Lab 3: Image Processing

Name: Gavin Abrigo Student ID: 205260732

1. Histogram equalization

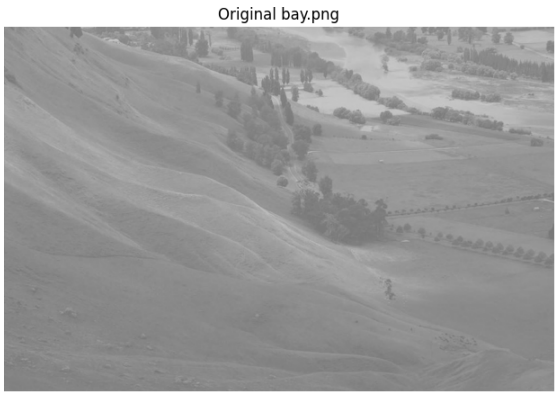
Compute and visualize histogram and cumulative distance function (CDF) of an input gray-scale image

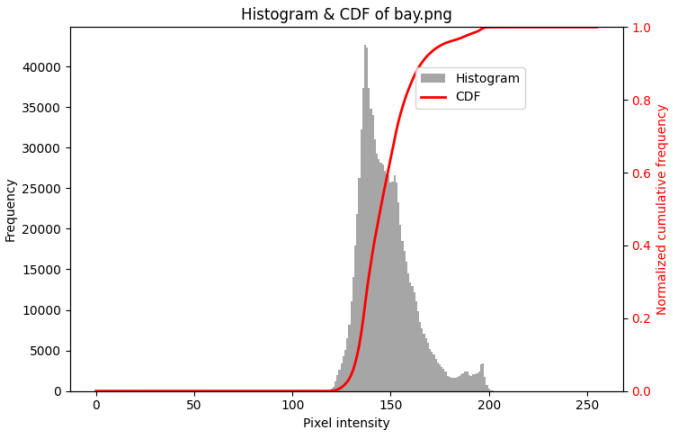
```
# Useful code snippet:
# read an image as gray-scale image
import cv2
import numpy as np
import matplotlib.pyplot as plt
img = cv2.imread('./bay.png', cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError('Could not load ./bay.png')

# compute image histogram
hist, bins = np.histogram(img.flatten(), 256, [0, 256])

# compute cumulative density function
cdf = hist.cumsum()
# normalize cdf
cdf_normalized = cdf / cdf.max()

# Visualize image with histogram and CDF overlay
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Show original image
axes[0].imshow(img, cmap='gray', vmin=0, vmax=255)
axes[0].set_title('Original bay.png')
axes[0].axis('off')

# Plot histogram and CDF together
ax1 = axes[1]
ax2 = ax1.twinx()

# Histogram as bar chart
```

```
ax1.bar(range(256), hist, width=1.0, color='gray', alpha=0.7,
label='Histogram')
ax1.set_xlabel('Pixel intensity')
ax1.set_ylabel('Frequency', color='black')
ax1.tick_params(axis='y', labelcolor='black')

# CDF as line plot
ax2.plot(cdf_normalized, color='red', linewidth=2, label='CDF')
ax2.set_ylabel('Normalized cumulative frequency', color='red')
ax2.tick_params(axis='y', labelcolor='red')
ax2.set_ylim([0, 1])

axes[1].set_title('Histogram & CDF of bay.png')
fig.legend(loc='upper right', bbox_to_anchor=(0.88, 0.85))

plt.tight_layout()
plt.show()

# histogram equalization using CDF
cdf_masked = np.ma.masked_equal(cdf, 0)
cdf_scaled = (cdf_masked - cdf_masked.min()) * 255 / (cdf_masked.max() -
cdf_masked.min())
cdf_scaled = np.ma.filled(cdf_scaled, 0).astype(np.uint8)

eq_img = cdf_scaled[img]

# Compute histogram and CDF of equalized image
eq_hist, _ = np.histogram(eq_img.flatten(), 256, [0, 256])
eq_cdf = eq_hist.cumsum()
eq_cdf_normalized = eq_cdf / eq_cdf.max()

# Show original vs equalized with their histograms and CDFs
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Left: Equalized image
axes[0].imshow(eq_img, cmap='gray', vmin=0, vmax=255)
axes[0].set_title('Histogram Equalized bay.png')
axes[0].axis('off')

# Right: Histogram and CDF of equalized image
ax1 = axes[1]
```

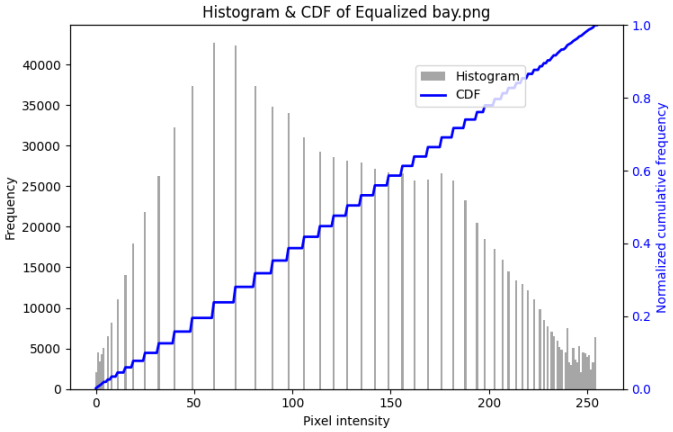
```
ax2 = ax1.twinx()

# Histogram
ax1.bar(range(256), eq_hist, width=1.0, color='gray', alpha=0.7,
label='Histogram')
ax1.set_xlabel('Pixel intensity')
ax1.set_ylabel('Frequency', color='black')
ax1.tick_params(axis='y', labelcolor='black')

# CDF
ax2.plot(eq_cdf_normalized, color='blue', linewidth=2, label='CDF')
ax2.set_ylabel('Normalized cumulative frequency', color='blue')
ax2.tick_params(axis='y', labelcolor='blue')
ax2.set_ylim([0, 1])

axes[1].set_title('Histogram & CDF of Equalized bay.png')
fig.legend(loc='upper right', bbox_to_anchor=(0.88, 0.85))

plt.tight_layout()
plt.show()
```

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

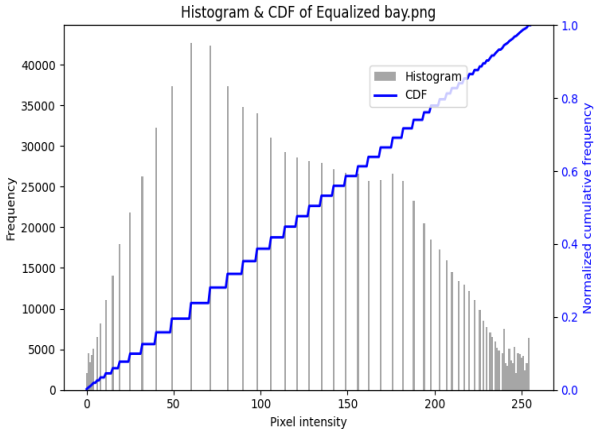




2. Apply histogram equalization using obtained CDF on the input image



3. Compute and visualize histogram of output image



2. Image denoising

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

```
img = cv2.imread("./lena.png")
if img is None:
    raise FileNotFoundError("Could not load ./lena.png")

lena_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

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

```
def add_gaussian_noise(
    image: np.ndarray,
    mean: float = 0.0,
    std: float = 15.0,
```

```
    rng: np.random.Generator | None = None,
) -> np.ndarray:
    """Add Gaussian noise to a grayscale image."""
    if rng is None:
        rng = np.random.default_rng()
    noise = rng.normal(mean, std, size=image.shape)
    noisy = image.astype(np.float32) + noise
    return np.clip(noisy, 0, 255).astype(np.uint8)


def add_salt_pepper_noise(
    image: np.ndarray,
    amount: float = 0.04,
    salt_vs_pepper: float = 0.5,
    rng: np.random.Generator | None = None,
) -> np.ndarray:
    """Add Salt & Pepper noise to a grayscale image."""
    if rng is None:
        rng = np.random.default_rng()

    noisy = image.copy()
    total_pixels = image.size
    num_salt = int(total_pixels * amount * salt_vs_pepper)
    num_pepper = int(total_pixels * amount * (1.0 - salt_vs_pepper))

    salt_coords = (
        rng.integers(0, image.shape[0], num_salt),
        rng.integers(0, image.shape[1], num_salt),
    )
    pepper_coords = (
        rng.integers(0, image.shape[0], num_pepper),
        rng.integers(0, image.shape[1], num_pepper),
    )

    noisy[salt_coords] = 255
    noisy[pepper_coords] = 0
    return noisy
```

3. Implement mean and median filtering in 5x5 windows

```
gaussian_noisy = add_gaussian_noise(lena_gray, std=20.0, rng=rng)
# Gaussian noise denoised with Gaussian blur
# 5x5 kernel keeps parity with other operations
gaussian_denoised = cv2.GaussianBlur(gaussian_noisy, (5, 5),
sigmaX=0)

salt_pepper_noisy = add_salt_pepper_noise(lena_gray, amount=0.03,
salt_vs_pepper=0.6, rng=rng)
# Salt & pepper noise denoised with median blur (distinct method)
salt_pepper_denoised = cv2.medianBlur(salt_pepper_noisy, 5)
```

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

```
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
fig.suptitle("Noise addition and denoising on lena.png", fontsize=14)

images = [
    (lena_gray, "Original"),
    (gaussian_noisy, "Gaussian noise"),
    (gaussian_denoised, "Gaussian blur (5x5)"),
    (lena_gray, "Original"),
    (salt_pepper_noisy, "Salt & pepper noise"),
    (salt_pepper_denoised, "Median blur (5x5)"),
]
```

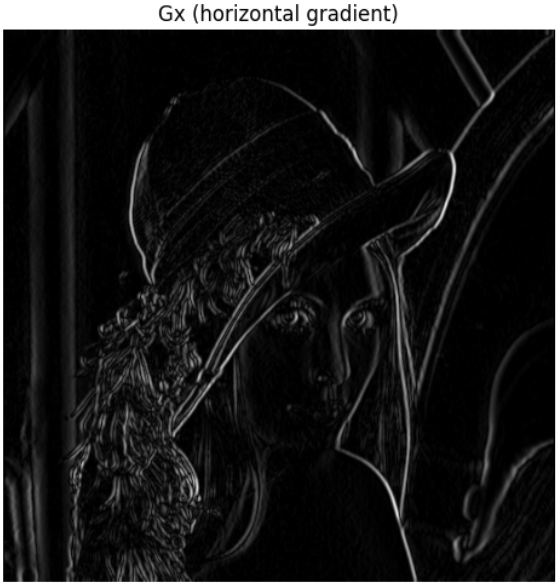
3. Image gradient

1. Compute image gradient in x and y direction respectively

```
img = cv2.imread("./lena.png")
lena_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# compute gradient
gx = cv2.Sobel(lena_gray, cv2.CV_64F, 1, 0, ksize=5)
gy = cv2.Sobel(lena_gray, cv2.CV_64F, 0, 1, ksize=5)
```

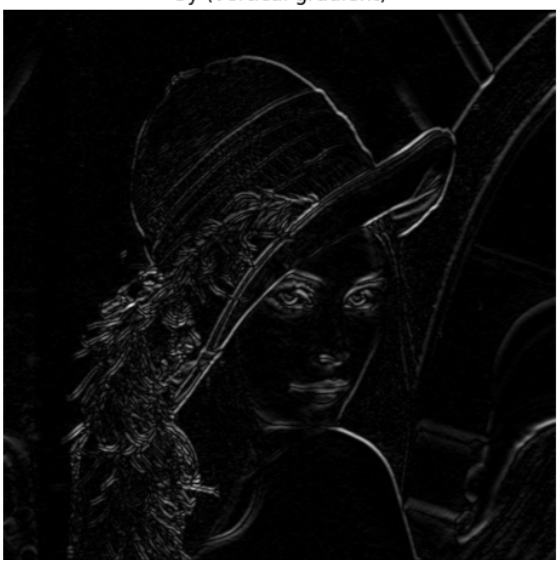
Original lena.png

Gx (horizontal gradient)



Original lena.png

Gy (vertical gradient)



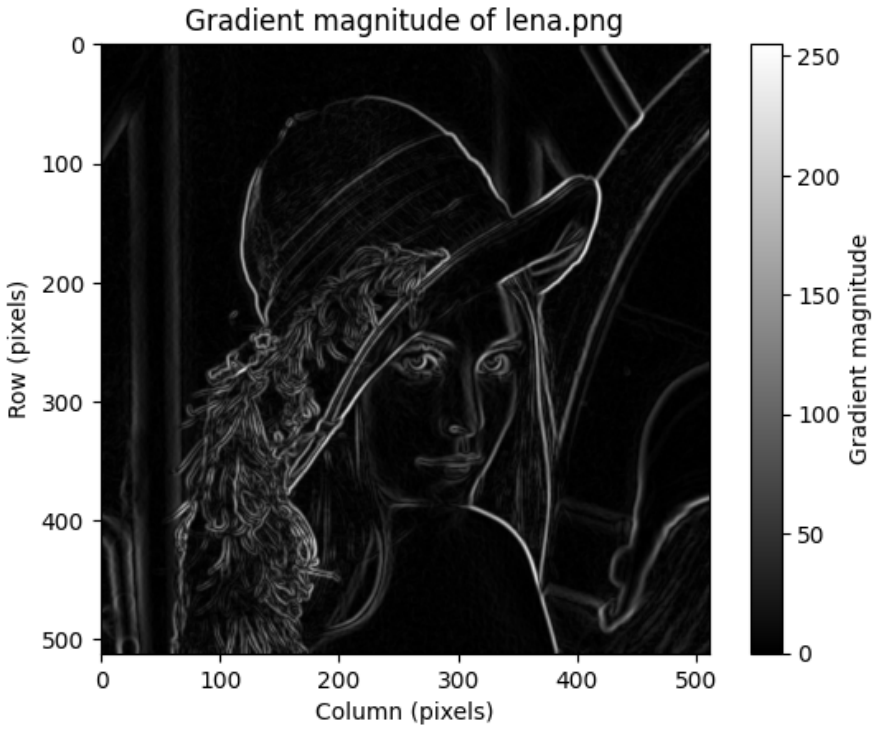
2. Read the input image and convert to a grayscale image

```
img = cv2.imread("./lena.png")
lena_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

Original lena.png

3. Compute magnitude of image gradient for each pixel

```
4. magnitude = cv2.magnitude(gx, gy)


magnitude_norm = cv2.normalize(magnitude, None, 0, 255,
cv2.NORM_MINMAX)
magnitude_uint8 = magnitude_norm.astype(np.uint8)
```



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

```
plt.imshow(magnitude_uint8, cmap="gray")
plt.title("Gradient magnitude of lena.png")
plt.xlabel("Column (pixels)")
plt.ylabel("Row (pixels)")
plt.colorbar(label="Gradient magnitude")
plt.show()
```