Lab Report-07

Course title: Digital Image Processing Laboratory

Course code: CSE-406

4th Year 1st Semester Examination 2023

Date of Submission: 20 October 2024

Submitted to-

Dr. Morium Akter
Professor

Dr. Md. Golam Moazzam
Professor

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka-1342

Sl	Class Roll	Exam Roll	Name
01	408	202220	Sudipta Singha

Department of Computer Science and Engineering
Jahangirnagar University
Savar, Dhaka, Bangladesh

Lab Report Title

Image Dilation with OpenCV

Introduction

This lab explores image dilation, a fundamental operation in morphological image processing. Dilation expands the boundaries of objects in an image, effectively enlarging bright regions and filling in small holes. We'll use the OpenCV library in Python to perform dilation and visualize its effects

Python code

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

# Load the image in grayscale
image = cv2.imread('input.jpg', cv2.IMREAD_GRAYSCALE)

# Create a structuring element (kernel)
kernel = np.ones((5,5), np.uint8)  # 5x5 square kernel

# Perform dilation
dilated_image = cv2.dilate(image, kernel, iterations=1)

# Display the original and dilated images using Matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(dilated_image, cmap='gray')
plt.title('Dilated Image')
plt.axis('off')

plt.tight_layout()
plt.show()

# Save the dilated image
plt.imsave('dilated_image.jpg', dilated_image, cmap='gray')
```

Input

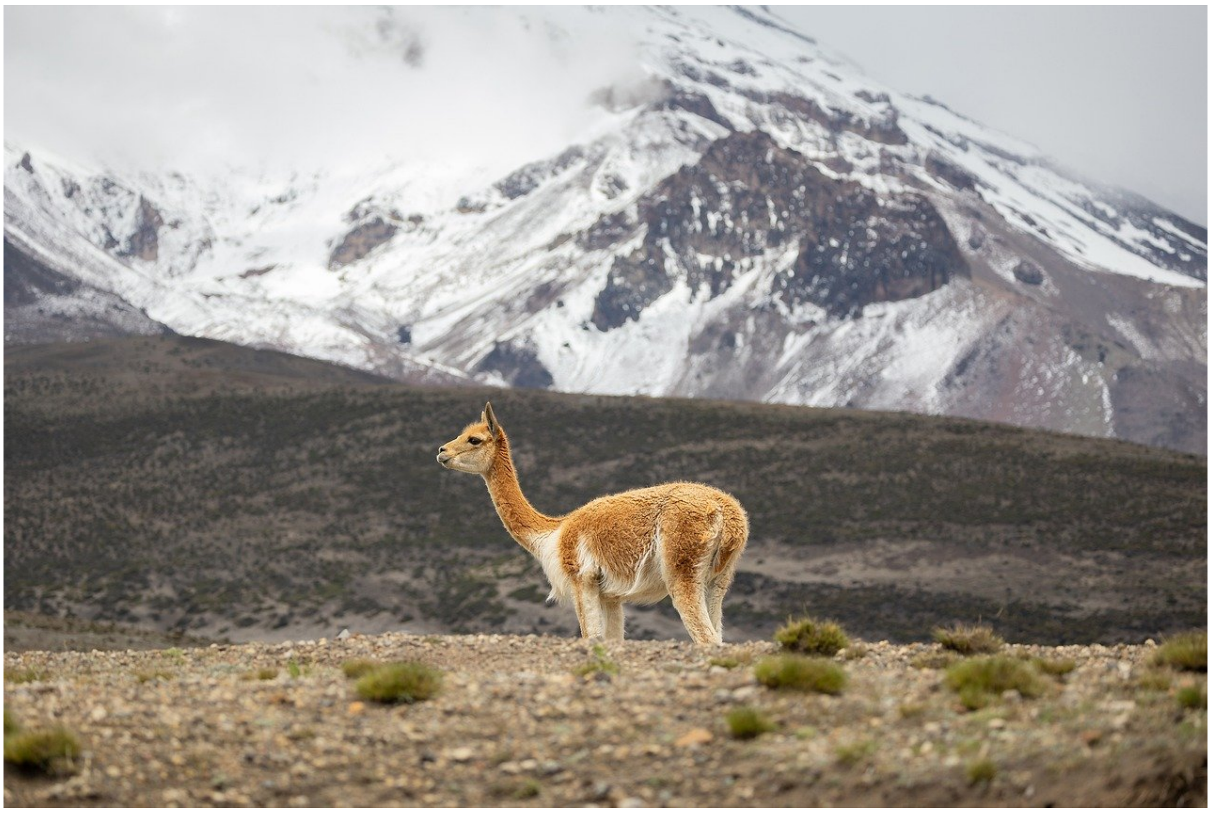

Figure 1:

Output

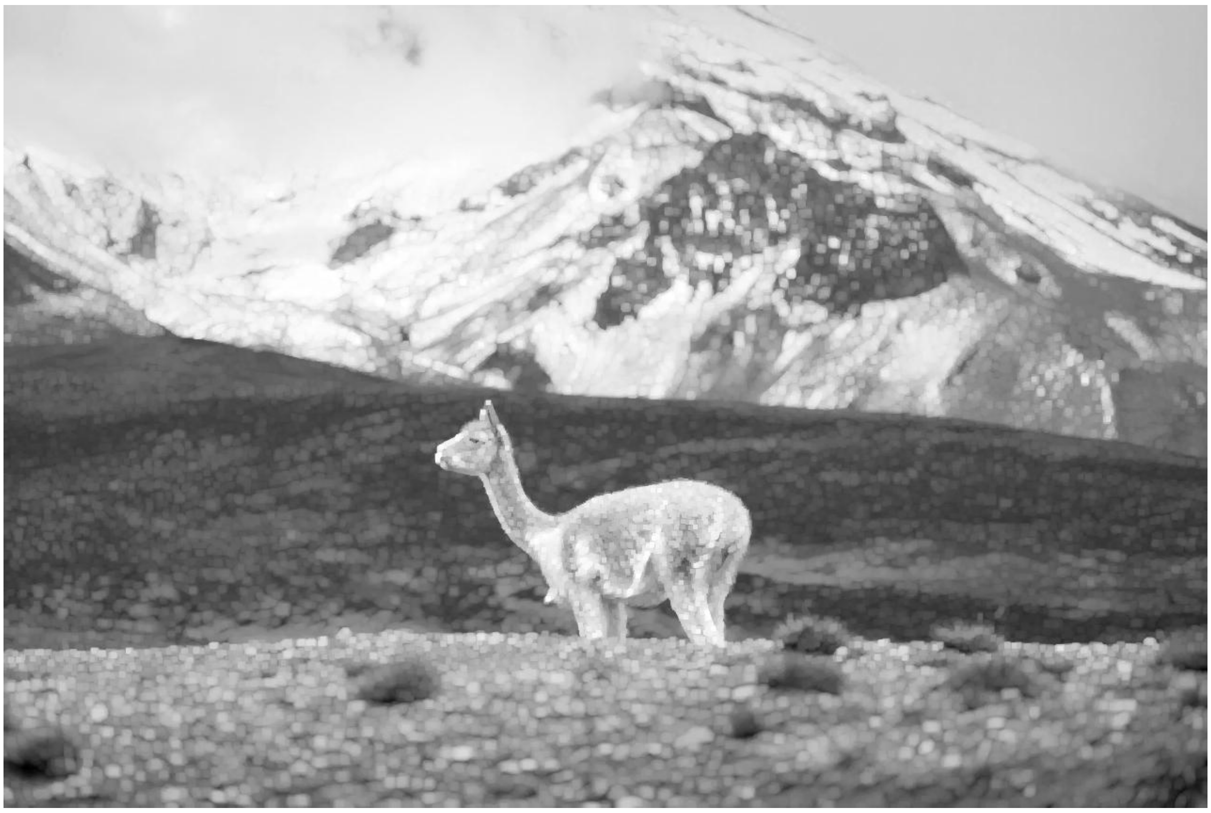

Figure 2:

Lab Report Title

Image Erosion with OpenCV

Introduction

This lab focuses on image erosion, another fundamental operation in morphological image processing. Erosion shrinks the boundaries of objects in an image, effectively making bright regions smaller and removing small protrusions or noise. We'll again utilize the OpenCV library in Python to perform erosion and visualize its effects.

Python code

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

# Load the image in grayscale
image = cv2.imread('input.jpg', cv2.IMREAD_GRAYSCALE)

# Create a structuring element (kernel)
kernel = np.ones((5,5), np.uint8)  # 5x5 square kernel

# Perform erosion
eroded_image = cv2.erode(image, kernel, iterations=1)

# Display the original and eroded images using Matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(eroded_image, cmap='gray')
plt.title('Eroded Image')
plt.axis('off')

plt.tight_layout()
plt.show()

# Save the eroded image
plt.imsave('eroded_image.jpg', eroded_image, cmap='gray')
```

Input

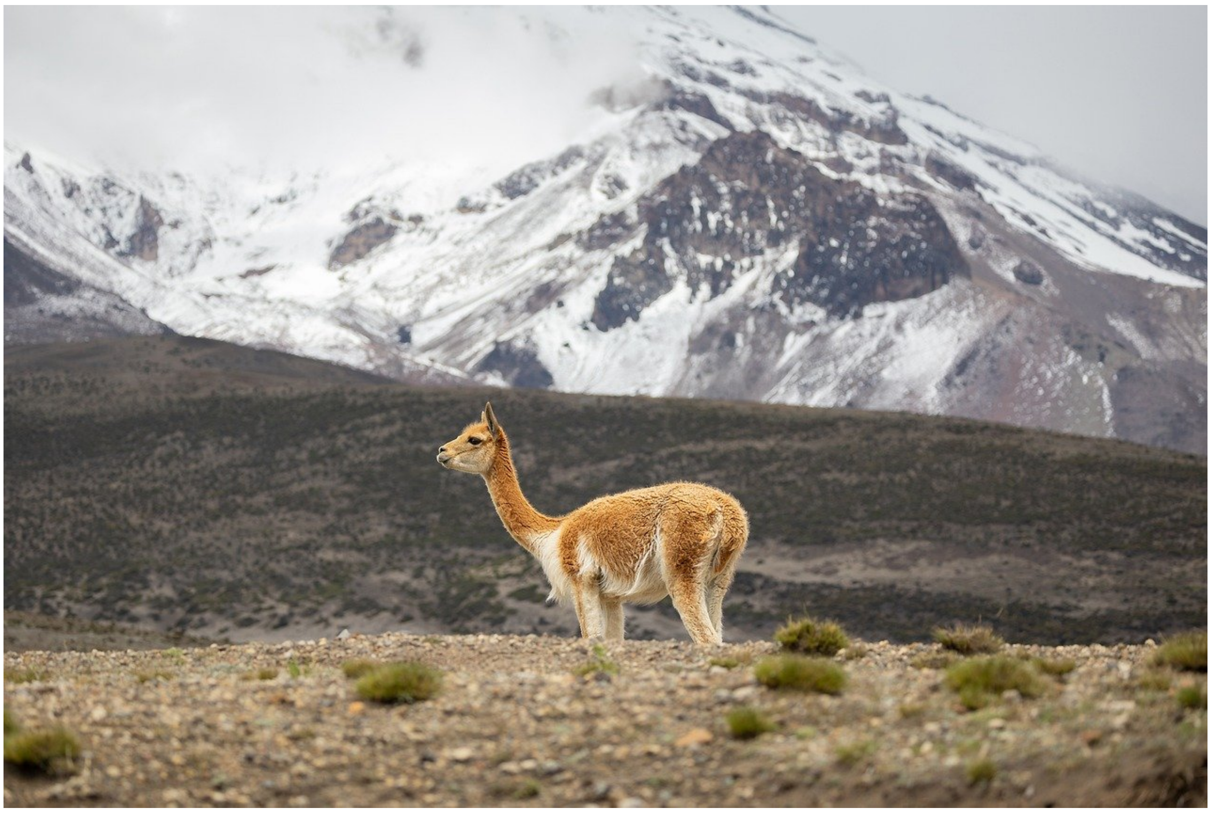

Figure 3:

Output

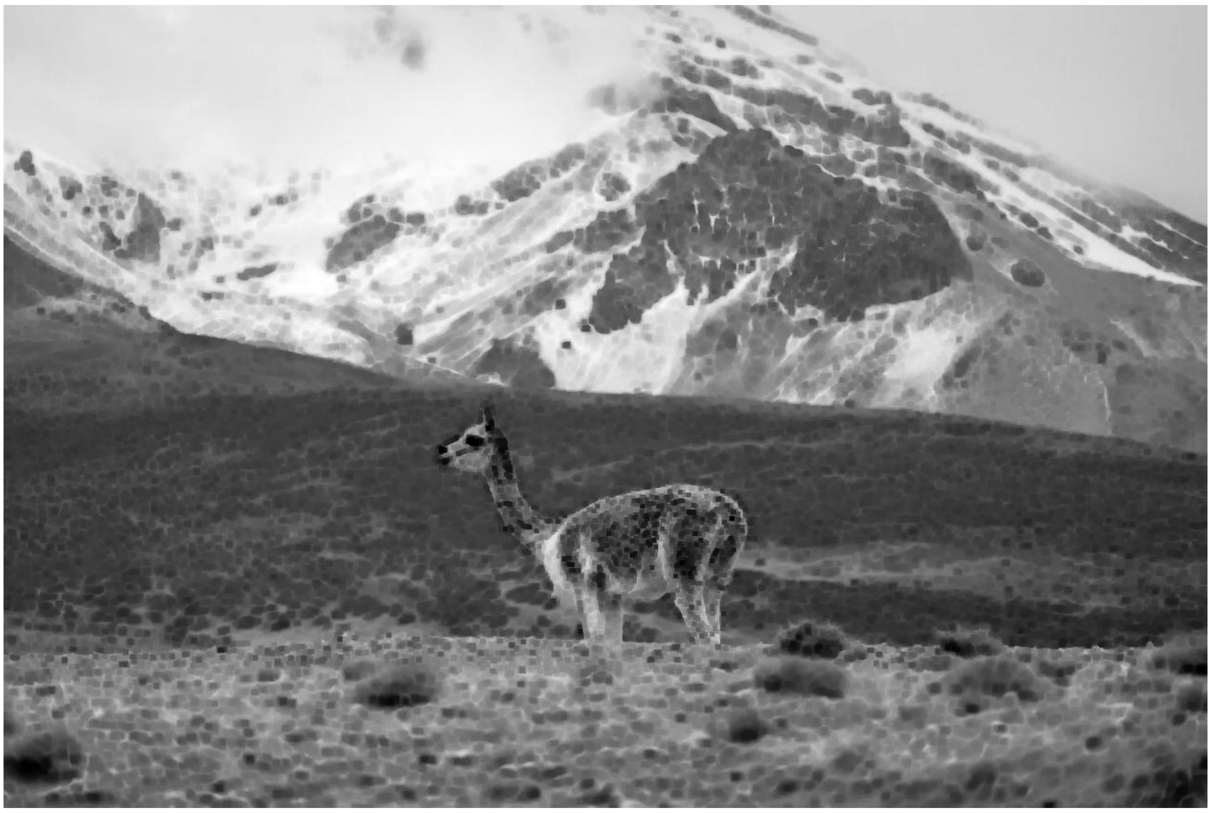

Figure 4:

Lab Report Title

Illumination Correction with Homomorphic Filtering

Introduction

This lab explores homomorphic filtering, a technique used to enhance images that suffer from uneven illumination or poor contrast. Homomorphic filtering operates in the frequency domain to separate the illumination and reflectance components of an image. By manipulating these components, we can improve the image's overall appearance. This lab will demonstrate how to apply homomorphic filtering using OpenCV and NumPy

Python code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image = cv2.imread('input.jpg')

# Convert the image to grayscale
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply logarithmic transformation
log_image = np.log1p(np.array(gray_image, dtype="float") /
   ↪ 255)

# Compute the 2D DFT
dft = cv2.dft(np.float32(log_image),
   ↪ flags=cv2.DFT_COMPLEX_OUTPUT)
dft_shift = np.fft.fftshift(dft)

# Define the homomorphic filter parameters
rows, cols = gray_image.shape
crow, ccol = rows // 2, cols // 2
gamma_h = 1.5
gamma_l = 0.5
c = 1
d0 = 20

# Create a Gaussian filter
gaussian_filter = np.zeros((rows, cols, 2), np.float32)
for i in range(rows):
    for j in range(cols):
        distance = np.sqrt((i - crow) ** 2 + (j - ccol) ** 2)
        gaussian_filter[i, j] = (gamma_h - gamma_l) * (1 -
          ↪ np.exp(-c * (distance ** 2) / (d0 ** 2))) +
          ↪ gamma_l

# Apply the filter to the shifted DFT
filtered_dft = dft_shift * gaussian_filter

# Shift back the zero-frequency component
filtered_dft_shift = np.fft.ifftshift(filtered_dft)

# Compute the inverse DFT
filtered_image = cv2.idft(filtered_dft_shift)
filtered_image = cv2.magnitude(filtered_image[:, :, 0],
  ↪ filtered_image[:, :, 1])

# Apply exponential transformation
filtered_image = np.exp(filtered_image) - 1
filtered_image = cv2.normalize(filtered_image, None,
  ↪ alpha=0, beta=255, norm_type=cv2.NORM_MINMAX,
  ↪ dtype=cv2.CV_8U)

# Display the original and filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(gray_image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(filtered_image, cmap='gray')
plt.title('Homomorphic Filtered Image')
plt.axis('off')

plt.tight_layout()
plt.show()

# Save the filtered image
plt.imsave('homomorphic_filtered_image.jpg', filtered_image,
  ↪ cmap='gray')
```

Input

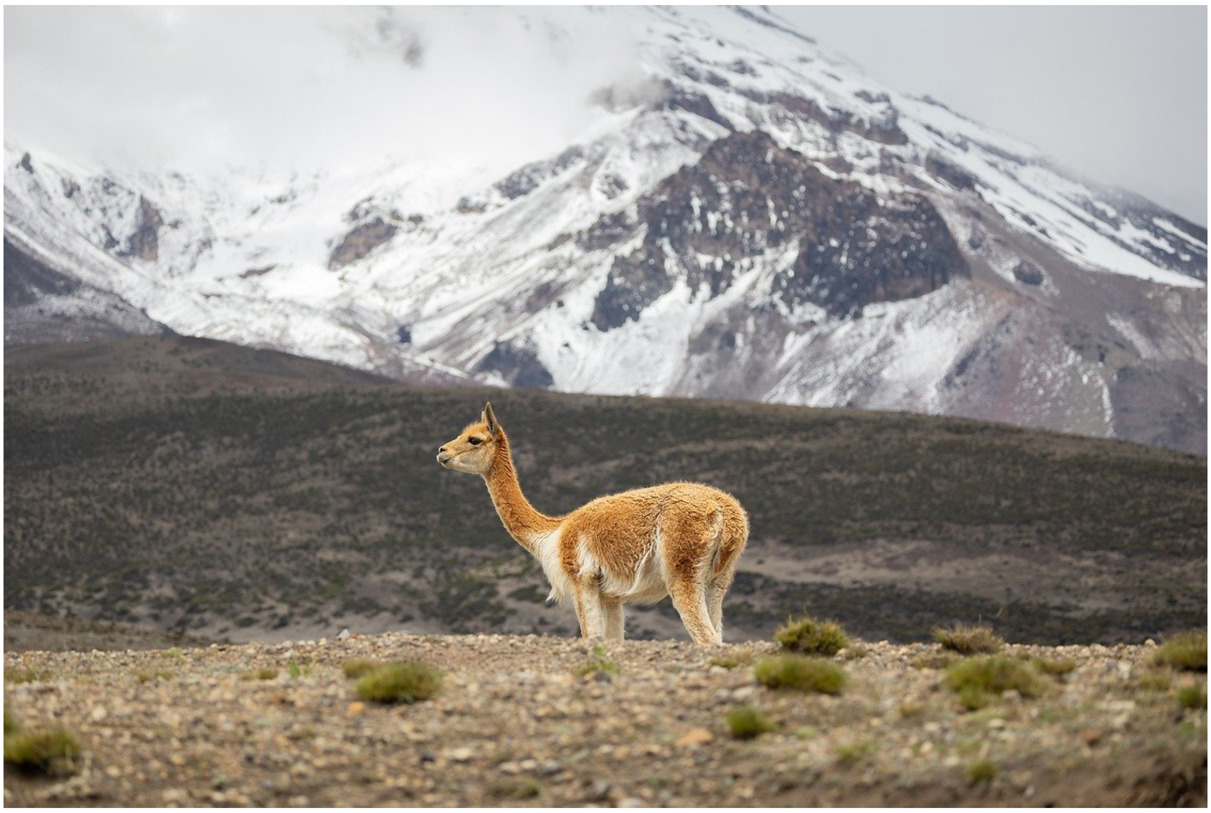

Figure 5:

Output

Figure 6:

Lab Report Title

Exploring the Frequency Domain: FFT of an Image

Introduction

This lab delves into the frequency domain representation of an image using the Fast Fourier Transform (FFT). The FFT decomposes an image into its constituent frequencies, revealing information about patterns, textures, and edges that might not be readily apparent in the spatial domain. We'll use OpenCV and NumPy to compute the FFT and visualize the frequency spectrum

Python code

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image in grayscale
image = cv2.imread('input.jpg', cv2.IMREAD_GRAYSCALE)

# Compute the 2D DFT
dft = cv2.dft(np.float32(image),
    ↪ flags=cv2.DFT_COMPLEX_OUTPUT)

# Shift the zero-frequency component to the center of the
    ↪ spectrum
dft_shift = np.fft.fftshift(dft)

# Compute the magnitude spectrum (logarithmic scale for
    ↪ better visualization)
magnitude_spectrum = 20 * np.log(cv2.magnitude(dft_shift[:,
    ↪ :, 0], dft_shift[:, :, 1]))

# Display the original and magnitude spectrum images using
    ↪ Matplotlib
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(magnitude_spectrum, cmap='gray')
plt.title('Magnitude Spectrum')
plt.axis('off')
plt.tight_layout()
plt.show()

# Save the magnitude spectrum image
plt.imsave('magnitude_spectrum.jpg', magnitude_spectrum,
    ↪ cmap='gray')
```

Input

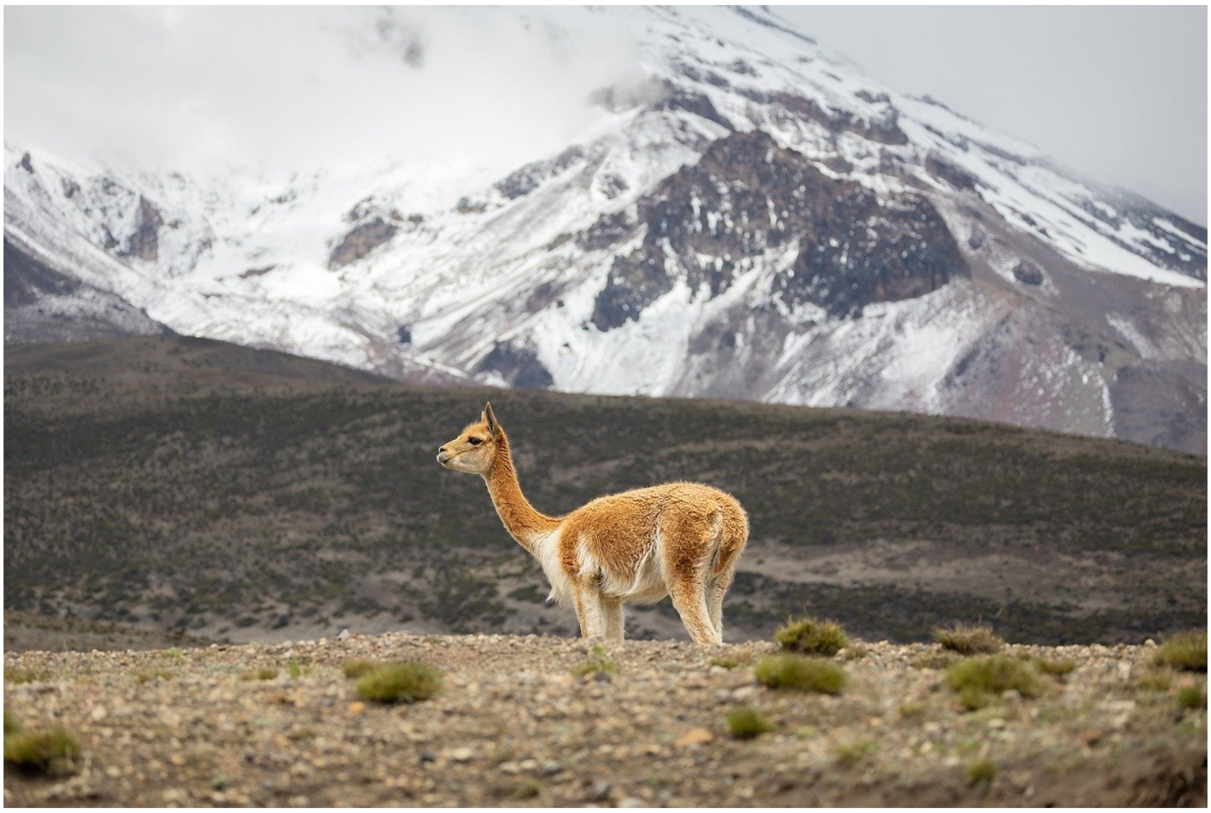

Figure 7:

Output

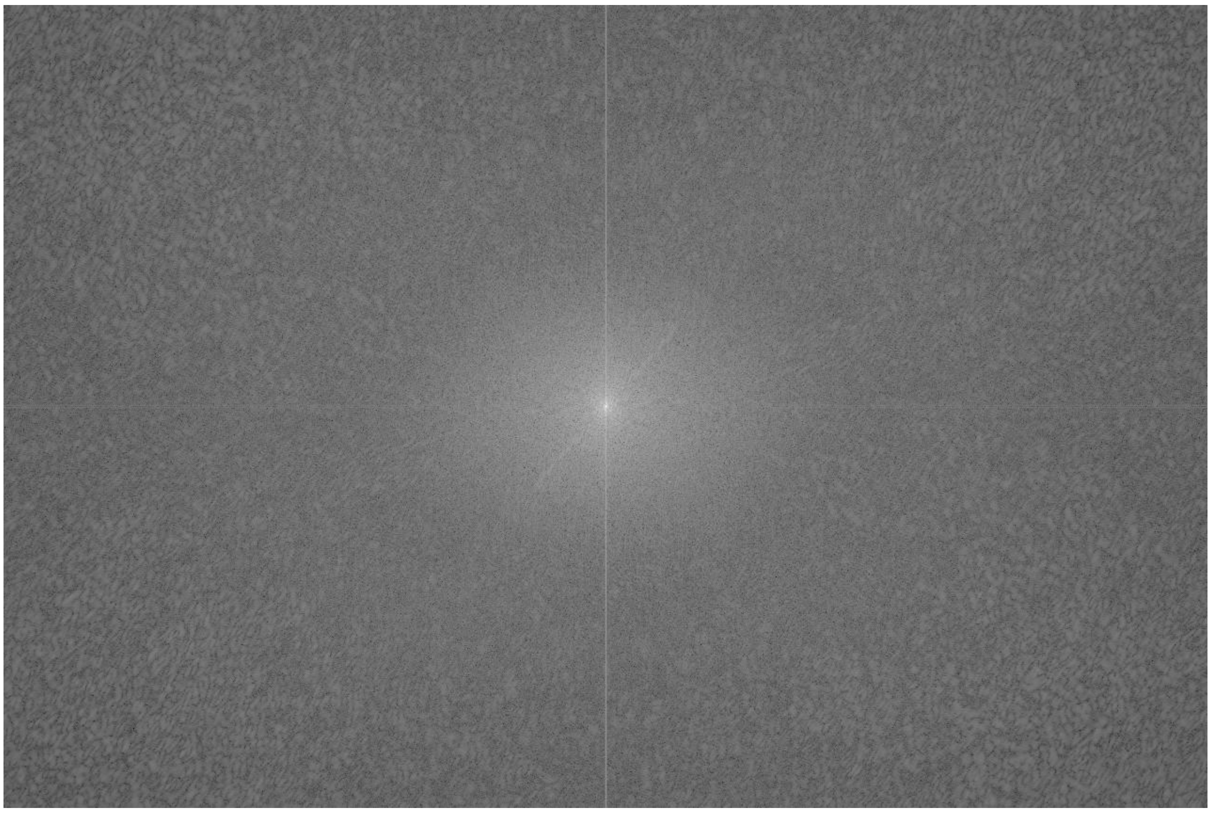

Figure 8:

Lab Report Title

Opening and Closing of an Image

Introduction

This lab provides a fundamental understanding of the opening and closing operations in morphological image processing. By experimenting with the code and exploring different parameters, you can gain a deeper understanding of how these techniques can be used for various image analysis and manipulation tasks.

Python code

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

# Load the image in grayscale
image = cv2.imread('input.jpg', cv2.IMREAD_GRAYSCALE)

# Create a structuring element (kernel)
kernel = np.ones((5,5), np.uint8)  # 5x5 square kernel

# Perform opening (erosion followed by dilation)
opening = cv2.morphologyEx(image, cv2.MORPH_OPEN, kernel)

# Perform closing (dilation followed by erosion)
closing = cv2.morphologyEx(image, cv2.MORPH_CLOSE, kernel)

# Display the original, opened, and closed images
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 3, 2)
plt.imshow(opening, cmap='gray')
plt.title('Opening')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(closing, cmap='gray')
plt.title('Closing')
plt.axis('off')
plt.tight_layout()
plt.show()

# Save the opened and closed images
plt.imsave('opening.jpg', opening, cmap='gray')
plt.imsave('closing.jpg', closing, cmap='gray')
```

Input

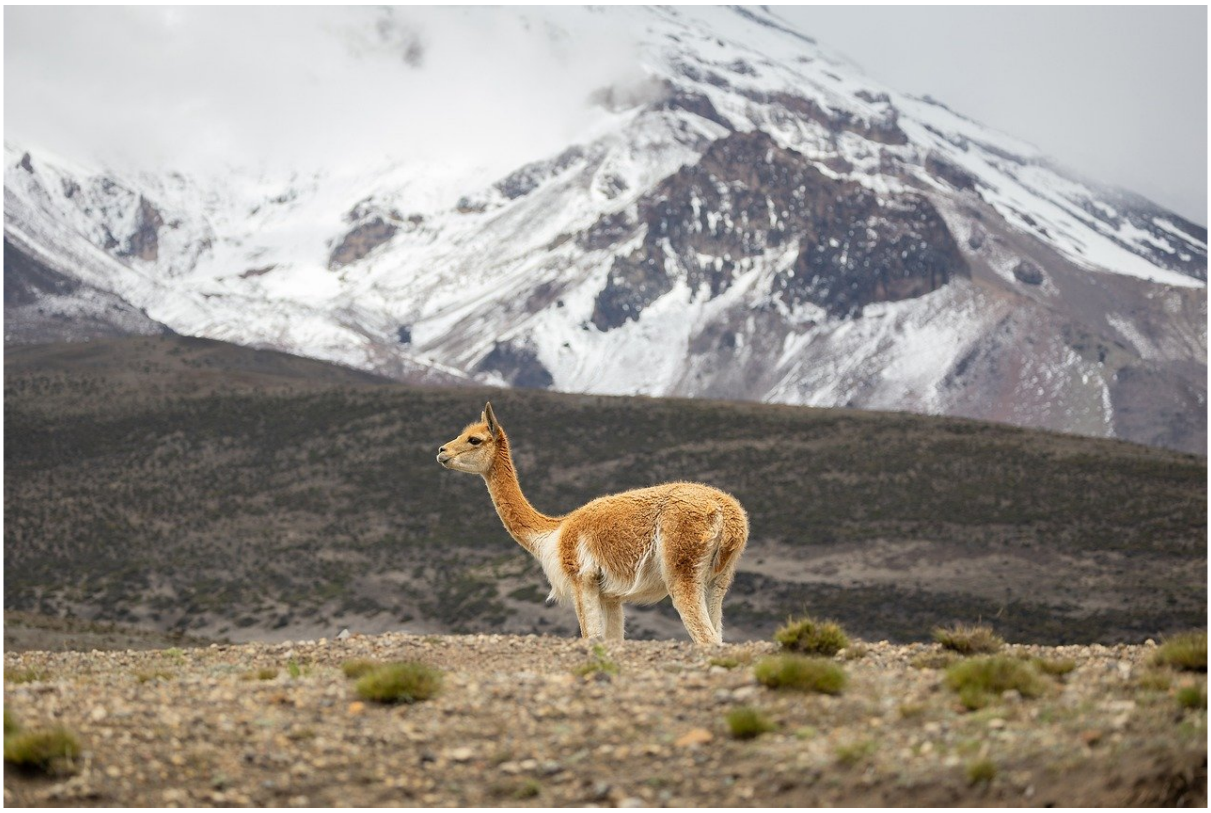

Figure 9:

Output

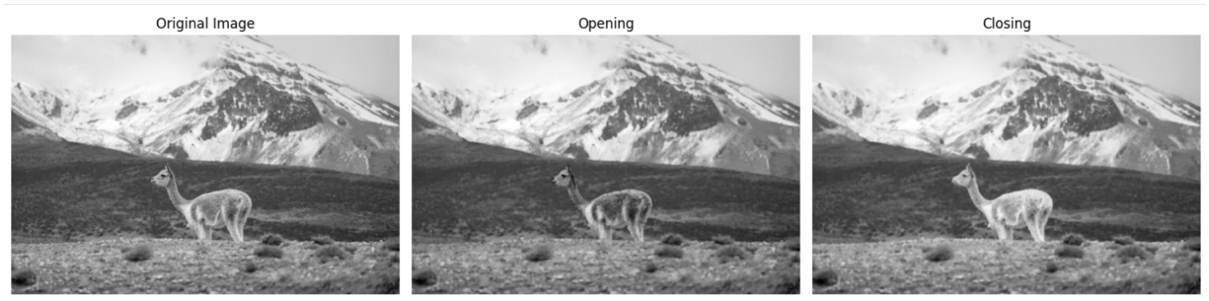

Figure 10: