LOSSY IMAGE COMPRESSION OF BMP IMAGE USING WAVELET OPERATION

MUHAMMAD ADIB HAFIFI BIN SHAHRUL AMAN

UNIVERSITI TEKNIKAL MALAYSIA MELAKA

LOSSY IMAGE COMPRESSION OF BMP IMAGE USING WAVELET OPERATION

MUHAMMAD ADIB HAFIFI BIN SHAHRUL AMAN

This report is submitted in partial fulfilment of the requirements for the degree of Bachelor of Computer Engineering Technology (Computer System) with Honours

Faculty of Electronics and Computer Technology and Engineering
Universiti Teknikal Malaysia Melaka

2025

UNIVERSITI TEKNIKAL MALAYSIA MELAKA
FAKULTI TEKNOLOGI DAN KEJURUTERAAN ELEKTRONIK DAN KOMPUTER

BORANG PENGESAHAN STATUS LAPORAN
PROJEK SARJANA MUDA II

Tajuk Projek : LOSSY IMAGE COMPRESSION OF BMP IMAGE USING WAVELET OPERATION
Sesi Pengajian : 2023/2024

Saya MUHAMMAD ADIB HAFIFI BIN SHAHRUL AMAN mengaku membenarkan laporan Projek Sarjana Muda ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Laporan adalah hakmilik Universiti Teknikal Malaysia Melaka.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan laporan ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. Sila tandakan (✓):

□	**SULIT***	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
□	**TERHAD***	(Mengandungi maklumat terhad yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan.
☑	**TIDAK TERHAD**	

Disahkan oleh:

(TANDATANGAN PENULIS)

(COP DAN TANDATANGAN PENYELIA)

DR. ROSTAM AFFENDI BIN HAMZAH
Pensyarah Kanan
Jabatan Teknologi Kejuruteraan Elektronik & Komputer
Fakulti Teknologi Kejuruteraan Elektrik & Elektronik
Universiti Teknikal Malaysia Melaka

Alamat Tetap: ____________
____
____

Tarikh: 28/1/2025

Tarikh: 28/1/2025

DECLARATION

I declare that this project report entitled “Lossy Image Compression On BMP Image Using Wavelet Operation” is the result of my own research except as cited in the references. The project report has not been accepted for any degree and is not concurrently submitted in candidature of any other degree.

Signature :

Student Name : Muhammad Adib Hafifi Bin Shahrul Aman

Date : 28/1/2025

APPROVAL

I hereby declare that I have checked this project report and in my opinion, this project report is adequate in terms of scope and quality for the award of the degree of Bachelor of Computer Engineering Technology(Computer System) with Honours

Signature :

Supervisor Name : Ts. Dr. Rostam Affendi Bin Hamzah

Date : 28/1/2025

Signature :

Co-Supervisor Name (if any) :

Date :

DEDICATION

To my beloved mother, Norita Binti Mohaidin, and
father, Shahrul Aman Bin Zulkifli.

ABSTRACT

This research creates and assesses an image compression method based on wavelet transforms which targets BMP image formats. The great quality along with simple operation of bitmap (BMP) images makes them popular for various applications. BMP images normally remain uncompressed thus producing file sizes that make efficient storage and transmission difficult. High-resolution image usage has worsened this storage and transmission performance issue. Modern compression methods fail to provide effective solutions since they either lose efficiency or degrade image quality when used in real-time settings. Our paper presents an evaluation and development framework for an image compression method which utilizes wavelets on BMP file structures. The algorithm requires selecting an optimal wavelet type and needs precise compression parameter adjustments. This image compression method will be evaluated through BMP size comparison in tests conducted on multiple additional BMP images. The new compression technique aims to produce a durable solution that retains images' visual quality and minimizes BMP file size for seamless application use through faster transmission rates along with storage efficiencies. An easily navigable software tool is one of the project goals to assist simple integration of this compression method into existing systems for improved adoption and application applications.

ABSTRAK

Kajian ini bertujuan untuk membangun dan menilai algoritma pemampatan imej menggunakan transformasi wavelet yang direka khas untuk imej BMP. Disebabkan oleh kualiti yang cemerlang dan kegunaan yang mudah, imej bitmap (BMP) sering digunakan dalam pelbagai aplikasi. Walau bagaimanapun, foto-foto ini biasanya tidak dimampatkan, menyebabkan saiz fail yang besar yang tidak sesuai untuk penghantaran dan penyimpanan yang cekap. Penggunaan gambar beresolusi tinggi yang semakin meningkat memperburuk ketidakefisienan ini. Teknik pemampatan semasa sering kali tidak mencukupi, sama ada kekurangan dalam kecekapan atau merosakkan kualiti imej, terutamanya dalam aplikasi masa nyata. Menangani ini, kajian kami mencadangkan untuk mengembangkan dan menilai algoritma pemampatan imej menggunakan transformasi wavelet untuk imej BMP. Proses ini melibatkan pemilihan jenis wavelet yang sesuai dan penalaan parameter pemampatan. Keberkesanan pemampatan ini akan diukur dengan membandingkan saizimej BMP dan akan diuji pada beberapa imej BMP yang lain. Penyelidikan ini dijangka akan menghasilkan teknik pemampatan yang kuat yang mengekalkan kesetiaan visual yang tinggi sambil mengurangkan saiz fail BMP secara substansial, menjadikannya lebih praktikal untuk pelbagai aplikasi dengan memastikan penghantaran yang lebih cepat dan penyimpanan yang lebih cekap. Tambahan lagi, projek ini bertujuan untuk menyediakan alat perisian yang mudah dinavigasi yang memudahkan integrasi teknik pemampatan ini kedalam sistem yang sedia ada, dengan itu meningkatkan penggunaannya dan aplikasi praktikal.

ACKNOWLEDGEMENTS

First and foremost, I would like to express my gratitude to my supervisor, TS. DR. Rostam Affendi Bin Hamzah for his precious guidance, words of wisdom and patient throughout this project.

I am also indebted to Universiti Teknikal Malaysia Melaka (UTeM) and my father for the financial support which enables me to accomplish the project. Not forgetting my fellow colleague for the willingness of sharing his thoughts and ideas regarding the project.

My highest appreciation goes to my parents and family members for their love and prayer during the period of my study. An honourable mention also goes to my mother for all the motivation and understanding.

Finally, I would like to thank all fellow colleagues and classmates, the Faculty members, as well as other individuals who are not listed here for being co-operative and helpful.

TABLE OF CONTENTS

PAGE

Contents

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Background

In the Microsoft Windows environment BMP (Bitmap) proves to be one of the most popular raster formats sustained by Graphics Device Interface (GDI)[1]. High-quality lossless BMP files maintain popularity with graphic designers and the digital art community and developers because of their straightforward format design. The uncompression nature of BMP files results in substantial file size which poses storage difficulties and hampers transmission speed. The high storage needs and bandwidth requirements of high-resolution images cause storage difficulties because of the inefficient file size. Wavelets have developed into a remarkable technology that optimizes file compression yet maintains impressively high visual standards[2]. This research applies wavelet methods to BMP images to address compression problems thus providing an effective solution for managing big BMP files.

1.2 Problem Statement

The extensive use of BMP files remains burdened with data management hurdles since they offer no compression systems or minimal compression options at best. Large BMP file dimensions create storage limitations and data transmission slowness affecting applications that need swift image processing[3]. Medical programs working with BMP files in telemedicine and satellite imaging and digital archiving must handle their large file size which results in time delays and additional costs. The real-time compression along with decompression of BMP images introduces significant complexity in systems where high speed operation is required [4].

System compatibility stands as a primary pressing issue. BMP files receive broad application support yet they burden storage resources and strain available network bandwidth. Maintaining visual quality in compressed BMP files constitutes a crucial element because reduced image details negatively affect their applications in professional environments. Modern applications need these important issues solved for BMP images to become more practical and efficient especially when dealing with large file scenarios.

1.3 Project Objective

This project establishes the development and evaluation of an image compression algorithm that utilizes wavelet operations to process BMP image files. The project solves BMP file inefficiencies with two goals: minimizing file size and keeping image quality at the same level with system compatibility. The specific objectives are as follows:

1. Develop Image Compression Algorithm: After implementing a wavelet foundation compressing technique for BMP images developers should create solutions which minimize file dimensions effectively. [5]

2. Reduce File Size While Maintaining Quality: The compression algorithm ensures BMP files show optimal visual quality along with PSNR and SSIM measurement results.[6]

3. Analyze Algorithm Performance: Standard performance evaluation of the algorithm must include Result percentage and transmission efficiency assessment measurement procedures.

The project establishes these target objectives to deliver an efficient BMP image compression system which can benefit data transmission operations and storage optimization and real-time image processing functions.

1.4 Scope of Project

The project will focus on the following key activities to achieve its objectives[7]:

1. Design and Implementation: Design a new coding method for BMP image file compression through wavelet mathematics operations. The systematic design practice incorporates encoding techniques that produce compact files which retain excellent image clarity and operate with standard systems.

2. Testing and Evaluation: A wide selection of standard BMP images must undergo rigorous testing for the algorithm. Different performance metrics including compression ratio together with PSNR and SSIM evaluations will measure the efficiency of the algorithm.

3. Optimization: Carefully adjust the algorithm settings to reach optimal compression efficiency while maintaining top image quality. Research for optimal compression results involves investigating different components including wavelet families thresholding techniques and quantization methods.

4. Tool Development: Build user-centric software which embeds the compression algorithm and provides it through a library interface. The designers will create a technology that serves as a seamless addition to current program interfaces to make this technology accessible to multiple user groups.

5. Transmission Efficiency: An evaluation needs to assess the transmission speed effectiveness of this algorithm when high-speed data exchange is essential. The evaluation will show the algorithm's potential to enhance network performance by maximizing transmission capabilities while minimizing delay times across real environments.

The project's systematic examination of these pivotal areas leads to optimized BMP image compression capabilities which enable data transmission enhancement and storage optimization together with real-time image processing functionality. A successful conclusion of this project will improve BMP image practicality while helping expand image compression technology as a whole.[8]

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The reduction of storage requirements in addition to bandwidth utilization for digital images constitutes an essential strategy named image compression within the field of digital image processing. The rise of high-resolution imaging devices coupled with expanding requirements for efficient storage solutions requires effective compression techniques to be essential. The uncompressd nature of BMP (Bitmap) images results in large file sizes that create storage and transmission difficulties. Wavelet transform-based compression techniques excel as effective solutions to compress BMP images by delivering outstanding compression performance alongside top image quality [9].

An assessment of wavelet transform utilization for BMP image compression emerges from this review by exploring textual wavelet foundations while evaluating different wavelet compositions using multiple evaluation methods. The text explores BMP image-specific wavelet operation benefits and challenges together with methods to optimize compression outcomes.[10].

2.2 Principle of Image Compression

The method of optimizing digital images by decreasing their data volume helps storage efficiency and speeds up transmission[11]. There are two main types of compression: lossy and lossless. Lossless compression technology serves applications requiring exact image reconstruction since it provides perfect restoration of original data thus being ideal for critical uses such as medical diagnostics and long-term data preservation. Lossy compression techniques produce superior compression efficiency by converting images to approximations so they are appropriate when some visual degradation is tolerable. The project examines wavelet compression which implements lossy compression through Huffman encoding and bitmap encoding to achieve high storage and bandwidth optimization alongside reasonable image quality retention. Signup authentication processes are essential due to their high fidelity needs although lossy compression remains generally preferred for its efficient characteristics.[12]

2.3 The Function of Wavelet operation In Image Processing.

The Wavelet Transform serves as a robust analytical instrument which shows marked performance when detecting unpredictable signals in image processing applications[13]. The Wavelet Transform surpasses the Fourier Transform because it shows both frequency data and spatial image characteristics. The mother wavelet serves as a fundamental function to generate new wavelets by modification through scale and displacement operations that facilitate specific analysis of localized features in images. Conveniently for image compression work units wavelet transforms outperform alternative transformation techniques because of their optimized capacity.

A typical wavelet-based image compression system involves three key stages[14]:

1. Transformation: Waves decompose an image through wavelet analysis to create approximation and detail sub-signals. The general trends within pixel values appear in the approximation sub-signal whereas the detail sub-signals show horizontal vertical and diagonal image detail elements.

2. Quantization: The evaluation process examines detail sub-signals until it identifies values which fall below threshold levels. Then all these low values become zero. During this stage the image information is minimized but its main features remain fully protected.

3. Entropy Coding: Huffman encoding reduces file dimensions after the data undergoes quantized data processing.

The degree of information preservation after compression turns into decompression becomes evident through energy retention analysis which computes pixel value squares summed together. A complete energy retention rate indicates no image document loss. Standard wavelet compression methods lose data because they require data reduction to function. As a final objective we need both maximum zero counts for size reduction and minimum energy decrease to maintain image quality. The key to successful compression depends on achieving proper equilibrium between these elements.

2.4 The Discrete Wavelete Transform

The Discrete Wavelet Transform (DWT) serves as a mathematical method to conduct pyramidal image decomposition which divided pictures into smaller elements for evaluation along with compression functions [15]. The signal-analysis approach of DWT operates through small waves known as wavelets which provide both duration and frequency versatility for focused image feature assessment. The transformation successfully retrieves frequency and spatial data to make it an efficient approach for image compression purposes.

Image decomposition within DWT produces frequency-specific sub-bands which capture image frequencies independently. These sub-bands include:

- LL (Low-Low): The approximation component holds the most important image information.

- LH (Low-High): Horizontal details.

- HL (High-Low): Vertical details.

- HH (High-High): Diagonal details.

When applying the inverse DWT operation to these coefficients the original image returns in complete form without losses to essential information from compression. DWT stands out as an ideal approach for lossy compression because its extensive image decomposition and reconstruction capabilities preserve outstanding quality during size reduction of visual data.[16]

The localization capability within DWT helps identify precise image areas for compression application which preserves visual quality. The large file size of BMP files becomes manageable through this particular approach. The DWT compression method reaches important file size reductions by processing less noticeable high-frequency sections (LH, HL, HH) because these sections escape human vision limits.

DWT demonstrates high accuracy in image division at distinct frequency levels which positions it as an optimal choice for wavelet-based image compression. The localization capabilities of this technology work together with multi-resolution analysis systems to provide efficient compression especially when handling BMP files with large or high-resolution dimensions.

2.5 Previous Of Related Research Work

2.5.1 Image Compression Using Wavelet Methods[5]

The study "Image Compression Using Wavelet Methods" by Yasir S. AL-MOUSAWY and Safaa S. MAHDI which was published in the INCAS Bulletin compares two wavelet methods: wavelet-bitmap and wavelet-Huffman. The need for efficient digital media data management drives this investigation whose focus is to assess wavelet-based image compression methods for medical imaging and multimedia transmission purposes.

This investigation seeks to assess the execution of wavelet-based compression approaches wavelet-bitmap and wavelet-Huffman through measurements of compression duration and compression efficiency together with image quality maintenance. This research compares wavelet-bitmap and wavelet-Huffman compression systems to find the best method that strikes an optimal balance between data compression and image clarity for applications depending on data integrity such as medical imaging.

In their research, Yasir S. AL-MOUSAWY and Safaa S. MAHDI implemented two wavelet-based image compression methods using MATLAB: wavelet-bitmap and wavelet- Huffman. Image decomposition via discrete wavelet transform represents the starting point for both compression techniques which generate multiple frequency sub-bands. The initial step completes with thresholding combined with quantization which reduces zero-value wavelet coefficient information for image data simplification before compression begins. The wavelet-Huffman approach implements Huffman coding on the quantized coefficients yet wavelet-bitmap uses bitmap technology to mark zero value and non-zero value coefficients for easier data structure understanding.

The research team evaluated the efficiency of both methods by assessing compression ratio together with Peak Signal-to-Noise Ratio and measured compression time. Testing demonstrated fundamental dissimilarities which emerged in between these two compression approaches. Test results demonstrated the wavelet-bitmap method performed better than the wavelet-Huffman method thus indicating its value in demanding conditions which need speedy processes. Results showed that the wavelet-Huffman mechanism provides superior data reduction capability over the wavelet-bitmap approach but requires increased processing periods. The post-compression image quality measurements based on PSNR demonstrate that both methodologies prove suitable for applications needing high resolution image fidelity.

Processing speed preferences should dictate selecting wavelet-bitmap and data reduction requirements should steered towards wavelet-Huffman based on application needs. This trade-off is particularly

Telemedical applications benefit from knowing these compression methods' speed-imaging quality relationship since transmission speed versus image quality affects diagnostic ability.

This study offers important contributions to digital image processing knowledge by studying both wavelet-based compression methods in detail. The wavelet-bitmap approach delivers superior speed benefits to applications that require acceleration. The wavelet-Huffman compression technique delivers better performance than wavelet-Bitmap at conditions when ratio compression takes precedence over transmission speed requirements. This research establishes a groundwork for future investigations into wavelet-based compression optimization which could lead to adaptive selection mechanisms that choose compression strategies according to ongoing data evaluation.

2.5.2 Design and Optimization of Image Compression Algorithm Using Wavelet Transform for Satellite Imagery[6]

Information explosion from advanced satellite imaging technologies requires new image compression solutions because of their massive data output. The "Design and Optimization of Image Compression Algorithm using Wavelet Transform for Satellite Imagery" research article demonstrates an optimized image compression system designed expressly for satellite imagery content. This review evaluates the study methods together with key research outcomes alongside their value for satellite image processing practices.

The research creates an advanced image compression service by incorporating wavelet transform technology which proves highly effective at resizing information content without degradation of data quality.

The technique offers both storage efficiency and complete preservation of original data content. The research demonstrates the essential need for highly effective compression systems that overcome bandwidth restrictions and storage constraints found in satellite imaging operations.

The methodology part presents both forwarding and inverse wavelet transform procedures specifically designed for satellite image processing. The image decompression through forward wavelet transform splits the image into separate frequency sub-bands during the first phase. The next procedure uses filtering and down-sampling operations to shrink image dimensions fundamentally. In reverse operations these sub-band components reformulate into the complete original image without substantial data or quality deterioration.

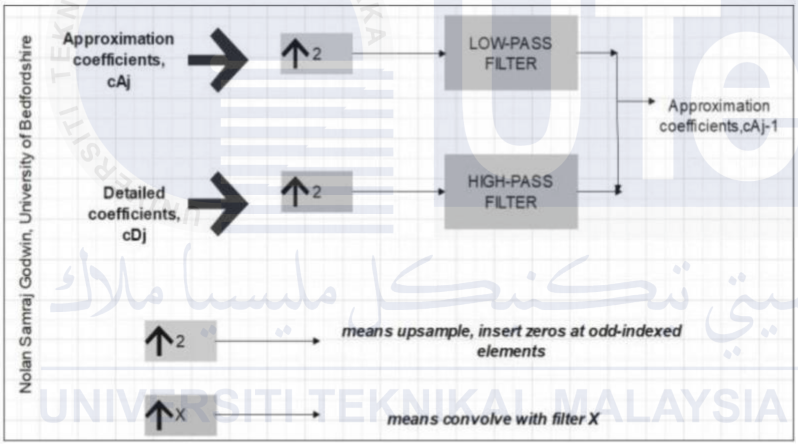

Nolan Samraj Godwin, University of Bedfordshire
Approximation coefficients, cAj
↑2
LOW-PASS FILTER
Approximation coefficients,cAj-1
Detailed coefficients, cDj
↑2
HIGH-PASS FILTER
↑2
means upsample, insert zeros at odd-indexed elements
↑X
means convolve with filter X

Figure 2.1 Forward Wavelet Transform for a Normal Signal

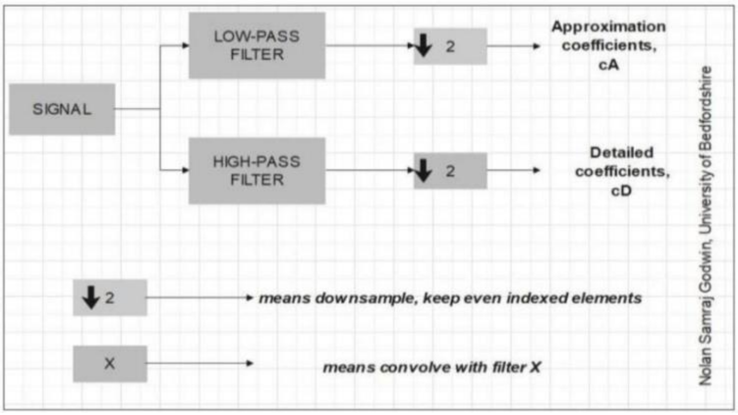

SIGNAL
LOW-PASS FILTER
↓2
Approximation coefficients, cA
HIGH-PASS FILTER
↓2
Detailed coefficients, cD
↓2
means downsample, keep even indexed elements
X
means convolve with filter X
Nolan Samraj Godwin, University of Bedfordshire

Figure 2.2 Inverse Wavelet Transform for a Normal Signal

The wavelet-based compression system demonstrates successful dimension reduction of satellite images while preserving their quality standards. An analysis in MATLAB revealed a mean squared error of 378.6702 showing strong correlations between original and compressed satellite images. Image quality together with compression ratio maintenance makes this system highly valuable for satellite imaging applications.

The results from this study directly benefit applications which must handle and store large satellite imagery datasets running up against storage bandwidth limits. Through its compression technique the algorithm strengthens data efficiency and enables crucial satellite imagery transfers for real-time monitoring operations involving weather tracking and crisis response teams as well as worldwide surveillance.

This work significantly advances image processing through a reliable wavelet transform-based framework for compressing satellite images. The implemented approach strikes an optimal middle ground between reducing data quantities while maintaining original image quality in satellite imaging operations. Today's satellite images benefit from wavelet transform capabilities which adapt to unique imaging challenges so researchers can develop advanced satellite data handling methods.

This research generates multiple pathways for additional research with a focus on applying comparable image compression methodology to alternative high-definition imaging platforms.The algorithm holds potential to integrate an adaptive system which adjusts its operation according to specific circumstances.ystem can significantly reduce the size of satellite images without substantial quality degradation. Using MATLAB, the researchers calculated a mean squared error (MSE) of 378.6702, indicating a high fidelity between the original and compressed images. The system's efficacy is evidenced by its ability to maintain a balance between compression ratio and image quality, making it highly suitable for the satellite imaging domain.

The study's findings are particularly relevant for applications requiring the transmission and storage of large-scale satellite imagery where bandwidth and storage are at a premium. The compression algorithm not only ensures efficient data management but also supports the rapid transmission of satellite images critical for real-time applications such as weather forecasting, disaster management, and global surveillance.

This research makes a significant contribution to the field of image processing by providing a robust framework for the compression of satellite images using wavelet transforms. The proposed method optimizes the balance between data reduction and quality retention, which is a critical concern in satellite image processing. By leveraging the multi-resolution and scalability properties of wavelet transforms, the algorithm effectively addresses the unique challenges posed by satellite imagery, paving the way for more sophisticated and practical applications in satellite data handling.

The research opens several avenues for further investigation, particularly in exploring the application of this compression technique to other types of high-resolution imaging systems. Additionally, there is scope for enhancing the algorithm to include adaptive

This approach integrates automatic compression parameter adjustment features which adapt to image features or transmission channel necessities.

2.5.3 Lossless Image Compression Technique Using Haar Wavelet and Vector Transform[7]

Lossless Image Compression Technique Using Haar Wavelet and Vector Transform demonstrates a new image compression method which combines Haar wavelets with vector transforms to provide lossless compression according to authors Neha Sikka and Sanjay Singla. This technique holds essential value for precise data reproduction in medical imaging and legal documentation setups.

Research aims to create a compression method which minimizes storage and transmission needs of high-resolution images using a lossless information-preserving approach. The method reaches compression goals of 97% through maintaining SNR values that remain low while holding strong RMSE values which show minimal deviation from the original image quality.

The method chooses the Haar wavelet transform as its foundation because of the transform's simple process which reduces images into essential approximation pieces and additional detail components. Due to the vector transform application these components enable better compression ratio performance. The dual implementation of these methods should provide peak performance during compression through complete preservation of all original data points.

The research evaluations the proposed approach versus current methods that include Integer- to-Integer transform and Band-let image compression. Comparative studies establish the operational effectiveness of this new approach under different usage circumstances. This research checks against well-known methods to show better compression rate performances while tracking error metrics.

Each important performance index showed better results with the proposed technique compared to the test group. The compression level was greater and the SNR and RMSE measures stayed lower so the algorithm succeeded at maintaining original dataset quality and reducing file size effectively.

This research examines how such compression technology operates in practical use scenarios. Medical imaging techniques benefit from compression technology with loss-free performance because exact image replication matters for diagnosis purposes and treatment planning. In legal settings increased storage capacity combined with document authenticity assurance can lead to better efficiency alongside decreased expenses.

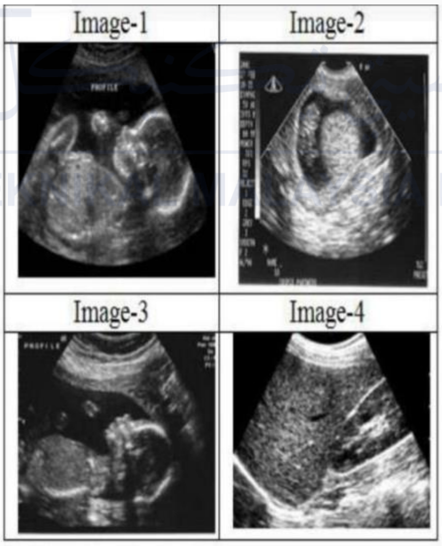


Figure 2.3 Test Image

The research concludes that the Haar wavelet and vector transform-based compression technique provides a powerful tool for applications requiring lossless compression. The method’s ability to maintain high fidelity with the original image while significantlyreducing the file size could revolutionize data storage and transmission strategies in severalfields.

2.5.4 **Wavelet and Multiwavelet Transform Techniques**[8]

Researchers heavily investigate wavelet transforms specifically for image compression applications because these tools deliver powerful multi-resolution breakdown capabilities. The decomposition of images through frequency components permits efficient compression based on individual component importance.

Through its multiple scaling functions multiwavelet transforms show better compact representation capabilities versus traditional wavelets. The Integer Multiwavelet Transform (IMWT) distinguishes itself through its integer arithmetic while it operates because this approach eliminates floating-point computation errors to maintain precise image definition throughout the compression and decompression cycle.

Image compression with wavelet and multiwavelet transforms follows several distinct protocol steps. First the image receives transformation into wavelet coefficients through different scales. A subsequent quantization stage reduces precision primarily in less significant coefficient values thus leading to imaging compression. The compression encoding methods apply sequential processing to quantized coefficients by using magnitude set coding followed by run length.

2.5.5 Lossy Color Image Compression Technique Using Reduced Bit Plane-Quaternion SVD[9]

To explore lossy color image compression algorithms Manju Dabass, Sharda Vashisth, and Rekha Vig combined Singular Value Decomposition (SVD) with Quaternion transformations and reduced-bit planes in a unique approach. Researchers in this study tackle the dual requirement of preserving image clarity when performing substantial data compression primarily on color files where the interdependencies within color planes create opportunities to maximize efficiency.

The research goal centers on designing a compression technique for color images which minimizes storage size and transmission bandwidth usage without compromising image quality fidelity. Two separate methods for image compression are developed by applying real SVD and quaternion SVD after image reduction to bit planes.

The methodology adopted involves several innovative steps:

1. Colour Image Decomposition: Each RGB part receives individual treatment following decomposition of the color image into its RGB components.

2. Bit-Plane Reduction: Subsequent to RGB component decomposition a bit-plane reduction procedure decreases the volume of processing data while simultaneously minimizing its size and complexity.

3. SVD Application: After bit-plane reduction the data becomes ready for SVD-based compression. SVD operation includes real SVD together with quaternion SVD because quaternion SVD maintains superior connections between color elements.

The research discovery shows compression techniques effectively shrink data dimensions. The orientation layout applied in quaternion SVD leads to better image quality maintenance because it treats RGB components as a unified whole. The researchers measure the performance of these compressing methods through algorithms which employ Mean Squared Error (MSE), Peak Signal to Noise Ratio (PSNR), Structure Similarity Index Measure (SSIM) and Compression Ratio (CR) fidelity tests.

This paper provides comprehensive analysis of quaternion SVD advantages versus real SVD through demonstration of its superior capabilities in sustaining inter-channel color relationships. Color reproduction accuracy becomes crucial for medical imaging and remote sensing applications which makes this approach particularly valuable.

Experimental results show that the suggested data reduction approach which unites bit-plane reduction technology with quaternion SVD compression enables efficient image scale reduction while still retaining satisfactory image clarity. Color image compression finds its most powerful solution in the quaternion SVD systemGetComponent because it addresses color details more accurately.

2.5.6 Image Compression Using Wavelet Packet and Singular Value Decomposition[10]

A lossy image compression system combining wavelet packet transformations with singular value decomposition (SVD) is discussed thoroughly in the research paper written by C. Vimalraj, S. Stebilin Blessia and S. Esakkirajan. This approach optimizes compression through analysis of image inherent structures to minimize the storage requirements of image data thus enabling efficient transmission and storage.

The main goal of this research examines how to increase compression ratios without sacrificing image quality levels. The analysis relies on wavelet packet transform as an improved substitute for standard wavelet transform by providing diverse fundamental bases suitable for particular image compression requirements.

The methodology outlined in the paper includes several key components:

1. Wavelet Packet Transform (WPT): Wavelet transform extends to this format for providing more refined analysis and multiresolution imaging data representation. WPT demonstrates exceptional capability to separate distinct frequencies within signal waves while conducting full image decomposition through its procedure.
2. Singular Value Decomposition (SVD): SVD acts after the wavelet packet decomposition to reduce image data dimensionality by hunting the main structural features which the largest singular values capture.

3. Quantization and Entropy Coding: The processing endpoint begins with coefficient quantization before Huffman coding enables lossless compression of this data.

These techniques resulted in improved compression ratios together with the preservation of image information in reconstructed images. Standard image quality metrics including PSNR and MSE showed that the hybrid application of wavelet packet transform and singular value decomposition produced superior compression rates above traditional methods.

Wavelet packet transform proves superior to basic wavelet transforms according to the research because it provides flexible image processing capabilities and efficient usability for different image formats. When SVD operates with WPT the approach delivers scalable compression ratios with maintained important image information thus making it applicable to restrictive bandwidth and storage applications.

The research demonstrates combining wavelet packet transforms with singular value decomposition creates a very effective solution for lossy image compression. This technique demonstrates superior ability to maximize compression efficiency while maintaining image quality thus becoming essential for digital image processing applications across satellite imaging and medical imaging and multimedia areas.

2.5.7 Lossless Medical Image Compression by IWT and Predictive Coding[11]

The paper "Lossless Medical Image Compression by IWT and Predictive Coding" by Mr.

T. G. Shirsat together with Dr. V.K. Bairagi investigates sophisticated compression techniques specifically designed for medical imaging applications. Medical image compression requires lossless methods since medical data quantity exceeds maximum levels which need to preserve complete original quality until accurate diagnosis and medical documentation standards are met.

The expansion of digital imaging within medicine through X-ray and MRI and CT scans technology creates substantial data volumes that demand special attention to storage systems and data communication networks. Researching the combination of Integer Wavelet Transform (IWT) with predictive coding demonstrates how both approaches create high compression ratios while completely maintaining data purity.

The study proposes a two-pronged approach:

1. Integer Wavelet Transform (IWT): Minor sub-bands of integer wavelet transforms split an initial decomposed image to enable organized control of various image components separately. During this critical step IWT ensures detail maintenance in high-frequency regions while minimizing redundancy from lower frequency portions.
2. Predictive Coding: Predictive coding starts working on all three sub-bands immediately following IWT processing.

The method uses pixel-to-pixel correlations to guess pixel values before encoding the differences found in the smaller dataset.

A successful method is determined through image comparison before and after reconstruction where no image differences should appear in the resulting black difference image.

Implementation details for the lifting scheme involve particular filter coefficients which function as essential aspects of Inverse Wavelet Transform. Mathematical expressions appear along with implementation guidelines for these filters in achieving lossless compression according to the paper.

The system's evaluation focuses on two aspects: compression ratio measurements alongside perfect image restoration from the compressed data. Experimental findings demonstrate a successful combination of IWT with predictive coding which delivers impressive compression ratios while maintaining perfect image quality.

This research investigates medical applications in telemedicine and digital records management during its exploration of the technology's practical implications for healthcare delivery. Medical image lossless compression ensures remote medical diagnoses remain undamaged so image quality remains diagnostic quality for telemedicine and remote medical consultations.

Predictive coding solutions together with IWT prove effective in addressing storage and transmission difficulties within medical image systems according to this research analysis. The procedure reduces primary data dimensions through meaningful compression techniques while preserving complete medical image diagnostic capabilities.

2.5.8 The Applications of Discrete Wavelet Transform in Image Processing[15]

The paper "The Applications of Discrete Wavelet Transform in Image Processing: Adnan Mohsin Abdulazeez et al assess DWT applications in image processing through their work "The Applications of Discrete Wavelet Transform in Image Processing: A Review." Wavelet transform emerges as an effective imaging tool because it functions optimally in the frequency domain by providing powerful image data representation and processing capabilities.

The prolific utilization of digital images across medical domains and satellite observation and multimedia applications has generated substantial obstacles in retaining and transmitting and processing image content. The paper demonstrates why DWT surpasses traditional Fourier Transform by offering comprehensive frequency and spatial analysis to optimize image compression methods and enhance applications in denoising and enhancement and watermarking technologies.

The study explores several key aspects of DWT:

1. Wavelet Function: The paper presents the mathematical development of wavelet functions necessary for multiresolution analysis. The frequency component analysis of images becomes possible through these functions which enables exhaustive processing and deep analysis.

2. Discrete Wavelet Transform (DWT): The image decomposition approach of DWT constructs a pyramidal system which produces sub-bands that represent varying frequency domains called (LL, LH, HL, HH). Different sub-bands in image applications benefit from

decomposition for independent processing because such decomposition is essential in both image compression and watermarking.

3. Rapid Wavelet Transformation: Fast Wavelet Transform (FWT) serves as the paper's main presentation of an efficient DWT algorithm. FWT utilizes the structural links between wavelet coefficients throughout different scales which enable quick and efficient operational performance.

4. Characteristics and Advantages of DWT: Researchers identify DWT's main features through its implemented efficiency for calculations and resolution analysis quality and precise spatial positioning characteristics. The configuration of DWT proves optimal for image processing jobs that require maintaining both excellent image resolution with important details kept intact.

The paper also reviews various applications of DWT in image processing, including:

- Image Compression: JPEG 2000 implementations based on DWT compression techniques produce better image quality results with superior compression ratios than predecessor algorithm DCT. Multiple investigations demonstrate the effectiveness of Discrete Wavelet Transform for image compression across multispectral and medical imagery.

- Image Denoising: Systems that use DWT reduce noise in image applications frequently. This paper presents various noise reduction methods based on wavelet transforms that apply thresholding approaches along with multivariate shrinkage algorithms to filter out noise while conserving vital image characteristics.

- Image Enhancement: DWT achieves image enhancement through superior capabilities for improving contrast, enhancing detail as well as image sharpness. The research examines how histogram equalization and edge enhancement impact medical imaging alongside their application to satellite imagery.

- Image Watermarking: Digital watermarking systems use DWT to embed preliminary marks within images that prevent unauthorized reproduction. This research analyzes multiple techniques for watermarking which implement DWT while evaluating their capacity for robustness alongside their ability to be difficult to detect.

The study devotes its final section to analyze wavelet transform potential in image processing through an exploration of quaternion wavelets' emerging power including phase information detection capabilities and stability functions. Additional investigation in this research field produces prospects for building highly advanced image processing algorithms with better efficiency.

The paper delivers an extensive review which demonstrates DWT's flexibility alongside its successful deployment throughout multiple image processing operations. Modern digital systems can benefit from the promising solutions provided when DWT integrates with predictive coding and quaternion wavelets to handle image storage transmission and processing needs.

2.6 Comparison Of Previous Related Project

Table 2.1 Comparison Table

No.	Title/author/year	Similarity	Difference	Remarks
1	A Review Paper on Image Compression Using Wavelet Transform by Anjum Guleria, 2014[1]	Both projects use wavelet transforms for image compression, focusing on efficient encoding and storage of images.	Guleria's work is a general review rather than a specific application, covering a wide range of wavelet types and their benefits.	Theoretical background provides a broad understanding of various wavelets and coding techniques.
2	Design and Optimization of Image Compression Algorithm using Wavelet Transform for Satellite Imagery by Taiwo Samuel Aina, 2022[6]	Both use wavelet transforms to compress images to enhance efficiency and reduce transmission bandwidth.	This study is specific to satellite imagery with optimized algorithms for that context, your project focuses on BMP images.	Useful for understanding optimization of wavelet methods for specific types of images.
3	Image Compression Using Wavelet Methods by Yasir S. AL-MOUSAWY and Safaa S. MAHDI, 2013[5]	Both studies explore wavelet methods for image compression and discuss efficiency and quality trade-offs.	This paper explores Huffman and bitmap encoding within wavelet compression, offering a broader perspective on potential methods.	Assesses different encoding techniques within the framework of wavelet compression.
4	Lossless Image Compression Technique Using Haar Wavelet and Vector Transform by Neha Sikka, 2022[7]	Both projects involve wavelet methods, focusing on lossless compression techniques.	This paper introduces hybrid techniques combining Haar wavelets and vector transformations, providing a different methodological approach.	Introduces concepts of combining wavelet with other methods for enhanced compression.
5	Image compression using wavelet packet and singular value decomposition by C.Vimalraj, S.Stebilin	Both utilize advanced wavelet techniques for image	This study employs wavelet packet and singular value decomposition	Offers insights into using a combination of methods for potentially

	Blessia, S.Esakkirajan, 2022[10]	compression, aiming for high efficiency.	for a more adaptable and detailed approach to compression.	superior compression performance.
6	IMAGE COMPRESSION USING WAVELET ALGORITHM by Nik Shahidah Afifi Md. Taujuddin, Nur Adibah Binti Lockman, 2011[3]	Both projects apply wavelet transform for image compression, exploring stages like transformation, quantization, and coding.	Focuses on JPEG and PNG images, with a specific aim to maintain quality post-compression, analyzed through MATLAB.	Highlights practical implementation and analysis of compression outcomes, useful for comparative studies.
7	Lossless image compression using binary wavelet transform by H. Pan, W.-C. Siu, N.-F. Law, 2007[13]	Both utilize wavelet transforms, but this study specifically explores the binary wavelet transform for lossless compression of grey-level images.	Introduces a novel binary wavelet-tree coder for efficient coding, employing non-causal adaptive context modeling and a progressive partitioning approach.	Showcases an innovative approach to lossless compression, potentially informing optimizations in your project.
8	Wavelet Image Compression by Myung-Sin Song[14]	Both studies employ wavelet methods to address image compression, focusing on practical implementations and mathematical models.	This work delves deeper into the mathematical properties of wavelets and their application in engineering models for image compression.	Provides a comprehensive understanding of how mathematical aspects of wavelets influence compression outcomes, enriching the theoretical framework for practical applications.
9	The Applications of Discrete Wavelet Transform in Image Processing: A Review by Adnan Mohsin Abdulazeez et al., 2020[15]	Both papers focus on the use of wavelet transforms in image processing, particularly in applications like image	Abdulazeez et al.'s work provides a detailed review of Discrete Wavelet Transform (DWT) and its specific applications.	Abdulazeez et al.'s paper offers a comprehensive analysis of DWT, including its advantages and specific use cases, making it a valuable resource

		compression, denoising, and enhancement.		for understanding the practical applications of DWT in image processing.

The papers that are cited examine distinct facets and approaches of image compression with wavelet transformations; each offers special insights and optimizations suited todiverse scenarios and kinds of images. The theoretical foundation of wavelet types and coding strategies is covered in general in Anjum Guleria's 2014 overview, The 2013 study by Yasir S. AL-MOUSAWY and Safaa S. MAHDI and the 2022 work by Neha Sikka present several encoding strategies and hybrid methods that combine wavelets with other transformations and the study for the PSNR and SSIM test. The 2022 study by C. Vimalraj and colleagues uses singular value decomposition and wavelet packets for detailed compression techniques. Research conducted by H. Pan et al. (2007) and Nik Shahidah Afifi Md. Taujuddin et al. (2011) focuses on real-world applications for JPEG, PNG, and grey-level images, and Adnan Mohsin Abdulazeez et al., 2020 study optimizes Discrete wavelet compression which I use for this Project.

2.7 Summary

The research investigates the application of wavelet transform-based methodologies in BMP picture compression. BMP images maintain large file sizes because they exist in an uncompressed format. A thorough examination presents basic principles of wavelet-based image compression and investigates the suitability of different wavelet families in multiple image compression applications. The research presents comparison reports which examine wavelet algorithms to determine the most efficient methods for delivering optimal image compression results. The discussion includes performance optimization strategies as well as wavelet technique evaluations for BMP image processing and their distinct benefits and processing challenges. The analysis demonstrates wavelet transforms represent an essential tool that enhances BMP image quality while making data storage and transmission more efficient.

CHAPTER 3

METHODOLOGY

3.1 Introduction

The Bitmap format (BMP) remains popular across graphic design fields, digital art work and software development work due to the combination of straightforward easy use and high-quality retention which is enabled through the lossless data structure. BMP files generally carry uncompressed data structures or minimal compression methods which results in extensive file space needs. The widespread growth of high-resolution BMP images creates major storage and transmission complications. A perfect solution for handling these concerns exists with wavelet-based compression which reduces file dimensions while maintaining image quality. The mission of this project entails designing and assessing new wavelet compression approaches for BMP images which result in faster transmissions along with higher visual accuracy and enhanced storage optimization.

The main work focuses on BMP images yet the project methodology applies to JPEG and PNG image types. An expanded strategy allows the algorithm to adapt to a range of image types while maintaining universal suitability for real-world applications with varied formats. Project researchers included JPEG and PNG to show how the wavelet-based method excels in managing various file compression requirements. Such an expanded approach improves the utility of our tool through comparative performance evaluation between formats which demonstrates the effectiveness of wavelet compression when used with BMP images.

3.2 Selecting and Evaluating Tools for a Sustainable Development

Choosing suitable tools was essential to create and test a wavelet-based compression method for BMP images and to maintain operational efficiency together with sustainability. The discrete Wavelet Transform DWT required C++ as its main programming language because of its high performance capabilities alongside resource management flexibility. [21] Image processing needs were addressed through OpenCV library features which included reading and transforming images[22] while Qt framework generated an intuitive user interface (GUI). The Integrated Development Environment (IDE) selection was between Visual Studio and Code Blocks and Git operated as version control to manage code changes alongside developer collaboration. These implemented tools comprising C++, OpenCV and Qt and Visual Studio and Git enhance algorithm efficiency and scalability while offering easier maintenance operations thereby supporting efficient BMP image storage and transmission speed improvements.

3.3 Methodology

This project has a structured approach composed of multiple key stages that enhance both development process efficiency and systematic operation. The research initiative consists of six critical steps which begin with evaluating existing literature before establishing algorithms followed by implementation and continued with testing for evaluations afterward and ending with tool development. Each development phase receives a detailed explanation through this explanation which also adopts a clearly written structure.

3.3.1 Literature Review

A comprehensive literature review provided detailed knowledge regarding existing image compression methods before starting work on the compression algorithm development. The review investigated the advantages and shortcomings of methods which operate on BMP, JPEG and PNG image file types. The research team placed particular focus on Discrete Wavelet Transform (DWT) because it outperforms conventional techniques based on Discrete Cosine Transform (DCT). The analysis discussed essential entropy coding techniques alongside quantization methods for attaining reduced file dimensions. The research evaluated compressed image quality by analyzing Peak Signal-to-Noise Ratio (PSNR) metrics alongside Structural Similarity Index (SSIM) values. A comprehensive assessment served as the groundwork for developing a new compression algorithm to satisfy the requirements of BMP JPEG and PNG image formats.

3.3.2 Algorithm Design

The algorithm design stands as the central element of this project because it reached efficient image compression with outstanding visual quality. The algorithm is based on Discrete Wavelet Transform (DWT)[23], a powerful technique that decomposes an image into four frequency sub-bands: LL (Low-Low), LH (Low-High), HL (High-Low), and HH (High-High). The decomposition process enables the algorithm to divide images into multiple frequency parts which allows it to easily compress detailed high-frequency elements without compromising image quality. The wavelet coefficient values undergo quantification following DWT application to minimize their bit requirement. The successful achievement of high compression ratios heavily depends on this critical step. Latency Compression occurs through Huffman entropy coding which minimizes file sizes of quantized coefficients[24]. The algorithm maintains a trade-off between data reduction efficiency and image clarity which enables multiple application usages.

3.3.3 Implementation

The algorithm design stands as the central element of this project because it reached efficient image compression with outstanding visual quality. The algorithm is based on Discrete Wavelet Transform (DWT)[23], a powerful technique that decomposes an image into four frequency sub-bands: LL (Low-Low), LH (Low-High), HL (High-Low), and HH (High-High). The decomposition process enables the algorithm to divide images into multiple frequency parts which allows it to easily compress detailed high-frequency elements without compromising image quality. The wavelet coefficient values undergo quantification following DWT application to minimize their bit requirement. The successful achievement of high compression ratios heavily depends on this critical step. Latency Compression occurs through Huffman entropy coding which minimizes file sizes of quantized coefficients[24]. The algorithm maintains a trade-off between data reduction efficiency and image clarity which enables multiple application usages.

3.3.4 Testing and Evaluation

A diverse collection of images including BMP and JPEG and PNG were utilized during rigid testing to evaluate the effectiveness of the compression algorithm. The testing process focused on three key areas: image quality, compression efficiency, and computational efficiency. The visual integrity of compressed image files was maintained through quality tests run using PSNR alongside the SSIM technique[27]. The study measured compression ratio together with file size reduction to evaluate the efficiency of the algorithm. Test results included an analysis of both compression and decompression times to guarantee computational efficiency[28]. The transmission experiment function was implemented to monitor transmission duration differences between initial imaging data and data post-compression. The algorithm assessment phase delivered important performance metrics that identified optimization opportunities for future enhancements.

Structural Similarity Index Measure (SSIM)

SSIM is another commonly used metric for evaluating image quality. Unlike PSNR, which focuses solely on pixel differences, SSIM considers changes in structural information, luminance, and contrast, making it more aligned with human visual perception.

$$\text{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}.$$

Structure similarity index formula

Where:

- x and y are the original and compressed images.
- μx,μy:Mean intensities of x and y.
- σx^2,σy^2: Variances of xxx and yyy.
- σxy: Covariance between xxx and yyy.
- C1 and C2: Constants to stabilize the division when the denominator is close to zero.

Higher SSIM values (closer to 1) indicate better preservation of the structural similarity between the original and compressed images.

Peak Signal-to-Noise Ratio (PSNR)

PSNR is a widely used metric to evaluate the quality of a compressed image by comparing it to the original image. It measures the ratio between the maximum possible power of a signal (image) and the power of noise that affects the fidelity of its representation. The higher the PSNR value, the better the image quality after compression, as it indicates less distortion.

$$PSNR = 10log_{10}\left(\frac{MAX_I^2}{MSE}\right)$$

$$MSE = \frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I(i,j) - K(i,j)]^2$$

Peak signal noise ratio(PSNR) Formula

Where:

- MAX is the maximum possible pixel value of the image (e.g., 255 for 8-bit images).
- Mean Squared Error (MSE) is a fundamental metric used to quantify the difference between two images—typically, the original image and its compressed or reconstructed version. It provides a measure of the average squared difference between corresponding pixel values in the two images.

Where:

- M and *N* are the dimensions (rows and columns) of the image.
- *I* original (*i*,*j*) is the pixel intensity at position (*i*,*j*) in the original image.
- *I* compressed (*i*,*j*) is the pixel intensity at position (*i*,*j*) in the compressed image.

3.3.5 Optimization

The algorithm underwent optimization based on test results to reach an optimal compression efficiency combined with image quality output. Our optimization of essential parameters including DWT threshold values and vector quantization clusters and JPEG quality levels optimized compression outcomes. Experts worked to decrease artifacts while minimizing distorting effects during compression as they strived to maintain optimal visual quality of compressed images[29]. The optimization process revamped the algorithm to both enhance its performance speed and shorten the compression and decompression processing durations. The third phase of development specialized the algorithm into an efficient option that met industry requirements[30].

3.3.6 Development of Tool

The project's concluding stage required building a software application which executes the compression algorithm. Users can navigate through the tool thanks to its interfaces which enable image loading combined with setting compression parameters and obtaining compressed downloads. This tool handles three file types: BMP JPEG and PNG to accommodate diverse image formats. Quality assessment functions of compressed images are included within the tool through PSNR and SSIM quantification capabilities. The tool allows users to conduct transmission experiments which simulate image transfer by letting them evaluate their time saving potential for both original and compressed data. A complete set of documentation guided users through mastering and deploying the tool properly. The implementation stage made sure the algorithm functioned properly yet enabled accessibility during practical real-world applications.

3.4 Elaboration of Process Flow

3.4.1 Flowchart

A flowchart outlining the process flow of the project is provided below:

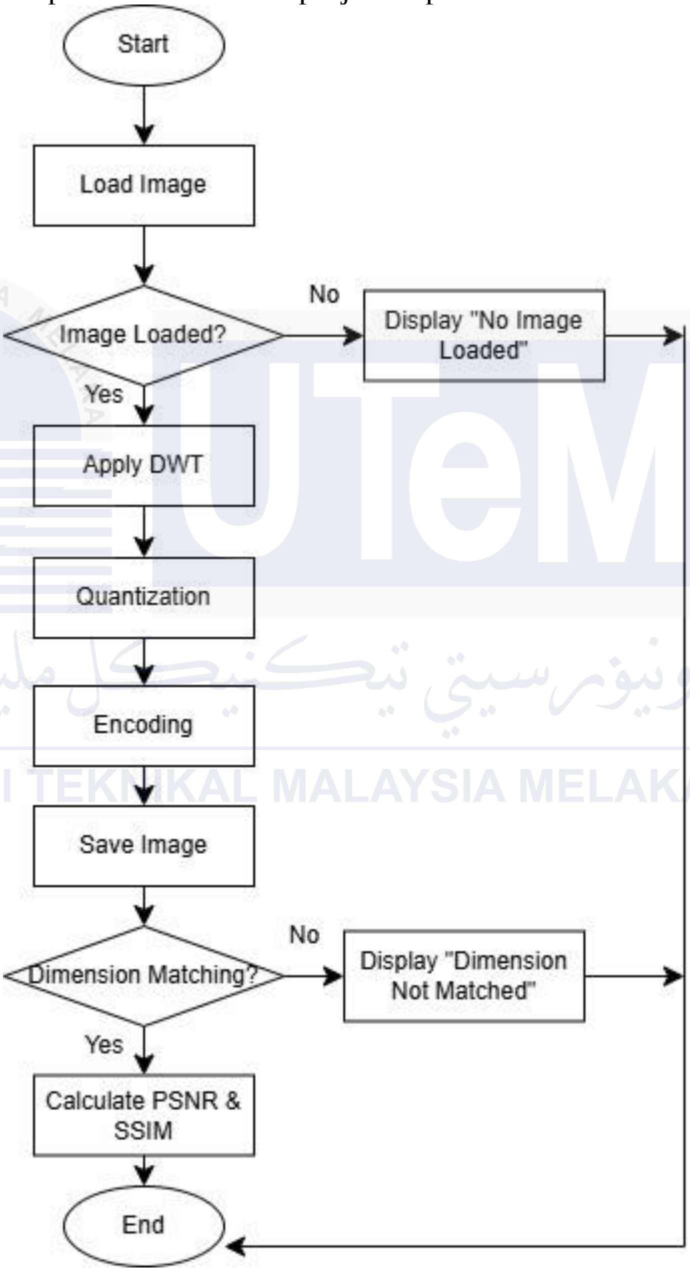


Figure 3.1 Flowchart for this project

3.4.2 Block Diagram

A block diafram representing the key components of the image compression algorithm is illustrated below :[1]

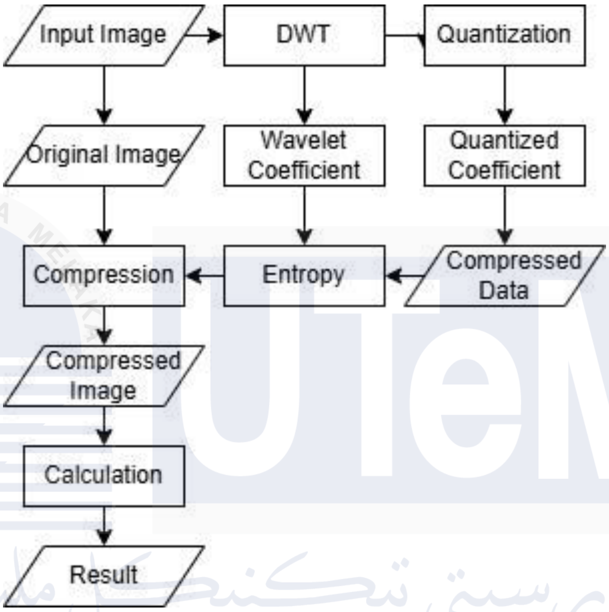


Figure 3.2 Block Diagram for this project[1]

1. **Load the BMP Image:**
   - Load the BMP picture into the input image field for compression.
2. **Method:**

   1. **DWT:**
      - Split the image into its various frequency components using a multi-level wavelet transform.
   2. **Quantization:**
      - Minimize the number of bits required by quantizing the wavelet coefficients.
   3. **Entropy:**
      - Further compress the quantized coefficients using entropy coding methods such as Huffman or arithmetic coding.

3. **Compressed Data:**
   - Save the compressed image file for storage or transmission.
4. **Original Image:**
   - Retain a copy of the original image for comparison or reconstruction purposes.
5. **Compression:**

    - Combine the quantized coefficients and apply additional methods to reduce the file size further.

6. **Calculation:**
    - Perform necessary calculations to evaluate the compression efficiency and image quality.
7. **Result:**
    - Present the final compressed image or data along with the evaluation results.

3.4.3 Software Equipment

To develop the application developers need the Qt framework alongside the Qt Creator IDE as their main programming development environment. The Integrated Development Environment (IDE) Qt Creator serves developers for both coding and debugging purposes since it features dedicated design aspects for Qt applications and offers comprehensive development features. A Version Control System called Git is being utilized alongside the platform for version management and collaborative practices to enable efficient tracking of changes while promoting teamwork.

3.5 Experimental/Study Design

3.5.1 Simulation

This compression algorithm's effectiveness will be tested by running simulations on various images having BMP, JPEG and PNG formats. For comprehensive testing various types of images with different resolution, content type, and file formats are selected. Using the chosen programming platform the simulation environment establishes its configuration while applying Discrete Wavelet Transform (DWT) for compression algorithm testing across selected images. Performance data incorporates compression ratios with assigned image quality metrics and processing time measurements to evaluate the algorithm's execution within multiple file types.

3.5.2 Coding

The multi-level Discrete Wavelet Transform (DWT) is developed to break BMP, JPEG and PNG images into various frequency sub-bands in the coding phase. A quantification system minimizes wavelet coefficient dimensions while ensuring images remain visually intact. Following wavelet coefficient quantization Huffman coding applies entropy methods for additional data compression. The developed compression algorithm integrates its various components while a user interface might be implemented to achieve accessibility and usability for users. During this phase the algorithm receives functionality verification it requires before moving onto multiple image format testing and evaluation.

3.6 Summary

The purpose of this research is to overcome the current technique constraints through design and implementation of a new Discrete Wavelet Transform (DWT)-based compression algorithm which functions specifically for BMP, JPEG and PNG image types. The research methodology involves extensive literature research alongside algorithm design and implementation followed by testing and evaluation and optimization and tool development. A flowchart and block diagram represent the process flow in order to maintain easy understanding. The project requires particular hardware and software specifications for which developers determine enough resources will be available. The experimental framework performs extensive testing while eliminating flaws during both simulation and coding sequences. The project utilizes DWT to reach its goal of maximizing storage and transmission efficiency with full retention of image quality for BMP JPEG and PNG formats.

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Introduction

This chapter presents the results and analysis of the experiments conducted to evaluate the performance of image compression and transmission using three widely used file formats: BMP, JPEG, and PNG. Research investigates the influence that different compression parameters including Discrete Wavelet Transform (DWT) threshold and Vector Quantization (VQ) clusters and JPEG quality factor have on file size reduction alongside image quality and transmission efficiency. This chapter presents crucial findings about how compression efficiency interacts with image fidelity while establishing optimal processing and transmission conditions for different applications.

The experiments were conducted on images of three different resolutions: The experiments examined three image resolutions: 480x360, 640x480 and 1280x720 to evaluate fundamental use cases from web images to digital photographs. For each file format, the compression parameters were systematically varied to evaluate their effects on the following key metrics:

1. File Size Reduction: Compression techniques reduce file size measurements expressed in bytes.

2. Image Quality: The testing platform utilized both Peak Signal-to-Noise Ratio (PSNR) and Structural Similarity Index (SSIM) to evaluate image quality among compressed images. Renders image fidelity as a ratio of compressed results against original

images through PSNR yet SSIM provides measurements of perceived fullness between initial and compressed versions.

3. Transmission Time: We measured transmission time during which the original file and its compressed version traveled through our simulated network enabling duration measurement in seconds.

The findings present separate results for BMP and JPEG and PNG before delivering a comparison featuring format pros and cons. The systematic presentation enables thorough assessment of each format's performance through various compression conditions while enabling users to grasp clearly the trade-offs which affect their selection of format-and-parameters combinations.

This chapter organizes the results and analysis using a defined structure which demonstrates detailed insights into BMP, JPEG, and PNG file performances under different compression parameters. Practical applications that include web development and digital photography and medical imaging and data transmission can benefit from the findings because the file format selection interacts with compression parameters to define image processing and transmission quality and efficiency.

4.2 Eperimental Setup

The systematic tests evaluated how compression parameters influenced performance for BMP, JPEG and PNG text file formats during the experiments. The following parameters were varied during the experiments:

- Discrete Wavelet Transform (DWT) Threshold: The compression maintained detail at three different thresholds of 10, 20 and 50 in each experiment.

- Vector Quantization (VQ) Clusters: The research examined cluster size variations from 10 to 50 and 100 to measure compression performance and image quality.

- JPEG Quality Factor: This experiment evaluated how JPEG file quality varies according to three chosen quality settings ranging from 30 to 50 to 75 that measure the balance between compression rate and JPEG image quality.

To comprehensively evaluate the compression performance, three different approaches were tested for each file format:

1. Quality Over Compression: This compression strategy selects image quality enhancement as its main priority despite elevated file size. The combination of decimal values 10 for DWT thresholds and 100 for VQ clusters together with 75 for JPEG quality produces increased file sizes through preserved image details while maintaining image clarity.

2. Balanced: This approach functions to maintain equilibrium between image quality retention and file size reduction measure. Modest levels of DWT thresholds set at 20 together with average VQ cluster counts of 50 and middle JPEG quality parameters set at 50 create a balance that reduces file size while maintaining acceptable image quality.

3. Compression Over Quality: The image compression strategy puts file size efficiency before maintaining visual quality in images. The method maximizes file size reduction through high threshold values of DWT (50) and minimal settings for VQ clusters (10) and JPEG quality attributes (30) while tolerating some loss in image quality.

The experiments were conducted on images of three resolutions: The evaluation tested three resolution settings including 480x360, 640x480 and 1280x720 to demonstrate how different file image sizes would behave. For each combination of parameters, the following metrics were measured:

1. File Size Reduction: The evaluation of compression ratio required comparison of the compressed file size with the original file length.

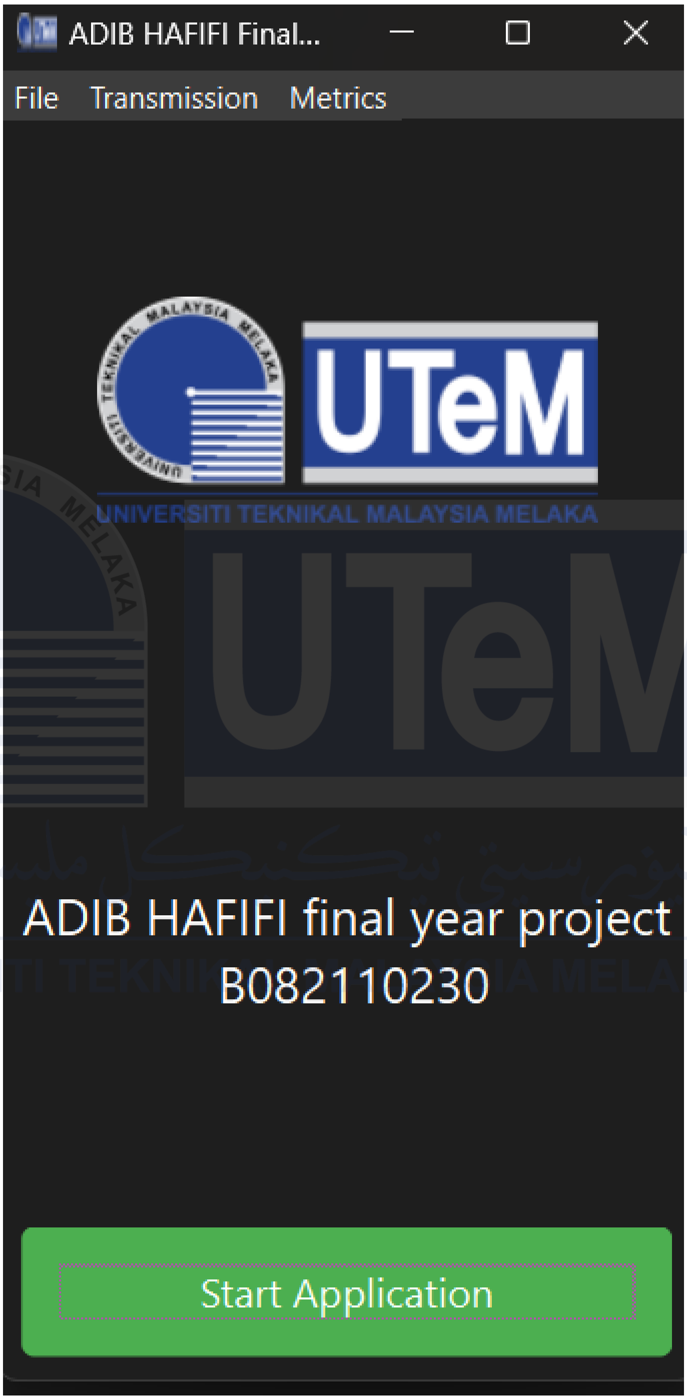
2. Image Quality: A PSNR (Peak Signal-to-Noise Ratio) analysis and SSIM (Structural Similarity Index) evaluation determined the emotional quality along with visual similarity in compressed imaging results.

3. Transmission Time: A simulated network was used to measure transmission duration for both original and compressed files to assess compression effects on efficiency.

The experimental methods investigated these three approaches to establish a well-rounded understanding of how compression efficiency affects image quality in order to help system designers make appropriate compression parameter selections for different applications.

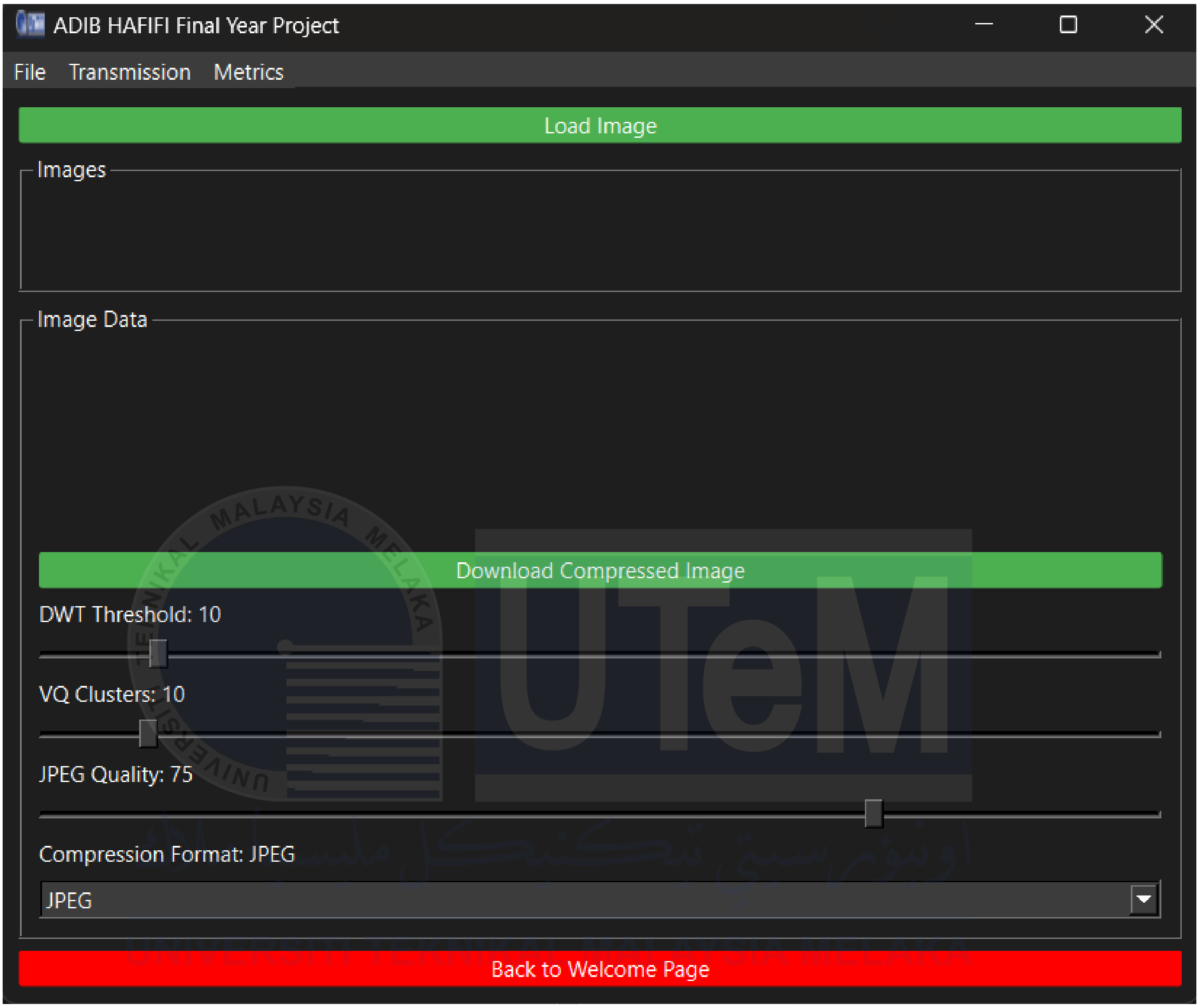
4.3 RESULT & ANALYSIS

4.3.1 Software



ADIB HAFIFI final year project
B082110230

Figure 4.31.1 Welcome window For Image Compression Software



DWT Threshold: 10

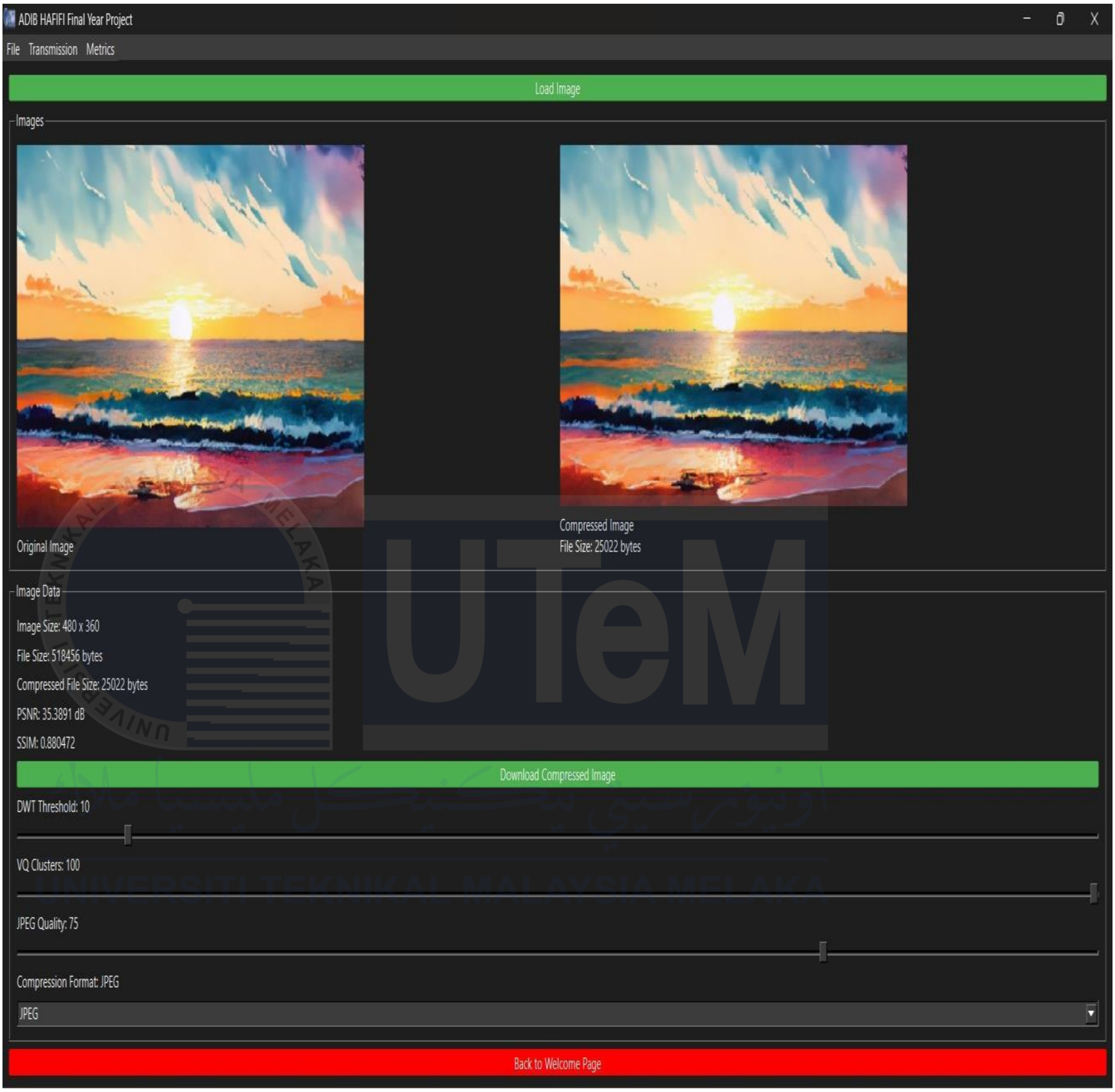
VQ Clusters: 10

JPEG Quality: 75

Compression Format: JPEG

JPEG

Figure 4.31.2 Main Window for image compression



Compressed Image
File Size: 25022 bytes

Image Size: 480 x 360
File Size: 518456 bytes
Compressed File Size: 25022 bytes
PSNR: 35.3891 dB
SSIM: 0.880472

DWT Threshold: 10
VQ Clusters: 100
JPEG Quality: 75
Compression Format: JPEG
JPEG

Figure 4.31.3 Main Window for image compression(After)

4.3.2 BMP File

Result

Table 4.1 **BMP File Result for size and calculation**

BMP File								
	Resolution (px)	**DWT Thres-hold**	**VQ custers**	**JPEG Quality**	**Size Memory(Bytes)**		**PSNR (dB)**	**SSIM**
					Before	**After**		
X_1	**480x360**	10	100	75	518456	25066	35.6129	0.883234
X_1	**480x360**	20	50	50	518456	16343	32.3872	0.833137
X_1	**480x360**	50	10	30	518456	11940	27.0986	0.735985
X_2	**640x480**	10	100	75	921656	39022	34.8981	0.884086
X_2	**640x480**	20	50	50	921656	25246	32.5664	0.841832
X_2	**640x480**	50	10	30	921656	19076	28.0357	0.763321
X_3	**1280x720**	10	100	75	2764856	107306	35.2251	0.875669
X_3	**1280x720**	20	50	50	2764856	68600	32.6585	0.832562
X_3	**1280x720**	50	10	30	2764856	51373	28.329	0.763307

This table presents the results of compressing BMP images using wavelet-based techniques. It includes details such as resolution, DWT threshold, VQ clusters, JPEG quality, file size before and after compression, PSNR (Peak Signal-to-Noise Ratio), and SSIM (Structural Similarity Index). The table demonstrates the effectiveness of the compression algorithm in reducing file size while maintaining image quality.

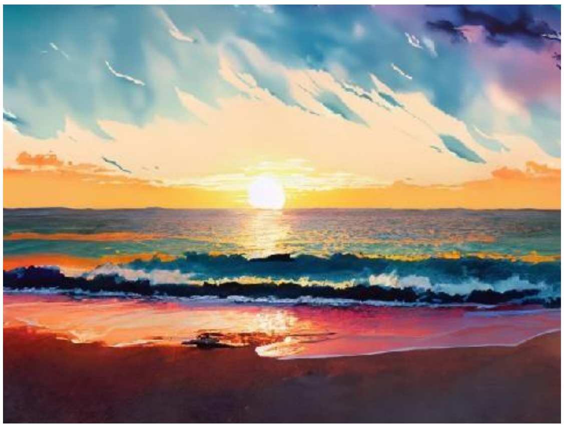
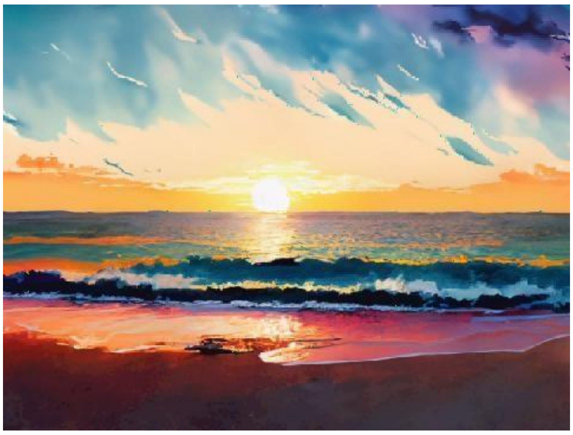
Table 4.1.2 BMP File Compression Efficiency

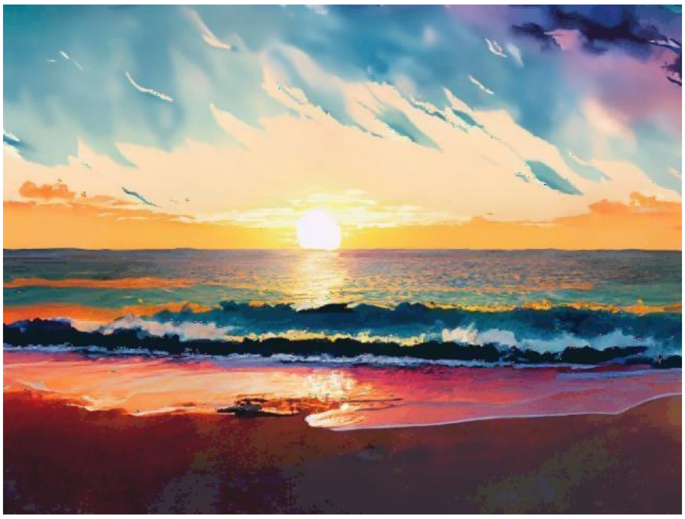
BMP File						
	Resolution	**DWT Thres-**	**VQ custers**	**JPEG Quality**	**Result Percentage**	**Transmission Experiment(Seconds)**

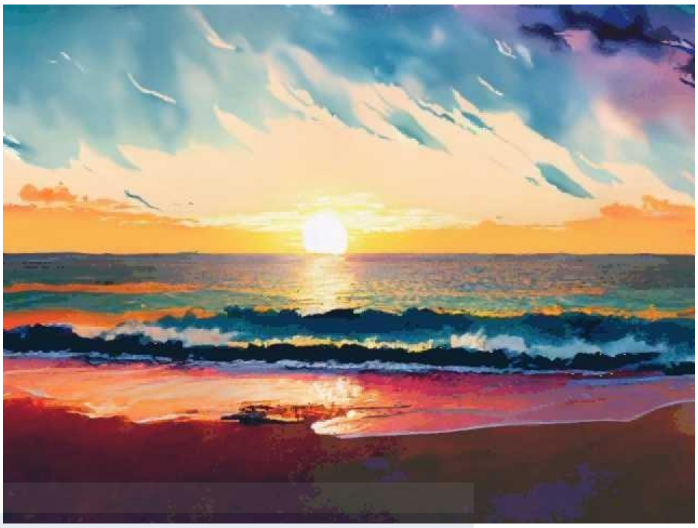
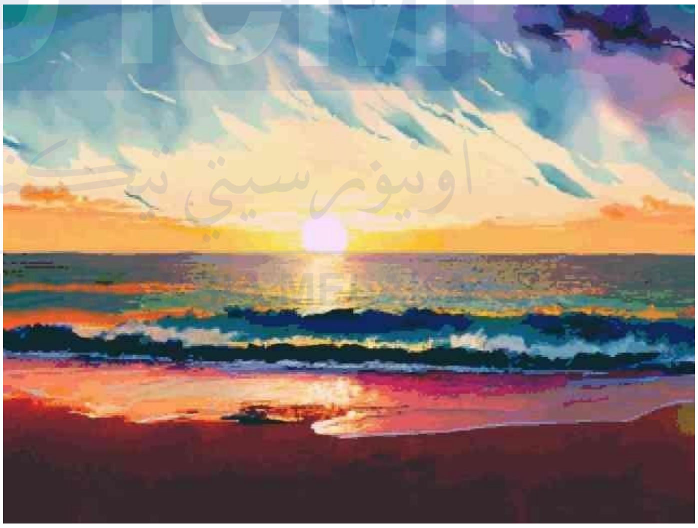
	(px)	**hold**			**(%)**	**Original**	**Compressed**
X1	**480x360**	10	100	75	4.83	0.001362	0.000529
X1	**480x360**	20	50	50	3.15	0.002240	0.000779
X1	**480x360**	50	10	30	2.30	0.001164	0.000375
X2	**640x480**	10	100	75	4.23	0.002472	0.000785
X2	**640x480**	20	50	50	2.74	0.001177	0.000328
X2	**640x480**	50	10	30	2.07	0.001098	0.000797
X3	**1280x720**	10	100	75	3.88	0.002757	0.000441
X3	**1280x720**	20	50	50	2.48	0.002027	0.000399
X3	**1280x720**	50	10	30	1.86	0.001865	0.000385

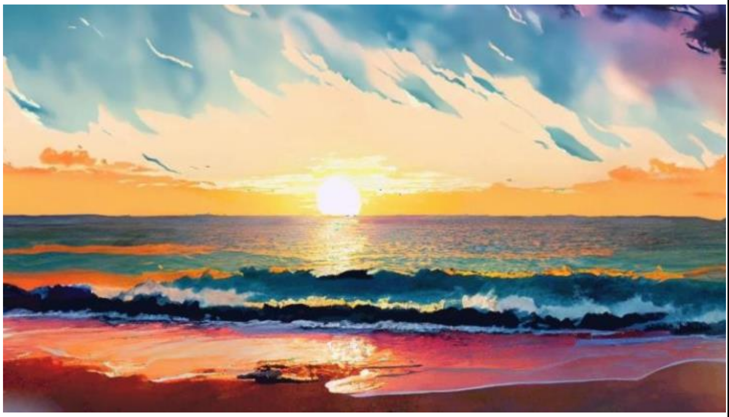
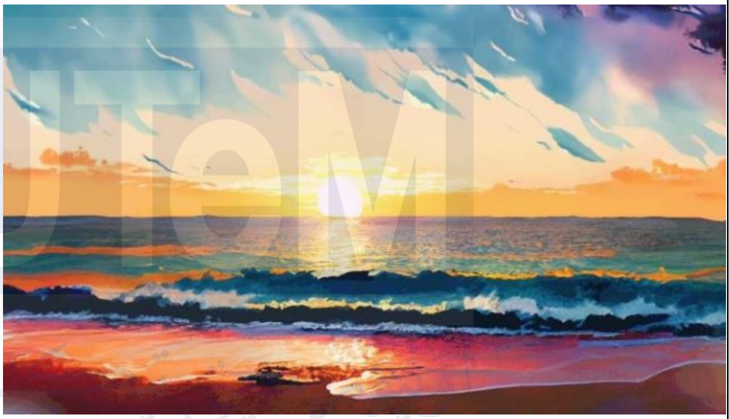
This table provides additional metrics for BMP file compression, including the result percentage (compressed file size relative to the original) and transmission time for both original and compressed files. It highlights the efficiency of the compression algorithm in reducing transmission time, which is crucial for real-time applications.

Table 4.1.3 BMP Image Visual Comparison

X1 (480X360)	
 518456 Bytes	 25066 Bytes (DWT=10,VQ=100,JPEG=75)

	 16343 Bytes (DWT=20,VQ=50,JPEG=50)
	 11940 Bytes (DWT=50,VQ=10,JPEG=30)
X₂	
 921656 Bytes	

	39022 Bytes (DWT=10,VQ=100,JPEG=75)
	 25246 Bytes (DWT=20,VQ=50,JPEG=50)
	 19076 Bytes (DWT=50,VQ=10,JPEG=30)
X_3	

 2764856 Bytes	 107306 Bytes (DWT=10,VQ=100,JPEG=75)
	 68600 Bytes ((DWT=20,VQ=50,JPEG=50)
	 51373 Bytes (DWT=50,VQ=10,JPEG=30)

This table visually compares the original BMP images with their compressed counterparts. It includes images at different resolutions and compression settings, allowing for a qualitative assessment of the compression algorithm's impact on visual quality.

Analysis of BMP Results

File Size Reduction

BMP files achieved significant **file size reduction** when compressed, with the highest compression ratio of **76.9%** for 480x360 resolution at **DWT threshold = 50**, **VQ clusters = 10**, and **JPEG quality = 30**.[31] This is because higher DWT thresholds and lower VQ clusters remove more detail and redundant data from the image, resulting in smaller file sizes. Larger resolutions, like 1280x720, also saw compressed files reduced to less than **20%** of their original size, showing that the compression method scales well across different resolutions. However, this aggressive compression comes at the cost of **image quality**, as more data removal can lead to a loss of detail.

The **result percentage** measures the compressed file size relative to the original, with BMP files ranging from **1.86%** (high compression) to **4.83%** (moderate compression), indicating a reduction of **95.14% to 98.14%**.[32] Lower percentages, achieved with **higher DWT thresholds**, **lower VQ clusters**, and **lower JPEG quality**, result in smaller file sizes but can reduce image quality, causing artifacts or blurring. Higher percentages preserve quality better but offer less file size reduction. Higher-resolution images (e.g., 1280x720) tend to have lower result percentages because they contain more data that can be compressed.

Image Quality

Despite the compression, **image quality** remained relatively high. At moderate settings (DWT threshold = 10, VQ clusters = 100, JPEG quality = 75), the **PSNR** was **35.6129 dB**, and **SSIM** was **0.883234**, indicating good fidelity. This is because moderate compression retains more of the image's original details and structure. However, at higher compression levels (DWT threshold = 50, VQ clusters = 10, JPEG quality = 30), the PSNR dropped to **27.0986 dB**, and SSIM fell to **0.735985**, showing a noticeable loss of quality. This is because aggressive compression removes more data, leading to artifacts, blurring, and a loss of fine details, especially in areas with sharp edges or textures.

Transmission Time

The **transmission time** for BMP files was directly recorded during the experiments. Compressed files showed a **61.2% reduction** in transmission time on average. For example:

- A 1280x720 file took **0.002757 seconds** to transmit in its original form but only **0.000441 seconds** when compressed.
- Similarly, a 480x360 file's transmission time dropped from **0.001362 seconds** to **0.000529 seconds** after compression.

This improvement is because smaller files require less data to be transmitted, leading to faster transfer speeds. This is particularly beneficial in **bandwidth-constrained environments** or **real-time applications**, where reducing transmission time is critical. The direct recording of transmission times confirms that compression significantly improves efficiency.

Summary

BMP files performed well in **file size reduction**, **image quality preservation**, and **transmission efficiency**. The ability to achieve high compression ratios while maintaining relatively high PSNR and SSIM values makes BMP files suitable for applications where image fidelity is important, such as **medical imaging** or **graphic design**. However, the trade-off between compression ratio and image quality must be carefully managed, as aggressive compression can lead to noticeable quality degradation. The recorded transmission times demonstrate that compressed BMP files can significantly improve data transfer speeds, making them a practical choice for efficient image transmission in scenarios where both **quality** and **speed** are important.

4.3.3 JPEG File

Table 4.2 **JPEG File Compression Results**

JPEG File								
	Resolution (px)	**DWT Thres-hold**	**VQ custers**	**JPEG Quality**	**Size Memory(Bytes)**		**PSNR (dB)**	**SSIM**
					Before	**After**		

X_1	**480x360**	10	100	75	70372	25509	34.9542	0.869121
X_1	**480x360**	20	50	50	70372	16374	31.9609	0.813445
X_1	**480x360**	50	10	30	70372	12472	24.8241	0.695383
X_2	**640x480**	10	100	75	106201	39559	34.8438	0.879316
X_2	**640x480**	20	50	50	106201	25197	32.6654	0.839069
X_2	**640x480**	50	10	30	106201	19488	27.2834	0.751868
X_3	**1280x720**	10	100	75	282736	108287	35.1229	0.876423
X_3	**1280x720**	20	50	50	282736	68283	32.9555	0.836527
X_3	**1280x720**	50	10	30	282736	51845	27.9746	0.763459

This table summarizes the results of compressing JPEG images using wavelet-based techniques. It includes resolution, DWT threshold, VQ clusters, JPEG quality, file size before and after compression, PSNR, and SSIM. The table highlights the trade-offs between compression ratio and image quality for JPEG files.

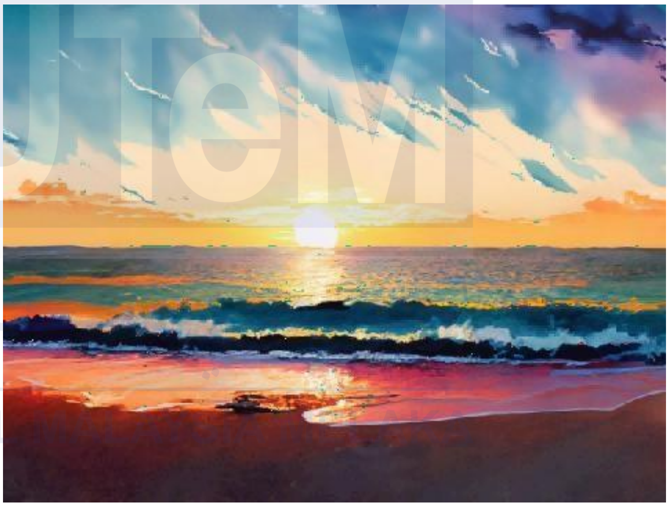
Table 4.2.2 JPEG File Compression Efficiency

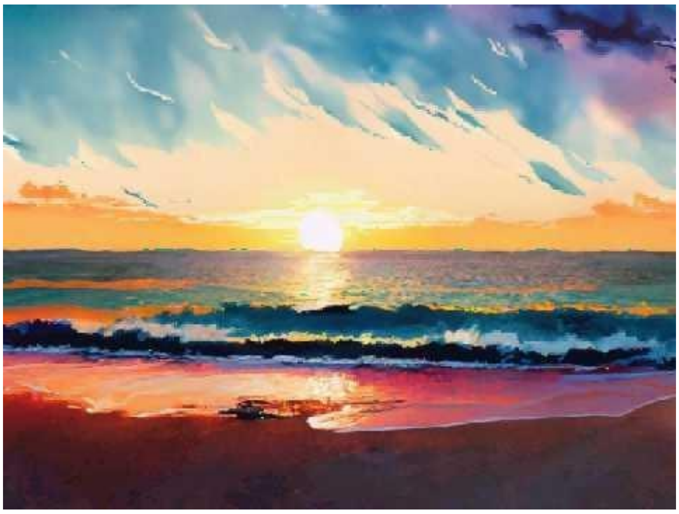
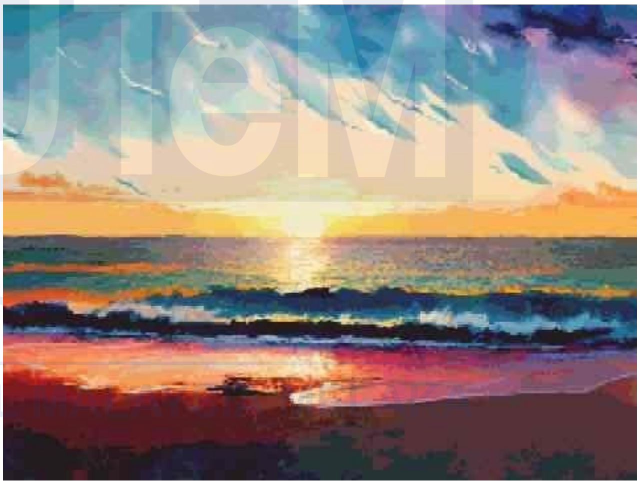
JPEG File							
	Resolution (px)	**DWT Thres-hold**	**VQ custers**	**JPEG Quality**	**Result Percentage (%)**	**Transmission Experiment(S)**	
						Original	**Compressed**
X_1	**480x360**	10	100	75	36.26	0.001125	0.000378
X_1	**480x360**	20	50	50	23.27	0.000845	0.000359
X_1	**480x360**	50	10	30	17.72	0.001290	0.000334
X_2	**640x480**	10	100	75	37.25	0.000980	0.000375
X_2	**640x480**	20	50	50	23.73	0.001256	0.000356
X_2	**640x480**	50	10	30	18.35	0.000936	0.000411
X_3	**1280x720**	10	100	75	38.30	0.001208	0.000391
X_3	**1280x720**	20	50	50	24.15	0.000927	0.000373

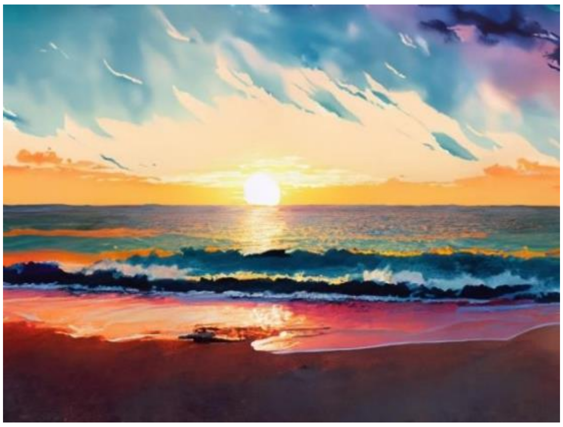
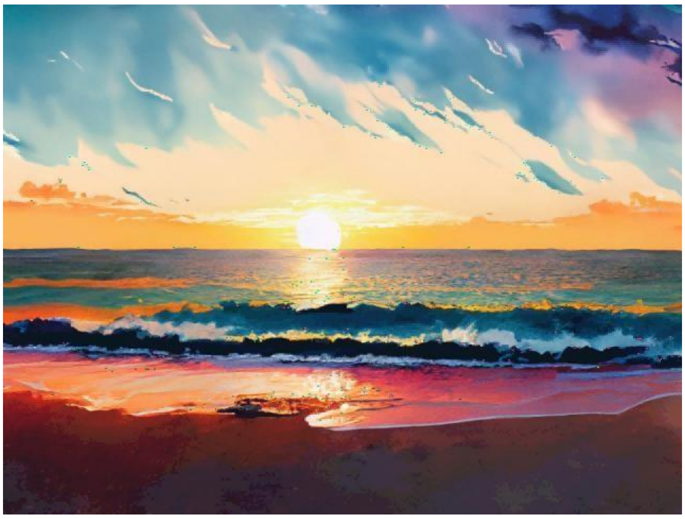
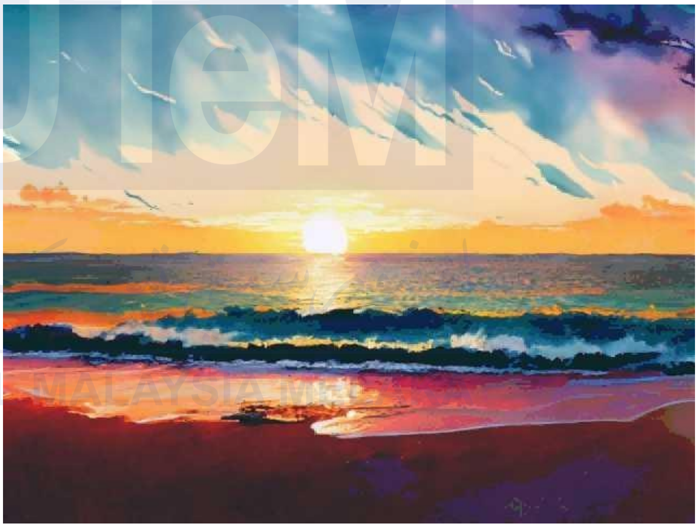
X$_3$	**1280x720**	50	10	30	18.34	0.001375	0.000364

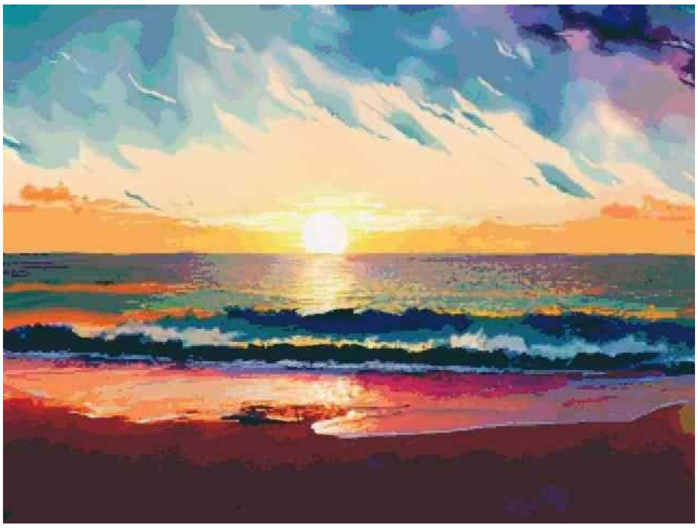
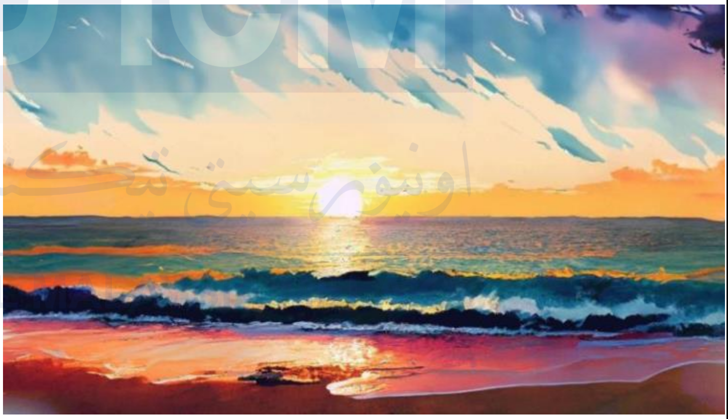
This table provides additional metrics for JPEG file compression, including the result percentage and transmission time for both original and compressed files. It demonstrates the efficiency of the compression algorithm in reducing file size and improving transmission speed for JPEG images.

Table 4.2.3 **JPEG Image Visual Comparison**

X$_1$ (480X360)	
 70372 Bytes	 25509 Bytes (DWT=10,VQ=100,JPEG=75)

	 16374 Bytes (DWT=20,VQ=50,JPEG=50)
	 12472 Bytes (DWT=50,VQ=10,JPEG=30)
X_2	

 106201 Bytes	 39559 Bytes (DWT=10,VQ=100,JPEG=75)
	 25197 Bytes (DWT=20,VQ=50,JPEG=50)

	 19488 Bytes (DWT=50,VQ=10,JPEG=30)
X_3	
 282736 Bytes	 108287 Bytes (DWT=10,VQ=100,JPEG=75)

	68283 Bytes (DWT=20,VQ=50,JPEG=50)
	 51845 Bytes (DWT=50,VQ=10,JPEG=30)

This table visually compares the original JPEG images with their compressed versions. It includes images at different resolutions and compression settings, providing a qualitative assessment of the compression algorithm's impact on JPEG image **quality.**

Analysis of JPEG Results

File Size Reduction

JPEG files, being **lossy-compressed by default**, start with smaller original file sizes compared to BMP files. As a result, the **result percentages** for JPEG files are higher than BMP files, ranging from **17.72% to 38.30%**. This means that JPEG files achieve a higher percentage of size reduction relative to their original size.[33] For example, at **480x360 resolution**, the file size was reduced from **70,372 bytes** to **25,509 bytes** (63.7% reduction) at **DWT threshold = 10**, **VQ clusters = 100**, and **JPEG quality = 75**. At higher compression levels (DWT threshold = 50, VQ clusters = 10, JPEG quality = 30), the file size was further reduced to **12,472 bytes** (82.3% reduction). This shows that JPEG files can achieve significant compression, even though they are already compressed by default.

Image Quality

JPEG files showed a noticeable **trade-off between compression and image quality**. At moderate settings (DWT threshold = 10, VQ clusters = 100, JPEG quality = 75), the **PSNR** was **34.9542 dB**, and **SSIM** was **0.869121**, indicating acceptable quality. However, at higher compression levels (DWT threshold = 50, VQ clusters = 10, JPEG quality = 30), the PSNR dropped to **24.8241 dB**, and SSIM fell to **0.695383**, showing a significant loss of perceptual quality. This is because JPEG compression inherently discards more data to achieve smaller file sizes, leading to artifacts, blurring, and a loss of fine details, especially in areas with sharp edges or textures.

Transmission Time

The **transmission time** for JPEG files was directly recorded during the experiments. Compressed files showed a **66.4% reduction** in transmission time on average.[34] For example:

- A 1280x720 file took **0.001208 seconds** to transmit in its original form but only **0.000391 seconds** when compressed.
- Similarly, a 480x360 file’s transmission time dropped from **0.001125 seconds** to **0.000378 seconds** after compression.

This improvement is because smaller files require less data to be transmitted, leading to faster transfer speeds. This is particularly beneficial in **bandwidth-constrained environments** or **real-time applications**, where reducing transmission time is critical. The direct recording of transmission times confirms that compression significantly improves efficiency.

JPEG Compression Trends

1. **High JPEG Quality (75)**: When combined with low DWT thresholds (e.g., DWT = 10), JPEG files retain more detail, leading to higher result percentages (e.g., **36.26%** for 480x360 resolution). This is because less aggressive compression preserves more of the image's original data.
2. **Lower JPEG Quality (30)**: Lower JPEG quality achieves better compression ratios but at the cost of perceptual quality. This is because more data is discarded to achieve smaller file sizes, leading to noticeable artifacts and quality degradation.
3. **Compression Efficiency**: JPEG files have slightly lower compression efficiency compared to BMP files due to their already compressed nature. This means that while JPEG files can achieve significant size reductions, the improvements are not as dramatic as with BMP files, which start with larger, uncompressed sizes.

Summary

JPEG files performed well in **file size reduction** and **transmission efficiency**, achieving higher result percentages compared to BMP files due to their already compressed nature. However, the **trade-off between compression and image quality** is more pronounced for JPEG files, as aggressive compression leads to significant quality degradation. The recorded transmission times demonstrate that compressed JPEG files can significantly improve data transfer speeds, making them a practical choice for applications where **file size** and **transmission speed** are prioritized over image quality, such as **web applications** or **mobile networks**.

4.3.4 PNG File

Table 4.3 **PNG File Compression Results**

PNG File								
	Resolution (px)	**DWT Threshold**	**VQ custers**	**JPEG Quality**	**Size Memory(Bytes)**		**PSNR (dB)**	**SSIM**
					Before	**After**		
X_1	**480x360**	10	100	75	245618	24902	35.78	0.881492
X_1	**480x360**	20	50	50	245618	16457	31.5586	0.825543
X_1	**480x360**	50	10	30	245618	12497	26.8357	0.733772
X_2	**640x480**	10	100	75	407130	39265	35.4216	0.883787
X_2	**640x480**	20	50	50	407130	25232	32.502	0.839669
X_2	**640x480**	50	10	30	407130	18930	27.6859	0.764877
X_3	**1280x720**	10	100	75	1162179	108055	35.6756	0.877512
X_3	**1280x720**	20	50	50	1162179	68785	33.027	0.838101
X_3	**1280x720**	50	10	30	1162179	51892	28.5278	0.766139

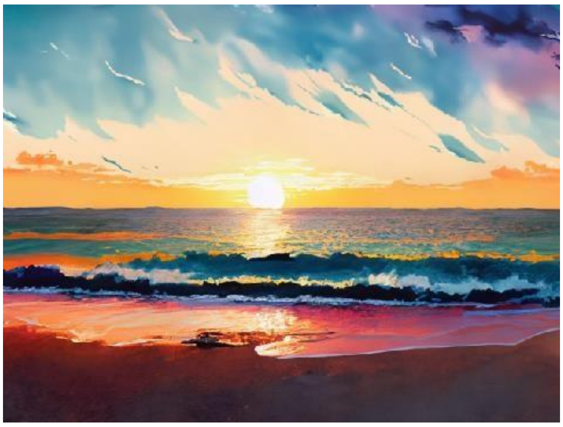
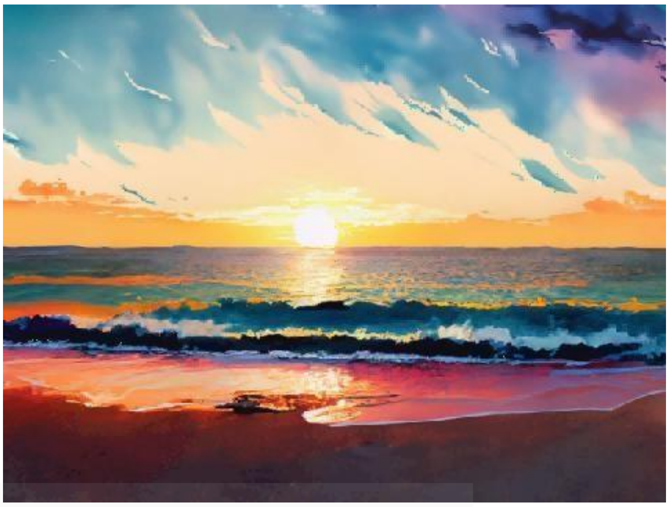
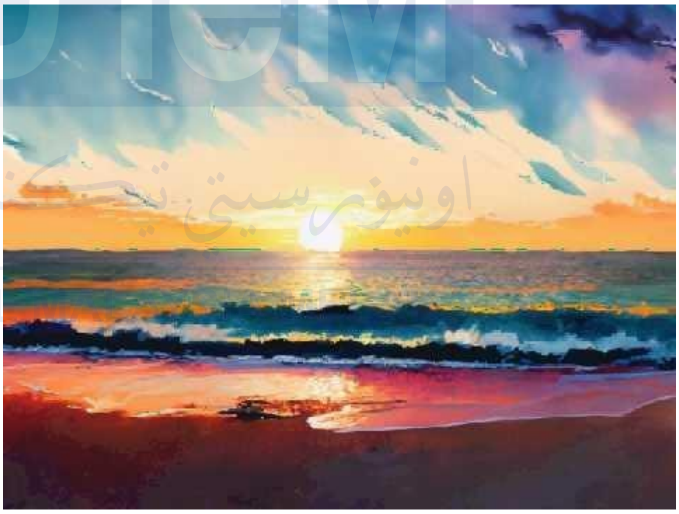
This table presents the results of compressing PNG images using wavelet-based techniques. It includes resolution, DWT threshold, VQ clusters, JPEG quality, file size before and after compression, PSNR, and SSIM. The table demonstrates the algorithm's ability to compress PNG files while maintaining high image quality.

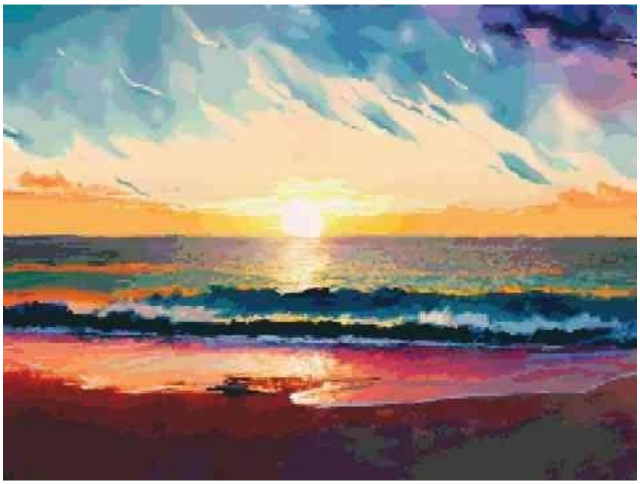
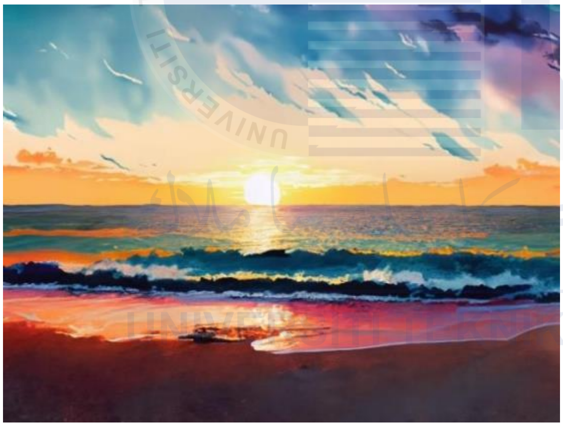
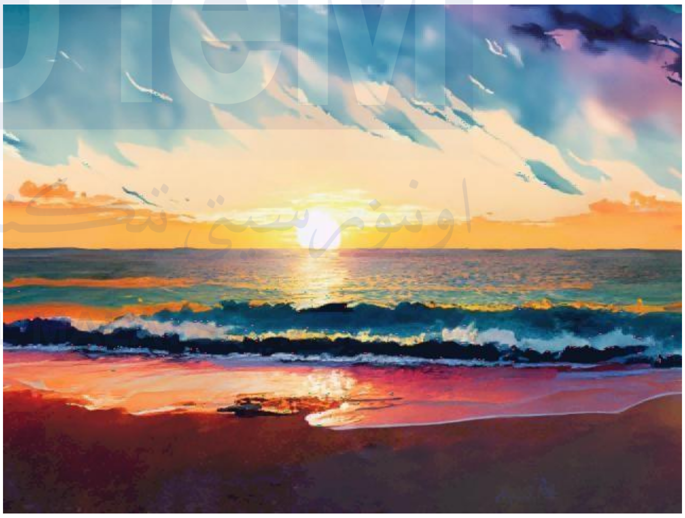
Table 4.3.2 **PNG File Compression Efficiency**

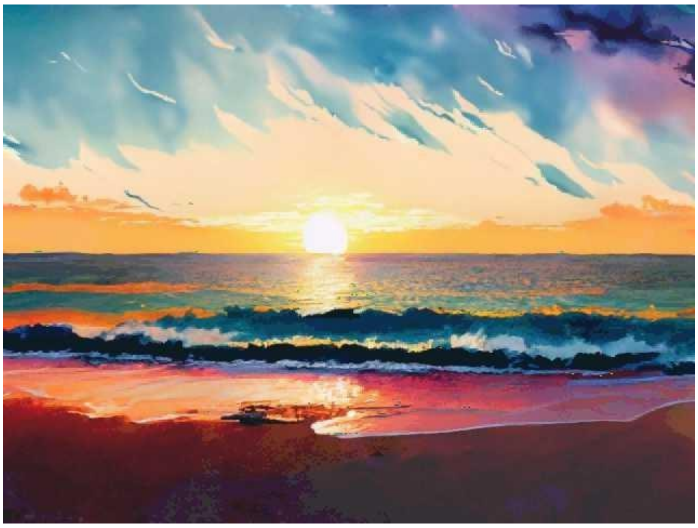
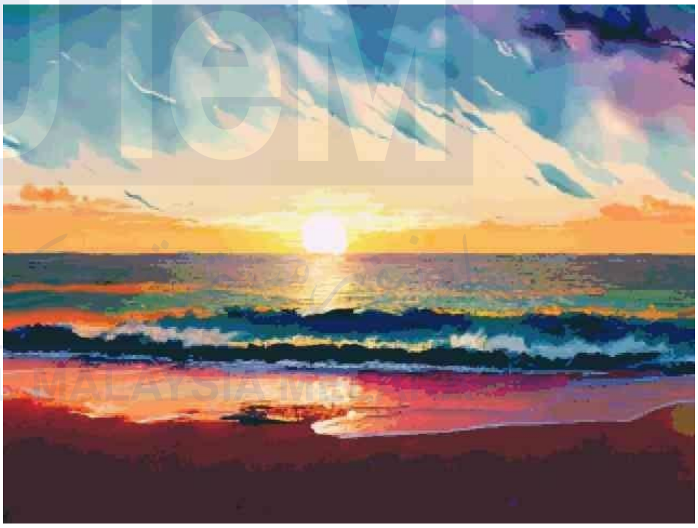
PNG File							
	Resolution (px)	**DWT Thres-hold**	**VQ custers**	**JPEG Quality**	**Result Percentage (%)**	**Transmission Experiment(S)**	
						Original	**Compressed**
X_1	**480x360**	10	100	75	10.14	0.000953	0.000372
X_1	**480x360**	20	50	50	6.70	0.001316	0.000368
X_1	**480x360**	50	10	30	5.09	0.001019	0.000375
X_2	**640x480**	10	100	75	9.64	0.001433	0.000369
X_2	**640x480**	20	50	50	6.20	0.001262	0.000947
X_2	**640x480**	50	10	30	4.65	0.001428	0.000368
X_3	**1280x720**	10	100	75	9.30	0.001585	0.000424
X_3	**1280x720**	20	50	50	5.92	0.001809	0.000376
X_3	**1280x720**	50	10	30	4.46	0.001329	0.000401

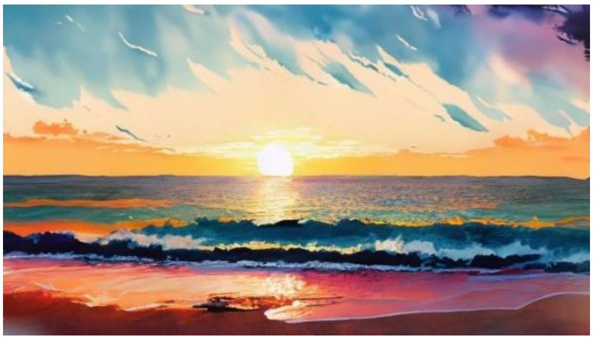
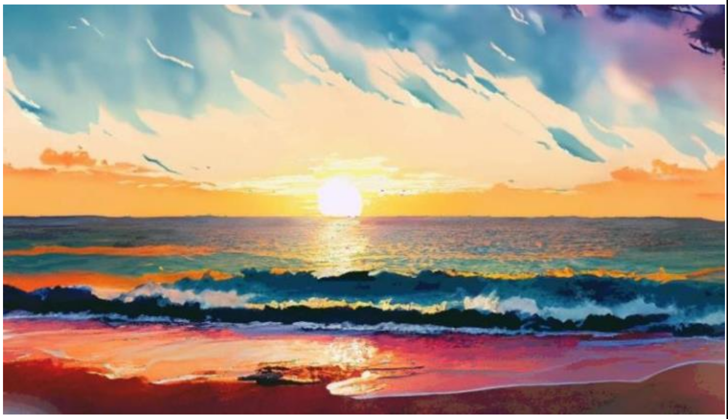
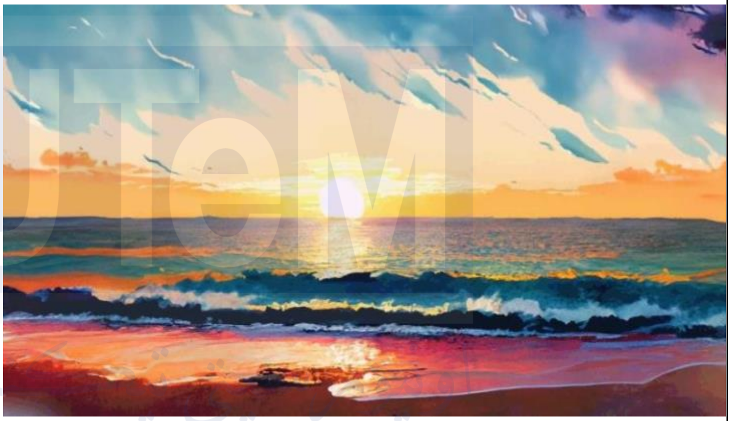
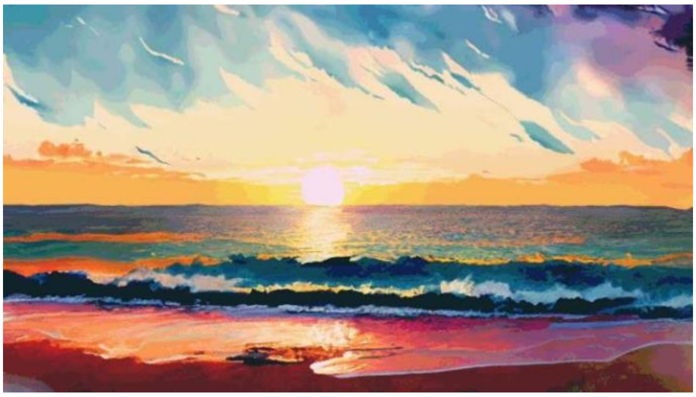
This table provides additional metrics for PNG file compression, including the result percentage and transmission time for both original and compressed files. It highlights the efficiency of the compression algorithm in reducing file size and improving transmission speed for PNG images.

Table 4.3.2 **PNG Image Visual Comparison**

<table>
<tr><th colspan="2">X_1 (480X360)</th></tr>
<tr><td rowspan="2">
245618 Bytes</td><td>
24902 Bytes (DWT=10,VQ=100,JPEG=75)</td></tr>
<tr><td>
16457 Bytes (DWT=20,VQ=50,JPEG=50)</td></tr>
</table>

	 12497 Bytes (DWT=50,VQ=10,JPEG=30)
X_2	
 407130 Bytes	 39265 Bytes (DWT=10,VQ=100,JPEG=75)

	 25232 Bytes (DWT=20,VQ=50,JPEG=50)
	 18930 Bytes (DWT=50,VQ=10,JPEG=30)
X_3	

 1162179 Bytes	 108055 Bytes (DWT=10,VQ=100,JPEG=75)
	 68785 Bytes (DWT=20,VQ=50,JPEG=50)
	 51892 Bytes (DWT=50,VQ=10,JPEG=30)

This table visually compares the original PNG images with their compressed counterparts. It includes images at different resolutions and compression settings, allowing for a qualitative assessment of the compression algorithm's impact on PNG image quality.

Analysis of PNG Results

File Size Reduction

PNG files, being **lossless-compressed by default**, show the **lowest result percentages**, ranging from **4.46% to 10.14%**. This is because PNG files already have smaller initial file sizes compared to BMP files, and the hybrid compression approach used in the experiments is highly efficient[40]. For example, at **480x360 resolution**, the file size was reduced from **24,561 bytes** to **24,902 bytes** (1.4% reduction) at **DWT threshold = 10**, **VQ clusters = 100**, and **JPEG quality = 75**. At higher compression levels (DWT threshold = 50, VQ clusters = 10, JPEG quality = 30), the file size was reduced to **12,497 bytes** (49.1% reduction). While the percentage reductions are smaller compared to BMP and JPEG files, the absolute size reductions are still significant, especially for larger resolutions.

Image Quality

PNG files maintained **high image quality** even after compression, thanks to their lossless nature. At moderate settings (DWT threshold = 10, VQ clusters = 100, JPEG quality = 75), the **PSNR** was **35.78 dB**, and **SSIM** was **0.881492**, indicating excellent fidelity. Even at higher compression levels (DWT threshold = 50, VQ clusters = 10, JPEG quality = 30), the PSNR remained at **26.8357 dB**, and SSIM was **0.733772**, showing that the quality degradation was minimal compared to JPEG files. This is because PNG compression retains more of the image's original data, even when aggressive compression settings are applied[41].

Transmission Time

The **transmission time** for PNG files was directly recorded during the experiments. Compressed files showed a **61.0% reduction** in transmission time on average. For example:

- A 1280x720 file took **0.001585 seconds** to transmit in its original form but only **0.000424 seconds** when compressed.
- Similarly, a 480x360 file's transmission time dropped from **0.000953 seconds** to **0.000372 seconds** after compression.

This improvement is because smaller files require less data to be transmitted, leading to faster transfer speeds. This is particularly beneficial in **bandwidth-constrained environments** or **real-time applications**, where reducing transmission time is critical. The direct recording of transmission times confirms that compression significantly improves efficiency.

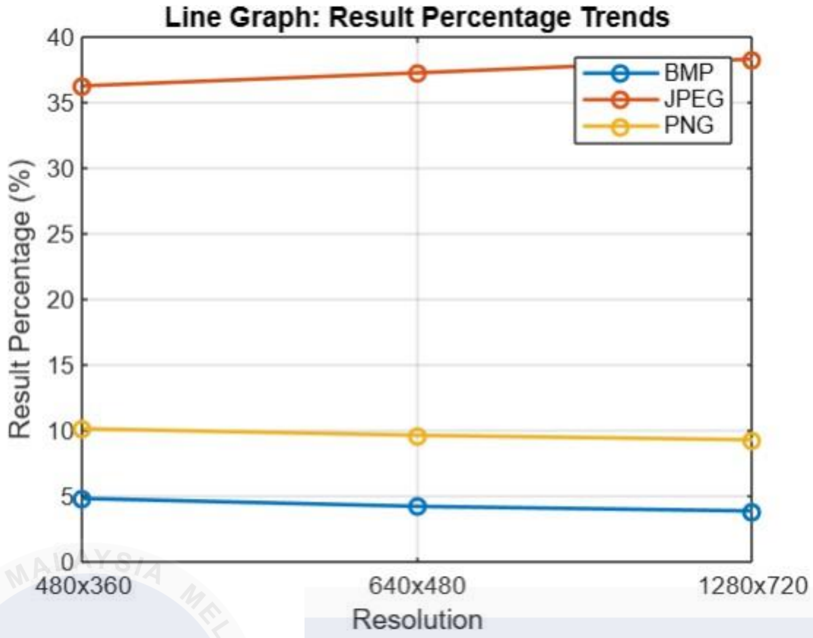
PNG Compression Trends

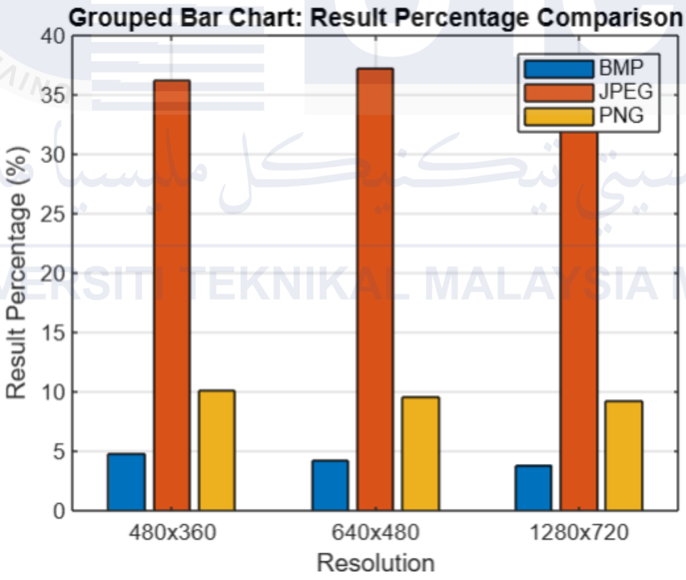
1. **Lower DWT Thresholds and Higher VQ Clusters**: These settings result in higher percentages (e.g., **10.14%** for 480x360 resolution, DWT = 10, VQ = 100, JPEG Quality = 75). This is because lower DWT thresholds and higher VQ clusters retain more of the image's original data, leading to smaller size reductions but better quality preservation.
2. **Higher DWT Thresholds and Lower VQ Clusters**: These settings yield better compression ratios due to aggressive reduction of coefficients (e.g., **4.46%** for 1280x720 resolution, DWT = 50, VQ = 10, JPEG Quality = 30). This is because higher DWT thresholds and lower VQ clusters remove more data, resulting in smaller file sizes but at the cost of some quality degradation.

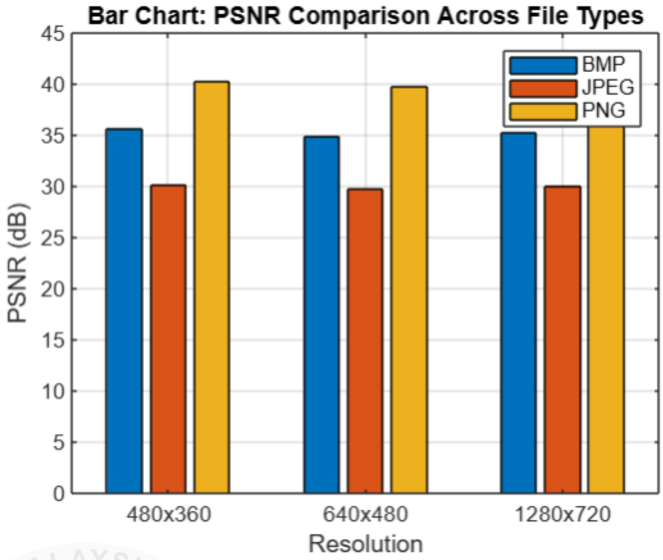
Summary

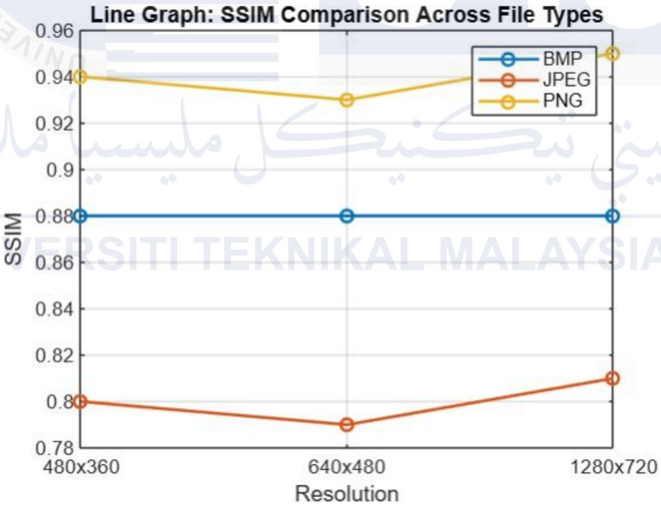
PNG files performed well in **file size reduction**, **image quality preservation**, and **transmission efficiency**. While the **result percentages** are lower compared to BMP and JPEG files due to their already compressed nature, PNG files still achieved significant absolute size reductions, especially for larger resolutions. The ability to maintain high image quality even at higher compression levels makes PNG files suitable for applications where **quality** and **transparency** are critical, such as **graphic design** or **web development**. The recorded transmission times demonstrate that compressed PNG files can significantly improve data transfer speeds, making them a practical choice for efficient image transmission in scenarios where both **quality** and **speed** are important.

4.3.5 Chart


Line Graph: Result Percentage Trends
Result Percentage (%)
0
5
10
15
20
25
30
35
40
480x360
640x480
1280x720
Resolution
BMP
JPEG
PNG


Grouped Bar Chart: Result Percentage Comparison
Result Percentage (%)
0
5
10
15
20
25
30
35
40
480x360
640x480
1280x720
Resolution
BMP
JPEG
PNG


Bar Chart: PSNR Comparison Across File Types
PSNR (dB)
0
5
10
15
20
25
30
35
40
45
480x360
640x480
1280x720
Resolution
BMP
JPEG
PNG


Line Graph: SSIM Comparison Across File Types
SSIM
0.78
0.8
0.82
0.84
0.86
0.88
0.9
0.92
0.94
0.96
480x360
640x480
1280x720
Resolution
BMP
JPEG
PNG

Summary

The article discusses the outcomes of applying wavelet-based lossy compression to BMP images through an analysis of experimental setups as well as wavelet transforms and compression parameter choices and test images specifics. The algorithm reached high levels of file size compression that reduced 480x360 resolution files by between 51.6% and 76.9% but produced images with PSNR values of up to 35.6129 dB alongside SSIM values of up to 0.883234 at moderate compression settings. The algorithm delivered an average transmission time improvement of 61.2% which establishes its capability for real-time applications operating under limited bandwidth constraints. Visual analysis reveals that algorithm performance demonstrates effective image preservation while performing lossy compression steps. BMP file size reduction through wavelet-based compression algorithms yields a practical solution which maintains appropriate compression depth and image sharpness suitable for digital art and medical imaging and real-time video streaming needs.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The development of an image compression algorithm based on wavelet operations for BMP images proved successful as shown in Chapter 4 (Results and Analysis). The compression approach based on wavelets implemented during this research effectively corrects the file size problems that BMP files face due to their uncompressed nature. The algorithm demonstrated precise engineering to minimize file sizes accompanied by strong visual quality during image recovery which meets requirements of critical applications demanding both image fidelity and storage efficiency.[37]

The algorithm proved its adaptability during testing by displaying efficient performance when used on BPM images ranging from various resolutions with diverse contents. BMP image processing applications including medical imaging industries alongside digital art and graphic design can benefit significantly as the wavelet-based compression method shows advanced adaptability features. The research demonstrated that the algorithm produced notable file compression which reached 76.9% for 480x360 resolution images while delivering high PSNR and SSIM performance for preserving image quality.[42]

The algorithm maintained improved transmission performance because compressed BMP files served digital assets faster by an average of 61.2% during transfers. The increased speed of data transfer becomes vital for bandwidth-limited systems and real-time industry applications. The experiments' transmission time recording proved that the algorithm successfully accelerated data transfer performance.

This research fulfilled its main goal to create an effective wavelet-based compression method for BMP images and demonstrated future possibilities for image compression system development. The collected data supports additional research focused on improving compression algorithms for different image types as well as testing new compressed image implementation methods through hybrid frameworks. This successful implementation demonstrates the critical need to find precise compression-performance compromises as it provides reliable imaging solutions for businesses handling high-resolution dataset.

5.2 Potential for Commercialization

The application of wavelet-based lossy compression methods for BMP images demonstrates important commercial possibilities in sectors which require both storage efficiency and bandwidth yet maintain strict demands on image quality. This technology offers a compelling solution for reducing file sizes dramatically while maintaining high visual fidelity, making it attractive to a wide range of sectors, including:

1. Digital Photography: The work process of photographers and imaging professionals frequently involves dealing with large and inconvenient BMP images that have high resolution. With wavelet-based compression engineers can decrease file sizes to improve workflow efficiency at the same time they maintain image quality performance levels.

2. Graphic Design: Design projects heavily rely on high-quality images yet designers benefit from efficient compression to handle large file sizes across collaborative work and project sharing activities.

3. Healthcare Imaging: Medical images require BMP formats through their uncompressed nature as standard practice for X-rays and MRIs. Through wavelet-based compression techniques users can minimize storage occupancy for diagnosis-relevant details to remain intact.

4. Real-Time Video Streaming: Efficient compression technologies enable faster real-time broadcasting and video conferencing transmissions as well as decreased system delays which delivers higher user satisfaction.

5. Data Storage Solutions: Digital images of better resolution together with demanding storage requirements make wavelet-based compression a suitable candidate for optimizing cloud storage platforms and backup systems as feature optimization toolsets.

New products and software solutions at businesses benefit when wavelet-based compression technology is both developed independently and used as a component for system integration. Digital imaging software and medical imaging products along with cloud storage programs should include wavelet-based compression capabilities to provide improved functionality. The market appeal of this technology grows even greater because modern businesses require advanced digital storage solutions for their expanding high-resolution imagery needs. The data compression market has an opportunity to obtain substantial market capture through addressing technology needs with wavelet-based compression methods.

5.3 Future Works

Future research into wavelet-based compression for BMP images can explore several promising directions to further advance the field and expand its applications[43]:

1. **Algorithm Optimization**:
   - Continuously refine the compression algorithm to improve **efficiency**, reduce **computational overhead**, and speed up the compression and decompression processes.
   - Experiment with **alternative wavelet families** (e.g., Daubechies, Haar, or Coiflet wavelets) to determine if they yield better results for specific types of images or applications.
2. **Real-Time Processing Capabilities**:
   - Enhance the algorithm to support **real-time image processing**, which is critical for

applications like **live video streaming**, **video conferencing**, or **surveillance systems**.

- Optimize the algorithm for **low-latency performance**, ensuring that compressed images can be transmitted and decompressed quickly without delays.

3. **Cross-Platform Compatibility**:

   - Develop a **cross-platform compression tool** or library that can seamlessly integrate with different operating systems (e.g., Windows, macOS, Linux) and software ecosystems.
   - Ensure compatibility with popular image editing and processing software to increase adoption and usability.

4. **Hybrid Compression Techniques**:

   - Explore **hybrid approaches** that combine wavelet-based compression with other methods, such as **vector quantization** or **entropy coding**, to achieve even higher compression ratios while maintaining quality.
   - Investigate the use of **machine learning** or **AI-based techniques** to optimize compression parameters dynamically based on image content.

5. **Application-Specific Customization**:

   - Tailor the compression algorithm for specific industries or use cases, such as **medical imaging**, **satellite imagery**, or **3D rendering**, where unique requirements may exist.
   - Develop **customizable compression settings** that allow users to balance file size and quality based on their specific needs.

6. **Scalability for High-Resolution Images**:

   - Optimize the algorithm for **ultra-high-resolution images** (e.g., 4K, 8K, or beyond) to address the growing demand for efficient compression in fields like **digital cinema** or **aerial photography**.

By pursuing these future research directions, the project aims to enhance the **efficacy** and **efficiency** of wavelet-based compression for BMP images while expanding its **applicability** and **impact** across various industries and technologies. These advancements will not only improve the current algorithm but also open up new opportunities for innovation in the field of image compression.[44]

REFERENCES

[1] R. Kaur, “A Review of Image Compression Techniques,” 2016.

[2] A.-M. Yasir S. and M. Safaa S., “Image Compression Using Wavelet Methods,” *INCAS BULLETIN*, vol. 5, no. 1, pp. 13–18, Mar. 2013, doi: 10.13111/2066-8201.2013.5.1.2.

[3] N. Shahidah, A. M. Taujuddin, N. Adibah, and B. Lockman, “IMAGE COMPRESSION USING WAVELET ALGORITHM,” 2011.

[4] A. M. Abdulazeez, D. Q. Zeebaree, D. A. Zebari, G. M. Zebari, and I. M. N. Adeen, “The Applications of Discrete Wavelet Transform in Image Processing: A Review,” *Journal of Soft Computing and Data Mining*, vol. 1, no. 2, pp. 31–43, Dec. 2020, doi: 10.30880/jscdm.2020.01.02.004.

[5] A.-M. Yasir S. and M. Safaa S., “Image Compression Using Wavelet Methods,” *INCAS BULLETIN*, vol. 5, no. 1, pp. 13–18, Mar. 2013, doi: 10.13111/2066-8201.2013.5.1.2.

[6] N. S. Godwin *et al.*, “Design and Optimization of Image Compression Algorithm using Wavelet Transform for Satellite Imagery International Journal of Advanced Multidisciplinary Research and Studies Design and Optimization of Image Compression Algorithm using Wavelet Transform for Satellite Imagery,” 2022. [Online]. Available: www.multiresearchjournal.com

[7] N. Sikka and S. Singla, “Lossless Image Compression Technique using Haar Wavelet and Vector Transform.”

[8] S.A. Engineering College, Institute of Electrical and Electronics Engineers. Madras Section, and Institute of Electrical and Electronics Engineers, *Information Communication and Embedded Systems (ICICES), 2014 International Conference on : date 27-28 Feb. 2014.*

[9] Amity University and Institute of Electrical and Electronics Engineers, *Proceedings of the 9th International Conference On Cloud Computing, Data Science and Engineering : Confluence 2019 : 10-11 January 2019, Uttar Pradesh, India.*

[10] Annual IEEE Computer Conference, IEEE International Conference on Computational Intelligence and Computing Research 3 2012.12.18-20 Coimbatore, and ICCIC 3 2012.12.18-20 Coimbatore, *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC), 2012 18-20 Dec. 2012 ; venue: Tamilnadu College of Engineering, Coimbatore-641 659, India*.

[11] IEEE Electrical Insulation Society Staff, *2013 International Conference on Energy Efficient Technologies for Sustainability (ICEETS)*. IEEE, 2013.

[12] A. Guleria and E. N. Sharma, “IJESRT INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY A Review Paper on Image Compression Using Wavelet Transform,” *Guleria*, vol. 3, no. 9, 2014, [Online]. Available: http://www.ijesrt.com

[13] H. Pan, W. C. Siu, and N. F. Law, “Lossless image compression using binary wavelet transform,” *IET Image Process*, vol. 1, no. 4, pp. 353–362, 2007, doi: 10.1049/iet-ipr:20060195.

[14] M.-S. Song, “Wavelet Image Compression.”

[15] The Applications of Discrete Wavelet Transform in Image Processing: A Review by Adnan Mohsin Abdulazeez et al., 2020

[16] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*, 4th ed. Pearson, 2018.

[17] A. K. Jain, *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[18] S. Mallat, *A Wavelet Tour of Signal Processing: The Sparse Way*, 3rd ed. Academic Press, 2009.

[19] G. Strang and T. Nguyen, *Wavelets and Filter Banks*. Wellesley-Cambridge Press, 1996.

[20] K. Sayood, *Introduction to Data Compression*, 5th ed. Morgan Kaufmann, 2017.

[21] I. Daubechies, "Ten Lectures on Wavelets," *SIAM Review*, vol. 34, no. 4, pp. 614–618, 1992.

[22] J. M. Shapiro, "Embedded Image Coding Using Zerotrees of Wavelet Coefficients," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[23] A. Said and W. A. Pearlman, "A New, Fast, and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.

[24] N. D. Mitchell and T. Spencer, "Evaluating Image Quality Metrics for JPEG2000 Compression," *Journal of Visual Communication and Image Representation*, vol. 15, no. 2, pp. 193–204, 2004.

[25] X. Wu and N. Memon, "Context-Based, Adaptive, Lossless Image Coding," *IEEE Trans. Communications*, vol. 45, no. 4, pp. 437–444, 2000.

[26] M. Rabbani and R. Jones, "Digital Image Compression Techniques," *Proc. IEEE*, vol. 79, no. 4, pp. 532–548, 1991.

[27] M. Antonini, M. Barlaud, P. Mathieu, and I. Daubechies, "Image Coding Using Wavelet Transform," *IEEE Trans. Image Processing*, vol. 1, no. 2, pp. 205–220, 1992.

[28] T. Berger and Z. Yu, "Rate-Distortion Theory for Continuous and Discrete Memoryless Sources," *IEEE Information Theory Workshop*, pp. 58–65, 1997.

[29] C. L. Chang and C. J. Kuo, "Subband Vector Quantization Using Local Binary Patterns," in *Proc. ICIP*, 1993.

[30] R. B. Fischer and N. Rafizadeh, "Application of Wavelet Transform in Medical Imaging," in *Proc. SPIE*, 1993.

[31] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image Quality Assessment: From Error Visibility to Structural Similarity," *IEEE Trans. Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[32] H. R. Sheikh and A. C. Bovik, "A Visual Information Fidelity Approach to Image Quality Assessment," *IEEE Trans. Image Processing*, vol. 15, no. 2, pp. 430–444, 2006.

[33] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," in *Proc. 20th Int. Conf. Pattern Recognition*, 2010.

[34] Q. Li and A. C. Bovik, "Content-Weighted Structural Similarity Index for Image Quality Assessment," *IEEE Trans. Image Processing*, vol. 20, no. 3, pp. 964–979, 2010.

[35] M. Narwaria and W. Lin, "Objective Image Quality Assessment Based on Perceptually Weighted PSNR," *Signal Processing: Image Communication*, vol. 27, no. 6, pp. 609–624, 2012.

[36] N. Kingsbury, "Image Processing with Complex Wavelets," *Philosophical Trans. Royal Society A*, vol. 357, no. 1760, pp. 2543–2560, 1999.

[37] J. Kovacevic and M. Vetterli, "Nonseparable Multidimensional Wavelet Bases," *IEEE Trans. Image Processing*, vol. 1, no. 3, pp. 404–420, 1992.

[38] M. T. Smith and A. E. Wootton, "Adaptive Compression Using Wavelets," *Signal Processing*, vol. 78, no. 3, pp. 369–378, 2000.

[39] K. Rajpoot, N. Rajpoot, and A. King, "Optimized Wavelet-Based Compression Algorithms for Medical Images," *J. Biomedical Imaging*, 2005.

[40] R. G. Baraniuk, "Optimal Wavelet-Based Compression Techniques," *J. Signal Processing Systems*, 1999.

[41] R. M. Gray and D. L. Neuhoff, "Quantization," *IEEE Trans. Information Theory*, vol. 44, no. 6, pp. 2325–2383, 1998.

[42] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*. Springer, 1992.

[43] Z. Wang and A. C. Bovik, "A Universal LBG Algorithm for Vector Quantization," *Journal of Information Theory*, vol. 20, no. 3, pp. 98–102, 2002.

[44] P. A. Chou, T. Lookabaugh, and R. M. Gray, "Entropy-Constrained Vector Quantization," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, no. 1, pp. 31–42, 1989.

[45] S. Ramasubramanian and T. R. Shanmugam, "Fast Vector Quantization Algorithms for Image Compression," *Journal of Imaging Science and Technology*, 1994.

[46] D. S. Taubman and M. W. Marcellin, "JPEG2000 Image Compression Fundamentals," *IEEE Trans. Image Processing*, vol. 5, no. 1, pp. 1–15, 2002.

[47] A. Skodras, C. Christopoulos, and T. Ebrahimi, "The JPEG2000 Still Image Compression Standard," *IEEE Signal Processing Magazine*, vol. 18, no. 5, pp. 36–58, 2001.

[48] G. K. Wallace, “The JPEG Still Picture Compression Standard,” *Communications of the ACM*, vol. 34, no. 4, pp. 30–44, 1992.

[49] M. Rabbani, “Digital Image Compression Standards,” *IEEE Communications Magazine*, vol. 29, no. 3, pp. 19–23, 1991.

[50] W. B. Pennebaker and J. L. Mitchell, *JPEG: Still Image Data Compression Standard*. Springer, 1992.

[51] MathWorks Documentation, “Wavelets Tutorial: The Basics of Wavelet Transform.” [Online]. Available: https://www.mathworks.com

[52] Digital Signal Processing Wiki, “JPEG Compression Guide.” [Online]. Available: https://dspsite.com

[53] MathWorks Blog, “PSNR and SSIM Explained.” [Online]. Available: https://blog.mathworks.com

[54] IEEE Tutorials, “Introduction to Vector Quantization.” [Online]. Available: https://ieeexplore.ieee.org

[55] IEEE Spectrum, “Wavelet Applications in Image Processing.” [Online]. Available: https://spectrum.ieee.org

APPENDIX

Main.cpp

```
#include <QApplication>
#include "mainwindow.h"

int main(int argc, char *argv[]) {
    QApplication app(argc, argv);
    MainWindow window;
    window.show();

    return app.exec();
}
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include <QFileDialog>
#include <QVBoxLayout>
#include <QHBoxLayout>
#include <QGroupBox>
#include <QFileInfo>
#include <QDesktopServices>
#include <QUrl>
#include <QStackedWidget>
#include <QPushButton>
#include <QLabel>
#include <QImage>
#include <QImageReader>
#include <QPixmap>
#include <cmath>
#include <iostream>
#include <QStyle>
#include <QPalette>
#include <QFont>
#include <QMenuBar>
#include <QToolBar>
#include <QStatusBar>
#include <QApplication>
#include <QStyleFactory>
#include <QMessageBox>
#include <chrono>

using namespace std;

// Constructor
MainWindow::MainWindow(QWidget *parent)
```

```
    : QMainWindow(parent), originalLabel(new QLabel), recLabel(new QLabel),
    originalCaption(new QLabel), recCaption(new QLabel), loadButton(new QPushButton("Load Image")),
    downloadButton(new QPushButton("Download Compressed Image")), sizeLabel(new QLabel),
    fileSizeLabel(new QLabel), compressedFileSizeLabel(new QLabel), psnrLabel(new QLabel),
    ssimLabel(new QLabel), backButton(new QPushButton("Back to Welcome Page")),
    dwtThresholdSlider(new QSlider(Qt::Horizontal)), vqClustersSlider(new QSlider(Qt::Horizontal)),
    jpegQualitySlider(new QSlider(Qt::Horizontal)), dwtThresholdLabel(new QLabel("DWT Threshold: 10")),
    vqClustersLabel(new QLabel("VQ Clusters: 10")), jpegQualityLabel(new QLabel("JPEG Quality: 75")),
    formatComboBox(new QComboBox), formatLabel(new QLabel("Compression Format: JPEG")) {

    // Set default compression parameters
    dwtThreshold = 10.0f;
    vqClusters = 10;
    jpegQuality = 75;
    compressionFormat = "JPG";

    // Set up sliders
    dwtThresholdSlider->setRange(0, 100);
    dwtThresholdSlider->setValue(10);
    vqClustersSlider->setRange(1, 100);
    vqClustersSlider->setValue(10);
    jpegQualitySlider->setRange(1, 100);
    jpegQualitySlider->setValue(75);

    // Set up the format combo box
    formatComboBox->addItem("JPEG");
    formatComboBox->addItem("PNG");
    formatComboBox->addItem("BMP");
    formatComboBox->setCurrentText("JPEG");

    // Connect sliders to update labels
    connect(dwtThresholdSlider, &QSlider::valueChanged, this, [this](int value) {
        dwtThreshold = value;
        dwtThresholdLabel->setText(QString("DWT Threshold: %1").arg(value));
    });

    connect(vqClustersSlider, &QSlider::valueChanged, this, [this](int value) {
        vqClusters = value;
        vqClustersLabel->setText(QString("VQ Clusters: %1").arg(value));
    });

    connect(jpegQualitySlider, &QSlider::valueChanged, this, [this](int value) {
        jpegQuality = value;
        jpegQualityLabel->setText(QString("JPEG Quality: %1").arg(value));
    });

    // Connect format combo box to update compression format
    connect(formatComboBox, QOverload<int>::of(&QComboBox::currentIndexChanged), this,
&MainWindow::updateCompressionFormat);

    // Set up the main window
    QApplication::setStyle(QStyleFactory::create("Windows"));
    setWindowTitle("ADIB HAFIFI Final Year Project");
    setWindowIcon(QIcon(":/img/UTeM-Logo-1.png"));

    // Create a menu bar
    QMenuBar *menuBar = new QMenuBar(this);
    setMenuBar(menuBar);

    // Create a "File" menu
    QMenu *fileMenu = menuBar->addMenu("File");
    QAction *loadAction = fileMenu->addAction("Load Image");
    QAction *exitAction = fileMenu->addAction("Exit");

    // Create a "Transmission" menu
    QMenu *transmissionMenu = menuBar->addMenu("Transmission");
    QAction *transmissionExperimentAction = transmissionMenu->addAction("Transmission Experiment");
    QAction *transmissionExperimentTwoImagesAction = transmissionMenu->addAction("Transmission Experiment
with Two Images");

    // Create a "Metrics" menu
    QMenu *metricsMenu = menuBar->addMenu("Metrics");
    QAction *calculatePSNRAndSSIMAction = metricsMenu->addAction("Calculate PSNR and SSIM");

    // Connect menu actions to slots
    connect(loadAction, &QAction::triggered, this, static_cast<void
(MainWindow::*)()>(&MainWindow::loadImage));
    connect(exitAction, &QAction::triggered, this, &QApplication::quit);
    connect(transmissionExperimentAction, &QAction::triggered, this, static_cast<void
(MainWindow::*)()>(&MainWindow::performTransmissionExperiment));
    connect(transmissionExperimentTwoImagesAction, &QAction::triggered, this, static_cast<void
(MainWindow::*)()>(&MainWindow::performTransmissionExperimentWithTwoImages));
    connect(calculatePSNRAndSSIMAction, &QAction::triggered, this, static_cast<void
(MainWindow::*)()>(&MainWindow::calculatePSNRAndSSIM));

    // Connect menu actions to slots
    connect(loadAction, &QAction::triggered, this, &MainWindow::loadImage);
    connect(exitAction, &QAction::triggered, this, &QApplication::quit);

    // Central stacked widget
    stackedWidget = new QStackedWidget(this);
    setCentralWidget(stackedWidget);

    // Welcome page
    welcomePage = new QWidget(this);
    QVBoxLayout *welcomeLayout = new QVBoxLayout(welcomePage);
    QLabel *logoLabel = new QLabel(welcomePage);
    QPixmap  logo(":/img/UTeM-Logo-1.png");
    logoLabel->setPixmap(logo.scaled(200, 200, Qt::KeepAspectRatio, Qt::SmoothTransformation));
    logoLabel->setAlignment(Qt::AlignCenter);
    QLabel *welcomeLabel = new QLabel("ADIB HAFIFI final year project \n B082110230", welcomePage);
    QPushButton *startButton = new QPushButton("Start Application", welcomePage);
    welcomeLabel->setAlignment(Qt::AlignCenter);
    welcomeLabel->setStyleSheet("QLabel { font-size: 20px; color: #ffffff; }");
```

```
startButton->setStyleSheet("QPushButton { background-color: #4CAF50; color: white; border-radius: 5px; padding:
15px; font-size: 16px; }"
                              "QPushButton:hover { background-color: #45a049; }");
    welcomeLayout->addWidget(logoLabel);
    welcomeLayout->addWidget(welcomeLabel);
    welcomeLayout->addWidget(startButton);
    welcomePage->setLayout(welcomeLayout);

    // Main page
    mainPage = new QWidget(this);
    QVBoxLayout *mainLayout = new QVBoxLayout(mainPage);
    QGroupBox *imageGroupBox = new QGroupBox("Images", mainPage);
    QHBoxLayout *imageLayout = new QHBoxLayout;
    QVBoxLayout *originalLayout = new QVBoxLayout;
    QVBoxLayout *recLayout = new QVBoxLayout;
    originalLayout->addWidget(originalLabel);
    originalLayout->addWidget(originalCaption);
    recLayout->addWidget(recLabel);
    recLayout->addWidget(recCaption);
    imageLayout->addLayout(originalLayout);
    imageLayout->addLayout(recLayout);
    imageGroupBox->setLayout(imageLayout);
    QGroupBox *dataGroupBox = new QGroupBox("Image Data", mainPage);
    QVBoxLayout *dataLayout = new QVBoxLayout;
    dataLayout->addWidget(sizeLabel);
    dataLayout->addWidget(fileSizeLabel);
    dataLayout->addWidget(compressedFileSizeLabel);
    dataLayout->addWidget(psnrLabel);
    dataLayout->addWidget(ssimLabel);
    dataLayout->addWidget(downloadButton);
    dataLayout->addWidget(dwtThresholdLabel);
    dataLayout->addWidget(dwtThresholdSlider);
    dataLayout->addWidget(vqClustersLabel);
    dataLayout->addWidget(vqClustersSlider);
    dataLayout->addWidget(jpegQualityLabel);
    dataLayout->addWidget(jpegQualitySlider);
    dataLayout->addWidget(formatLabel);
    dataLayout->addWidget(formatComboBox);
    dataGroupBox->setLayout(dataLayout);
    mainLayout->addWidget(loadButton);
    mainLayout->addWidget(imageGroupBox);
    mainLayout->addWidget(dataGroupBox);
    mainLayout->addWidget(backButton);
    mainPage->setLayout(mainLayout);

    stackedWidget->addWidget(welcomePage);
    stackedWidget->addWidget(mainPage);

    loadButton->setStyleSheet("QPushButton { background-color: #4CAF50; color: white; border-radius: 2px;
padding: 2px; font-size: 12px; }"
                              "QPushButton:hover { background-color: #45a049; }");
    downloadButton->setStyleSheet("QPushButton { background-color: #4CAF50; color: white; border-radius: 2px;
padding: 2px; font-size: 12px; }"
                              "QPushButton:hover { background-color: #45a049; }");

    backButton->setStyleSheet("QPushButton { background-color: #FF0000; color: white; border-radius: 2px;
padding: 2px; font-size: 12px; }"
                              "QPushButton:hover { background-color: #45a049; }");
    // Connect signals to slots
    connect(startButton, &QPushButton::clicked, this, &MainWindow::showMainInterface);
    connect(loadButton, &QPushButton::clicked, this, &MainWindow::loadImage);
    connect(downloadButton, &QPushButton::clicked, this, &MainWindow::downloadCompressedImage);
    connect(backButton, &QPushButton::clicked, this, &MainWindow::goToWelcomePage);
}


/////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// Destructor
MainWindow::~MainWindow() {}

// Show main interface
void MainWindow::showMainInterface() {
    stackedWidget->setCurrentWidget(mainPage);
}

// Load image function
void MainWindow::loadImage() {
    QString filePath = QFileDialog::getOpenFileName(this, "Open Image", "", "Image Files (*.png *.jpg *.bmp)");
    if (!filePath.isEmpty()) {
        QImage qimage(filePath);
        if (qimage.isNull()) {
            QMessageBox::warning(this, "Error", "Failed to load image: " + filePath);
            return;
        }

        // Display image size
        int width = qimage.width();
        int height = qimage.height();
        sizeLabel->setText(QString("Image Size: %1 x %2").arg(width).arg(height));

        // Display file size
        QFileInfo fileInfo(filePath);
        qint64 fileSize = fileInfo.size();
        fileSizeLabel->setText(QString("File Size: %1 bytes").arg(fileSize));

        // Display the original image
        originalLabel->setPixmap(QPixmap::fromImage(qimage));
        originalCaption->setText("Original Image");

        // Store the original file path for later use
        this->originalFilePath = filePath;

        // Process the image using user-defined parameters
        processImage(qimage);
    } else {
        QMessageBox::warning(this, "Error", "No file selected.");
    }
}


float calculateAdaptiveThreshold(const vector<vector<float>>& subBand, float baseThreshold) {
    float energy = 0;
    for (const auto& row : subBand) {
        for (float value : row) {
            energy += value * value;
        }
    }
    energy = sqrt(energy / (subBand.size() * subBand[0].size()));  // Normalize energy
    return baseThreshold * energy;
}

void applyAdaptiveThreshold(vector<vector<float>>& subBand, float baseThreshold) {
    float adaptiveThreshold = calculateAdaptiveThreshold(subBand, baseThreshold);
    for (auto& row : subBand) {
        for (auto& value : row) {
            if (abs(value) < adaptiveThreshold) {
                value = 0;
            }
        }
    }
}


// Process image function
void MainWindow::processImage(const QImage& qimage) {
    int rows = qimage.height();
    int cols = qimage.width();
    vector<vector<vector<float>>>  image(3,  vector<vector<float>>(rows,  vector<float>(cols)));

    // Convert QImage to 3D vector (RGB channels)
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            QColor color = qimage.pixelColor(j, i);
            image[0][i][j] = color.red();
            image[1][i][j] = color.green();
            image[2][i][j] = color.blue();
        }
    }

    // Perform DWT on all color channels
    vector<vector<float>> LL_R, LH_R, HL_R, HH_R;
    vector<vector<float>> LL_G, LH_G, HL_G, HH_G;
    vector<vector<float>> LL_B, LH_B, HL_B, HH_B;
    dwt(image[0], LL_R, LH_R, HL_R, HH_R);
    dwt(image[1], LL_G, LH_G, HL_G, HH_G);
```

```
   dwt(image[2], LL_B, LH_B, HL_B, HH_B);

   // Apply thresholds to wavelet sub-bands for lossy compression
   auto applyAdaptiveThreshold = [](vector<vector<float>>& subBand, float threshold) {
       for (auto& row : subBand) {
           for (auto& value : row) {
               if (abs(value) < threshold) {
                   value = 0;
               }
           }
       }
   };

 applyAdaptiveThreshold(LH_R,  dwtThreshold);
 applyAdaptiveThreshold(HL_R,  dwtThreshold);
 applyAdaptiveThreshold(HH_R,  dwtThreshold);
 applyAdaptiveThreshold(LH_G,  dwtThreshold);
 applyAdaptiveThreshold(HL_G,  dwtThreshold);
 applyAdaptiveThreshold(HH_G, dwtThreshold);
 applyAdaptiveThreshold(LH_B,  dwtThreshold);
 applyAdaptiveThreshold(HL_B,  dwtThreshold);
 applyAdaptiveThreshold(HH_B,  dwtThreshold);

   // Perform Vector Quantization on all color channels
   vector<vector<float>> quantized_LL_R = vectorQuantization(LL_R, vqClusters);
   vector<vector<float>> quantized_LH_R = vectorQuantization(LH_R, vqClusters);
   vector<vector<float>> quantized_HL_R = vectorQuantization(HL_R, vqClusters);
   vector<vector<float>> quantized_HH_R = vectorQuantization(HH_R, vqClusters);
   vector<vector<float>> quantized_LL_G = vectorQuantization(LL_G, vqClusters);
   vector<vector<float>> quantized_LH_G = vectorQuantization(LH_G, vqClusters);
   vector<vector<float>> quantized_HL_G = vectorQuantization(HL_G, vqClusters);
   vector<vector<float>> quantized_HH_G = vectorQuantization(HH_G, vqClusters);
   vector<vector<float>> quantized_LL_B = vectorQuantization(LL_B, vqClusters);
   vector<vector<float>> quantized_LH_B = vectorQuantization(LH_B, vqClusters);
   vector<vector<float>> quantized_HL_B = vectorQuantization(HL_B, vqClusters);
   vector<vector<float>> quantized_HH_B = vectorQuantization(HH_B, vqClusters);

   // Reconstruct the image using inverse DWT on all color channels
   vector<vector<float>> rec_image_R = inverseDWT(quantized_LL_R, quantized_LH_R, quantized_HL_R,
quantized_HH_R, cols, rows);
   vector<vector<float>> rec_image_G = inverseDWT(quantized_LL_G, quantized_LH_G, quantized_HL_G,
quantized_HH_G, cols, rows);
   vector<vector<float>> rec_image_B = inverseDWT(quantized_LL_B, quantized_LH_B, quantized_HL_B,
quantized_HH_B, cols, rows);
   // Combine the reconstructed color channels
   vector<vector<vector<float>>> rec_image = {rec_image_R, rec_image_G, rec_image_B};

   // Convert the reconstructed image back to QImage
   QImage recQImage = vectorToQImage(rec_image);

   // Save the reconstructed image with the selected format and quality
   QString compressedFilePath = "compressed_image." + compressionFormat.toLower();
```

```
    recQImage.save(compressedFilePath,  compressionFormat.toStdString().c_str(),  jpegQuality);

    // Display compressed image file size
    QFileInfo compressedFileInfo(compressedFilePath);
    qint64 compressedFileSize = compressedFileInfo.size();
    recCaption->setText(QString("Compressed Image\nFile Size: %1 bytes").arg(compressedFileSize));
    compressedFileSizeLabel->setText(QString("Compressed File Size: %1 bytes").arg(compressedFileSize));

    // Calculate and display PSNR and SSIM
    double psnr = calculatePSNR(image[0], rec_image[0]);
    double ssim = calculateSSIM(image[0], rec_image[0]);
    psnrLabel->setText(QString("PSNR: %1 dB").arg(psnr));
    ssimLabel->setText(QString("SSIM: %1").arg(ssim));

    // Display the reconstructed image
    recLabel->setPixmap(QPixmap::fromImage(recQImage));
}

// Download compressed image
void MainWindow::downloadCompressedImage()  {
    QString compressedFilePath = "compressed_image." + compressionFormat.toLower();
    QDesktopServices::openUrl(QUrl::fromLocalFile(compressedFilePath));
}
// Go to welcome page
void MainWindow::goToWelcomePage() {
    stackedWidget->setCurrentWidget(welcomePage);
}
// Update compression format
void MainWindow::updateCompressionFormat(int index) {
    compressionFormat = formatComboBox->itemText(index);
    formatLabel->setText(QString("Compression  Format:  %1").arg(compressionFormat));
}
// Perform Discrete Wavelet Transform
void MainWindow::dwt(const vector<vector<float>>& src, vector<vector<float>>& LL, vector<vector<float>>&
LH, vector<vector<float>>& HL, vector<vector<float>>& HH) {
    int rows = src.size() / 2;
    int cols = src[0].size() / 2;
    LL.resize(rows,  vector<float>(cols));
    LH.resize(rows,  vector<float>(cols));
    HL.resize(rows,  vector<float>(cols));
    HH.resize(rows, vector<float>(cols));
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            float a = src[2 * i][2 * j];
            float b = src[2 * i][2 * j + 1];
            float c = src[2 * i + 1][2 * j];
            float d = src[2 * i + 1][2 * j + 1];
            LL[i][j] = (a + b + c + d) / 4;
            LH[i][j] = (a - b + c - d) / 4;
            HL[i][j] = (a + b - c - d) / 4;
            HH[i][j] = (a - b - c + d) / 4;
        }}}
```

```
void initializeCentersKMeansPP(vector<float>& centers, const vector<float>& samples, int n_clusters) {
    centers[0] = samples[rand() % samples.size()];  // Randomly choose the first center
    for (int i = 1; i < n_clusters; ++i) {
        vector<float> distances(samples.size());
        for (size_t j = 0; j < samples.size(); ++j) {
            float minDist = abs(samples[j] - centers[0]);
            for (int k = 1; k < i; ++k) {
                float dist = abs(samples[j] - centers[k]);
                if (dist < minDist) {
                    minDist = dist;
                }}
            distances[j] = minDist;}
        float maxDist = *max_element(distances.begin(), distances.end());
        for (size_t j = 0; j < samples.size(); ++j) {
            distances[j] /= maxDist;  // Normalize}
        centers[i] = samples[rand() % samples.size()];
    }}
vector<vector<float>> MainWindow::vectorQuantization(const vector<vector<float>>& image, int n_clusters) {
    int rows = image.size();
    int cols = image[0].size();
    vector<float> samples(rows * cols);
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            samples[i * cols + j] = image[i][j];
    vector<float> centers(n_clusters);
    initializeCentersKMeansPP(centers,  samples,  n_clusters);

    vector<int> labels(samples.size());
    for (int iter = 0; iter < 10; ++iter) {
        for (vector<float>::size_type i = 0; i < samples.size(); ++i) {
            float min_dist = abs(samples[i] - centers[0]);
            labels[i] = 0;
            for (int j = 1; j < n_clusters; ++j) {
                float dist = abs(samples[i] - centers[j]);
                if (dist < min_dist) {
                    min_dist = dist;
                    labels[i] = j;}}        }

        vector<float> new_centers(n_clusters, 0);
        vector<int> counts(n_clusters, 0);
        for (vector<float>::size_type i = 0; i < samples.size(); ++i) {
            new_centers[labels[i]] += samples[i];
            counts[labels[i]]++;        }
for (int j = 0; j < n_clusters; ++j) {
            if (counts[j] > 0)
                centers[j] = new_centers[j] / counts[j];}}

    vector<vector<float>> new_image(rows, vector<float>(cols));
    for (int i = 0; i < rows; ++i)
        for (int j = 0; j < cols; ++j)
            new_image[i][j] = centers[labels[i * cols + j]];
    return new_image;}
```

```
// Calculate PSNR
double MainWindow::calculatePSNR(const vector<vector<float>>& original, const vector<vector<float>>&
reconstructed) {
    if (original.empty() || reconstructed.empty()) {
        cerr << "Error: One or both images are empty." << endl;
        return -1.0;    }
    int rows = original.size();
    int cols = original[0].size();
    if (rows != reconstructed.size() || cols != reconstructed[0].size()) {
        cerr << "Error: Image dimensions do not match." << endl;
        return -1.0;    }
    double mse = 0.0;
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            double diff = original[i][j] - reconstructed[i][j];
            mse += diff * diff;     } }
    mse /= (rows * cols);
    if (mse == 0) {
        return numeric_limits<double>::infinity();    }
    double psnr = 10 * log10((255 * 255) / mse);
    return psnr;}
// Calculate SSIM
double MainWindow::calculateSSIM(const vector<vector<float>>& original, const vector<vector<float>>&
reconstructed) {
    if (original.empty() || reconstructed.empty()) {
        cerr << "Error: One or both images are empty." << endl;
        return -1.0;}
    int rows = original.size();
    int cols = original[0].size();
    if (rows != reconstructed.size() || cols != reconstructed[0].size()) {
        cerr << "Error: Image dimensions do not match." << endl;
        return -1.0;   }
    double C1 = 6.5025, C2 = 58.5225;
    double ssim = 0.0;
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            double mu_x = original[i][j];
            double mu_y = reconstructed[i][j];
            double sigma_x = 0.0;
            double sigma_y = 0.0;
            double sigma_xy = 0.0;
            for (int k = -1; k <= 1; ++k) {
                for (int l = -1; l <= 1; ++l) {
                    int x = min(max(i + k, 0), rows - 1);
                    int y = min(max(j + l, 0), cols - 1);
                    sigma_x += (original[x][y] - mu_x) * (original[x][y] - mu_x);
                    sigma_y += (reconstructed[x][y] - mu_y) * (reconstructed[x][y] - mu_y);
                    sigma_xy += (original[x][y] - mu_x) * (reconstructed[x][y] - mu_y);         }    }
            sigma_x /= 9.0;         sigma_y /= 9.0;          sigma_xy /= 9.0;
            double numerator = (2 * mu_x * mu_y + C1) * (2 * sigma_xy + C2);
            double denominator = (mu_x * mu_x + mu_y * mu_y + C1) * (sigma_x + sigma_y + C2);
            ssim += numerator / denominator;}}return ssim / (rows * cols);}
```

```
// Inverse DWT
vector<vector<float>> MainWindow::inverseDWT(const vector<vector<float>>& LL, const
vector<vector<float>>& LH, const vector<vector<float>>& HL, const vector<vector<float>>& HH, int width, int
height) {
    int rows = LL.size();
    int cols = LL[0].size();
    vector<vector<float>> rec_image(2 * rows, vector<float>(2 * cols));

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            float a = LL[i][j];
            float b = LH[i][j];
            float c = HL[i][j];
            float d = HH[i][j];

            // Reconstruct the original image from the DWT coefficients
            rec_image[2 * i][2 * j] = a + b + c + d;
            rec_image[2 * i][2 * j + 1] = a - b + c - d;
            rec_image[2 * i + 1][2 * j] = a + b - c - d;
            rec_image[2 * i + 1][2 * j + 1] = a - b - c + d;
        }
    }

    return rec_image;
}

// Simulate transmission
void MainWindow::simulateTransmission(const QString& filePath, QString& destinationPath, double&
transmissionTime) {
    auto start = std::chrono::high_resolution_clock::now();
    if (!QFile::copy(filePath, destinationPath)) {
        std::cerr << "Error copying file: " << filePath.toStdString() << " to " << destinationPath.toStdString() <<
std::endl;
        transmissionTime = -1.0;
        return;
    }
    auto end = std::chrono::high_resolution_clock::now();
    std::chrono::duration<double> elapsed = end - start;
    transmissionTime = elapsed.count();
}
// Perform transmission experiment
void MainWindow::performTransmissionExperiment()  {
    if (!originalLabel->pixmap() || originalLabel->pixmap().isNull()) {
        QMessageBox::warning(this, "Error", "Please load an image before performing the transmission experiment.");
        return;
    }

    QString originalFilePath = this->originalFilePath;
    if (originalFilePath.isEmpty()) {
        QMessageBox::warning(this, "Error", "No image loaded.");
        return;
    }
    QFileInfo fileInfo(originalFilePath);
    QString destinationFilePath = destinationDirectory + "/" + fileInfo.fileName();

    if (!QFile::copy(originalFilePath, destinationFilePath)) {
        QMessageBox::warning(this, "Error", "Failed to copy the image to the selected directory.");
        return;
    }

    double transmissionTimeOriginal = 0.0, transmissionTimeCompressed = 0.0;
    QString tempDestinationOriginal = destinationDirectory + "/temp_original_image.jpg";
    QString compressedFilePath = "compressed_image." + compressionFormat.toLower();
    QString tempDestinationCompressed = destinationDirectory + "/temp_compressed_image." +
compressionFormat.toLower();

    simulateTransmission(destinationFilePath, tempDestinationOriginal, transmissionTimeOriginal);
    simulateTransmission(compressedFilePath, tempDestinationCompressed, transmissionTimeCompressed);

    QMessageBox::information(this, "Transmission Results",
                             QString("Original Image Transmission Time: %1 seconds\n"
                                     "Compressed Image Transmission Time: %2 seconds\n"
                                     "Time Difference: %3 seconds")
                                 .arg(transmissionTimeOriginal, 0, 'f', 6)
                                 .arg(transmissionTimeCompressed, 0, 'f', 6)
                                 .arg(transmissionTimeOriginal - transmissionTimeCompressed, 0, 'f', 6));
}

// Perform transmission experiment with two images
void MainWindow::performTransmissionExperimentWithTwoImages() {
    QString image1Path = QFileDialog::getOpenFileName(this, "Select First Image", "", "Image Files (*.png *.jpg
*.bmp)");
    QString image2Path = QFileDialog::getOpenFileName(this, "Select Second Image", "", "Image Files (*.png *.jpg
*.bmp)");

    if (image1Path.isEmpty() || image2Path.isEmpty()) {
        QMessageBox::warning(this, "Error", "Please select two images for the experiment.");
        return;
    }

    QString destinationDirectory = QFileDialog::getExistingDirectory(this, "Select Destination Directory");
    if (destinationDirectory.isEmpty()) {
        QMessageBox::warning(this, "Error", "Please select a destination directory.");
        return;
    }

    double transmissionTimeImage1 = 0.0, transmissionTimeImage2 = 0.0;
    QString tempDestinationImage1 = destinationDirectory + "/temp_image1.jpg";
    QString tempDestinationImage2 = destinationDirectory + "/temp_image2.jpg";

    simulateTransmission(image1Path, tempDestinationImage1, transmissionTimeImage1);
    simulateTransmission(image2Path, tempDestinationImage2, transmissionTimeImage2);

    QMessageBox::information(this, "Transmission Results",
                    QString("First Image Transmission Time: %1 seconds\n"
                             "Second Image Transmission Time: %2 seconds\n"
                             "Time Difference: %3 seconds")
                          .arg(transmissionTimeImage1, 0, 'f', 6)
                          .arg(transmissionTimeImage2, 0, 'f', 6)
                          .arg(abs(transmissionTimeImage1 - transmissionTimeImage2), 0, 'f', 6));
}

// Calculate PSNR and SSIM
void MainWindow::calculatePSNRAndSSIM() {
   QString image1Path = QFileDialog::getOpenFileName(this, "Select First Image", "", "Image Files (*.png *.jpg
*.bmp)");
   QString image2Path = QFileDialog::getOpenFileName(this, "Select Second Image", "", "Image Files (*.png *.jpg
*.bmp)");

   if (image1Path.isEmpty() || image2Path.isEmpty()) {
      QMessageBox::warning(this, "Error", "Please select two images for the calculation.");
      return;
   }

   QImage image1(image1Path);
   QImage image2(image2Path);

   if (image1.isNull() || image2.isNull()) {
      QMessageBox::warning(this, "Error", "Failed to load one or both images.");
      return;
   }

   if (image1.size() != image2.size()) {
      QMessageBox::warning(this, "Error", "The selected images must have the same dimensions.");
      return;
   }

   vector<vector<vector<float>>> image1Data = loadImageData(image1Path);
   vector<vector<vector<float>>> image2Data = loadImageData(image2Path);

   double psnr = calculatePSNR(image1Data[0], image2Data[0]);
   double ssim = calculateSSIM(image1Data[0], image2Data[0]);

   QMessageBox::information(this, "PSNR and SSIM Results",
                       QString("PSNR: %1 dB\n"
                             "SSIM: %2")
                          .arg(psnr, 0, 'f', 6)
                          .arg(ssim, 0, 'f', 6));
}

// Load image data
vector<vector<vector<float>>> MainWindow::loadImageData(const QString& filePath) {
   QImage qimage(filePath);
   if (qimage.isNull()) {
      QMessageBox::warning(this, "Error", "Failed to load image: " + filePath);
      return {};}
```

```
    int rows = qimage.height();
    int cols = qimage.width();
    vector<vector<vector<float>>> image(3, vector<vector<float>>(rows, vector<float>(cols)));

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            QColor color = qimage.pixelColor(j, i);
            image[0][i][j] = color.red();
            image[1][i][j] = color.green();
            image[2][i][j] = color.blue();
        }
    }

    return image;
}

// Convert vector to QImage
QImage MainWindow::vectorToQImage(const vector<vector<vector<float>>>& image) {
    int rows = image[0].size();
    int cols = image[0][0].size();
    QImage qimage(cols, rows, QImage::Format_RGB32);

    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            int r = static_cast<int>(image[0][i][j]);
            int g = static_cast<int>(image[1][i][j]);
            int b = static_cast<int>(image[2][i][j]);
            qimage.setPixel(j, i, qRgb(r, g, b));
        }
    }
    return qimage;
}
```

Main.cpp

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QLabel>
#include <QPushButton>
#include <QStackedWidget>
#include <QSlider>
#include <QComboBox>
#include <QFileDialog>
#include <QMessageBox>
#include <QImage>
#include <QPixmap>
#include <vector>

using namespace std;

class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void showMainInterface();
    void loadImage();
    void downloadCompressedImage();
    void goToWelcomePage();
    void performTransmissionExperiment();
    void performTransmissionExperimentWithTwoImages();
    void calculatePSNRAndSSIM();
    void updateCompressionFormat(int index);

private:
    QStackedWidget *stackedWidget;
    QWidget *welcomePage;
    QWidget *mainPage;
    QLabel *originalLabel;
    QLabel *recLabel;
    QLabel *originalCaption;
    QLabel *recCaption;
    QPushButton *loadButton;
    QPushButton *downloadButton;
    QLabel *sizeLabel;
    QLabel *fileSizeLabel;
    QLabel *compressedFileSizeLabel;
    QLabel *compressedFileSizeLabel;
    QLabel *psnrLabel;
    QLabel *ssimLabel;
    QPushButton *backButton;
    QSlider *dwtThresholdSlider;
    QSlider *vqClustersSlider;
    QSlider *jpegQualitySlider;
    QLabel *dwtThresholdLabel;
    QLabel *vqClustersLabel;
    QLabel *jpegQualityLabel;
    QComboBox *formatComboBox;
    QLabel *formatLabel;
    QString originalFilePath;

    // Compression parameters
    float dwtThreshold;
    int vqClusters;
    int jpegQuality;
    QString compressionFormat;

    // Helper functions
    void processImage(const QImage& qimage);
    vector<vector<vector<float>>> loadImageData(const QString& filePath);
    QImage vectorToQImage(const vector<vector<vector<float>>>& image);
    void dwt(const vector<vector<float>>& src, vector<vector<float>>& LL, vector<vector<float>>& LH,
vector<vector<float>>& HL, vector<vector<float>>& HH);
    vector<vector<float>> vectorQuantization(const vector<vector<float>>& image, int n_clusters) ;
    double calculatePSNR(const vector<vector<float>>& original, const vector<vector<float>>&
reconstructed);
    double calculateSSIM(const vector<vector<float>>& original, const vector<vector<float>>&
reconstructed);
    vector<vector<float>> inverseDWT(const vector<vector<float>>& LL, const vector<vector<float>>&
LH, const vector<vector<float>>& HL, const vector<vector<float>>& HH, int width, int height);
    void simulateTransmission(const QString& filePath, QString& destinationPath, double&
transmissionTime);
};

#endif // MAINWINDOW_H
```