**Banaras Hindu University**  
**Institute of Science**  
**Department of Computer Science**



**Subject: “Image Processing”**

**Submitted To:**  
**Dr.Ankita Vaish**  
Department of Computer Science

**Submitted By:**  
Sagar Timalsena

(24419MSC025)

**Academic Year:**

2024-2025

1. Least Significant Bit (LSB) Modification in an Image

1. Load an Image: Choose a grayscale.
2. Modify the Image: Convert the image into its pixel matrix.
3. Set the Least Significant Bit (LSB) of each pixel to zero.
4. Save the modified image.
5. Display the Images: Display the original image and Display the modified image.
6. Compute and display the difference image (original - modified).
7. Analyze the Difference: the visual and numerical differences between the images.

|  |
| --- |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  def lsb\_modify(image\_path, save\_path):  # Load image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Modify LSB by setting it to 0  modified\_img = original\_img & 0b11111110    # Save the modified image  cv2.imwrite(save\_path, modified\_img)    # Compute the difference image  difference\_img = original\_img - modified\_img    # Display images  plt.figure(figsize=(10, 4))    plt.subplot(1, 3, 1)  plt.imshow(original\_img, cmap='gray')  plt.title("Original Image")  plt.axis("off")    plt.subplot(1, 3, 2)  plt.imshow(modified\_img, cmap='gray')  plt.title("Modified Image (LSB Zeroed)")  plt.axis("off")    plt.subplot(1, 3, 3)  plt.imshow(difference\_img, cmap='gray')  plt.title("Difference Image")  plt.axis("off")    plt.show()    # Numerical analysis  print("Maximum pixel difference:", np.max(difference\_img))  print("Minimum pixel difference:", np.min(difference\_img))  print("Mean pixel difference:", np.mean(difference\_img))  # Example usage  image\_path = "image.png" # Provide the path to your grayscale image  save\_path = "modified\_image.png"  lsb\_modify(image\_path, save\_path) |

Output:



Normalize an image using the following steps:

* Loads an image in grayscale mode using OpenCV.
* Converts the image to floating point representation for calculations.
* Calculates the minimum and maximum pixel values in the image.
* Normalizes the image using the formula round((image\_float - min\_val) / (max\_val - min\_val))\*R.  if R=128, it will normalized image to [0-127] range.
* Displays the original and normalized images using OpenCV.

|  |
| --- |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  def lsb\_modify(image\_path, save\_path):  # Load image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Modify LSB by setting it to 0  modified\_img = original\_img & 0b11111110    # Save the modified image  cv2.imwrite(save\_path, modified\_img)    # Compute the difference image  difference\_img = original\_img - modified\_img    # Display images  plt.figure(figsize=(10, 4))    plt.subplot(1, 3, 1)  plt.imshow(original\_img, cmap='gray')  plt.title("Original Image")  plt.axis("off")    plt.subplot(1, 3, 2)  plt.imshow(modified\_img, cmap='gray')  plt.title("Modified Image (LSB Zeroed)")  plt.axis("off")    plt.subplot(1, 3, 3)  plt.imshow(difference\_img, cmap='gray')  plt.title("Difference Image")  plt.axis("off")    plt.show()    # Numerical analysis  print("Maximum pixel difference:", np.max(difference\_img))  print("Minimum pixel difference:", np.min(difference\_img))  print("Mean pixel difference:", np.mean(difference\_img))  def normalize\_image(image\_path, R=128):  # Load image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Convert image to float  image\_float = original\_img.astype(np.float32)    # Calculate min and max pixel values  min\_val, max\_val = np.min(image\_float), np.max(image\_float)    # Normalize the image  normalized\_img = np.round(((image\_float - min\_val) / (max\_val - min\_val)) \* R)  normalized\_img = normalized\_img.astype(np.uint8)    # Display images  cv2.imshow("Original Image", original\_img)  cv2.imshow("Normalized Image", normalized\_img)  cv2.waitKey(0)  cv2.destroyAllWindows()  # Example usage  image\_path = "image.png"  save\_path = "modified\_image.png"  lsb\_modify(image\_path, save\_path)  normalize\_image(image\_path) |

# Output:



Image Encryption: Encrypt the image

1. Load the input grayscale image using an image processing library like OpenCV.
2. Generate a random key matrix of the same size as the input image. Each element in the key matrix should be a random integer between 0 and 255.
3. Perform element-wise addition of the input image and the key matrix.
4. Take the modulo 256 of each sum to ensure the result stays within the 0-255 range.
5. The resulting matrix represents the encrypted image.

Decryption: Decrypt the Image

1. Perform element-wise subtraction of the encrypted image and the key matrix.
2. Take the modulo 256 of each difference to ensure the result stays within the 0-255 range.
3. The resulting matrix represents the decrypted image.
4. Display the original input image, the encrypted image, and the decrypted image for comparison.
5. Verify that the decrypted image matches the original input image, confirming the correctness of the encryption and decryption process.

|  |
| --- |
| import cv2  import numpy as np  import matplotlib.pyplot as plt  def lsb\_modify(image\_path, save\_path):  # Load image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Modify LSB by setting it to 0  modified\_img = original\_img & 0b11111110    # Save the modified image  cv2.imwrite(save\_path, modified\_img)    # Compute the difference image  difference\_img = original\_img - modified\_img    # Display images  plt.figure(figsize=(10, 4))    plt.subplot(1, 3, 1)  plt.imshow(original\_img, cmap='gray')  plt.title("Original Image")  plt.axis("off")    plt.subplot(1, 3, 2)  plt.imshow(modified\_img, cmap='gray')  plt.title("Modified Image (LSB Zeroed)")  plt.axis("off")    plt.subplot(1, 3, 3)  plt.imshow(difference\_img, cmap='gray')  plt.title("Difference Image")  plt.axis("off")    plt.show()    # Numerical analysis  print("Maximum pixel difference:", np.max(difference\_img))  print("Minimum pixel difference:", np.min(difference\_img))  print("Mean pixel difference:", np.mean(difference\_img))  def normalize\_image(image\_path, R=128):  # Load image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Convert image to float  image\_float = original\_img.astype(np.float32)    # Calculate min and max pixel values  min\_val, max\_val = np.min(image\_float), np.max(image\_float)    # Normalize the image  normalized\_img = np.round(((image\_float - min\_val) / (max\_val - min\_val)) \* R)  normalized\_img = normalized\_img.astype(np.uint8)    # Display images  cv2.imshow("Original Image", original\_img)  cv2.imshow("Normalized Image", normalized\_img)  cv2.waitKey(0)  cv2.destroyAllWindows()  def encrypt\_decrypt\_image(image\_path):  # Load the image in grayscale  original\_img = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)    if original\_img is None:  print("Error: Could not load image.")  return    # Generate random key matrix  key\_matrix = np.random.randint(0, 256, original\_img.shape, dtype=np.uint8)    # Encrypt the image  encrypted\_img = (original\_img + key\_matrix) % 256    # Decrypt the image  decrypted\_img = (encrypted\_img - key\_matrix) % 256    # Display images  plt.figure(figsize=(15, 5))    plt.subplot(1, 3, 1)  plt.imshow(original\_img, cmap='gray')  plt.title("Original Image")  plt.axis("off")    plt.subplot(1, 3, 2)  plt.imshow(encrypted\_img, cmap='gray')  plt.title("Encrypted Image")  plt.axis("off")    plt.subplot(1, 3, 3)  plt.imshow(decrypted\_img, cmap='gray')  plt.title("Decrypted Image")  plt.axis("off")    plt.show()    # Verification  if np.array\_equal(original\_img, decrypted\_img):  print("Decryption successful: The decrypted image matches the original image.")  else:  print("Decryption failed: The decrypted image does not match the original image.")  # Example usage  image\_path = "image.png" # Provide the path to your grayscale image  save\_path = "modified\_image.png"  lsb\_modify(image\_path, save\_path)  normalize\_image(image\_path)  encrypt\_decrypt\_image(image\_path) |

