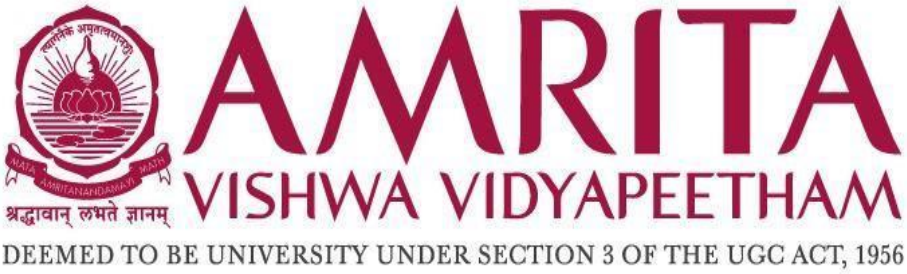
ASSIGNMENT - 3

An assignment
Submitted in fulfillment of the Completion of

the course 19CCE447 Image Processing under the Faculty of

Electronics and Communication Engineering

By:
SANTOSH [CB.EN.U4CCE20006]



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

AMRITA VISHWA VIDYAPEETHAM

COIMBATORE CAMPUS

FACULTY IN-CHARGE: SUGUNA G.

ASSIGNMENT 3 – ENTROPY

AIM:

To explore different methods for improving the visual quality of digital images.

SOFTWARE:

Product version: Anaconda3 2021.11
Spyder version: 5.1.5
Python version: 3.9.7 64-bit

Qt version: 5.9.7
PyQt5 version: 5.9.2
Operating System: Windows 10

THEORY AND PYTHON CODES:

I. Data Compression in Image Processing

Data compression is a technique used to minimize the size of data for storage or transmission purposes. It involves two primary approaches: lossless compression and lossy compression.

Lossless compression aims to compress data without any loss of information. It identifies patterns or redundancies within the data and encodes them more efficiently. Methods like Huffman coding assign shorter codes to frequently occurring patterns, while LZW replaces repeated patterns with shorter codes. Lossless compression is particularly useful for scenarios where preserving all original data is crucial, such as in text documents or program files.

Lossy compression, on the other hand, sacrifices some data fidelity to achieve higher compression ratios. It discards less important or less noticeable data based on human perception. Popular in multimedia applications, lossy compression algorithms like JPEG for images and MP3 for audio exploit perceptual limitations to reduce file size.

Data compression can be implemented through entropy coding or dictionary-based compression. Entropy coding, such as Huffman coding, utilizes statistical properties to assign shorter codes to more frequent symbols. Dictionary-based compression, like LZW, replaces repeated patterns with references.

Choosing a compression algorithm depends on factors such as data type, desired compression ratio, available resources, and the trade-off between compression efficiency and decompression speed. Continuous research and advancements aim to improve compression techniques for efficient data storage and transmission.

In summary, data compression reduces data size through lossless or lossy methods. It utilizes various algorithms such as Huffman coding and LZW, considering factors like data fidelity and compression efficiency, to optimize storage and transmission.

Q. Write a program in Python to implement Data Compression and to observe its effect on the image.

```
import os # import os module for file system operations
from PIL import Image # import Pillow module to work with images
import concurrent.futures # import concurrent.futures to perform multi-
threading
def resizeAndCompress(file):
    # Get the current working directory and join it with the file name
    filepath = os.path.join(os.getcwd(), file)
    picture = Image.open(filepath) # Open the image file
    # Resize the image to half of its original size
    width, height = picture.size
    picture.thumbnail((width//2, height//2))
    # Save the compressed image with custom quality
    picture.save("Compressed_" + file, "JPEG", optimize=True, quality=30)
    return file
def main():
    formats = ('.jpg', '.jpeg') # tuple containing image file extensions
    files_to_compress = [file for file in os.listdir() if
os.path.splitext(file)[1].lower() in formats] # list comprehension to get
all image files in current directory
    # Use ThreadPoolExecutor to compress multiple files simultaneously
    with concurrent.futures.ThreadPoolExecutor(max_workers=4) as executor:
        results = list(executor.map(resizeAndCompress, files_to_compress))
    print(f"Compressed {len(results)} files.") # print the number of files
that were compressed
if __name__ == "__main__":
    main()
```

Figure 1. Original Image

Figure 2. Compressed Image

II. Spatial Resolution in Image Processing

Spatial resolution refers to the level of detail in an image, measured by the number of pixels per unit area. It determines the ability of an imaging system to capture fine details. A higher spatial resolution means more pixels packed into a given area, resulting in a sharper and more detailed image.

Digital imaging systems use image sensors or display devices to define spatial resolution. A higher-resolution image sensor captures finer details, while a high-resolution display reproduces them with clarity. Spatial resolution is commonly expressed in DPI or PPI, indicating the number of pixels per inch.

Spatial resolution plays a critical role in various applications. In medical imaging, high spatial resolution is essential for accurately examining intricate structures. Satellite imaging benefits from higher resolution to identify small objects on the Earth's surface. Surveillance systems rely on spatial resolution to capture clear facial features or license plate numbers.

However, increasing spatial resolution leads to larger file sizes and greater computational demands. Therefore, it is crucial to strike a balance between detail and practical considerations. Understanding spatial resolution helps optimize image quality for specific requirements while managing storage and processing resources effectively.

In summary, spatial resolution in image processing refers to the level of detail or fineness of an image representation in terms of pixels per unit area. It is determined by the number of pixels in the image sensor or display device. Higher spatial resolution provides greater detail and is essential in various applications where capturing fine details is critical. However, it is crucial to consider the trade-offs between spatial resolution, file size, and computational requirements.

Q. Write a program in Python to implement Spatial Resolution and to observe its effect on the image.

```
import cv2
import matplotlib.pyplot as plt

# Read the original image and know its type
original_image = cv2.imread('Lighthouse.jpg', 0)

# Obtain the size of the original image
height, width = original_image.shape

# Show the original image
plt.imshow(original_image, cmap="gray")
plt.title("Original Image")
```

```
plt.xlabel("Width")
plt.ylabel("Height")
plt.show()

# Down sampling
downsampling_rate = 4
downsampled_image = cv2.resize(original_image, (width // downsampling_rate,
height // downsampling_rate))

# Show down-sampled image
plt.imshow(downsampled_image, cmap="gray")
plt.title("Down Sampled Image")
plt.xlabel("Width")
plt.ylabel("Height")
plt.show()

# Up-sampling
upsampled_image = cv2.resize(downsampled_image, (width, height),
interpolation=cv2.INTER_NEAREST)

# Plot the up-sampled image
plt.imshow(upsampled_image, cmap="gray")
plt.title("Up Sampled Image")
plt.xlabel("Width")
plt.ylabel("Height")
plt.show()
```

Figure 3. Original Image

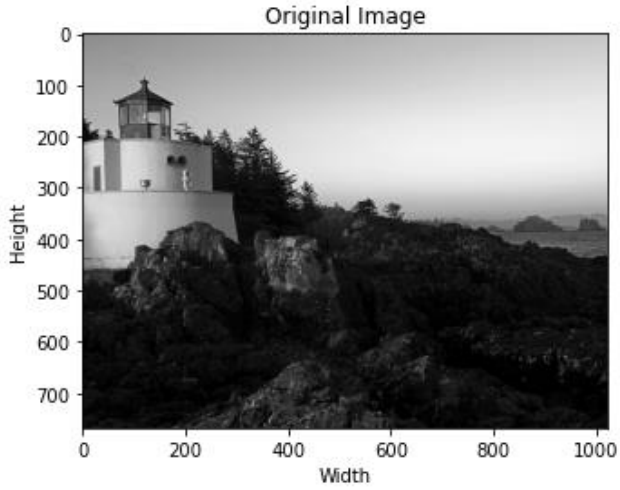


Figure 4. Grayscale Image

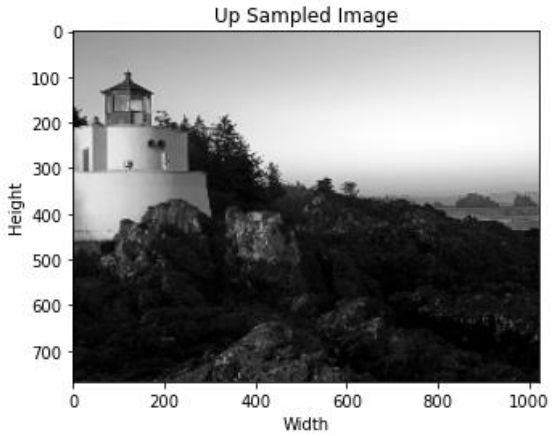


Figure 5. Up Sampled Image

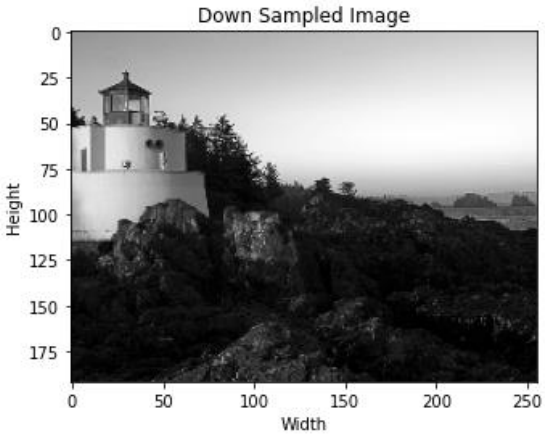


Figure 6. Down Sampled Image

III. Entropy in Image Processing

Entropy is a fundamental concept used in image processing to measure the amount of information or uncertainty present in an image. In the context of image analysis, entropy provides a quantitative measure of the randomness or disorder within the distribution of pixel intensities.

The concept of entropy originates from information theory, where it is used to quantify the amount of information contained in a signal or data set. In image processing, entropy is utilized to assess the complexity and richness of information within an image.

To compute entropy, the histogram of an image is first calculated. The histogram represents the frequency distribution of pixel intensities in the image. Then, using the histogram, the probability of occurrence of each intensity level is determined by dividing the frequency of pixels with a particular intensity by the total number of pixels in the image.

Once the probabilities are obtained, the entropy is computed by summing up the product of each probability and its corresponding logarithm (usually in base 2) for all intensity levels present in the image. The negative sign is often applied to the summation to yield a positive entropy value.

High entropy values indicate that the image contains a wide range of intensities with no dominant patterns or structures. This implies a high level of complexity or randomness in the image. On the other hand, low entropy values suggest a more predictable distribution of pixel intensities, indicating the presence of repetitive patterns or uniform regions.

Entropy has various applications in image processing. It is commonly used in image segmentation, where it helps in identifying regions of interest based on the differences in entropy values across the image. Higher entropy regions often correspond to boundaries or regions with rich texture and detail. Entropy can also be used for image compression, as areas with lower entropy can be encoded with fewer bits, while preserving the essential information.

In summary, entropy is a valuable measure in image processing that quantifies the amount of information or randomness within an image. It aids in understanding the complexity and structure of an image, and finds applications in tasks like image segmentation and compression.

Q. Write a program in Python to implement Entropy and to observe its effect on the image.

```
import warnings
import numpy as np
import matplotlib.pyplot as plt
import skimage.io as io
from skimage.util import img_as_ubyte
from skimage.filters.rank import entropy
from skimage.morphology import disk
from skimage.color import rgb2gray
warnings.filterwarnings('ignore')

def get_entropy_image(image_gray, radius):
    return entropy(image_gray, disk(radius))

def plot_disk_iterations(image_gray):
    fig, axes = plt.subplots(3, 3, figsize=(15, 15))
    radii = range(1, 10)
    fig.suptitle('Disk Iterations', fontsize=24)

    for radius, ax in zip(radii, axes.flatten()):
        ax.set_title(f'Radius at {radius}', fontsize=20)

        entropy_image = get_entropy_image(image_gray, radius)
        ax.imshow(entropy_image, cmap='magma')
        ax.set_xlabel('X Label')
        ax.set_ylabel('Y Label')
    fig.tight_layout()
    return fig, axes

def plot_threshold_checker(image_gray):
    entropy_image = get_entropy_image(image_gray, 6)
    scaled_entropy = entropy_image / entropy_image.max()
    fig, axes = plt.subplots(2, 5, figsize=(17, 10))
```

```
    thresholds = [0.1 * (i+1) for i in range(10)]
    fig.suptitle('Threshold Checker', fontsize=24)
    for threshold, ax in zip(thresholds, axes.flatten()):
        ax.set_title(f'Threshold: {threshold:.2f}', fontsize=16)
        threshold_mask = scaled_entropy > threshold
        ax.imshow(threshold_mask, cmap='gist_stern_r')
        ax.axis('off')
    fig.tight_layout()
    return fig, axes
def plot_entropy_mask_viz(image_gray):
    entropy_image = get_entropy_image(image_gray, 6)
    scaled_entropy = entropy_image / entropy_image.max()
    fig, axes = plt.subplots(1, 2, figsize=(17, 10))
    f_size = 24
    titles = ['Greater Than Threshold', 'Less Than Threshold']
    for condition, title, ax in zip([scaled_entropy > 0.8, scaled_entropy <
0.8], titles, axes.flatten()):
        masked_image = np.stack([image_gray]*3, axis=-1)
        masked_image[~condition] = 0
        ax.imshow(masked_image)
        ax.axis('off')
        ax.set_title(title, fontsize=f_size)
    fig.suptitle('Entropy Mask Visualization', fontsize=24)
    fig.tight_layout()
    return fig, axes

# Read image and save to variable for later use
shawls = io.imread('Lighthouse.jpg')

# Convert to grayscale and unsigned byte
shawl_gray = img_as_ubyte(rgb2gray(shawls))

# Show original and grayscale images
fig, axes = plt.subplots(1, 2, figsize=(12, 6))
titles = ['Original Image', 'Grayscale Image']
images = [shawls, shawl_gray]

for ax, image, title in zip(axes.flatten(), images, titles):
    ax.imshow(image)
    ax.set_xlabel('X Label')
```

```
    ax.set_ylabel('Y Label')
    ax.set_title(title, fontsize=20)

plt.tight_layout()

# Calculate entropy image and show it
entropy_image = get_entropy_image(shawl_gray, 6)
fig, ax = plt.subplots(1, 1, figsize=(7, 7))
ax.imshow(entropy_image, cmap='magma')
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_title('Entropy Image', fontsize=20)
plt.tight_layout()

# Show entropic images for different disk radii
plot_disk_iterations(shawl_gray)

# Show thresholded versions of the entropic image
plot_threshold_checker(shawl_gray)

# Show masked grayscale image based on entropy threshold
plot_entropy_mask_viz(shawl_gray)

plt.show()
```

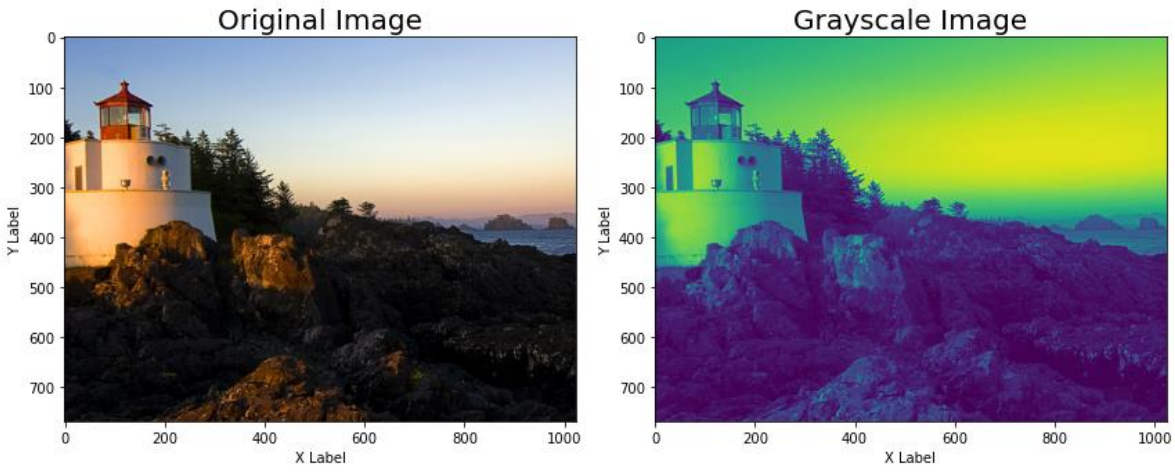


Figure 7. (a) Original Image (b) Grayscale Image

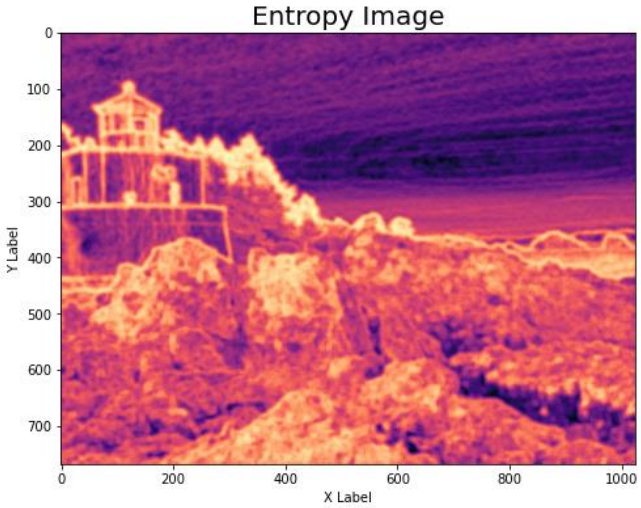


Figure 8. Entropy Image

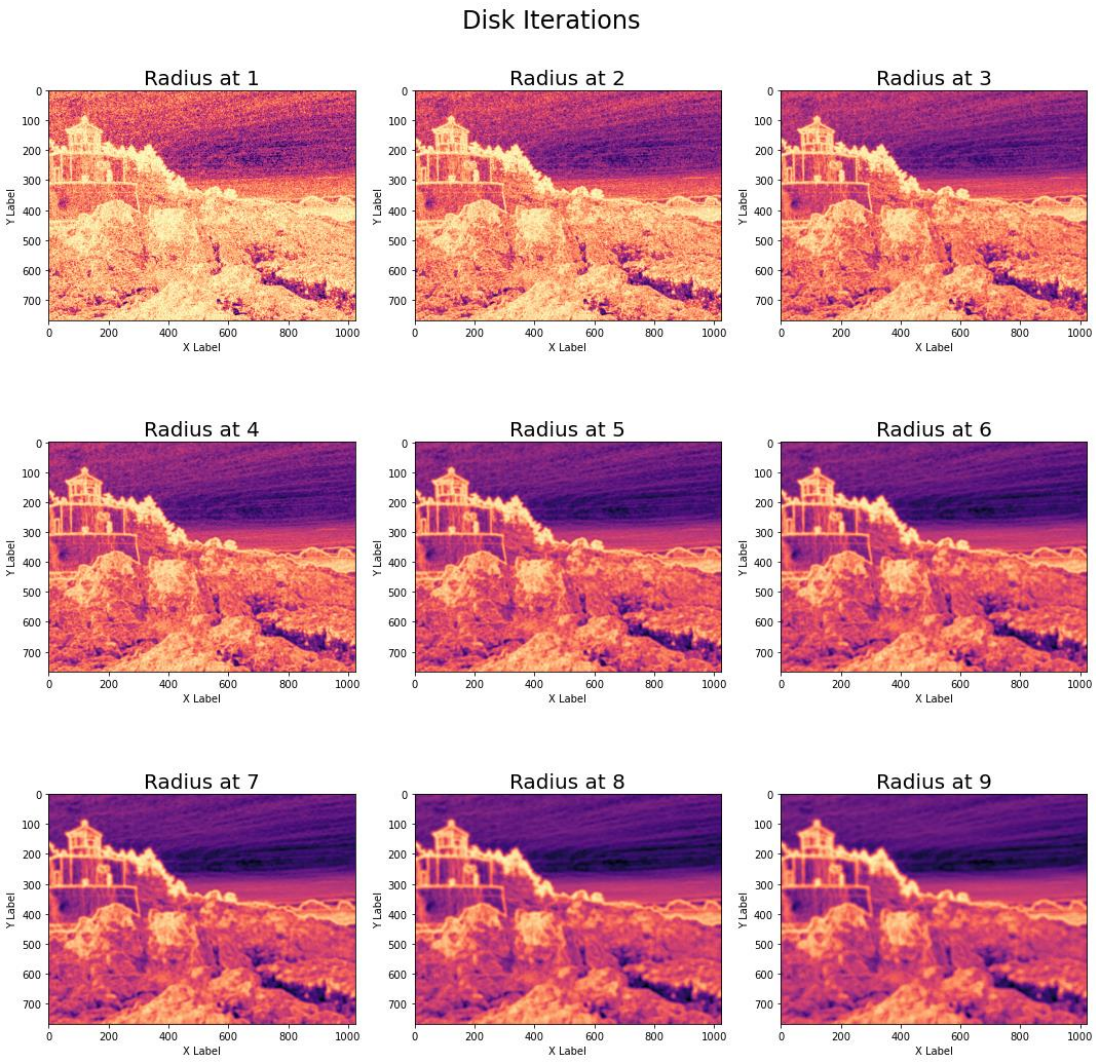


Figure 9. Disk Iterations

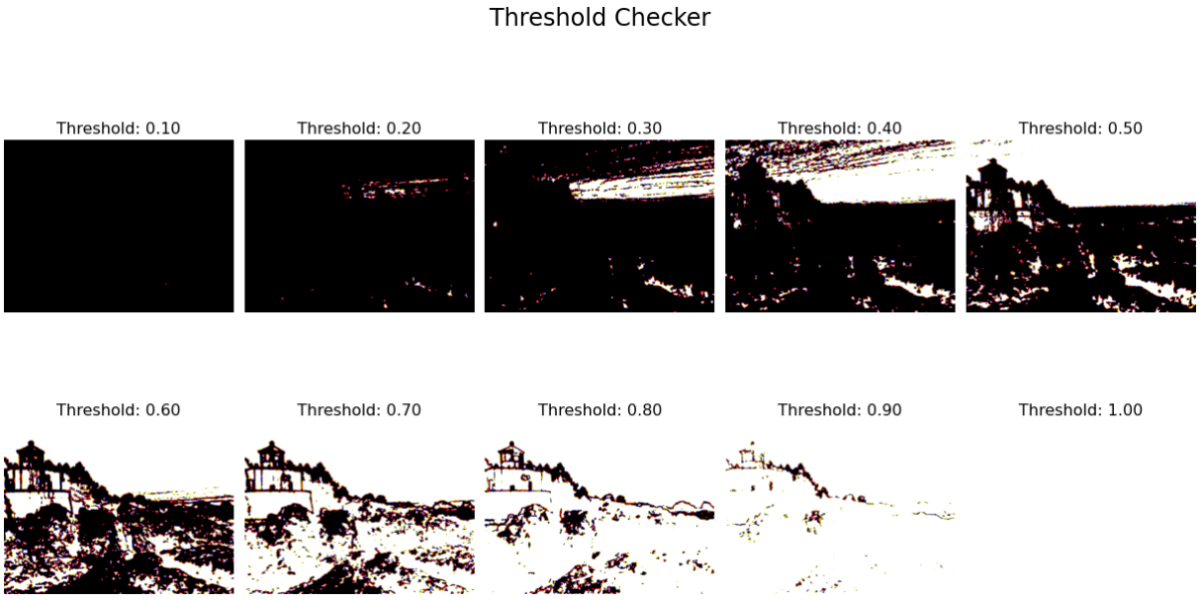


Figure 10. Threshold Checker

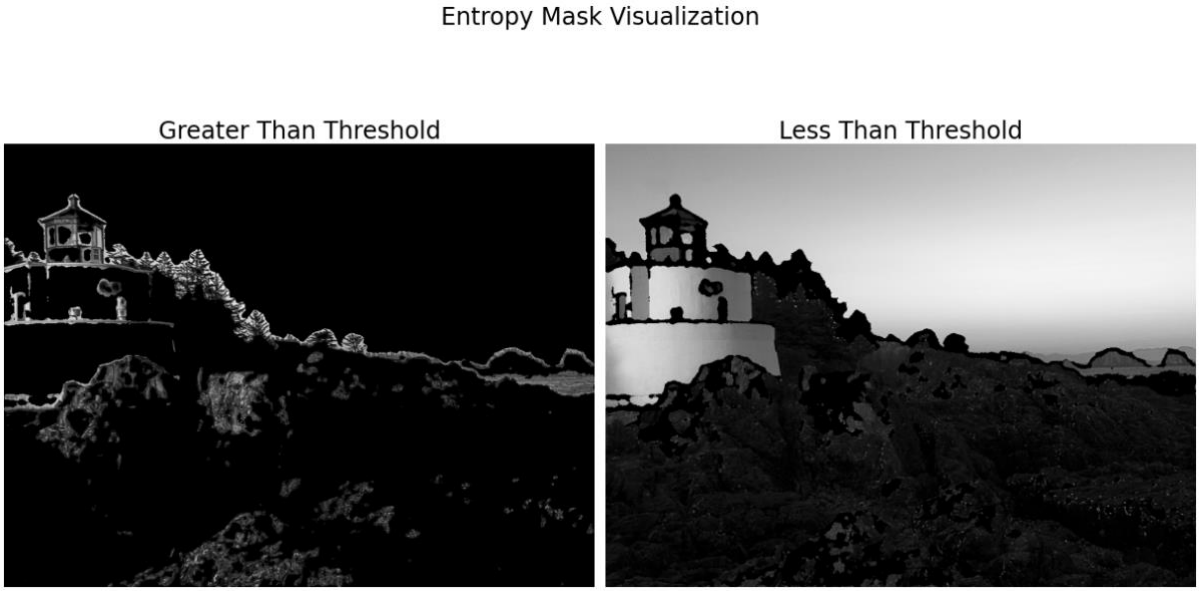


Figure 11. Entropy Mask Visualisation

INFERENCE:

- The image processing assignment on entropy has been completed, and various methods for quantifying the information content and randomness within digital images have been explored.
- Thorough research on the theoretical background of entropy in image processing, along with practical implementation using Python programming, has provided a comprehensive understanding of different entropy-based methods and their applications.
- The documentation includes labeled images of input and output figures, demonstrating the effectiveness of the methods in quantifying image complexity and randomness.
- The provided references offer additional resources for delving deeper into the topic of entropy in image processing.
- Overall, this assignment has significantly contributed to the development of skills in understanding and utilizing entropy as a measure of information and randomness in digital images, as well as proficiency in applying Python programming for entropy analysis in image processing tasks.

REFERENCES:

Serial Number	Online Portal	Content	URL
1	Geeks For Geeks	How to compress images using Python and PIL?	https://www.geeksforgeeks.org/how-to-compress-images-using-python-and-pil/
2		Spatial Resolution (down sampling and up sampling) in image processing	https://www.geeksforgeeks.org/spatial-resolution-down-sampling-and-up-sampling-in-image-processing/
3	Medium	Image Processing with Python — Working with Entropy	https://towardsdatascience.com/image-processing-with-python-working-with-entropy-b05e9c84fc36

Textbooks:

1. Digital Image Processing, Fourth Edition, Rafael C. Gonzalez, Richard E. Woods, Pearson, 2018
2. Digital Image Processing, S Jayaraman, S Esakkirajan, T Veerakumar, Tata McGraw Hill, 2009
3. Fundamentals of Digital Image Processing, Anil K. Jain, Pearson Education, Prentice Hall

References:

1. Digital Image Processing - PIKS Scientific Inside, Fourth Edition, Willian K Pratt, Wiley-Interscience, 2007
2. Multidimensional Signal, Image, and Video Processing and Coding, Second Edition, John W. Woods, Elsevier, 2012