To understand and implement the process of 2D-DCT (Discrete Cosine Transform) and its inverse (2D-IDCT) for image compression, we'll break down the entire process step-by-step. This guide will also highlight the relevant concepts and definitions you'll encounter in your homework.

### Overview of Image Compression Using DCT

Image compression aims to reduce the amount of data required to represent an image while maintaining acceptable quality. The DCT is widely used in image compression (like JPEG) because it separates the image into parts of differing importance, allowing for lossy compression. Here's a broad overview of the steps involved:

1. \*\*Convert the Image to Grayscale\*\*

2. \*\*Apply 2D-DCT\*\*

3. \*\*Visualize DCT Coefficients\*\*

4. \*\*Reconstruct the Image Using 2D-IDCT\*\*

5. \*\*Evaluate the PSNR (Peak Signal-to-Noise Ratio)\*\*

6. \*\*Implement Fast Algorithm using Two 1D-DCT\*\*

7. \*\*Compare Runtime between 2D-DCT and Two 1D-DCT\*\*

### Step-by-Step Process

#### 1. Convert the Image to Grayscale

\*\*Knowledge Applied: Color Models\*\*

- \*\*Concept\*\*: Images are typically represented in color (RGB), where each pixel has three values (red, green, blue). For DCT, we often convert the image to grayscale, which simplifies the data to a single channel.

- \*\*Implementation\*\*: Use a library (like OpenCV in Python) to read the image and convert it to grayscale.

\*\*Code Example\*\*:

```python

import cv2

# Load the image

image = cv2.imread('lena.png')

# Convert to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

```

#### 2. Apply 2D-DCT

\*\*Knowledge Applied: Discrete Cosine Transform\*\*

- \*\*Concept\*\*: The DCT transforms the spatial domain (pixel values) into the frequency domain. It separates the image into a sum of cosine functions of varying magnitudes and frequencies.

- \*\*Implementation\*\*: Implement the 2D-DCT manually (without using library functions) by using the DCT formula. For an image of size \(N \times M\):

\[

C(u, v) = \alpha(u) \alpha(v) \sum\_{x=0}^{N-1} \sum\_{y=0}^{M-1} f(x, y) \cos\left(\frac{(2x + 1)u\pi}{2N}\right) \cos\left(\frac{(2y + 1)v\pi}{2M}\right)

\]

where \(f(x,y)\) is the pixel value.

\*\*Code Example\*\*:

```python

import numpy as np

def dct2(a):

return np.dot(np.dot(np.cos(np.pi \* (2 \* np.arange(a.shape[0])[:, None] + 1) \* np.arange(a.shape[0]) / (2 \* a.shape[0]))), a),

np.cos(np.pi \* (2 \* np.arange(a.shape[1]) + 1) \* np.arange(a.shape[1]) / (2 \* a.shape[1])))

dct\_image = dct2(gray\_image)

```

#### 3. Visualize DCT Coefficients

\*\*Knowledge Applied: Logarithmic Scaling for Visualization\*\*

- \*\*Concept\*\*: DCT coefficients contain a wide range of values. Logarithmic scaling is often used for visualization to emphasize smaller coefficients.

- \*\*Implementation\*\*: Scale and clip the DCT coefficients for better visualization.

\*\*Code Example\*\*:

```python

import matplotlib.pyplot as plt

# Scale and clip the DCT coefficients

dct\_image\_log = np.log1p(np.abs(dct\_image)) # Use log(1 + |x|) for scaling

plt.imshow(dct\_image\_log, cmap='gray')

plt.title('DCT Coefficients')

plt.colorbar()

plt.show()

```

#### 4. Reconstruct the Image Using 2D-IDCT

\*\*Knowledge Applied: Inverse Discrete Cosine Transform\*\*

- \*\*Concept\*\*: The IDCT reconstructs the image from its DCT coefficients. The formula is the inverse of DCT.

- \*\*Implementation\*\*: Implement the 2D-IDCT manually using the inverse formula.

\*\*Code Example\*\*:

```python

def idct2(a):

return np.dot(np.dot(a, np.cos(np.pi \* (2 \* np.arange(a.shape[0])[:, None] + 1) \* np.arange(a.shape[0]) / (2 \* a.shape[0]))),

np.cos(np.pi \* (2 \* np.arange(a.shape[1]) + 1) \* np.arange(a.shape[1]) / (2 \* a.shape[1])))

reconstructed\_image = idct2(dct\_image)

plt.imshow(reconstructed\_image, cmap='gray')

plt.title('Reconstructed Image')

plt.show()

```

#### 5. Evaluate the PSNR

\*\*Knowledge Applied: Peak Signal-to-Noise Ratio\*\*

- \*\*Concept\*\*: PSNR measures the quality of the reconstructed image compared to the original image. It is defined as:

\[

PSNR = 10 \cdot \log\_{10}\left(\frac{MAX\_I^2}{MSE}\right)

\]

where \(MAX\_I\) is the maximum possible pixel value, and MSE (Mean Squared Error) is calculated between the original and reconstructed images.

- \*\*Implementation\*\*: Calculate PSNR using the original and reconstructed images.

\*\*Code Example\*\*:

```python

def psnr(original, reconstructed):

mse = np.mean((original - reconstructed) \*\* 2)

max\_pixel = 255.0

return 10 \* np.log10(max\_pixel \*\* 2 / mse)

psnr\_value = psnr(gray\_image, reconstructed\_image)

print(f'PSNR: {psnr\_value} dB')

```

#### 6. Implement Fast Algorithm using Two 1D-DCT

\*\*Knowledge Applied: Efficient Computation Techniques\*\*

- \*\*Concept\*\*: Instead of computing the 2D-DCT directly, you can compute the DCT along one dimension (rows) and then along the other dimension (columns). This reduces computational complexity.

- \*\*Implementation\*\*: Implement a function to perform 1D-DCT and then apply it twice.

\*\*Code Example\*\*:

```python

def dct1d(x):

N = len(x)

return np.array([sum(x[k] \* np.cos(np.pi \* (2 \* j + 1) \* k / (2 \* N)) for k in range(N)) for j in range(N)])

def dct2\_fast(image):

# Apply DCT row-wise and then column-wise

dct\_row = np.array([dct1d(row) for row in image])

dct2\_fast\_image = np.array([dct1d(col) for col in dct\_row.T]).T

return dct2\_fast\_image

dct\_fast\_image = dct2\_fast(gray\_image)

```

#### 7. Compare Runtime between 2D-DCT and Two 1D-DCT

\*\*Knowledge Applied: Performance Analysis\*\*

- \*\*Concept\*\*: Measuring the execution time of both methods gives insight into their efficiency. Python's `time` module can be used for this purpose.

- \*\*Implementation\*\*: Measure and compare the time taken for both methods.

\*\*Code Example\*\*:

```python

import time

# Measure time for 2D-DCT

start\_time = time.time()

dct\_image = dct2(gray\_image)

end\_time = time.time()

print(f'Time for 2D-DCT: {end\_time - start\_time} seconds')

# Measure time for Two 1D-DCT

start\_time = time.time()

dct\_fast\_image = dct2\_fast(gray\_image)

end\_time = time.time()

print(f'Time for Two 1D-DCT: {end\_time - start\_time} seconds')

```

### Summary of Knowledge Topics Applied

- \*\*Color Models\*\*: Understanding how to convert RGB images to grayscale.

- \*\*Discrete Cosine Transform (DCT)\*\*: Learning how DCT works and its mathematical foundation.

- \*\*Logarithmic Scaling\*\*: Using logarithmic functions for better visualization of DCT coefficients.

- \*\*Inverse DCT\*\*: Understanding the reconstruction of images from their DCT coefficients.

- \*\*Peak Signal-to-Noise Ratio (PSNR)\*\*: Evaluating image quality after reconstruction.

- \*\*Efficient Computation Techniques\*\*: Implementing algorithms that reduce computational complexity.

- \*\*Performance Analysis\*\*: Measuring and comparing execution times for different algorithms.

### Submission Instructions

Once you have completed the implementation, ensure to follow these steps for submission:

1. \*\*Create a PDF Report\*\*:

- Summarize the steps, explain the code, include visualizations, and document the PSNR results.

- Name it `VC\_HW2\_[student\_id].pdf`.

2. \*\*Prepare Your Code\*\*:

- Zip all source code files along with a README file explaining how to run your code.

- Name it `VC\_HW2\_[student\_id].zip`.

3. \*\*Upload to E3\*\*: Ensure both files are correctly named and uploaded before the deadline.

By following these steps, you will not only complete your homework but also develop a deeper understanding of image compression using DCT and its practical implementations. If you have any further questions or need clarification on any specific points, feel free to ask!