, 120, 150]
edge_maps = threshold_edges(mag, thresholds)
for t, edge in edge_maps.items():
    cv2.imwrite(f'lena_edges_T{t}.png', edge)

plt.figure(figsize=(14, 8))
plt.suptitle('Image Gradients and Edge Maps', fontsize=16)

plt.subplot(2, 3, 1)
plt.imshow(image_gray, cmap='gray')
plt.title('Input (Gray)')
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(normalize_to_uint8(grad_x), cmap='gray')
plt.title('Sobel Gx')
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(normalize_to_uint8(grad_y), cmap='gray')
plt.title('Sobel Gy')
plt.axis('off')

plt.subplot(2, 3, 4)
plt.imshow(normalize_to_uint8(mag), cmap='gray')
plt.title('Gradient Magnitude')
plt.axis('off')

t_values = thresholds[:2]
plt.subplot(2, 3, 5)
plt.imshow(edge_maps[t_values[0]], cmap='gray')
plt.title(f'Edges T={t_values[0]}')
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(edge_maps[t_values[1]], cmap='gray')
plt.title(f'Edges T={t_values[1]}')
plt.axis('off')
```

```
    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()


if __name__ == '__main__':
    main()
```