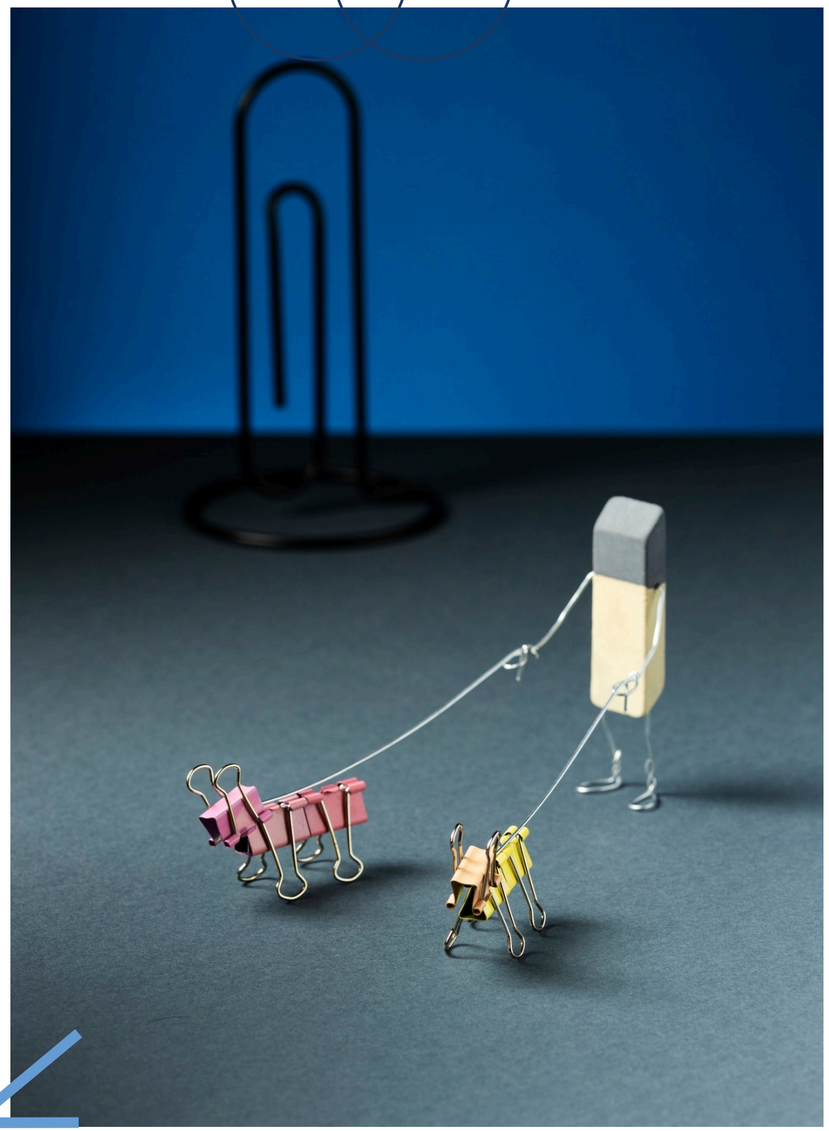
Unlocking Visual Potential: The Art of Compressed Image Sensing and Encryption using CNN and GAN



Guided By: Professor Shyamalendu Kandar
By group
Satvik Wazir (2021ITB018)
Shridhar Reddy (2021ITB023)
Suryansh Pandey (2021ITB095)

Table Of Contents

Introduction

- In today's digital era, image compression and encryption play a vital role in efficient storage and secure transmission of visual data.
- Compression techniques reduce the data size without significant loss of quality and encryption ensures the confidentiality of the image data.
- This project combines Discrete Wavelet Transform (DWT), Discrete Cosine Transform (DCT), and Chaotic Maps to achieve both compression and encryption, providing an optimized and secure image processing solution.

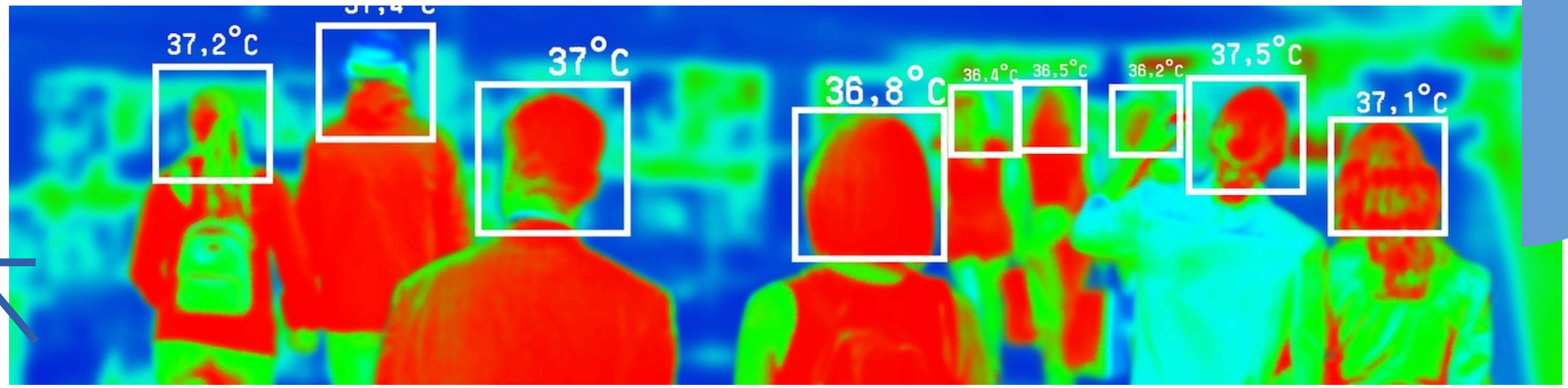
Objective

The primary objectives of this project are:

- Compress an image using DWT and DCT.
- Encrypt the image using a chaotic map and XOR operation for security.
- Decrypt and reconstruct the original image while evaluating quality using error metrics.

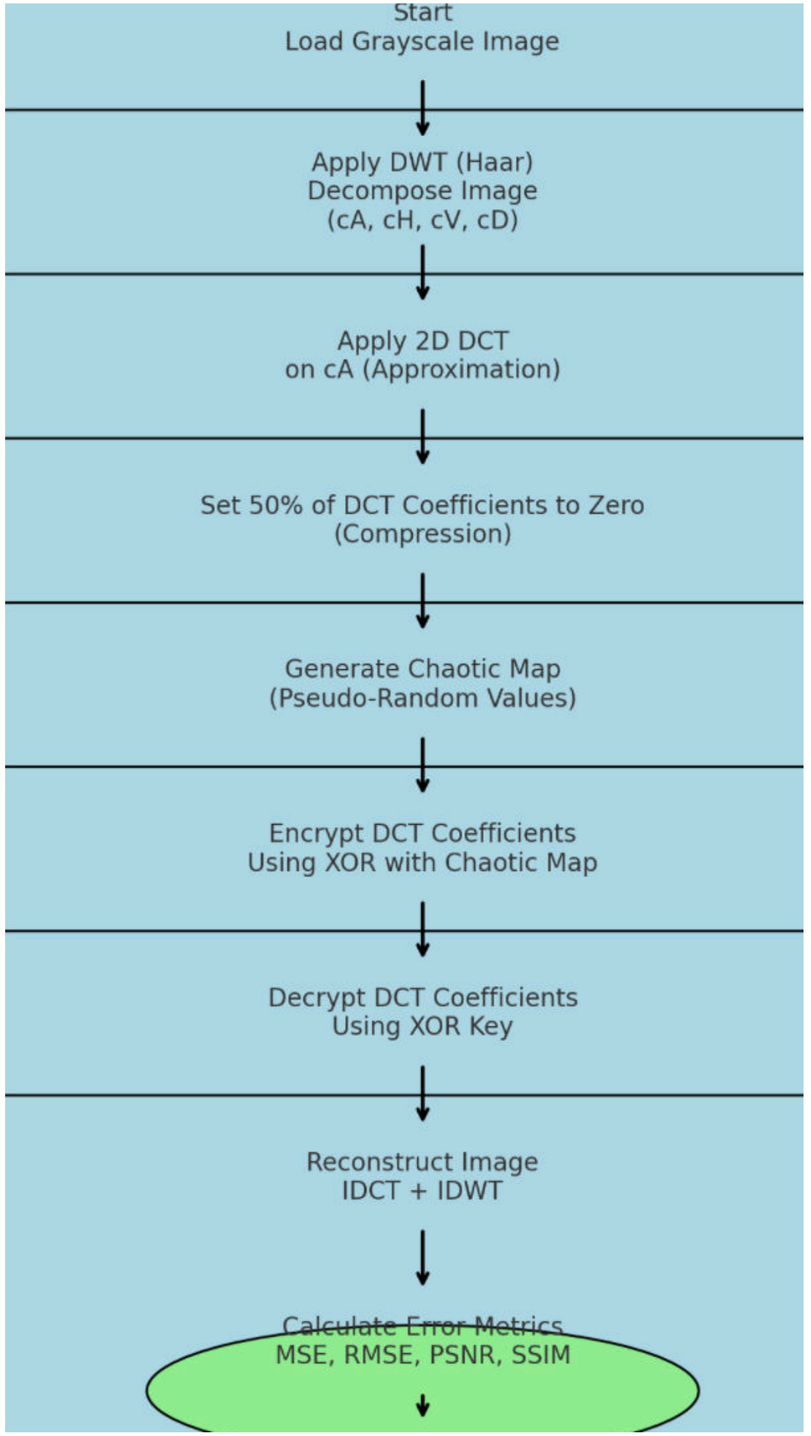
Introduction to Compressed Sensing

Compressed Sensing (CS)is a signal processing technique that enables the reconstruction of a signal (such as an image or audio) from a significantly smaller number of measurements than traditionally required.In the realm of **image processing**, **compressed sensing** offers a revolutionary approach to capture and reconstruct images efficiently. By exploiting **sparsity**, it allows for reduced data acquisition, leading to faster processing and enhanced performance in various applications.



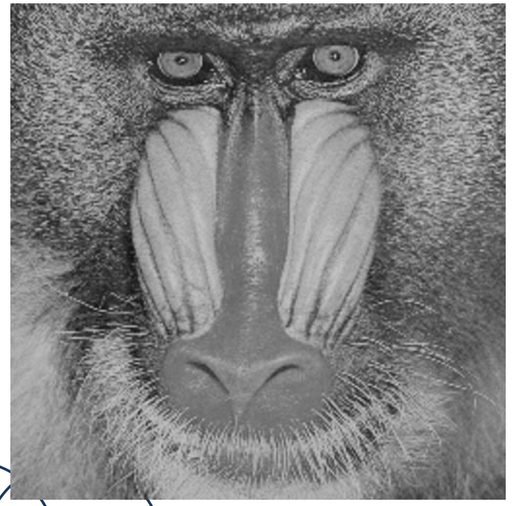
Flowchart

Here is the flowchart illustrating each step of the image compression and encryption process.



Step 1: Load the Image

- The process starts by loading a grayscale image using thefunction from the OpenCV library.
- Grayscale images are chosen as they are simpler to process compared to colored images. The pixel values range from 0 (black) to 255 (white).



Step 2: Apply Discrete Wavelet Transform (DWT)

- The Discrete Wavelet Transform (DWT) decomposes an image into frequency subbands. It separates the image into:
- Approximation Coefficients (cA): Represent the low-frequency part (most important visual information).
- Detail Coefficients (cH, cV, cD): Represent horizontal, vertical, and diagonal high-frequency details.
- Here, the 'Haar' wavelet is used, which is computationally efficient and commonly used in image processing.

Step 3: Apply Discrete Cosine Transform (DCT)

- The Discrete Cosine Transform (DCT) converts spatial domain data (pixels) into frequency domain coefficients. Most of the significant information is concentrated in the low-frequency coefficients.
- Compression: Small-magnitude coefficients (high frequencies) are set to zero.
- Reconstruction: The remaining coefficients are used to reconstruct the image.

Step 4: Generate Chaotic Map

Chaotic maps are deterministic systems that generate pseudo-random sequences. Here, a sinusoidal and cosine-based iterative map generates values that appear random but are reproducible.

Step 5: XOR-Based Encryption

To encrypt the DCT coefficients, the chaotic matrix is scaled and then XORed with a predefined encryption key.

Step 6: Encrypt the DCT Coefficients

Explanation: The non-zero DCT coefficients are multiplied element-wise by the encrypted chaotic matrix.This scrambling encrypts the DCT coefficients.

Step 7: Decrypt and Reconstruct

The encrypted matrix is XORed again with the same encryption key to retrieve the chaotic matrix.Reconstruction:Unscramble the DCT coefficients using the decrypted chaotic matrix.Perform the Inverse DCT (IDCT) to recover the approximation coefficients.Reconstruct the original image using the Inverse DWT (IDWT).

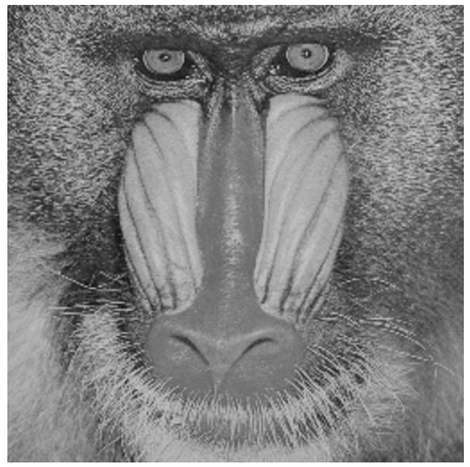
Step 8: Calculate Metrics

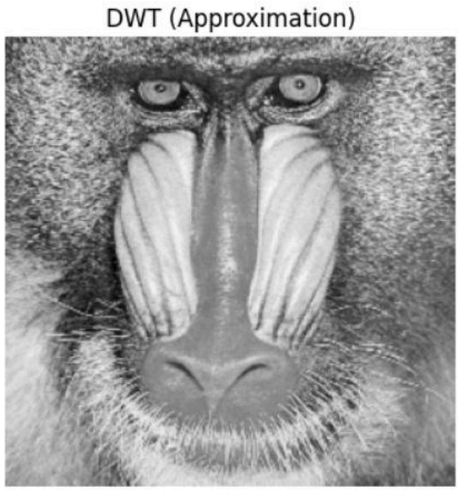
Error metrics are used to evaluate the quality of the reconstructed image.

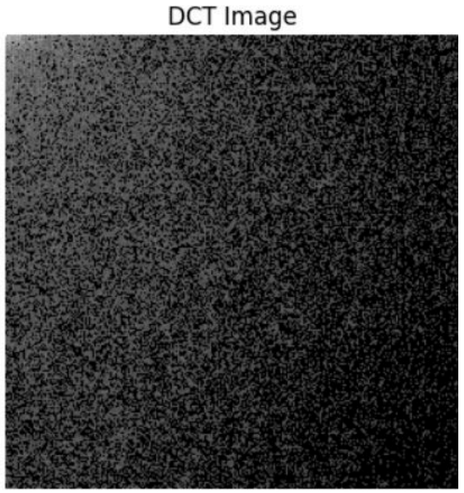
- Mean Squared Error (MSE): Measures the average squared difference between original and reconstructed pixels
- Root Mean Squared Error (RMSE): Square root of MSE for interpretability.
- Peak Signal-to-Noise Ratio (PSNR): Measures image quality in decibels (higher PSNR = better quality).
- Structural Similarity Index (SSIM): Measures the perceived similarity between two images.

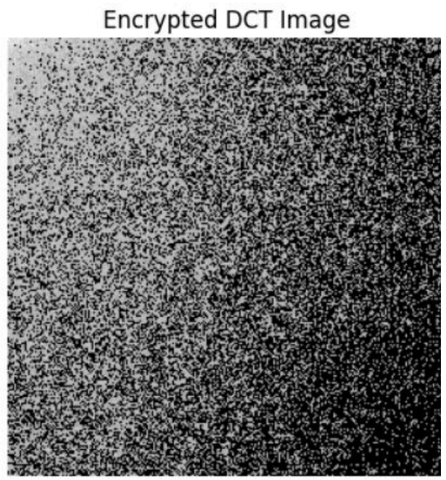
Step 9: Results Visualization

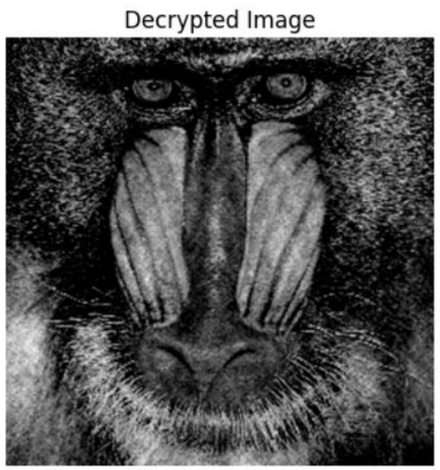
- The original image, encrypted DCT coefficients, and decrypted image are displayed. Histograms show changes in pixel distribution before and after encryption.



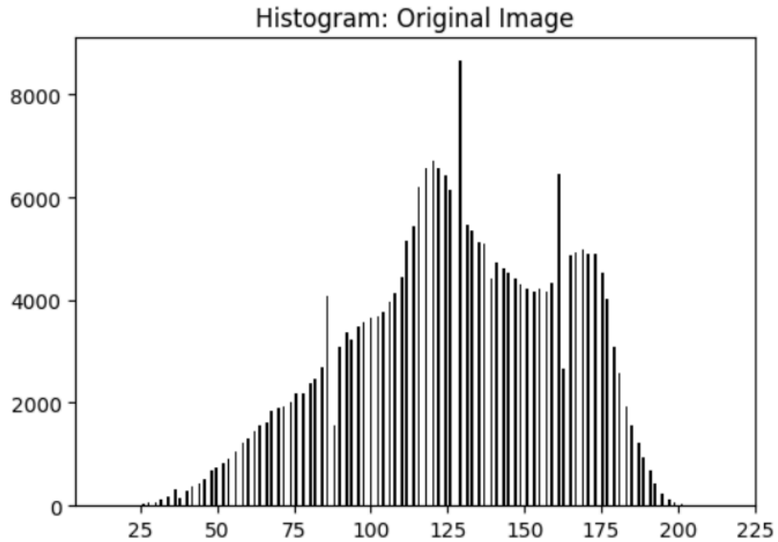


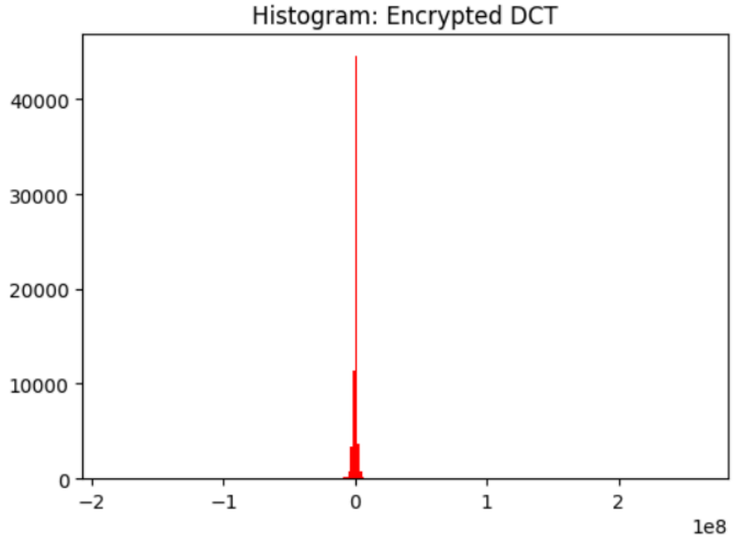


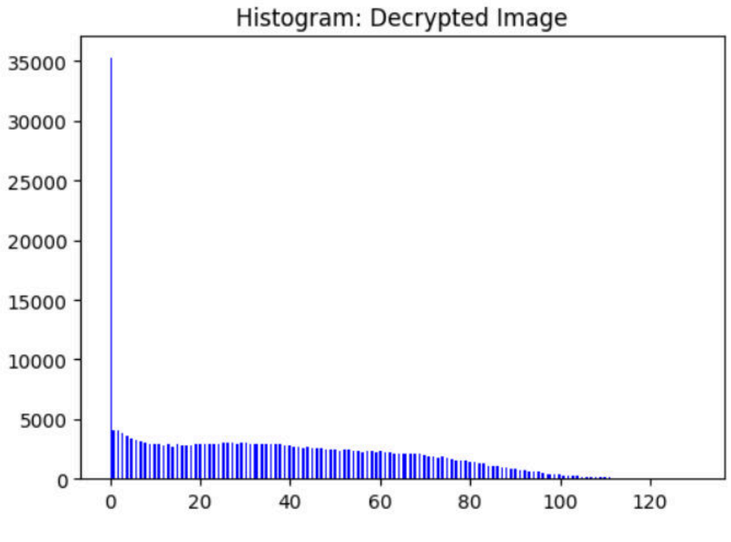




Step 10: Results Visualization







Error Metrics:

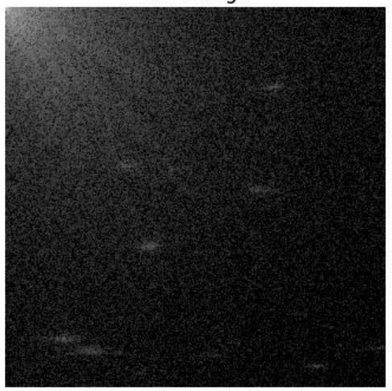
- MSE: 112.9207
- RMSE: 10.6264
- PSNR: 27.60 dB
- SSIM: 0.3187

Original Image

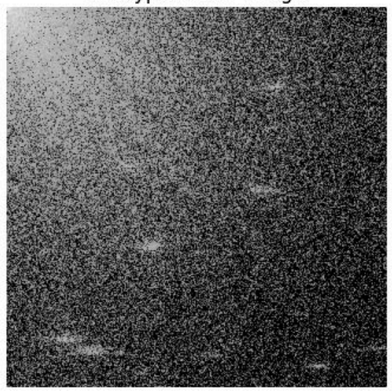
Histogram: Original Image

DWT (Approximation)

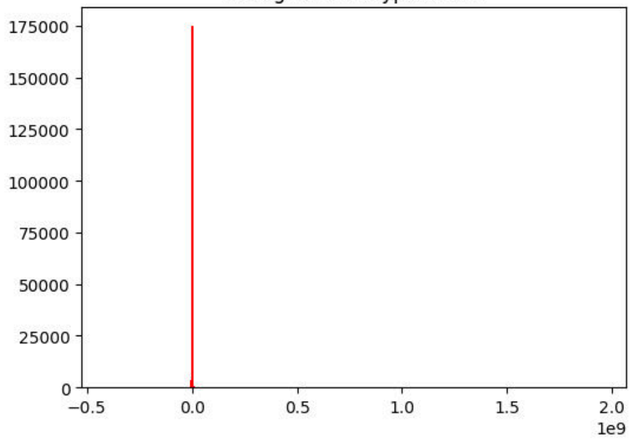
DCT Image



Encrypted DCT Image

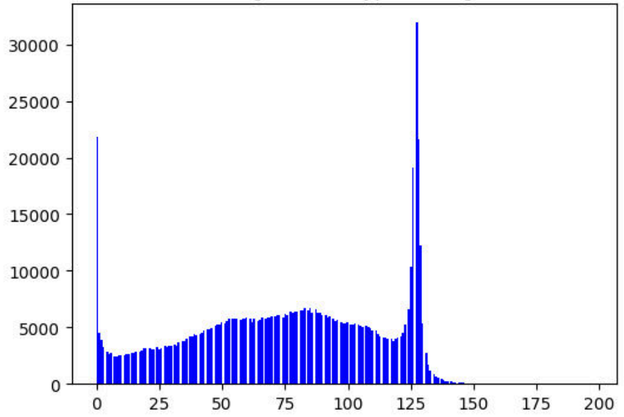


Histogram: Encrypted DCT



Decrypted Image

Histogram: Decrypted Image



Error Metrics

MSE: 108.3830

RMSE: 10.4107

PSNR: 27.78 dB

SSIM: 0.7079

Code Implementation

```
import numpy as np
import cv2
import pywt
from scipy.fftpack import dct, idct
import matplotlib.pyplot as plt
from skimage.metrics import structural_similarity as ssim
from math import log10, sqrt
```

- **numpy**: Used for numerical operations, arrays, and matrix manipulations.
- **cv2**: Used for image loading, manipulation, and processing.
- **pywt**: Provides functions for performing Discrete Wavelet Transform (DWT) and its inverse (IDWT).
- **scipy.fftpack.dct**: Computes the Discrete Cosine Transform (DCT) for image compression.
- **scipy.fftpack.idct**: Performs the inverse DCT (IDCT) to reconstruct compressed data.
- **matplotlib.pyplot**: Used for visualizing images, histograms, and other plots.
- **skimage.metrics.ssim**: Computes the Structural Similarity Index (SSIM) to evaluate image quality.
- **math.log10**: Used to compute the logarithm base 10 for PSNR calculation.
- **math.sqrt**: Computes the square root, essential for RMSE and PSNR calculations.

Code Implementation

```python
# Load the image
img = cv2.imread('/kaggle/input/lanapic/The-recovered-test-image-Elaine-obtained-from-an-authorized-set-of-four-shadow-images-one.ppm.png', cv2.IMREAD_GRAYSCALE)

# ---------------- 1. Apply DWT to the Image ---------------- #
coeffs = pywt.dwt2(img, 'haar')  # Single-level DWT using 'haar' wavelet
cA, (cH, cV, cD) = coeffs  # Approximation, Horizontal, Vertical, Diagonal components

# ---------------- 2. Apply DCT to the Approximation Image ---------------- #
dct_img = dct(dct(cA.astype(float), axis=0, norm='ortho'), axis=1, norm='ortho')

# Set 50% of the DCT coefficients to zero (reduce compression loss)
n = dct_img.size
nz = int(0.5 * n)  # Keep 50% of coefficients
indices = np.unravel_index(np.argsort(-np.abs(dct_img), axis=None), dct_img.shape)
zero_mask = np.zeros_like(dct_img, dtype=bool)
zero_mask[indices[0][nz:], indices[1][nz:]] = True
dct_img[zero_mask] = 0
```

Code Implementation

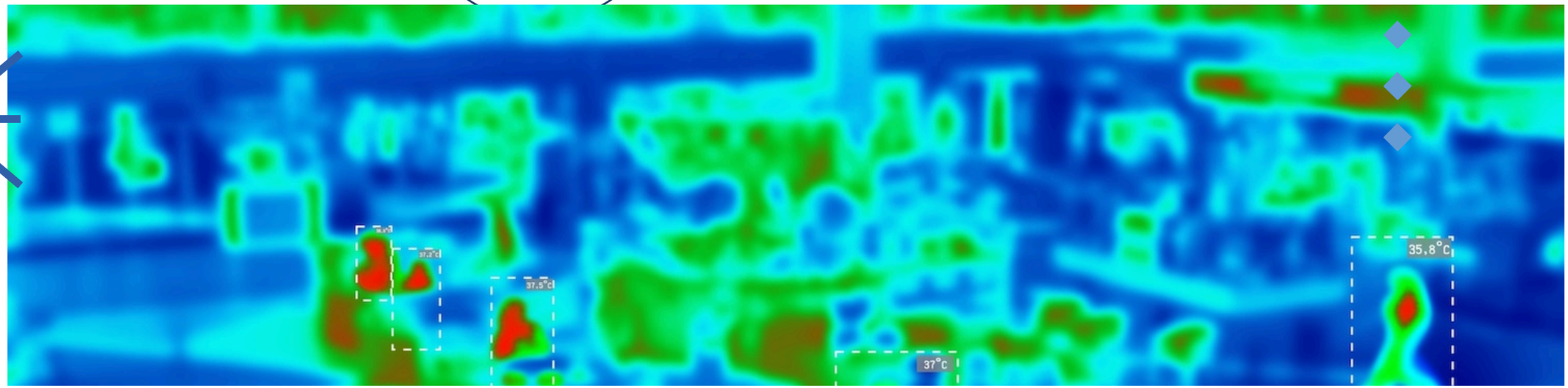
```
# ---------------- 3. Create the Chaotic Map ---------------- #a
chaotic_mat = np.zeros_like(dct_img)
x0, y0, z0 = 0.2, 0.4, 0.6
a = 10
for i in range(chaotic_mat.size):
    x = np.mod(np.sin(a * y0) + np.cos(a * z0), 1)
    y = np.mod(np.sin(a * z0) + np.cos(a * x0), 1)
    z = np.mod(np.sin(a * x0) + np.cos(a * y0), 1)
    chaotic_mat.flat[i] = x + y + z
    x0, y0, z0 = x, y, z

# Scale chaotic map to match DCT range
chaotic_mat = (chaotic_mat - chaotic_mat.min()) / (chaotic_mat.max() - chaotic_mat.min())
chaotic_mat *= np.max(np.abs(dct_img))

# Encrypt the chaotic matrix using XOR encryption
encryption_key = 12345
chaotic_mat_uint32 = chaotic_mat.astype(np.uint32)  # Convert chaotic map to integers
encrypted_mat_uint32 = np.bitwise_xor(chaotic_mat_uint32, encryption_key)
encrypted_mat = encrypted_mat_uint32.astype(np.float64)
```

Code Implementation

```
# ---------------- 4. Encrypt the DCT Coefficients ---------------- #
scrambled_dct = np.copy(dct_img)
non_zero_mask = scrambled_dct != 0
scrambled_dct[non_zero_mask] *= encrypted_mat[non_zero_mask]

# ---------------- 5. Decrypt the Encrypted Image ---------------- #
decrypted_mat_uint32 = np.bitwise_xor(encrypted_mat_uint32, encryption_key)
decrypted_mat = decrypted_mat_uint32.astype(np.float64)

# Rescale the decrypted matrix to ensure it's within a valid range
decrypted_mat = np.clip(decrypted_mat, 0, np.max(dct_img))

# Unscramble the non-zero DCT coefficients using the decrypted chaotic matrix
reconstructed_dct = np.copy(scrambled_dct)
reconstructed_dct[non_zero_mask] /= decrypted_mat[non_zero_mask]

# Reconstruct the approximation coefficients using IDCT
reconstructed_cA = idct(idct(reconstructed_dct, axis=0, norm='ortho'), axis=1, norm='ortho')
reconstructed_cA = np.clip(reconstructed_cA, 0, 255).astype(np.uint8)

# Perform inverse DWT to get the decrypted image
decrypted_img = pywt.idwt2((reconstructed_cA, (cH, cV, cD)), 'haar')
decrypted_img = np.clip(decrypted_img, 0, 255).astype(np.uint8)
```

Code Implementation

```python
# ---------------- Error Metrics ---------------- #
def calculate_metrics(original, compressed):
    # Resize the images to the same shape (if required)
    if original.shape != compressed.shape:
        compressed = cv2.resize(compressed, (original.shape[1], original.shape[0]))

    mse = np.mean((original - compressed) ** 2)
    rmse = sqrt(mse)
    psnr = 20 * log10(255 / sqrt(mse)) if mse != 0 else float('inf')
    ssim_index = ssim(original, compressed, data_range=compressed.max() - compressed.min())
    return mse, rmse, psnr, ssim_index

mse, rmse, psnr, ssim_index = calculate_metrics(img, decrypted_img)
```



Applications in Real Life

The integration of **compressed image sensing** and **GANs** has transformative applications in various domains. From **satellite imagery** to **medical diagnostics**, the ability to reconstruct images from minimal data can significantly improve efficiency and effectiveness in critical areas.

Future Work

- **Incorporation of GAN and CNN:**Use Generative Adversarial Networks (GANs) to reconstruct compressed images with higher visual quality by learning the complex patterns and distributions in the image data.Implement Convolutional Neural Networks (CNNs) to enhance compression by automatically learning optimal features and encoding them efficiently, reducing the reliance on manual transforms like DWT and DCT.
- **Robustness Against Attacks:**Test the system against various cryptographic attacks, such as brute force, statistical, and differential attacks, to enhance security.

References

- MNIST dataset: LeCun, Y., et al. (1998). "Gradient-Based Learning Applied to Document Recognition." Available at http://yann.lecun.com/exdb/mnist.
- A fast image encryption algorithm based on compressive sensing and hyper chaotic map by Qiaoyun Xu ,Kehui Sun ,Chun Cao, Congxu Zhu.
- A novel image encryption algorithm using chaotic compressive sensingand nonlinear exponential function by Farhan Musanna, Sanjeev Kumar.

Thanks!

DO YOU HAVE ANY QUESTIONS?