Discrete Cosine Transform (DCT)

Devesh Pant(MA24M007),Biswajit Gorai(MA24M005),Anurag Bantu(MA24M003),Aditya Kumar(MA24M001)

Introduction to DCT

- The Discrete Cosine Transform (DCT) transforms a sequence of data points into a sum of cosine functions oscillating at different frequencies.
- Commonly used in image and audio compression (e.g., JPEG, MP3).
- Efficient at energy compaction, concentrating most of the signal's information in a few coefficients.

1D DCT-II Definition

Formula:

$$X_k = \alpha(k) \sum_{n=0}^{N-1} x_n \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right) k\right) \tag{1}$$

Where:

- x_n: Original data point at index n.
- X_k: DCT coefficient at index k.
- $\alpha(k)$: Normalization factor,

$$\alpha(k) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } k = 0 \\ \sqrt{\frac{2}{N}} & \text{if } k > 0 \end{cases} \tag{2}$$

2D DCT-II for Images

Formula:

$$X_{k,l} = \alpha(k)\alpha(l) \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x_{m,n} \cos\left(\frac{\pi}{M}\left(m+\frac{1}{2}\right)k\right) \cos\left(\frac{\pi}{N}\left(n+\frac{1}{2}\right)l\right) \tag{3}$$

Where:

- $x_{m,n}$: Pixel value at position (m, n).
- $X_{k,l}$: DCT coefficient at position (k, l).
- $\alpha(k)$ and $\alpha(l)$: Normalization factors.

Properties of DCT

- **Orthogonality:** Cosine functions are orthogonal, meaning frequency components are independent.
- **Energy Compaction:** Most of the signal's energy is concentrated in the low-frequency coefficients.
- **Efficiency:** Useful for compression as high-frequency coefficients can often be discarded.

Inverse Discrete Cosine Transform (IDCT)

1D IDCT Formula:

$$x_n = \sum_{k=0}^{N-1} \alpha(k) X_k \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right) \tag{4}$$

2D IDCT Formula:

$$x_{m,n} = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} \alpha(k)\alpha(l) X_{k,l} \cos\left(\frac{\pi}{M}\left(m + \frac{1}{2}\right)k\right) \cos\left(\frac{\pi}{N}\left(n + \frac{1}{2}\right)l\right) \tag{5}$$

Signal Compression and Reconstruction Using DCT

In this example, we demonstrate signal compression and reconstruction using the Discrete Cosine Transform (DCT).

- We generate a sine wave as the sample signal.
- Apply the DCT to compress the signal by keeping a small number of coefficients.
- Reconstruct the signal from the compressed data using the inverse DCT.
- Visualize the original, compressed, and reconstructed signals.

Python Code Example (DCT Signal Compression - Part 1)

Here is the Python code for compressing and reconstructing a signal using the DCT:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct

# Generate a sample signal (sine wave)
fs = 100  # Sampling frequency
t = np.linspace(0, 1, fs, endpoint=False)  # Time vector
frequency = 5  # Frequency of the sine wave
signal = np.sin(2 * np.pi * frequency * t)  # Original signal

# Function to compress the signal
def compress_signal(signal, compression_ratio):
    # Perform DCT
    transformed_signal = dct(signal, norm='ortho')

    # Determine the number of coefficients to keep
    num_coefficients = int(len(transformed_signal) * compression_ratio)
```

Python Code Example (DCT Signal Compression - Part 2)

```
    # Zero out the rest of the coefficients
    compressed_signal = np.zeros_like(transformed_signal)
    compressed_signal[:num_coefficients] = transformed_signal[:num_coefficients]

    return compressed_signal

# Function to reconstruct the signal from compressed data
def reconstruct_signal(compressed_signal):
    # Perform inverse DCT
    reconstructed_signal = idct(compressed_signal, norm='ortho')
    return reconstructed_signal

# Compress the signal
compression_ratio = 0.1  # Keep 10% of the coefficients
compressed_signal = compress_signal(signal, compression_ratio)

# Reconstruct the signal
reconstructed_signal = reconstruct_signal(compressed_signal)

# Plot the original and reconstructed signals
plt.figure(figsize=(12, 6))
plt.subplot(3, 1, 1)
plt.plot(t, signal)
plt.title('Original Signal')
plt.grid()
```

Python Code Example (DCT Signal Compression - Part 3)

```
plt.subplot(3, 1, 2)
plt.plot(t, compressed_signal)
plt.title('Compressed Signal (DCT Coefficients)')
plt.grid()

plt.subplot(3, 1, 3)
plt.plot(t, reconstructed_signal)
plt.title('Reconstructed Signal')
plt.grid()

plt.tight_layout()
plt.show()
```

Results Visualization

- The original signal is a simple sine wave with a frequency of 5 Hz.
- After compression, most of the DCT coefficients are set to zero, keeping only the most important 10
- The reconstructed signal closely resembles the original, but with some minor losses due to the compression ratio(figure 1).

Figure 1

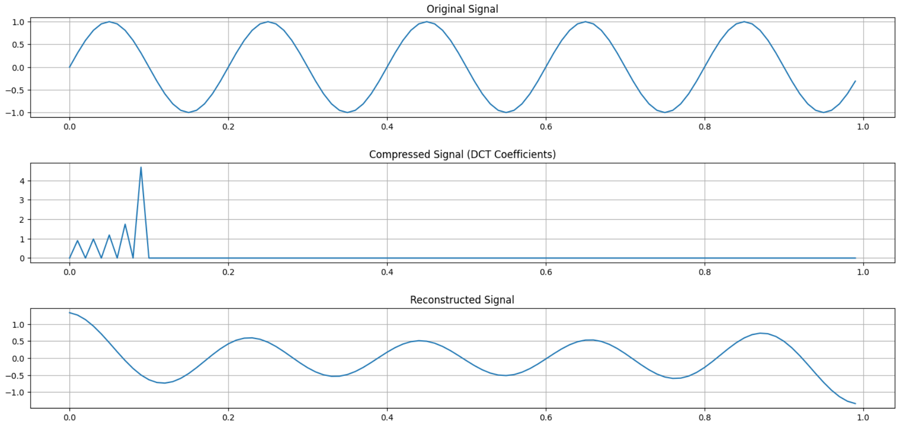

Original Signal
Compressed Signal (DCT Coefficients)
Reconstructed Signal
1.0
0.5
0.0
−0.5
−1.0
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

Image Compression Using 2D DCT in RGB Channels

This slide presents the compression of a color image using the 2D Discrete Cosine Transform (DCT) applied to each RGB channel.

- Load a color image in RGB format.
- Apply 2D DCT to each of the Red (R), Green (G), and Blue (B) channels.
- Compress the image by keeping only the top-left 100x100 DCT coefficients.
- Reconstruct the image using the Inverse DCT (IDCT).
- Visualize the original and the compressed image side-by-side.

Python Code Example (Image Compression Using 2D DCT)

Below is the Python code for compressing and reconstructing an image using the 2D DCT on each RGB channel:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import dct, idct
import cv2

# Load the color image in RGB format
image = cv2.imread('armas.jpg')
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Separate the color channels
R, G, B = cv2.split(image_rgb)

# Apply 2D DCT to each channel
dct_R = dct(dct(R.T, norm='ortho').T, norm='ortho')
dct_G = dct(dct(G.T, norm='ortho').T, norm='ortho')
dct_B = dct(dct(B.T, norm='ortho').T, norm='ortho')
```

Python Code Example (Image Compression Using 2D DCT - Part 2)

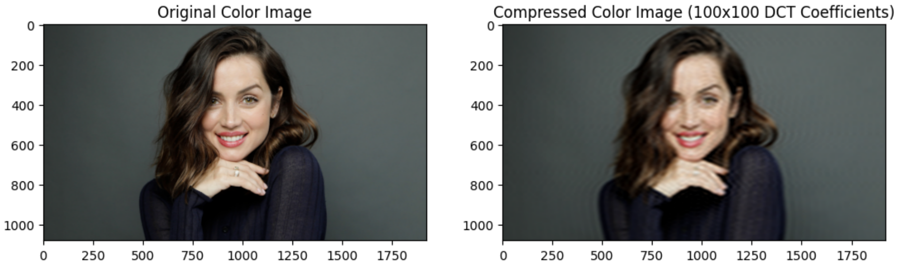
```
# Compress by keeping only the top-left 100x100 DCT coefficients
compressed_dct_R = np.zeros_like(dct_R)
compressed_dct_R[:100, :100] = dct_R[:100, :100]
compressed_dct_G = np.zeros_like(dct_G)
compressed_dct_G[:100, :100] = dct_G[:100, :100]
compressed_dct_B = np.zeros_like(dct_B)
compressed_dct_B[:100, :100] = dct_B[:100, :100]

# Reconstruct each channel using Inverse DCT
reconstructed_R = idct(idct(compressed_dct_R.T, norm='ortho').T, norm='ortho')
reconstructed_G = idct(idct(compressed_dct_G.T, norm='ortho').T, norm='ortho')
reconstructed_B = idct(idct(compressed_dct_B.T, norm='ortho').T, norm='ortho')

# Merge the channels back into an RGB image
reconstructed_image = cv2.merge([reconstructed_R, reconstructed_G, reconstructed_B])
```

Visualization of Original and Compressed Image

After applying the DCT compression, we compare the original and compressed images.

- The original image is shown on the left.
- The compressed image (with only the top-left 100x100 DCT coefficients kept) is shown on the right(Figure 2).

Figure 2


Original Color Image
Compressed Color Image (100x100 DCT Coefficients)
0
200
400
600
800
1000
0
250
500
750
1000
1250
1500
1750

Practical Interpretation

- **Low-Frequency Coefficients:** Represent the average or smoothness of the signal or image.
- **High-Frequency Coefficients:** Represent fine details and edges.
- **Compression:** Discarding high-frequency components can reduce file size with minimal loss of quality.

Applications of DCT

- Image Compression (e.g., JPEG)
- Audio Compression (e.g., MP3)
- Video Compression (e.g., MPEG)
- Data Analysis and Signal Processing

Conclusion

- The DCT is a powerful tool for signal and image compression.
- It efficiently compacts energy into a few coefficients, enabling effective data reduction.
- Understanding DCT is crucial for modern multimedia technologies.