cat1

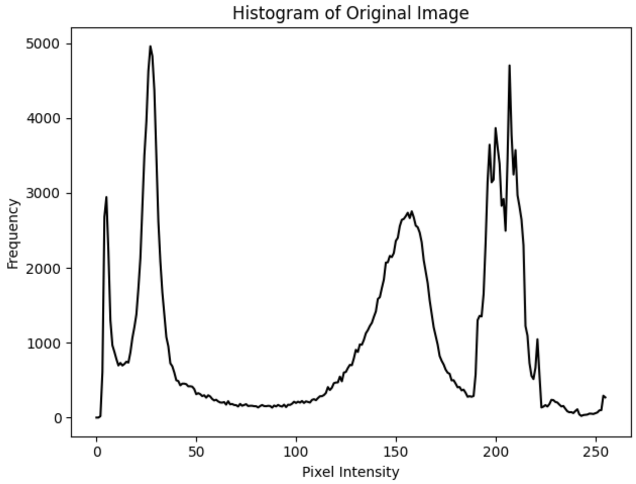
April 8, 2025

QUESTION 1

[1]:
```
%%capture --no-stderr

%pip install --quiet numpy matplotlib scikit-image opencv-python
```

[ ]:
```
#(i)Load a sample grayscale image in a Jupyter Notebook and analyze its␣
↪histogram. Display both the image and its histogram side by side.

import numpy as np
import matplotlib.pyplot as plt
from skimage import data, exposure

# Load a sample grayscale image (the "camera" image)
image = data.camera()

# Compute the histogram
hist, bins = np.histogram(image.flatten(), bins=256, range=[0,256])

# Display image and histogram side by side
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Grayscale Image')
axes[0].axis('off')

axes[1].plot(bins[:-1], hist, color='black')
axes[1].set_title('Histogram of Original Image')
axes[1].set_xlabel('Pixel Intensity')
axes[1].set_ylabel('Frequency')

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

Original Grayscale Image

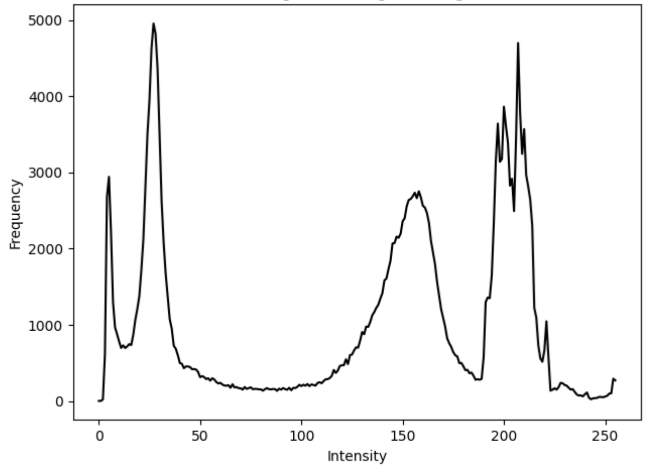


[2]:
```
# (ii) Modify the contrast of the image (increase and decrease it) and generate
↪the histograms for each version. Compare the histograms and explain your
↪observations.

# Increase contrast using contrast stretching
p2, p98 = np.percentile(image, (2, 98))
image_high_contrast = exposure.rescale_intensity(image, in_range=(p2, p98))

# Decrease contrast by compressing the intensity range e.g. by mapping the
↪intensities to a smaller range
image_low_contrast = exposure.rescale_intensity(image, in_range=(0, 255),
↪out_range=(100, 150))

# Compute histograms for the modified images
hist_high, bins_high = np.histogram(image_high_contrast.flatten(), bins=256,
↪range=[0,256])
hist_low, bins_low = np.histogram(image_low_contrast.flatten(), bins=256,
↪range=[0,256])

# Display images and histograms
fig, axes = plt.subplots(2, 2, figsize=(12, 10))

# Original image and histogram for reference
axes[0,0].imshow(image, cmap='gray')
axes[0,0].set_title('Original Image')
axes[0,0].axis('off')
axes[0,1].plot(bins[:-1], hist, color='black')
axes[0,1].set_title('Histogram of Original Image')
axes[0,1].set_xlabel('Intensity')
axes[0,1].set_ylabel('Frequency')
```

```python
# High contrast image and its histogram
axes[1,0].imshow(image_high_contrast, cmap='gray')
axes[1,0].set_title('High Contrast Image')
axes[1,0].axis('off')
axes[1,1].plot(bins_high[:-1], hist_high, color='blue')
axes[1,1].set_title('Histogram of High Contrast Image')
axes[1,1].set_xlabel('Intensity')
axes[1,1].set_ylabel('Frequency')

plt.tight_layout()
plt.show()

# For the low contrast image, we can plot separately:
fig, axes = plt.subplots(1, 2, figsize=(12, 5))
axes[0].imshow(image_low_contrast, cmap='gray')
axes[0].set_title('Low Contrast Image')
axes[0].axis('off')
axes[1].plot(bins_low[:-1], hist_low, color='red')
axes[1].set_title('Histogram of Low Contrast Image')
axes[1].set_xlabel('Intensity')
axes[1].set_ylabel('Frequency')

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

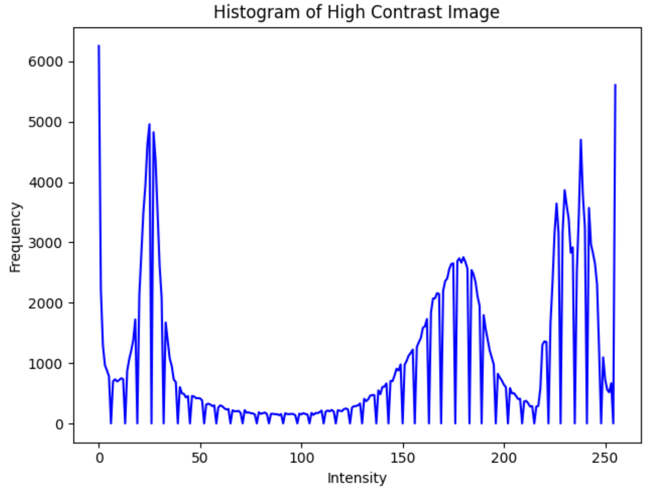
Original Image

Histogram of Original Image



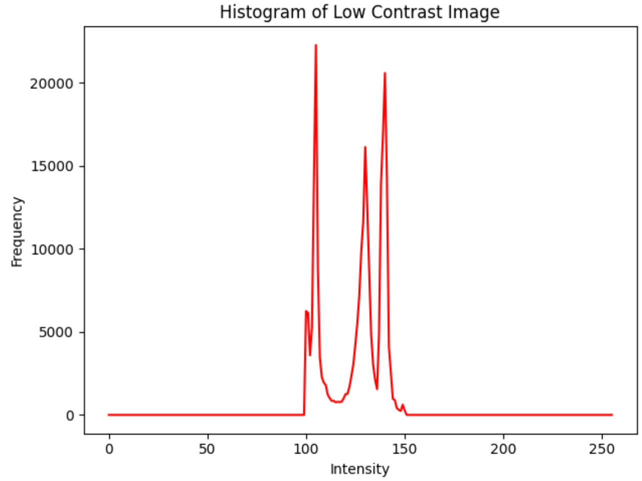
High Contrast Image

Histogram of High Contrast Image



Low Contrast Image

Histogram of Low Contrast Image



OBSERVATIONS

High Contrast: The histogram of the high contrast image tends to be more “spread out” across the intensity range (i.e. a wider spread of pixel values) compared to the original one.

Low Contrast: The low contrast image’s histogram is compressed into a narrow band (in our example, between intensity values 100 and 150). The pixel values are concentrated over a smaller range, making the image appear “washed out” or flat.

Question 3

(i)

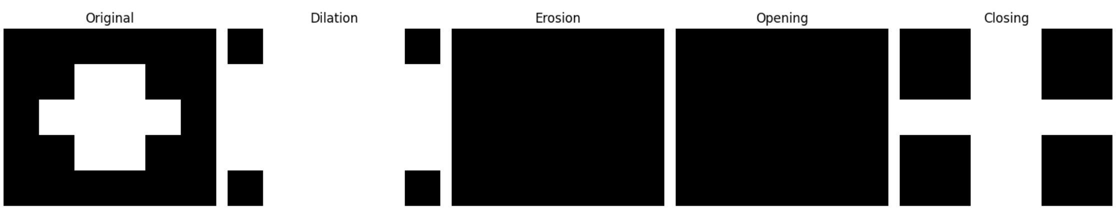
Morphological operations are non-linear transformations applied to binary or grayscale images that process structures based on their shape. These operations use a probe called a structuring element (SE) to inspect and modify the geometrical structure of an image. Useful in shape analysis, noise removal, image segmentation, and object extraction.

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Sample binary image
img = np.array([[0, 0, 0, 0, 0, 0],
                [0, 0, 1, 1, 0, 0],
                [0, 1, 1, 1, 1, 0],
                [0, 0, 1, 1, 0, 0],
                [0, 0, 0, 0, 0, 0]], dtype=np.uint8) * 255

# Structuring Element (3x3 square), a rectangular SE
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
```

[4]:
```
# a) Dilation

# Effect: Expands white regions (foreground), fills small holes, bridges gaps.

dilated = cv2.dilate(img, kernel)

# b) Erosion

# Effect: Shrinks white regions (foreground), removes small objects or noise.

eroded = cv2.erode(img, kernel)

# c) Opening

# Effect: Smoothes contours, removes small foreground noise.

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)
```

```
# d) Closing

# Effect: Closes small holes, connects narrow breaks.

closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)
```

[5]:
```
#Visualisation

titles = ['Original', 'Dilation', 'Erosion', 'Opening', 'Closing']
images = [img, dilated, eroded, opening, closing]

plt.figure(figsize=(15, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```



[7]:
```
# (iii) Creating custom structuring elements and applying morphological␣
↪operations to images then compare the results.

# Creating custom structuring elements (cross SE)
custom_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (3, 3))

# Applying morphological operations
dilated_custom = cv2.dilate(img, custom_kernel)
eroded_custom = cv2.erode(img, custom_kernel)
opening_custom = cv2.morphologyEx(img, cv2.MORPH_OPEN, custom_kernel)
closing_custom = cv2.morphologyEx(img, cv2.MORPH_CLOSE, custom_kernel)


#Visualisation

titles = ['Original', 'Dilation cross', 'Erosion cross', 'Opening cross',␣
↪'Closing cross']
images = [img, dilated_custom, eroded_custom, opening_custom, closing_custom]
```

```
plt.figure(figsize=(15, 3))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(images[i], cmap='gray')
    plt.title(titles[i])
    plt.axis('off')
plt.tight_layout()
plt.show()
```