Image Compression

1. Prediction-Based Coding (Lossless)

Based on lecture pages **10–13**, prediction coding reduces image entropy by coding *differences* instead of absolute pixel values.

Objective

Students implement simple 1-D and 2-D predictors and compute the **prediction error image**.

Tasks

1. Load a grayscale image.
2. Predict each pixel using one of the predictors from the slides (page 10):
   - Predictor A: Left neighbor
   - Predictor C: Upper neighbor
   - Predictor (A + B – C), etc.
3. Compute the **difference/error image**:

   error = original − predicted

4. Display:
   - original image
   - predicted image
   - error image

Python functions to use

- `cv2.imread()`
- `numpy.roll()` or direct indexing
- Display using `matplotlib.pyplot.imshow()`

---

2. DCT-Based Compression (JPEG-Like)

This corresponds to lecture pages **14–23**, including DCT, quantization, zigzag, etc.

Objective

Students apply:

1. **Blockwise 8×8 DCT**
2. **Quantization / Dequantization**
3. **Inverse DCT**
4. Observe compression artifacts

Tasks

1. Convert image to grayscale.
2. Split into 8×8 blocks.
3. For each block:
   - Apply **2D DCT**
   - Apply a quantization matrix Q (provided)
   - Apply **inverse quantization**
   - Apply **Inverse DCT**
4. Reconstruct the image and compare with the original.

Python functions to use

- **DCT:**
  - `cv2.dct(block.astype(np.float32))`
- **Inverse DCT:**
  - `cv2.idct(block)`
- **Block processing:**
  - `numpy.reshape()`
  - manual looping over blocks
- **Visualization:**
  - `matplotlib.pyplot.imshow()`

Expected Student Outputs

- Original image
- DCT coefficient images (log-scaled)
- Reconstructed image after quantization
- Difference image

---

3. Wavelet Compression (JPEG2000-Like)

Based on lecture pages **34–70**, especially DWT decomposition and subbands.

Objective

Students implement:

1. **2D Wavelet transform**

2. **Coefficient quantization**
3. **Inverse Wavelet transform**
4. Understand LL, LH, HL, HH subbands

Tasks

1. Load grayscale image.
2. Apply 1-level and 2-level **2D DWT** using the Haar or 9/7 wavelet.
3. Zero or quantize some detail coefficients (LH, HL, HH).
4. Reconstruct using inverse DWT.
5. Compare compressed vs. original.

Python functions to use

Using **PyWavelets (pywt)**:

- **Forward DWT:**
  - `pywt.dwt2(image, 'haar')`
    or
  - `pywt.wavedec2(image, 'bior4.4', level=2)` (JPEG2000-like)
- **Inverse DWT:**
  - `pywt.idwt2(coeffs, 'haar')`
    or
  - `pywt.waverec2(coeffs, 'bior4.4')`

Expected Student Outputs

- Subbands: LL, LH, HL, HH (display each)
- Quantized wavelet coefficients
- Reconstructed image
- Comparison to original

Use the following measure to measure the image distortion:

the total error between the two images (whose sizes are $M \times N$) is

$$E = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} |f'(x, y) - f(x, y)|$$

The RMS error, e_{rms}, between $f(x, y)$ and $f'(x, y)$ can be calculated by

$$e_{\mathrm{rms}} = \sqrt{\frac{1}{MN} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[f'(x, y) - f(x, y) \right]^2}$$

If we consider the resulting image, $f'(x, y)$, as the "signal" and the error as "noise," we can define the RMS signal to noise ratio (SNR_{rms}) between modified and original images as

$$\mathrm{SNR}_{\mathrm{rms}} = \sqrt{\frac{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f'(x, y)^2}{\sum_{x=0}^{M-1} \sum_{y=0}^{N-1} \left[f'(x, y) - f(x, y) \right]^2}}$$

If we express the SNR as a function of the peak value of the original image and the RMS error (equation (17.5)), we obtain another metric, the PSNR, defined as

$$\mathrm{PSNR} = 10 \log_{10} \frac{(L-1)^2}{(e_{\mathrm{rms}})^2}$$

where L is the number of gray levels (for 8 bits/pixel monochrome images, $L = 256$).