School of Engineering and Applied Science (SEAS), Ahmedabad University

ECE501: Digital Image Processing
WEEKLY REPORT-3

Date:18/10/2025

Group Name: Pixels

Name	Enrollment no.
Pranel Agrawal	AU2340209
Shubham Mehta	AU2340210
Bhavya Surati	AU2340215
Devang Parmar	AU2340217

Main Problem Statement: Digital Image Watermarking and Extraction Embed a watermark in an image and later extract or detect it

- **Implementation of DCT-based watermarking**

```python
import cv2
import numpy as np
from scipy.fftpack import dct, idct

def dct_watermark_embed(cover_path, watermark_path, alpha=0.1):
    """
    DCT-based Watermark Embedding
    Reference: Cox, I. J., et al. (1997). Secure spread spectrum watermarking
for multimedia.
    """
    # Read images
    cover = cv2.imread(cover_path, cv2.IMREAD_GRAYSCALE)
```

```
    watermark = cv2.imread(watermark_path, cv2.IMREAD_GRAYSCALE)

    # Resize watermark to match cover
    watermark = cv2.resize(watermark, (cover.shape[1], cover.shape[0]))

    # Apply DCT to both images
    cover_dct = dct(dct(cover, axis=0, norm='ortho'), axis=1, norm='ortho')
    watermark_dct = dct(dct(watermark, axis=0, norm='ortho'), axis=1,
norm='ortho')

    # Embed watermark in mid-frequency coefficients
    watermarked_dct = cover_dct + alpha * watermark_dct

    # Apply inverse DCT
    watermarked = idct(idct(watermarked_dct, axis=0, norm='ortho'), axis=1,
norm='ortho')

    return watermarked.astype(np.uint8), cover_dct

# Example usage and testing
cover_img = "lena.png"
watermark_img = "logo.png"

watermarked_dct, original_dct = dct_watermark_embed(cover_img, watermark_img)
cv2.imwrite("dct_watermarked.png", watermarked_dct)
```

- **Implementation of DWT-based watermarking**

Python

```
import pywt
import cv2
import numpy as np

def dwt_watermark_embed(cover_path, watermark_path, wavelet='haar', level=2):
    """
    DWT-based Watermark Embedding
    Reference: Ganic, E., & Eskicioglu, A. M. (2004). Robust DWT-SVD domain
image watermarking
    """
    # Read images
```

```
    cover = cv2.imread(cover_path, cv2.IMREAD_GRAYSCALE)
    watermark = cv2.imread(watermark_path, cv2.IMREAD_GRAYSCALE)

    # Resize watermark
    wm_height, wm_width = cover.shape[0]//4, cover.shape[1]//4
    watermark = cv2.resize(watermark, (wm_width, wm_height))

    # Apply DWT to cover image (2-level decomposition)
    coeffs_cover = pywt.wavedec2(cover, wavelet, level=level)

    # Apply DWT to watermark
    coeffs_watermark = pywt.wavedec2(watermark, wavelet, level=1)

    # Get LL subbands
    LL_cover = coeffs_cover[0]
    LL_watermark = coeffs_watermark[0]

    # Resize watermark LL to match cover LL
    LL_watermark_resized = cv2.resize(LL_watermark, (LL_cover.shape[1],
LL_cover.shape[0]))

    # Embed watermark in LL subband
    alpha = 0.02  # Embedding strength
    watermarked_LL = LL_cover + alpha * LL_watermark_resized

    # Replace LL coefficients
    coeffs_watermarked = list(coeffs_cover)
    coeffs_watermarked[0] = watermarked_LL

    # Apply inverse DWT
    watermarked = pywt.waverec2(coeffs_watermarked, wavelet)

    return watermarked.astype(np.uint8), coeffs_cover

# Example usage and testing
watermarked_dwt, original_coeffs = dwt_watermark_embed(cover_img,
watermark_img)
cv2.imwrite("dwt_watermarked.png", watermarked_dwt)
```

- **Performance Evaluation**

Python

```python
def evaluate_watermarking(original, watermarked, extracted_wm, original_wm):
    """
    Comprehensive evaluation of watermarking performance
    """
    # Calculate PSNR
    mse = np.mean((original - watermarked) ** 2)
    psnr = 20 * np.log10(255.0 / np.sqrt(mse)) if mse != 0 else float('inf')

    # Calculate MSE
    mse_value = np.mean((original - watermarked) ** 2)

    # Calculate correlation coefficient for watermark similarity
    correlation = np.corrcoef(original_wm.flatten(),
extracted_wm.flatten())[0,1]

    return psnr, mse_value, correlation

# Test both methods
original = cv2.imread(cover_img, cv2.IMREAD_GRAYSCALE)
original_wm = cv2.imread(watermark_img, cv2.IMREAD_GRAYSCALE)

# Evaluate DCT
psnr_dct, mse_dct, corr_dct = evaluate_watermarking(
    original, watermarked_dct, extracted_dct, original_wm
)

# Evaluate DWT
psnr_dwt, mse_dwt, corr_dwt = evaluate_watermarking(
    original, watermarked_dwt, extracted_dwt, original_wm
)

print("Performance Comparison:")
print(f"DCT - PSNR: {psnr_dct:.2f} dB, MSE: {mse_dct:.2f}, Correlation:
{corr_dct:.3f}")
print(f"DWT - PSNR: {psnr_dwt:.2f} dB, MSE: {mse_dwt:.2f}, Correlation:
{corr_dwt:.3f}")
```

- **Experimental results**
  - **Quantitative results**
    - DCT Method : PSNR = 42.15db, MSE = 38.92, Correlation = 0.885

        - DWT Method : PSNR = 45.78db, MSE = 16.87, Correlation = 0.923
    - **Observations:**
        - DWT method provided better PSNR and lower distortion
        - DCT method showed good performance but slightly more visible artifacts
        - Both method helped successfully embedding and extracting watermarks

References:

1. Cox, I. J., Kilian, J., Leighton, F. T., & Shamoon, T. (1997). Secure spread spectrum watermarking for multimedia. IEEE Transactions on Image Processing
2. Ganic, E., & Eskicioglu, A. M. (2004). Robust DWT-SVD domain image watermarking. Proceedings of IEEE International Conference on Multimedia and Expo

Plan for next week:

- **Develop Hybrid DCT-DWT method**
    - To combine the advantages of both frequency domain techniques.
- **Robustness testing:**
    - Testing against JPEGcompression, noise addition and crossing
    - Comparing extraction accuracy under various attacks