Digital Signal Processing

Mini-Project Report

Submitted by

Name : Ekta Sai Meda

Register Number : 220200019

E-mail ID : medaekta.s-26@scds.saiuniversity.edu.in

School of Study : CDS

Year of Study :2nd

Declaration

I hereby declare that the work presented in this mini-project report is my own. All sources used in this work have been properly cited and referenced.

<Signature>

<Name>

Date: _____

INTRODUCTION TO ENTROPY & DATA COMPRESSION

1. Problem Statement

In this report we will focuses on introduction to entropy and techniques for data compression

2. Motivation

The problem of image processing and compression is crucial due to the exponential growth in digital visual data. Effective approaches become crucial when dealing with resource limitations, bandwidth restrictions, and the requirement for real-time applications. In order to maximize storage, expedite data transfer, and guarantee data security, compression is essential. These methods are essential in domains where high-resolution photographs are often used, such as medicine and remote sensing. They also have an effect on corporate operations, scientific research, and general economic efficiency. If implemented successfully, it has the potential to completely change how we process and interpret visual data in a variety of fields and applications.

3. Methodology

INFORMATION ENTROPY :

Introduction:

Entropy is a concept from information theory that measures the average amount of information or uncertainty contained in a set of data. It is often used to quantify the amount of randomness or unpredictability in a given data source. In the context of data compression, entropy is a fundamental concept. The entropy of a discrete random variable X with possible values {x1, x2, ..., xn} and

associated probabilities {p(x1), p(x2), ..., p(xn)} is defined by the following formula:

$$H(X) = -\Sigma(p(xi) * \log2(p(xi)))$$

where log2 denotes the logarithm base 2. The entropy is measured in bits, and it represents the minimum average number of bits required to encode each symbol from the source efficiently. In other words, it provides an upper bound on the compression ratio that can be achieved for the given data.

Intuitively, if a source has low entropy, it means that it has a high degree of predictability, and the data can be compressed efficiently because there is less information or uncertainty to convey. On the other hand, a high-entropy source implies that the data is less predictable, and compression may be less effective because more information needs to be transmitted.

Overall, entropy provides a quantitative measure of disorder, randomness, or uncertainty in different systems. It plays a crucial role in understanding the behavior of physical systems, the transmission and storage of information, and the overall organization or disorder in various processes.

Image Processing in Entropy :

Entropy in image processing measures the information content or randomness in an image. It is computed by analyzing the pixel intensity distribution through the image histogram. The probability distribution is derived from the histogram, representing the likelihood of encountering each pixel intensity. Entropy is then calculated as the sum of the product of each probability and its logarithm (base 2), weighted by the probability itself. Higher entropy values indicate greater complexity and diversity in the image, while lower values suggest more

repetitive or homogeneous patterns. Entropy is commonly used for tasks like image compression, segmentation, and quality assessment, providing insights into the image's content and information richness

Data compression :

Data compression is the process of reducing the size of data files or streams, making them occupy less storage space or require less transmission bandwidth. The goal of data compression is to eliminate or minimize redundant or irrelevant information in a dataset while preserving essential information.

Data compression has numerous benefits, including:

• Reduced Storage Requirements: Compressed data occupies less storage space, allowing for efficient utilization of storage resources. This is particularly useful when dealing with large datasets or limited storage capacities.

• Faster Data Transmission: Compressed data requires less bandwidth to transmit, leading to faster data transfer rates. This is crucial in scenarios where data needs to be transmitted over networks with limited bandwidth or in real-time applications.

• Cost Savings: By reducing storage requirements and transmission bandwidth, data compression can result in cost savings. It allows organizations to optimize their storage infrastructure and minimize network infrastructure expenses.

There are two main types of data compression techniques:

• Lossless Compression: Lossless compression is a type of data compression that reduces the size of data without losing any information. The compressed data can be completely reconstructed to its original form, bit for bit, without any loss or degradation. Lossless compression is commonly used for data types where preserving all details and accuracy is crucial, such as text documents, databases, program files, and any data that cannot tolerate even minor errors or alterations.

• *Lossy Compression: Lossy compression is a data compression technique that reduces the size of data by permanently discarding or approximating certain information that is considered less important or less perceptually significant. Unlike lossless compression, which preserves all the original data, lossy compression intentionally introduces some degree of loss or degradation in the compressed data. Lossy compression is commonly used for multimedia data, such as images, audio, and video, where minor losses in quality can be tolerated without significant impact on the overall perception or usefulness of the data. Some commonly used data compression algorithms and file formats include:*
Lossless Compression: ZIP, GZIP, PNG, FLAC
Lossy Compression: JPEG, MP3, MPEG, H.264

Image processing using DCT :
Introduction :
Image processing through the Discrete Cosine Transform (DCT) is a commonly used technique for compressing and manipulating digital images. The DCT is a mathematical transform that converts an image from the spatial domain to the frequency domain, allowing for efficient representation and manipulation of image data.
Here's an overview of the process:
1. Image Representation: Digital images are typically represented as a matrix of pixels, where each pixel contains information about its color or intensity. The image is divided into small, non-overlapping blocks, such as 8x8 or 16x16 pixels.
2. Color Space Conversion: If the image is in a color space such as RGB, it is often converted to a different color space, such as YUV or YCbCr. This conversion separates the luminance (Y) and chrominance (U and V or Cb and Cr) components of the image. The DCT is then applied separately to each component.

3. Blockwise DCT Transformation: Each block of the image is transformed using the DCT. The DCT operation calculates the frequency components of the image within the block. The DCT coefficients represent different frequencies, ranging from low to high.
4. Quantization: The DCT coefficients are quantized to reduce their precision and remove higher-frequency components that are less perceptually significant. Quantization involves dividing the coefficients by a quantization matrix, where the values determine the level of compression and quality. Higher compression is achieved by using larger quantization values, leading to greater loss of information.
5. Compression: After quantization, the quantized DCT coefficients are typically encoded using entropy coding techniques, such as Huffman coding or arithmetic coding, to achieve further compression. The encoding step assigns shorter codes to frequently occurring coefficients, reducing the overall data size.
6. Reconstruction: To reconstruct the image, the quantized DCT coefficients are inverse quantized, and the inverse DCT is applied to each block. The inverse DCT converts the coefficients back to the spatial domain, resulting in an approximate representation of the original image. DCT-based image processing is not limited to compression.
It is also used for various image manipulation tasks, such as:
• Image Enhancement: By modifying or filtering the DCT coefficients, specific frequency components can be emphasized or suppressed, leading to enhancements in image contrast or details.
• Image Watermarking: The DCT coefficients can be modified by embedding a watermark signal, allowing for copyright protection or authentication of digital images.
• Image Denoising: DCT coefficients can be analyzed to identify and suppress noise components, improving image quality by reducing noise artifacts.

• Image Compression Standards: DCT-based compression algorithms like JPEG and JPEG2000 are widely used in image compression standards to achieve efficient storage and transmission of digital images.
The Discrete Cosine Transform has become a fundamental tool in image processing, offering a balance between compression efficiency and visual quality. Its efficient frequency domain representation and manipulation capabilities make it suitable for a wide range of applications in image compression, analysis, and manipulation.

Image processing through SVD :
Introduction :
Image processing through Singular Value Decomposition (SVD) is another technique commonly used for various image manipulation tasks, including compression, noise reduction, and image reconstruction. SVD is a matrix factorization method that decomposes an image matrix into three matrices: U, Σ, and V^T, where U and V are orthogonal matrices, and Σ is a diagonal matrix containing singular values.
Here's an overview of image processing through SVD:
• Image Representation: Like in other image processing techniques, digital images are represented as matrices, where each element represents the pixel value of the corresponding location in the image.
• SVD Decomposition: The image matrix is decomposed using SVD, resulting in three matrices: U, Σ, and V^T. The U matrix represents the left singular vectors, Σ contains the singular values, and V^T represents the right singular vectors.
• Compression: SVD allows for image compression by reducing the number of singular values and corresponding columns and rows in U and V^T. By keeping only the most significant singular values, the image can be reconstructed using a

reduced number of coefficients, leading to reduced storage space or transmission bandwidth.

• Denoising: SVD can be used for noise reduction in images. The noisy image matrix is decomposed using SVD, and the singular values associated with noise are suppressed or set to zero. By reconstructing the image using the modified singular values, noise can be effectively reduced.

• Image Reconstruction: To reconstruct the image, the compressed or modified singular values are combined with the U and V matrices. The product of U, Σ, and V^T yields the reconstructed image matrix, which can be displayed as an image.

When an image is transformed using SVD, it is not compressed, but the data takes a form in which the first singular value has a great amount of the image information. This implies that few singular values can be used to represent the image with little differences from the original image.When SVD is applied to the input image A,it can be written as

$$A = UDV^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T$$

Thus the input image A can be represented by the outer product expansion

$$A = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots\ldots\ldots + \sigma_r u_r v_r^T$$

When compressing the image, the sum is not performed to the very last singular values, and the singular values with smaller values are dropped. After truncating the smaller singular values, the reconstructed image using the first k larger singular values is given by

$$A_k = \sigma_1 u_1 v_1^T + \sigma_2 u_2 v_2^T + \ldots\ldots\ldots + \sigma_k u_k v_k^T$$

The total storage for the reconstructed image Ak is k (m+n+1). The integer k can be chosen to be less than n, and the digital image corresponding to Ak still will be closer to the original image.One of the performance measures of image compression is compression ratio.The compression ratio using SVD is given below:

$$\text{Compression ratio} = \frac{m \times n}{k(m+n+1)}$$

4. Implementation

Explanation for the code 1:

- *The code begins by loading an image using the **imread** function. The image is stored in the image variable.*
- *The **calculateImageEntropy** function is then defined. It takes the **image** as input and returns the entropy value.*
- *Inside the **calculateImageEntropy** function, a check is performed to see if the image is in RGB format. If so, the image is converted to grayscale using the **rgb2gray** function. This step ensures that the entropy is calculated on a single channel.*
- *The histogram of the image is computed using the **imhist** function. The histogram represents the distribution of pixel intensities in the image.*

• *The probability distribution is obtained by dividing each histogram bin value by the total sum of the histogram values. This yields the probability of encountering each pixel intensity.*
• *To avoid NaN (Not a Number) values in the entropy calculation, zero probabilities are removed from the probability distribution.*
• *The entropy value is then calculated using the formula: - sum(probabilities .* log2(probabilities)). This involves taking the sum of the product of each probability and its logarithm (base 2) and negating the result.*
• *Finally, the calculated entropy value is displayed using the* ***disp*** *function.*

Explanation for code 2:

The code performs image compression using the Discrete Cosine Transform (DCT) algorithm. Here's a breakdown of the code:

1. *The code starts by reading an image from a file using the `imread` function. The image is stored in the `image` variable.*
2. *It checks if the image is in color (RGB) or grayscale. If the image has three channels (size(image, 3) > 1), indicating it is a color image, it converts it to grayscale using the `rgb2gray` function.*
3. *The original image is displayed using the `imshow` function and shown in the first subplot of a figure.*
4. *The image data is converted to double precision using the `double` function and stored in the `imageData` variable.*
5. *The 2D DCT (Discrete Cosine Transform) is performed on the image data using the `dct2` function, and the result is stored in the `dctData` variable.*
6. *The desired number of DCT coefficients to retain for compression is set in the `numCoefficients` variable.*

7. *The code creates a copy of the DCT coefficients in the `compressedData` variable.*
8. *All DCT coefficients beyond the `numCoefficients` value are set to zero, effectively removing high-frequency components from the image.*
9. *The inverse DCT (IDCT) is applied to the compressed data using the `idct2` function to reconstruct the image.*
10. *The reconstructed image is displayed in the second subplot of the figure using the `imshow` function. The result of this code is a compressed version of the original image, where the number of retained DCT coefficients determines the level of compression. By reducing the number of coefficients, the image loses some details, resulting in a smaller file size.*

Explanation for code-3:

1. *The code reads an image named 'evee.jpg' and converts it into a double-precision matrix('a').*
2. *The size of the image and its color channels (m, n) is obtained.*
3. *A new matrix b is initialized with zeros, having the same size as the input image.*
4. *The user is prompted to enter a value for k, which determines the level of compression.*
5. *The code then enters a loop over each color channel (assuming the image is RGB).*
6. *For each channel, it performs SVD on the pixel values.*
7. *Singular Value Decomposition (SVD) is a matrix factorization technique.*
8. *The loop runs for the specified value of k.*
9. *The compressed image b is reconstructed using the first k singular values and corresponding vectors from the SVD.*
10. *The compressed image (b) is displayed using imshow.*

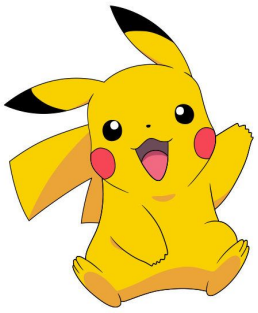
11. *The values are floored to ensure they are integers for display purposes.*
12. *The compression ratio is the ratio of the original image size to the size after compression, providing insights into the level of compression achieved. providing visual output of the compressed image and indicating the achieved compression ratio. The variable k controls the trade-off between compression and image quality, with a higher k resulting in less compression and potentially better image quality.*

5. Simulation Results

output of code 1:

Entropy of the image is

```
>> untitled
Image Entropy: 2.0966
>>
```



DCT Result:

Compression for image using DCT

numCoefficients=number of coefficients to retain for compression

```
>> untitled
Compression Ratio: 0.99999
>>
```

numCoefficients=1

Reconstructed Image


```
>> untitled
Compression Ratio: 0.99955
>>
```

numCoefficients=10

Reconstructed Image

```
>> untitled
Compression Ratio: 0.99821
>>
```

numCoefficients=20

Reconstructed Image

```
>> untitled
Compression Ratio: 0.95712
>>
```

numCoefficients=100

Reconstructed Image

```
>> untitled
Compression Ratio: 0.84805
```

numCoefficients=200

Reconstructed Image

Original Image

S.no	*numCoefficients*	*Compression ratio*
1.	*1*	*0.99999*
2.	*10*	*0.99955*
3.	*20*	*0.99821*
4.	*100*	*0.95712*
5.	*200*	*0.84805*

SVD Result:

```
Enter the value of k:
1
Compression Ratio: 235.9979
>>
```

```
Enter the value of k:
10
Compression Ratio: 23.5998
>>
```

```
Enter the value of k:
20
Compression Ratio: 11.7999
>>
```

```
Enter the value of k:
100
Compression Ratio: 2.36
>>
```

```
Enter the value of k:
200
Compression Ratio: 1.18
>>
```

original image :

Compression ratio against k :

S.no	*Value of k*	*Compression ratio*
1	*1*	*235.9979*
2	*10*	*23.5998*
3	*20*	*11.7999*
4	*100*	*2.36*
5	*200*	*1.18*

6. Conclusion

These methods are essential for solving issues created by the exponential increase in digital visual data. They offer solutions for real-time applications, quick data transfer, and effective storage.These methods have an impact on a wide range of fields, such as economic efficiency, corporate management, scientific research, medical, and remote sensing. Trade-offs between compression ratios, information loss, and image quality must be considered when selecting a compression strategy. Every technique meets various needs and situations. The way that visual data is handled in various applications will probably continue to change as long as image processing and compression techniques continue to

progress. Future research endeavors may focus on developing hybrid techniques and optimizing algorithms for particular fields.

References

[1]"Entropy and Information Theory First Edition, Corrected," 2013. Available: https://ee.stanford.edu/~gray/it.pdf

[1]Wikipedia Contributors, "Entropy (information theory)," Wikipedia, Apr. 02, 2019. https://en.wikipedia.org/wiki/Entropy_(information_theory)

[1]"What is data compression? - Definition from WhatIs.com," SearchStorage. https://www.techtarget.com/searchstorage/definition/compression

[1]"Lossy compression (article)," Khan Academy. https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:digital-information/xcae6f4a7ff015e7d:data-compression/a/lossy-compression

[1]A. Watson, "Image Compression Using the Discrete Cosine Transform The One-Dimensional Discrete Cosine Transform." Available: http://sites.apam.columbia.edu/courses/ap1601y/Watson_MathJour_94.pdf

[1]"Discrete Cosine Transform (Algorithm and Program)," GeeksforGeeks, Jul. 18, 2017. https://www.geeksforgeeks.org/discrete-cosine-transform-algorithm-program/

*[1]احمد محمد عبد المنعم خليل, "Digital Image Processing by S. Jayaraman (z-lib.org)," www.academia.edu, Accessed: Nov. 19, 2023. [Online]. Available: https://www.academia.edu/65668507/Digital_Image_Processing_by_S_Jayaraman_z_lib_org_*

Appendix:

Code-1:

```
>> % Load the image from file
image = imread('image.jpg');

% Calculate and display the entropy of the image
entropyValue = calculateImageEntropy(image);
disp(['Image Entropy: ', num2str(entropyValue)]);

% Function to calculate the entropy of an image
function entropyValue = calculateImageEntropy(image)
    % Check if the image is in color (RGB)
    if size(image, 3) == 3
        % Convert color image to grayscale
        image = rgb2gray(image);
    end

    % Compute the histogram of pixel intensities
    histogram = imhist(image);

    % Calculate probabilities of each intensity level
    probabilities = histogram / sum(histogram);

    % Remove zero probabilities to avoid issues in the entropy formula
    probabilities(probabilities == 0) = [];

    % Calculate entropy using the formula: H(X) = -Σ(p(xi) * log2(p(xi)))
    entropyValue = -sum(probabilities .* log2(probabilities));
end
```

Code-2:

```
>> % Load the image from file
image = imread('DSP.jpg');

% Check if the image is in color (RGB), convert to grayscale if necessary
if size(image, 3) > 1
    image = rgb2gray(image);
end

% Display the original image
figure;
subplot(1, 2, 1);
imshow(image);
title('Original Image');

% Convert image data to double for processing
imageData = double(image);

% Apply 2D Discrete Cosine Transform (DCT) to image data
dctData = dct2(imageData);

% Set the number of coefficients to retain for compression
numCoefficients = 100;
```

```
% Compress the DCT data by zeroing out coefficients beyond the specified limit
compressedData = dctData;
compressedData(numCoefficients+1:end, :) = 0;
compressedData(:, numCoefficients+1:end) = 0;

% Reconstruct the image using the inverse 2D DCT
reconstructedImage = idct2(compressedData);

% Display the reconstructed image
subplot(1, 2, 2);
imshow(uint8(reconstructedImage));
title('Reconstructed Image');
```

Code-3:

```
>> clc
clear all
close all

a = double(imread('DSP.jpg'));
[m, n, ~] = size(a);  % Get the size of the image, including the color channels
b = zeros(size(a));

% Prompt the user to enter the value of k once
k = input('Enter the value of k:');

for c = 1:3  % Loop over each color channel
    [u, d, v] = svd(a(:, :, c));

    for j = 1:k
        b(:, :, c) = b(:, :, c) + d(j, j) * u(:, j) * v(:, j).';
    end
end

b = floor(b);
imshow(uint8(b))

% Compute and display the compression ratio
compressionRatio = (m * n) / (k * (m + n + 1));
disp(['Compression Ratio: ', num2str(compressionRatio)]);
```