Name: Damya K. Shukla

Roll No: 23

Stream: AI&ML

Subject: Image Processing

Unit-4 Image Compression Assignment (Theory)

---

1. Why is Image Compression Necessary in Multimedia Applications? How Does Compression Affect Storage and Transmission Efficiency?

Answer:

Image compression is crucial in multimedia because uncompressed images demand significant storage space and bandwidth. Images contain millions of pixels, each requiring memory, leading to large file sizes. Without compression, multimedia applications would require substantial storage and network resources, limiting the efficiency of transmission and display.

Compression reduces file sizes by eliminating redundancy, allowing images to occupy less space and be transmitted faster. This is critical in applications like streaming, online photo storage, and video conferencing, where bandwidth and storage constraints must be managed. For example, an uncompressed HD image of 5 MB might be compressed to 500 KB, reducing both storage usage and upload/download times.

2. What is Redundancy in Image Compression? Explain Three Types of Redundancy.

Answer:

Redundancy in image compression refers to the unnecessary or repetitive data in an image that can be reduced or eliminated without affecting essential information. Reducing redundancy allows us to represent the image in a more compact form, which leads to compression.

Spatial Redundancy: This occurs due to similarities between neighboring pixel values in an image. Adjacent pixels often have similar colors, especially in images with gradual color changes or low-detail areas.

Spectral Redundancy: Found in color images where different color channels (like Red, Green, Blue) contain correlated data. For instance, in an RGB image, each color channel might have similar patterns, creating redundancy.

Temporal Redundancy: Found in video compression, where consecutive frames are similar, resulting in repeated information over time. Reducing this temporal redundancy can significantly compress video files.

3. What is Coding Redundancy? How Does It Help Reduce Image File Sizes? Provide Examples.

Answer:

Coding redundancy refers to assigning fewer bits to represent frequently occurring values (symbols) in an image and more bits for less common values. This reduces the average number of bits per pixel, helping in compression.

Example: Huffman coding is a technique that assigns shorter codes to common pixel values and longer codes to less common ones. For instance, in an image where black pixels are more common than white ones, Huffman coding assigns a shorter binary code to black pixels and a longer one to white pixels, reducing the total file size.

4. Explain Inter-pixel Redundancy and How It Is Exploited in Image Compression Algorithms. Provide Examples.

Answer:

Inter-pixel redundancy exists because neighboring pixels in an image often have similar or identical values. Compression algorithms reduce this redundancy by encoding the difference between pixels rather than each pixel value individually.

Examples of Methods to Reduce Inter-pixel Redundancy:

Run-Length Encoding (RLE): Compresses sequences of identical pixels by recording the pixel value and its frequency, which is effective in large, uniform areas.

Differential Pulse Code Modulation (DPCM): Encodes the difference between consecutive pixels. For example, rather than encoding two neighboring pixels as separate values, the algorithm records the difference between them, which is smaller and can be represented with fewer bits.

5. What is the Difference Between Lossy and Lossless Image Compression Techniques? Provide Examples of When Each Type Is Suitable.

Answer:

Lossy Compression: This type of compression discards some data to reduce file size, leading to a loss of detail. Lossy methods, like JPEG, are used in applications where high compression is necessary, and some quality loss is acceptable, such as in web images.

Advantages: Achieves much smaller file sizes.

Disadvantages: Leads to image quality loss and potential compression artifacts.

Examples: JPEG for photos on the web, MP3 for audio files.

Lossless Compression: Lossless compression retains all original data, preserving image quality, making it ideal for situations where data integrity is critical, such as medical or scientific images.

Advantages: Maintains original quality without any loss.

Disadvantages: Generally, larger file sizes than lossy compression.

Examples: PNG for web graphics, TIFF for medical imaging.

6. Define Compression Ratio and Provide an Example. What Other Metrics Help Assess Compression Quality?

Answer:

The Compression Ratio (CR) is a measure of how much an image has been compressed. It is calculated as:

Compression Ratio = Original File Size / Compressed File Size

Example: For an image of 10 MB that compresses to 2 MB, the compression ratio is 5:1.

Other Metrics:

Peak Signal-to-Noise Ratio (PSNR): Assesses the quality of the compressed image by comparing it to the original. A higher PSNR indicates better quality.

Structural Similarity Index (SSIM): Measures perceived quality based on structural information, offering a better indication of visual quality than PSNR alone.

7. Discuss the Pros and Cons of the Following Compression Algorithms:

Answer:

I. Huffman Coding

Pros: Simple, easy to implement, and effective for data with predictable frequency distributions.

Cons: Limited efficiency for small symbol sets or when data lacks a clear probability distribution.

II. Arithmetic Coding

Pros: More efficient than Huffman coding for data with slight probability variations, as it encodes an entire message as a single interval.

Cons: Complex implementation and higher computational cost, especially for real-time applications.

III. LZW Coding

Pros: Effective for data with repeated patterns and widely used in formats like GIF.

Cons: Less effective for data with little repetition, leading to minimal compression.

IV. Transform Coding (e.g., DCT)

Pros: Compresses data by reducing frequency redundancies, widely used in JPEG compression.

Cons: Computationally intensive and may introduce artifacts in lossy settings if block sizes are too small.

V. Run-Length Coding (RLE)

Pros: Simple and efficient for areas of uniform color, commonly used in simple graphics.

Cons: Not effective for complex or highly detailed images, where redundancy is minimal.

8. Perform Huffman Coding on a Set of Pixel Values and Calculate the Compression Ratio

Given pixel values and frequencies:

A (5), B (9), C (12), D (13), E (16), F (45)

Answer:

To perform Huffman coding on the given set of pixel values and calculate the compression ratio, we'll follow these steps:

Step 1: Given Pixel Frequencies

The symbols and their frequencies are:

A: 5

B: 9

C: 12

D: 13

E: 16

F: 45

Step 2: Building the Huffman Tree

Sort the symbols based on frequency in ascending order: A (5), B (9), C (12), D (13), E (16), F (45).

Combine nodes iteratively to create the Huffman tree:

First, combine A (5) and B (9) to form a node with a combined frequency of 14.

Now we have nodes with frequencies: 14, 12, 13, 16, 45.

Combine C (12) and D (13) to form a node with frequency 25.

Now we have nodes with frequencies: 14, 16, 25, 45.

Combine the node with frequency 14 (from A and B) and E (16) to form a new node with frequency 30.

Now we have nodes with frequencies: 25, 30, 45.

Combine the nodes with frequencies 25 (from C and D) and 30 (from A, B, and E) to form a node with frequency 55.

Now we have two nodes left with frequencies 45 and 55.

Finally, combine these last two nodes, with frequencies 45 (F) and 55, to form the root node with a total frequency of 100.

Assign binary codes based on tree structure:

Starting from the root, assign 0 to the left branches and 1 to the right branches until reaching each symbol.

Step 3: Assign Huffman Codes

Following the Huffman tree, the codes for each symbol are:

F = 0

C = 100

D = 101

A = 1100

B = 1101

E = 111

Step 4: Calculate Total Bits After Huffman Coding

Using the Huffman codes and their frequencies, we calculate the total number of bits needed:

Total Bits =(5×4)+(9×4)+(12×3)+(13×3)+(16×3)+(45×1)

Total Bits=(5×4)+(9×4)+(12×3)+(13×3)+(16×3)+(45×1)

Breaking it down:

A (5 occurrences, 4 bits each) = 20 bits

B (9 occurrences, 4 bits each) = 36 bits

C (12 occurrences, 3 bits each) = 36 bits

D (13 occurrences, 3 bits each) = 39 bits

E (16 occurrences, 3 bits each) = 48 bits

F (45 occurrences, 1 bit each) = 45 bits

Adding these up gives:

Total Compressed Bits=20+36+36+39+48+45=224 bits

Total Compressed Bits=20+36+36+39+48+45=224 bits

Step 5: Calculate Total Uncompressed Bits

If each symbol were encoded with a fixed-length binary code, we would need

$\lceil \log_2(6) \rceil$ = 3 bits per symbol (since there are 6 symbols). With a total frequency count of 100 symbols:

Total Uncompressed Bits = 100×3= 300 bits

Step 6: Calculate the Compression Ratio

The compression ratio is the ratio of uncompressed to compressed size:

Compression Ratio=Uncompressed Bits/Compressed Bits = 300/224 ≈ 1.34

Summary

Uncompressed Bits: 300

Compressed Bits (Huffman): 224

Compression Ratio: 1.34:1

Huffman coding reduces the data size by approximately 1.34 times, achieving significant compression.

9. Explain Arithmetic Coding and How It Differs from Huffman Coding. Why Is Arithmetic Coding More Efficient in Some Cases?

Answer:

Arithmetic coding represents an entire message as a range within [0, 1), encoding more efficiently by adjusting interval size to symbol probabilities. Unlike Huffman coding, which uses fixed integer-length codes, arithmetic coding encodes the probability distribution precisely, allowing finer compression.

Efficiency Reason: Arithmetic coding performs better with symbols of similar probability, as it adapts interval lengths continuously, which Huffman coding cannot achieve with fixed-length binary representations.

10. Provide an Example of LZW Coding on a Sequence of Pixel Values.

Answer:

1. **Initialize the Dictionary**: Start with each unique symbol in the sequence. In this example:
    - A is assigned code 65 (ASCII value for A).
    - B is assigned code 66 (ASCII value for B).

So, the initial dictionary contains:

A: 65, B: 66

2. **Encoding Process**:
    - Go through each symbol in the sequence and check if the current sequence is already in the dictionary.
    - If a sequence is in the dictionary, move to the next symbol to expand the sequence.

- If the sequence is not in the dictionary, assign it the next available code, add it to the dictionary, and start a new sequence with the last symbol.

3. **Step-by-Step Encoding of "ABABABA"**:
    - Start with A, which is in the dictionary, so we output 65.
    - Next is B, also in the dictionary, so we output 66.
    - Form AB, which is not in the dictionary, so we add it with code 256.
    - The next sequence is BA, also not in the dictionary, so we add BA with code 257.
    - Finally, we form ABA, which isn't in the dictionary, so we add it with code 258.
4. **Result**:
    - The **Encoded Output** is: [65, 66, 256, 258, 65].
    - The **Final Dictionary** created during encoding:

A: 65

B: 66

AB: 256

BA: 257

ABA: 258

Summary:

The encoded output [65, 66, 256, 258, 65] represents the compressed version of "ABABABA". The dictionary now includes patterns that appear repeatedly in the sequence, allowing for efficient compression. This example shows how LZW builds a dictionary dynamically as it processes the input, achieving data compression by encoding repeated patterns with shorter codes.

11. What is Transform Coding, and How Does It Help Compress Images by Reducing Redundancies in the Frequency Domain?

Answer:

Transform coding is a key technique in image compression that reduces data size by transforming image information from the spatial domain (where data is represented as individual pixels) to the frequency domain (where data is represented by frequency components). This shift makes it possible to capture essential information more compactly, exploiting the way image data is structured and perceived.

How Transform Coding Works:

Conversion to Frequency Domain:

Transform coding methods, like the Discrete Cosine Transform (DCT), analyze an image by converting it from its original pixel-based form (spatial domain) into a form based on frequencies

(frequency domain). In this domain, different frequencies represent different types of image information:

Low frequencies: Capture broad, smooth regions and essential structural features.

High frequencies: Capture fine details and edges, but are often less noticeable to the human eye.

Identifying Redundancies:

In the frequency domain, image redundancies become more noticeable. Large areas of similar colors or smooth gradients, which contribute little detailed information, are represented by low-frequency components. These areas can be compressed significantly because they don't require high-frequency detail to look clear to the human eye.

Transform coding leverages this by encoding only the most essential frequencies. In many images, much of the detail carried by high-frequency components is unnecessary for viewers, allowing these details to be minimized or removed during compression.

Data Compression via Frequency Reduction:

In transform coding, only the most important frequencies are kept, typically the low-frequency ones, which hold the majority of the image's visual information. Higher frequencies are discarded or simplified, resulting in a compressed image that retains its main structure and visual integrity.

For instance, in JPEG compression (a commonly used method employing DCT), the transformation allows high frequencies (fine details) to be quantized to zero. This results in minimal visual loss but achieves a high compression ratio.

Advantages of Transform Coding in Image Compression:

Compression Efficiency: By discarding high-frequency components, transform coding reduces the amount of data needed to represent an image, leading to significant storage and bandwidth savings.

Adaptability to Human Perception: The human eye is more sensitive to low-frequency information (like shape and contrast) than to high-frequency details. Transform coding aligns with this by preserving what's most noticeable while simplifying or discarding what's less important.

Noise Reduction: Since high-frequency components often contain noise or minor details, removing or simplifying them in the compression process can enhance the overall visual quality of the image.

Example of Transform Coding in Practice: In JPEG compression, each 8x8 block of pixels is transformed using DCT, and only the most essential frequency components are retained. This allows JPEG to achieve high compression ratios while still maintaining good image quality, especially in photographs and natural images with smooth gradients.

Summary:

Transform coding is a powerful method in image compression because it reduces data by focusing on significant frequencies, leveraging human visual perception. By keeping essential low frequencies and discarding unnecessary high frequencies, transform coding enables effective data reduction while maintaining image quality.

12. Discuss the significance of sub-image size selection and blocking in image compression. How do these factors impact compression efficiency and image quality?

Answer:

In image compression, sub-image size selection and blocking are essential in dividing an image into smaller, manageable sections, often called "blocks" or "sub-images." This process enables compression algorithms to focus on smaller areas of the image, which can be compressed individually. This method is crucial for techniques like JPEG compression, where images are commonly divided into 8x8 pixel blocks.

Impact on Compression Efficiency:

Localized Data Processing: By breaking down the image into blocks, the algorithm can identify and exploit redundancies within each block rather than needing to analyze the entire image at once. This localized approach improves processing efficiency.

Efficient Transform Application: Transform-based compression techniques, such as the Discrete Cosine Transform (DCT), operate on these blocks. Smaller block sizes allow for easier transformation and quantization, leading to better compression ratios.

Data Storage and Transmission: Smaller compressed blocks require less memory and bandwidth, which enhances storage efficiency and speeds up data transmission, especially important in applications like streaming and network transfers.

Impact on Image Quality:

Block Artifacts: While smaller blocks improve compression efficiency, they can sometimes introduce visual artifacts, known as "blockiness," especially if the compression level is high. This is because each block is compressed independently, leading to visible boundaries between blocks when the image is reconstructed.

Balance Between Quality and Compression: The choice of block size is a trade-off. Smaller blocks may allow for more detailed compression but can result in more visible artifacts. Larger blocks may improve visual consistency across the image but can reduce the overall compression ratio.

Adaptive Block Sizes: Modern algorithms often use variable block sizes to balance quality and compression based on image content. Areas with more detail might use smaller blocks, while less detailed areas can use larger blocks to minimize artifacts.

Summary: Choosing the correct block size in image compression is a balance between achieving high compression efficiency and maintaining image quality. Smaller blocks enable higher compression but may introduce artifacts, whereas larger blocks provide smoother images at the cost of lower compression ratios.

13. Explain the process of implementing Discrete Cosine Transform (DCT) using Fast Fourier Transform (FFT). Why is DCT preferred in image compression?

Answer:

The Discrete Cosine Transform (DCT) is widely used in image compression due to its ability to compact energy into fewer coefficients, which allows for effective data reduction. The DCT transforms spatial image data (pixel values) into frequency domain data, making it easier to separate high-frequency details from low-frequency components.

Implementing DCT using FFT:

Relationship Between DCT and FFT: Although DCT and FFT are different, they are mathematically related. FFT computes both sine and cosine components of a signal, while DCT focuses only on the cosine components. This makes DCT ideal for images, as they generally have more low-frequency (smooth) information than high-frequency details.

DCT Using FFT: To implement DCT using FFT, the input signal can be extended symmetrically (mirrored) to double its length. This symmetry allows the Fourier Transform to approximate the cosine-only DCT result. The FFT algorithm is then applied to this mirrored signal, and specific mathematical adjustments are made to obtain the final DCT values.

Computational Efficiency: Using FFT to approximate DCT reduces computational complexity, making it faster and more efficient. Since FFT is a well-optimized algorithm, it provides computational benefits in large-scale applications like image and video processing.

Why DCT is Preferred in Image Compression:

Energy Compaction: DCT tends to concentrate most of the image's energy in a few low-frequency components, which means many higher-frequency components (often representing minor details or noise) can be discarded without significantly affecting image quality.

Human Visual System Compatibility: The human eye is less sensitive to high-frequency changes in an image. DCT allows compression algorithms to retain low-frequency components (important for overall image structure) while discarding high-frequency components (less noticeable to viewers), making the compression more efficient.

Lossy Compression: DCT is ideal for lossy compression methods like JPEG, where some data loss is acceptable. By quantizing the high-frequency coefficients to zero, DCT-based compression effectively reduces data size without a substantial loss in visual quality.

Summary: The DCT, often implemented via FFT for efficiency, is preferred in image compression because it effectively compacts image energy into fewer coefficients, is aligned with human visual perception, and enables efficient lossy compression with minimal perceptual loss.

14. Describe how run-length coding is used in image compression, particularly for images with large areas of uniform color. Provide an example to illustrate your explanation.

Answer:

Run-length coding (RLC) is a simple and effective compression technique used to encode sequences of repeating values, particularly in images with large uniform color regions (e.g., graphics, icons, or simple color backgrounds). RLC compresses data by identifying consecutive pixels with the same value and encoding this sequence as a single value and a count.

How Run-Length Coding Works:

Identify Runs: RLC scans the image data to identify consecutive pixels with the same color value. Instead of storing each pixel individually, RLC stores the pixel value followed by the count of its consecutive occurrences.

Encode as (Value, Count): Each "run" (a sequence of identical values) is replaced with a pair (Value, Count), where "Value" is the pixel color, and "Count" is the number of times it repeats. This reduces the storage space needed, especially for large areas of uniform color.

Example of Run-Length Coding:

Suppose we have a row of pixels in an image: [A, A, A, B, B, C, C, C, C].

Using RLC, this sequence would be encoded as: [(A, 3), (B, 2), (C, 4)].

This representation compresses the data by capturing the repeating values as a single entry, reducing the total number of entries needed to store the row.

Advantages and Limitations:

Advantages: RLC is particularly effective in compressing images with large areas of solid color or patterns, as it can significantly reduce data size by encoding repetitive data concisely.

Limitations: RLC is less effective for images with a lot of detail or noise, as high-frequency changes in pixel values (e.g., photographs) lead to short runs, diminishing the compression benefit. This is why RLC is typically used in images with simple color schemes rather than complex, detailed images.

Summary: Run-length coding efficiently compresses images with large uniform regions by encoding consecutive identical pixels as single entries with their count. While it works well for simple images with repeating patterns, it is less effective for detailed or noisy images, where pixel values change frequently.