Laporan Tugas Kecil 2 IF2211 - Strategi Algoritma

Semester II Tahun Akademik 2024/2025

Kompresi Gambar Menggunakan Metode Quadtree dengan Algoritma Divide and Conquer

Disusun oleh:

Fajar Kurniawan	13523027
Nayaka Ghana Subrata	13523090

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

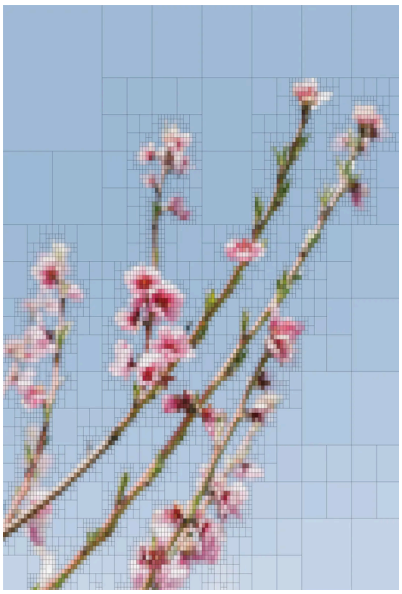
DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I: DESKRIPSI MASALAH

1.1. Pengertian Quadtree



Gambar 1. Quadtree dalam kompresi gambar
(Sumber: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00)

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

1.2. Tugas Kecil Quadtree Compression

Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. **Inisialisasi dan Persiapan Data**

   Memasukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:
   a. **Metode perhitungan variansi**: memilih metode perhitungan variansi berdasarkan opsi yang tersedia pada Tabel 1.
   b. **Threshold variansi**: nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
   c. **Minimum block size**: ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. **Perhitungan Error**

   Untuk setiap blok gambar yang sedang diproses, menghitung nilai variansi menggunakan metode yang dipilih sesuai Tabel 1.

3. **Pembagian Blok**

   Membandingkan nilai variansi blok dengan threshold:
   a. Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
   b. Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. **Normalisasi Warna**

   Untuk blok yang tidak lagi dibagi, dilakukan normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

5. **Rekursi dan Penghentian**

   Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:
   a. Error blok berada di bawah threshold.
   b. Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

6. **Penyimpanan dan Output**

   Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.

1.3 Parameter Compression

1. **Error Measurement Methods** (Metode Pengukuran Error)

   Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika error dalam blok melebihi ambas batas (threshold), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Tabel 1. Metode Pengukuran Error

Metode	Formula
Variance	$\sigma_c^2 = \frac{1}{N}\sum_{i=1}^{N}(P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok $P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c μ_c = Nilai rata-rata tiap piksel dalam satu blok N = Banyaknya piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N}\sum_{i=1}^{N}\lvert P_{i,c} - \mu_c \rvert$
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Variansi tiap kanal warna c (R, G, B) dalam satu blok $P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c

	μ_c = Nilai rata-rata tiap piksel dalam satu blok N = Banyaknya piksel dalam satu blok
Max Pixel Difference	$D_c = max(P_{i,c}) - min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok μ_c = Nilai rata-rata tiap piksel dalam satu blok
Entropy	$H_c = -\sum_{i=1}^{N} P_c(i) log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok $P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)
Structural Similarity Index (SSIM)	$SSIM_c(x,y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

2. **Threshold** (Ambang Batas)

   Threshold adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.

3. **Minimum Block Size** (Ukuran Blok Minimum)

Minimum block size (luas piksel) adalah ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Jika ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah ukuran minimum yang telah dikonfigurasi, maka blok tersebut tidak akan dibagi lebih lanjut, meskipun error masih di atas threshold.

4. **Compression Percentage** (Persentase Kompresi)

   Presentasi kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

$$Persentase\ Kompresi\ =\ (1\ -\ \frac{Ukuran\ Gambar\ Terkompresi}{Ukuran\ Gambar\ Asli})\ \times\ 100\%$$

BAB II: TEORI, PENYELESAIAN, DAN ALGORITMA

2.1. Divide and Conquer

Algoritma *Divide and Conquer* adalah teknik pemecahan masalah yang digunakan untuk memecahkan masalah dengan membagi masalah utama menjadi submasalah, menyelesaikannya satu per satu, dan kemudian menggabungkannya untuk menemukan solusi dari masalah utama. *Divide and Conquer* terutama berguna ketika kita membagi masalah menjadi submasalah yang berdiri sendiri. Jika kita memiliki submasalah yang saling tumpang tindih, maka kita menggunakan *Dynamic Programming*.

Berikut ini adalah beberapa algoritma standar yang mengikuti algoritma *Divide and Conquer*: *Binary Search, Quicksort, Merge Sort, Closest Pair of Points, Strassen's Algorithm, Cooley–Tukey Fast Fourier Transform (FFT) algorithm, Karatsuba algorithm for fast multiplication,* dan sebagainya.

Berikut merupakan komponen dari *Divide and Conquer:*

1. **Divide**: membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama).
2. **Conquer**: menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
3. **Combine**: menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

2.2. Quadtree Compression

Pada deskripsi masalah telah disajikan cara untuk membuat *quadtree*. Langkah-langkah yang disebutkan tersebut sesuai dengan konsep algoritma *Divide and Conquer* yaitu dengan membagi gambar awal menjadi beberapa bagian (*divide*). Kemudian sub-bagian gambar dilakukan *conquer* dengan diperiksa apakah warnanya seragam, jika seragam, proses berhenti, jika tidak, pembagian dilanjutkan. Selanjutnya dilakukan *combine* dengan menggabungkan semua sub-bagian gambar tadi menjadi gambar utuh yang terkompresi.

2.3. Algoritma Divide and Conquer

Program ini menggunakan Quadtree sebagai struktur data utama untuk melakukan kompresi. Program pertama-tama akan meminta input file berbentuk gambar dengan ekstensi .jpg, .jpeg, dan .png, yang setelah itu program akan melakukan proses terhadap gambar dengan membaca nilai RGB dan akan memprosesnya dengan menggunakan algoritma *divide and conquer*.

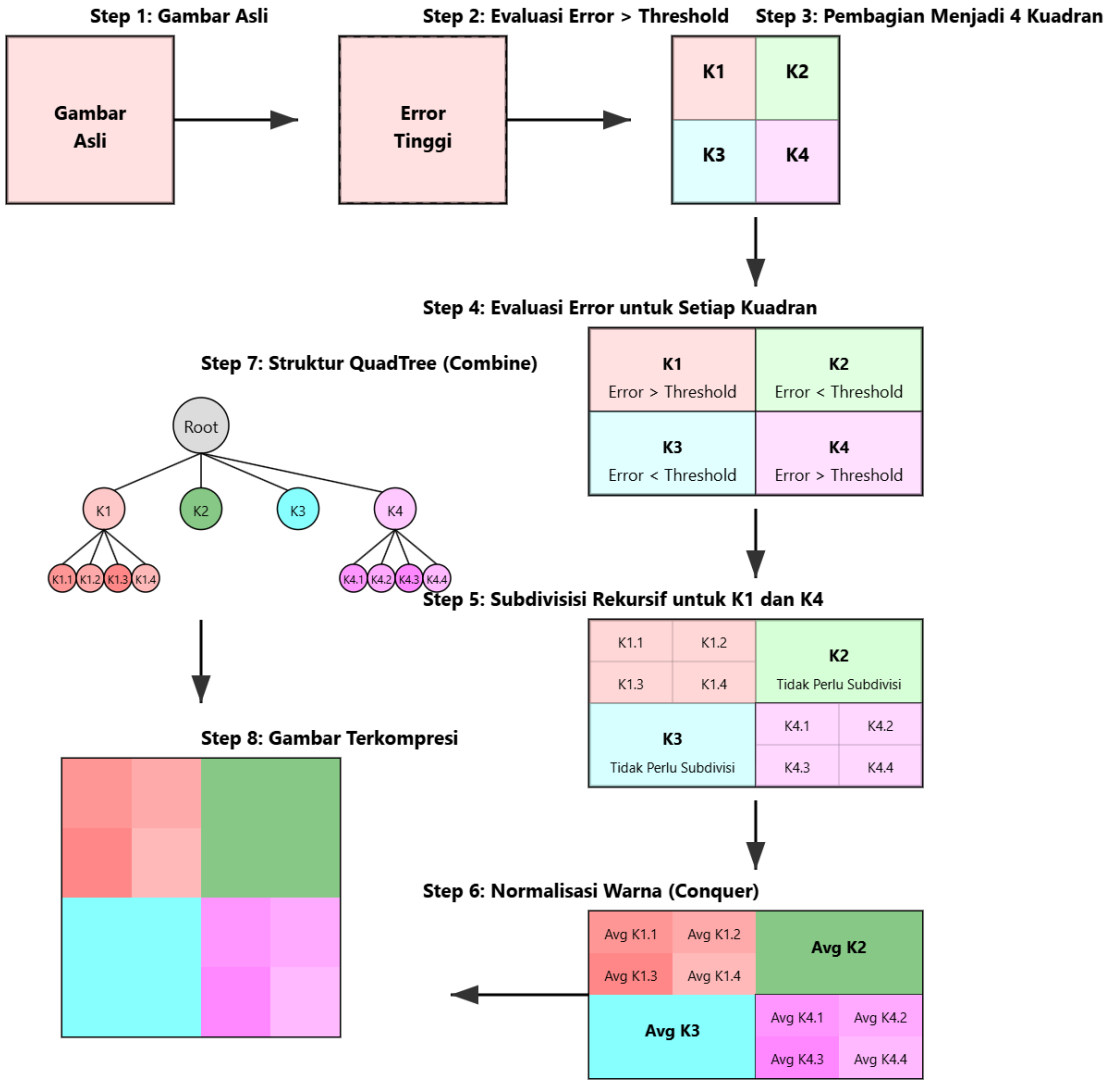
Setelah melakukan *processing* pada gambar, selanjutnya program akan masuk ke tahap kompresi dengan menggunakan algoritma *divide and conquer*. Algoritmanya pun cukup sederhana, yakni menggunakan pendekatan rekursif, dengan langkah sebagai berikut:

1. **(Divide)** Basis: Anggap gambar sebagai sebuah region yang besar, Coba bagi gambar menjadi 4 bagian, dengan pembagiannya adalah membagi gambar menjadi beberapa blok (subregion): blok kiri atas, kanan atas, kiri bawah, dan kanan bawah; dengan setiap blok memiliki ukuran yang relatif sama besar (sehingga memungkinkan bentuk per blok adalah persegi panjang). Setiap blok merepresentasikan simpul (*node*) dari Quadtree yang akan dibangun.
2. **(Divide)** Rekurens: Untuk blok (subregion) yang masih bisa dan perlu untuk dibagi, bagi tiap blok menjadi 4 bagian lagi dengan bentuk metode pembagian yang sama secara rekursif. Namun, ada aturan dalam membaginya, yakni sebagai berikut:
    a. Hitung nilai *error* (berdasarkan metode yang dipilih), nilai ini harus ada di atas threshold, ukuran blok harus lebih besar dari ukuran minimum blok (ukuran minimum didefinisikan sendiri), dan ukuran calon anak (*candidate children*) juga harus lebih besar dari ukuran minimum blok.

    Dari aturan tersebut, secara rekursif, bagi blok menjadi lebih kecil sampai salah satu aturan dari poin 2a berhasil kita langgar. Jika kita melanggar salah satu aturan ini, hentikan rekursi.

3. **(Conquer)** Dari proses rekursi, untuk blok yang tidak dapat dibagi lagi, normalisasikan warna untuk setiap blok dengan nilai rata-rata RGB pada blok yang bersangkutan.
4. **(Combine)** Gabungkan semua simpul (*node*) untuk membentuk struktur Quadtree utuh, dengan *root node* sebagai representasi keseluruhan gambar, dan *leaf node* yang merepresentasikan blok dengan warna yang ternormalisasi.

2.4. Visualisasi Sederhana



Gambar 2. Visualisasi sederhana dari algoritma *divide and conquer* yang digunakan pada program

(Sumber: arsip penulis)

2.5. Pseudocode

```
function buildQuadTree(input pixels : LarikPixel, input x, y,
width, height, depth : integer) : QuadTreeNode
{ Membangun QuadTree secara rekursif dengan metode Divide and
Conquer
  Masukan: pixels[y][x], posisi x, y, width, height, dan depth
  Luaran: node QuadTree untuk region tersebut }

Deklarasi
  node : QuadTreeNode
  error : real
  shouldDivide : boolean
  halfWidth, remainderWidth, halfHeight, remainderHeight,
subBlockArea : integer
  topLeft, topRight, bottomLeft, bottomRight : QuadTreeNode

Algoritma:
  { Buat node untuk region ini }
  node ← new QuadTreeNode(x, y, width, height)

  { Set warna rata-rata untuk node ini }
  node.setColor(calculateAverageColor(x, y, width, height))

  { Hitung error untuk region ini }
  error ← calculateError(pixels, x, y, width, height)
  shouldDivide ← (error > threshold)

  { Menghitung dimensi subdivision }
  halfWidth ← width / 2
  remainderWidth ← width - halfWidth
  halfHeight ← height / 2
  remainderHeight ← height - halfHeight
  subBlockArea ← halfWidth * halfHeight

  { DIVIDE: Periksa apakah perlu membagi (subdivide) region ini
}
  if (shouldDivide) and (subBlockArea >= minBlockSize) then
    { CONQUER: Proses rekursif untuk empat kuadran }
    topLeft ← buildQuadTree(pixels, x, y, halfWidth,
halfHeight, depth + 1)
    topRight ← buildQuadTree(pixels, x + halfWidth, y,
remainderWidth, halfHeight, depth + 1)
    bottomLeft ← buildQuadTree(pixels, x, y + halfHeight,
halfWidth, remainderHeight, depth + 1)
```

```
    bottomRight ← buildQuadTree(pixels, x + halfWidth, y +
halfHeight, remainderWidth, remainderHeight, depth + 1)

    { COMBINE: Tambahkan child nodes yang valid }
    if (topLeft) then
      node.addChild(topLeft)
    endif

    if (topRight) then
      node.addChild(topRight)
    endif

    if (bottomLeft) then
      node.addChild(bottomLeft)
    endif

    if (bottomRight) then
      node.addChild(bottomRight)
    endif
  endif

  return node
endfunction
```

BAB III: STRUKTUR REPOSITORY DAN SOURCE CODE

3.1. Struktur Repository

```
📂 Tucil2_13523027_13523090/
├── 📂 bin/
│   ├── .gitignore
│   ├── quadtree_compression
│   └── quadtree_compression.exe
│
├── 📂 doc/
│   ├── .gitkeep
│   └── Tucil2_13523027_13523090.pdf
│
├── 📂 src/
│   ├── 📂 comps/
│   │   ├── BasicInputManager.cpp
│   │   ├── CompressionAnalyzer.cpp
│   │   ├── ErrorCalculation.cpp
│   │   ├── GifGenerator.cpp
│   │   ├── ImageProcessor.cpp
│   │   ├── InputManager.cpp
│   │   ├── QuadTree.cpp
│   │   └── main.cpp
│   │
│   ├── 📂 header/
│   │   ├── BasicInputManager.hpp
│   │   ├── CompressionAnalyzer.hpp
│   │   ├── CompressionParams.hpp
│   │   ├── ErrorCalculator.hpp
│   │   ├── GifGenerator.hpp
│   │   ├── ImageProcessor.hpp
│   │   ├── InputManager.hpp
│   │   ├── Pixel.hpp
│   │   └── QuadTree.hpp
│   │
│   └── .gitkeep
│
├── 📂 test/
```

```
│   ├── 📂 input/
│   │   ├── W.jpg
│   │   ├── bocchi.jpg
│   │   ├── coffee.jpg
│   │   ├── flower.jpg
│   │   ├── isagi.jpg
│   │   ├── kaneki.jpeg
│   │   ├── kosong.png
│   │   ├── lappland.jpg
│   │   ├── misteri.jpg
│   │   ├── sample1.jpg
│   │   ├── texas.jpg
│   │   └── tragedy.png
│   │
│   └── 📂 output/
│   │   ├── W256.jpg
│   │   ├── W256gif.gif
│   │   ├── W64.jpg
│   │   ├── W64gif.gif
│   │   ├── Wlow.jpg
│   │   ├── Wlowgif.gif
│   │   ├── bocchlow.jpg
│   │   ├── bocchlowgif.gif
│   │   ├── flower.gif
│   │   ├── lappgif.gif
│   │   ├── lapplow.jpg
│   │   ├── mygoat.jpg
│   │   ├── mygoat2.jpg
│   │   ├── rcoffee.jpg
│   │   ├── rkaneki.jpeg
│   │   ├── rtragedy.png
│   │   ├── sample1a.jpg
│   │   └── texlow2.jpg
├── CMakeLists.txt
├── README.md
├── build.bat
├── build.sh
├── clean.bat
├── clean.sh
├── run.bat
└── run.sh
```

Pada skema di atas, sudah diberikan visualisasi dari struktur repositori dari program ini. Berikut merupakan penjelasan struktur dari isi *repository* tugas kecil *divide and conquer*.

1. Folder **src** yang berisi source code program.
2. Folder **bin** yang berisi executable file.
3. Folder **test** yang berisi solusi jawaban dari data uji yang digunakan dalam laporan.
4. Folder **doc** yang berisi laporan tugas kecil dalam bentuk PDF.
5. **README** yang berisi:
    a. Penjelasan singkat program yang dibuat.
    b. Requirement program dan instalasi tertentu bila ada.
    c. Cara mengkompilasi program bila perlu dikompilasi (pastikan dengan langkah yang jelas dan benar).
    d. Cara menjalankan dan menggunakan program (pastikan dengan langkah yang jelas dan benar).
    e. Author / identitas pembuat.
6. **CMakeLists.txt** yang berisi template builder dari program.
7. **build.bat** (Windows) dan **build.sh** (Linux) yang berisi instruksi untuk *construct* program dan *executable file*.
8. **run.bat** (Windows) dan **run.sh** (Linux) yang berisi instruksi untuk menjalankan program, yang terdiri dari dua argumen:
    a. Basic, argumen untuk menjalankan program dengan tampilan *default*.
    b. Page, argumen untuk menjalankan program dengan *paging mode*.
9. **clean.bat** (Windows) dan **clean.sh** (Linux) yang berisi instruksi untuk melakukan *clean* pada folder bin (membersihkan *executable file* sesuai dengan *environment* dari perangkat yang digunakan).

3.2. Source Code

Berikut adalah source code dari program ini yang berupa kode asli dari program, untuk lebih lengkapnya, source code dapat diakses pada tautan ini. Untuk cara menjalankan program, silakan untuk me-*refer* ke file "README.md" yang ada pada repository, atau bisa diakses pada tautan ini.

3.2.1 BasicInputManager.hpp

```
#ifndef _BASIC_INPUT__HPP
#define _BASIC_INPUT__HPP

// include lib files
```

```
#include <cctype>
#include <string>
#include <iostream>
#include <algorithm>
#include <filesystem>

// include header file
#include "CompressionParams.hpp"


// namespace
using namespace std;

class BasicInputManager {
    public:
        BasicInputManager(); // Ctor
        ~BasicInputManager(); // Dtor

        // Get compression parameters
        CompressionParams getCompressionParams();

    private:
        // ANSI color definitions
        const std::string ANSI_RESET    = "\033[0m";
        const std::string ANSI_BOLD     = "\033[1m";
        const std::string ANSI_RED      = "\033[31m";
        const std::string ANSI_GREEN    = "\033[32m";
        const std::string ANSI_YELLOW   = "\033[33m";
        const std::string ANSI_BLUE     = "\033[34m";
        const std::string ANSI_MAGENTA  = "\033[35m";
        const std::string ANSI_CYAN     = "\033[36m";
        const std::string ANSI_WHITE    = "\033[37m";

        // Helper methods for validation
        bool isImageFile(const std::string& filePath);
        bool fileExists(const std::string& filePath);
};

#endif
```

3.2.2 BasicInputManager.cpp

```
// include header file
#include "BasicInputManager.hpp"


BasicInputManager::BasicInputManager() {
    // cons
}

BasicInputManager::~BasicInputManager() {
```

```
    // dtor
}

// image checker
bool BasicInputManager::isImageFile(const string& filePath) {
    size_t pos = filePath.find_last_of(".");
    if (pos == string::npos) return false;

    string ext = filePath.substr(pos);
    for (auto& c : ext) c = tolower(c);

    return (ext == ".jpg" || ext == ".jpeg" || ext == ".png");
}

// check apakah file ada atau enggak
bool BasicInputManager::fileExists(const string& filePath) {
    return filesystem::exists(filePath);
}

CompressionParams BasicInputManager::getCompressionParams() {
    CompressionParams params;
    string input;

    cin.clear();

    // 1. Input image path
    cout << "1. Enter absolute path to image for compression: ";
    getline(cin, input);
    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Input path cannot be empty.
Please enter a valid file path." << ANSI_RESET << endl;
            cout << "Enter absolute path to image for compression: ";
            getline(cin, input);
            continue;
        }

        if (!fileExists(input)) {
            cout << ANSI_RED << "Error: File does not exist. Please
check the path and try again." << ANSI_RESET << endl;
            cout << "Enter absolute path to image for compression: ";
            getline(cin, input);
            continue;
        }

        if (!isImageFile(input)) {
            cout << ANSI_RED << "Error: File is not a valid image file
(.jpg, .jpeg, .png)." << ANSI_RESET << endl;
            cout << "Enter absolute path to image for compression: ";
            getline(cin, input);
            continue;
        }

        break;
    }
    params.inputImagePath = input;
```

```
    cout << endl << endl;

    // 2. Error calculation method
    cout << "2. Select error calculation method (1-5):" << endl;
    cout << "   1. Variance" << endl;
    cout << "   2. Mean Absolute Deviation (MAD)" << endl;
    cout << "   3. Max Pixel Difference" << endl;
    cout << "   4. Entropy" << endl;
    cout << "   5. Structural Similarity Index (SSIM)" << endl;
    cout << "Enter choice (1-5): ";
    getline(cin, input);
    int methodChoice;
    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Method choice cannot be empty.
Please enter a number between 1 and 5." << ANSI_RESET << endl;
            cout << "Enter choice (1-5): ";
            getline(cin, input);
            continue;
        }

        try {
            methodChoice = stoi(input);
            if (methodChoice >= 1 && methodChoice <= 5) break;
            cout << ANSI_RED << "Error: Please enter a number between 1
and 5." << ANSI_RESET << endl;
        } catch (const exception&) {
            cout << ANSI_RED << "Error: Please enter a valid number." <<
ANSI_RESET << endl;
        }
        cout << "Enter choice (1-5): ";
        getline(cin, input);
    }
    params.errorMethod = static_cast<ErrorMethod>(methodChoice);
    cout << endl << endl;

    // 3. Threshold
    double min = 0.0, max = 0.0, defaultValue = 0.0;
    string rangeDescription;

    switch (params.errorMethod) {
        case ErrorMethod::VARIANCE:
            min = 0.0; max = 16256.25; defaultValue = 500.0;
            rangeDescription = "For Variance method (valid range:
0.0-16256.25)\n"
                               "* Low threshold (high detail): 50-500\n"
                               "* Medium threshold: 500-2000\n"
                               "* High threshold (less detail):
2000-5000++";
            break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            min = 0.0; max = 127.5; defaultValue = 15.0;
            rangeDescription = "For MAD method (valid range:
0.0-127.5)\n"
                               "* Low threshold (high detail): 5-15\n"
                               "* Medium threshold: 15-30\n"
```

```
                                "* High threshold (less detail): 30-50++";
            break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            min = 0.0; max = 255.0; defaultValue = 30.0;
            rangeDescription = "For Max Pixel Difference (valid range:
0.0-255.0)\n"
                                "* Low threshold (high detail): 10-30\n"
                                "* Medium threshold: 30-60\n"
                                "* High threshold (less detail):
60-100++";
            break;
        case ErrorMethod::ENTROPY:
            min = 0.0; max = 8.0; defaultValue = 1.0;
            rangeDescription = "For Entropy method (valid range:
0.0-8.0)\n"
                                "* Low threshold (high detail): 0.1-1.0\n"
                                "* Medium threshold: 1.0-2.0\n"
                                "* High threshold (less detail):
2.0-4.0++";
            break;
        case ErrorMethod::STRUCTURAL_SIMILARITY:
            min = 0.0; max = 1.0; defaultValue = 0.05;
            rangeDescription = "For SSIM method (valid range:
0.0-1.0)\n"
                                "* Low threshold (high detail):
0.01-0.05\n"
                                "* Medium threshold: 0.05-0.15\n"
                                "* High threshold (less detail):
0.15-0.3++";
            break;
    }

    cout << "3. Enter threshold value for error calculation:" << endl;
    cout << ANSI_BOLD << ANSI_RED << "Valid threshold range: " << min <<
" to " << max << ANSI_RESET << endl;
    cout << rangeDescription << endl;
    cout << "Suggested value: " << defaultValue << endl;
    cout << "Enter threshold: ";
    getline(cin, input);

    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Threshold value cannot be empty.
Please enter a value within the range "
                 << min << " to " << max << "." << ANSI_RESET << endl;
            cout << "Enter threshold: ";
            getline(cin, input);
            continue;
        }

        try {
            double threshold = stod(input);
            if (threshold >= min && threshold <= max) {
                params.threshold = threshold;
                break;
            }
```

```
            cout << ANSI_RED << "Error: Value out of range. Please enter
a value between " << min << " and " << max << "." << ANSI_RESET << endl;
        } catch (const exception&) {
            cout << ANSI_RED << "Error: Please enter a valid number." <<
ANSI_RESET << endl;
        }
        cout << "Enter threshold: ";
        getline(cin, input);
    }
    cout << endl << endl;

    // 4. Minimum block size
    cout << "4. Enter minimum block size in square pixels (area)" <<
endl;
    cout << "   This is the smallest area (width x height) that will be
used during compression." << endl;
    cout << "   Recommended values (only applies if the dimensions are
relatively big):" << endl;
    cout << "   * 4 (2x2) - High detail" << endl;
    cout << "   * 16 (4x4) - Good detail" << endl;
    cout << "   * 64 (8x8) - Medium detail" << endl;
    cout << "   * 256 (16x16++) - Low detail" << endl;
    cout << "Enter minimum block size: ";
    getline(cin, input);

    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Minimum block size cannot be
empty. Please enter a positive integer value." << ANSI_RESET << endl;
            cout << "Enter minimum block size: ";
            getline(cin, input);
            continue;
        }

        try {
            int blockSize = stoi(input);
            if (blockSize >= 1) {
                params.minBlockSize = blockSize;
                break;
            }
            cout << ANSI_RED << "Error: Block size must be at least 1
square pixel." << ANSI_RESET << endl;
        } catch (const exception&) {
            cout << ANSI_RED << "Error: Please enter a valid integer
number." << ANSI_RESET << endl;
        }
        cout << "Enter minimum block size: ";
        getline(cin, input);
    }
    cout << endl << endl;

    // 5. Target compression percentage
    cout << "5. Enter target compression percentage (0.0-1.0)" << endl;
    cout << "   This is a bonus feature that automatically adjusts the
threshold." << endl;
    cout << "   Enter 0 to disable automatic threshold adjustment." <<
```

```
endl;
    cout << "Enter target compression (0.0-1.0): ";
    getline(cin, input);

    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Target compression value cannot
be empty. Please enter a value between 0.0 and 1.0." << ANSI_RESET <<
endl;
            cout << "Enter target compression: ";
            getline(cin, input);
            continue;
        }

        try {
            double percentage = stod(input);
            if (percentage >= 0.0 && percentage <= 1.0) {
                params.targetCompressionPercentage = percentage;
                break;
            }
            cout << ANSI_RED << "Error: Please enter a value between 0.0
and 1.0." << ANSI_RESET << endl;
        } catch (const exception&) {
            cout << ANSI_RED << "Error: Please enter a valid number." <<
ANSI_RESET << endl;
        }
        cout << "Enter target compression: ";
        getline(cin, input);
    }
    cout << endl << endl;

    // 6. Output image path
    cout << "6. Enter absolute path for output image:" << endl;

    filesystem::path inputPath(params.inputImagePath);
    string inputExt = inputPath.extension().string();
    for (auto& c : inputExt) c = tolower(c);

    cout << ANSI_BOLD << ANSI_RED << "NOTE: The output file MUST use the
same extension as the input file ("
         << inputExt << ")" << ANSI_RESET << endl;

    cout << "Enter output path: ";
    getline(cin, input);

    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Output path cannot be empty.
Please specify a valid file path with the "
                 << inputExt << " extension." << ANSI_RESET << endl;
            cout << "Enter output path: ";
            getline(cin, input);
            continue;
        }

        filesystem::path outputPath(input);
```

```
        string outputExt = outputPath.extension().string();

        if (outputExt.empty()) {
            cout << ANSI_RED << "Error: No file extension provided." <<
endl;
            cout << "The output file must have the same extension as the
input file (" << inputExt << ")." << ANSI_RESET << endl;
            cout << "Enter output path: ";
            getline(cin, input);
            continue;
        } else {
            for (auto& c : outputExt) c = tolower(c);

            if (outputExt != inputExt) {
                cout << ANSI_RED << "Error: Output extension must match
input extension (" << inputExt << ")." << ANSI_RESET << endl;
                cout << "Enter output path: ";
                getline(cin, input);
                continue;
            }
        }

        filesystem::path parentPath = outputPath.parent_path();
        if (!parentPath.empty() && !filesystem::exists(parentPath)) {
            cout << ANSI_RED << "Error: Output directory does not
exist." << ANSI_RESET << endl;
            cout << "Enter output path: ";
            getline(cin, input);
            continue;
        }

        params.outputImagePath = input;
        break;
    }
    cout << endl << endl;

    // 7. GIF output path (optional)
    cout << "7. Do you want to generate a GIF showing the compression
process? (y/n): ";
    getline(cin, input);

    bool generateGif = false;
    while (true) {
        if (input.empty()) {
            cout << ANSI_RED << "Error: Input cannot be empty. Please
enter 'y' or 'n'." << ANSI_RESET << endl;
            cout << "Do you want to generate a GIF showing the
compression process? (y/n): ";
            getline(cin, input);
            continue;
        }

        string lowerInput = input;
        for (auto& c : lowerInput) c = tolower(c);

        if (lowerInput == "y" || lowerInput == "yes") {
```

```
            generateGif = true;
            break;
        } else if (lowerInput == "n" || lowerInput == "no") {
            generateGif = false;
            break;
        } else {
            cout << ANSI_RED << "Invalid input. Please enter 'y' or
'n'." << ANSI_RESET << endl;
            cout << "Do you want to generate a GIF showing the
compression process? (y/n): ";
            getline(cin, input);
        }
    }

    params.generateGif = generateGif;

    if (generateGif) {
        cout << "Enter absolute path for output GIF (must end with
.gif): ";
        getline(cin, input);

        while (true) {
            if (input.empty()) {
                cout << ANSI_RED << "Error: GIF output path cannot be
empty. Please specify a valid file path with .gif extension." <<
ANSI_RESET << endl;
                cout << "Enter absolute path for output GIF: ";
                getline(cin, input);
                continue;
            }

            filesystem::path gifPath(input);
            string gifExt = gifPath.extension().string();
            for (auto& c : gifExt) c = tolower(c);

            if (gifExt != ".gif") {
                cout << ANSI_RED << "Error: Output file must have .gif
extension." << ANSI_RESET << endl;
                cout << "Enter absolute path for output GIF: ";
                getline(cin, input);
                continue;
            }

            filesystem::path parentPath = gifPath.parent_path();
            if (!parentPath.empty() && !filesystem::exists(parentPath))
{
                cout << ANSI_RED << "Error: Output directory does not
exist." << ANSI_RESET << endl;
                cout << "Enter absolute path for output GIF: ";
                getline(cin, input);
                continue;
            }

            params.gifOutputPath = input;
            break;
        }
```

```
    }
    cout << endl << endl;


    // Hasil
    cout << "\nParameters summary:" << endl;
    cout << "-------------------" << endl;
    cout << "Input image: " << params.inputImagePath << endl;
    cout << "Error method: ";
    switch (params.errorMethod) {
        case ErrorMethod::VARIANCE: cout << "Variance"; break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION: cout << "Mean
Absolute Deviation"; break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE: cout << "Max Pixel
Difference"; break;
        case ErrorMethod::ENTROPY: cout << "Entropy"; break;
        case ErrorMethod::STRUCTURAL_SIMILARITY: cout << "Structural
Similarity Index"; break;
    }
    cout << endl;

    cout << "Threshold: " << params.threshold << endl;
    cout << "Minimum block size: " << params.minBlockSize << " square
pixels" << endl;
    cout << "Target compression: ";
    if (params.targetCompressionPercentage > 0.0) {
        cout << (params.targetCompressionPercentage * 100) << "%" <<
endl;
    } else {
        cout << "Disabled" << endl;
    }

    cout << "Output image: " << params.outputImagePath << endl;
    if (params.generateGif) {
        cout << "GIF output: " << params.gifOutputPath << endl;
    } else {
        cout << "GIF generation: Disabled" << endl;
    }

    cout << "\nPress Enter to start compression or Ctrl+C to cancel..."
<< endl;
    getline(cin, input);

    return params;
}
```

3.2.3 CompressionAnalyzer.hpp

```
#ifndef _COMPRESSION_ANALYZER_HPP
#define _COMPRESSION_ANALYZER_HPP
```

```
// include lib files
#include <cstddef>
#include <iostream>
#include <iomanip>

// include header file
#include "QuadTree.hpp"


// namespace
using namespace std;

class CompressionAnalyzer {
    public:
        CompressionAnalyzer(size_t originalSize, size_t compressedSize,
const QuadTree& quadTree); //Ctor
        ~CompressionAnalyzer(); //Dtor

        // Calculate compression metrics
        double calculateCompressionPercentage() const;

        // Display results
        void displayResults(double executionTimeSeconds) const;

    private:
        // Params
        size_t originalImageSize;
        size_t compressedImageSize;
        const QuadTree& quadTree;
    };

#endif
```

3.2.4 CompressionAnalyzer.cpp

```
// Include header file
#include "CompressionAnalyzer.hpp"


CompressionAnalyzer::CompressionAnalyzer(size_t originalSize, size_t
compressedSize, const QuadTree& quadTree) :
originalImageSize(originalSize), compressedImageSize(compressedSize),
quadTree(quadTree) {
    // cons
}

CompressionAnalyzer::~CompressionAnalyzer() {
    // dtor
}
```

```
// percentage calc
double CompressionAnalyzer::calculateCompressionPercentage() const {
    if (originalImageSize == 0) {
        return 0.0;
    }
    return (1.0 - static_cast<double>(compressedImageSize) /
originalImageSize) * 100.0;
}

// handler
void CompressionAnalyzer::displayResults(double executionTimeSeconds)
const {
    cout << "\n======================================" << endl;
    cout << "       Compression Results" << endl;
    cout << "======================================" << endl;

    cout << "Execution Time: " << fixed << setprecision(3)
         << executionTimeSeconds << " seconds" << endl;

    cout << "Original Image Size: " << originalImageSize << " bytes";
    if (originalImageSize >= 1024) {
        cout << " (" << fixed << setprecision(2)
             << (originalImageSize / 1024.0) << " KB)";
    }
    if (originalImageSize >= 1024 * 1024) {
        cout << " (" << fixed << setprecision(2)
             << (originalImageSize / (1024.0 * 1024.0)) << " MB)";
    }
    cout << endl;

    cout << "Compressed Image Size: " << compressedImageSize << "
bytes";
    if (compressedImageSize >= 1024) {
        cout << " (" << fixed << setprecision(2)
             << (compressedImageSize / 1024.0) << " KB)";
    }
    if (compressedImageSize >= 1024 * 1024) {
        cout << " (" << fixed << setprecision(2)
             << (compressedImageSize / (1024.0 * 1024.0)) << " MB)";
    }
    cout << endl;

    cout << "Compression Percentage: " << fixed << setprecision(2)
         << calculateCompressionPercentage() << "%" << endl;

    cout << "QuadTree Depth: " << quadTree.getDepth() << endl;
    cout << "QuadTree Node Count: " << quadTree.getNodeCount() << endl;

    cout << "======================================" << endl;
}
```

3.2.5 CompressionParams.hpp

```
#ifndef _COMPRESSION_PARAMS_HPP
#define _COMPRESSION_PARAMS_HPP


// include lib file
#include <string>


// namespace
using namespace std;

// Error method list
enum class ErrorMethod {
    VARIANCE = 1,
    MEAN_ABSOLUTE_DEVIATION = 2,
    MAX_PIXEL_DIFFERENCE = 3,
    ENTROPY = 4,
    STRUCTURAL_SIMILARITY = 5
};

// Compression parameters structure
struct CompressionParams {
    string inputImagePath;
    ErrorMethod errorMethod;
    double threshold;
    int minBlockSize;
    double targetCompressionPercentage;
    string outputImagePath;
    string gifOutputPath;
    bool generateGif;

    CompressionParams() :
        errorMethod(ErrorMethod::VARIANCE),
        threshold(0.0),
        minBlockSize(1),
        targetCompressionPercentage(0.0),
        generateGif(false) {}
};

#endif
```

3.2.6 ErrorCalculator.hpp

```
#ifndef _ERROR_CALCULATOR_HPP
#define _ERROR_CALCULATOR_HPP
```

```
// include lib files
#include <map>
#include <array>
#include <cmath>
#include <memory>
#include <vector>
#include <algorithm>


// include header files
#include "CompressionParams.hpp"
#include "Pixel.hpp"

// namespace
using namespace std;


class ErrorCalculator {
    public:
        virtual ~ErrorCalculator() = default; // Dtor
        virtual double calculateError(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height) = 0;

        // Factory method to create appropriate error calculator
        static unique_ptr<ErrorCalculator> create(ErrorMethod method);

    protected:
        // Helper method to validate region bounds
        bool isValidRegion(const vector<vector<Pixel>>& pixels, int x,
int y, int width, int height) const;
};



// Variance
class VarianceErrorCalculator : public ErrorCalculator {
    public:
        // Method for calculating error using variance
        double calculateError(const vector<vector<Pixel>>& pixels, int
x, int y, int width, int height) override;

    private:
        // Helper method to calculate variance for each rgb channel
        double calculateVarianceForChannel(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height, int channel);
};



// Mean Absolute Deviation (MAD)
class MADErrorCalculator : public ErrorCalculator {
    public:
        // Method for calculating error using MAD
        double calculateError(const vector<vector<Pixel>>& pixels, int
x, int y, int width, int height) override;
```

```
    private:
        // Helper method to calculate MAD for each rgb channel
        double calculateMADForChannel(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height, int channel);
};



// Max Pixel Difference
class MaxPixelDifferenceCalculator : public ErrorCalculator {
    public:
        // Method for calculating error using max pixel difference
        double calculateError(const vector<vector<Pixel>>& pixels, int
x, int y, int width, int height) override;

    private:
        // Helper method to calculate max pixel difference for each rgb
channel
        double calculateMaxDiffForChannel(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height, int channel);
};



// Entropy
class EntropyCalculator : public ErrorCalculator {
    public:
        // Method for calculating error using entropy
        double calculateError(const vector<vector<Pixel>>& pixels, int
x, int y, int width, int height) override;

    private:
        // Helper method to calculate entropy for each rgb channel
        double calculateEntropyForChannel(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height, int channel);
};



// Structural Similarity Index (SSIM) - Bonus
class SSIMCalculator : public ErrorCalculator {
    public:
        // Method for calculating error using SSIM
        double calculateError(const vector<vector<Pixel>>& pixels, int
x, int y, int width, int height) override;

    private:
        // Helper method to calculate SSIM for each rgb channel
        double calculateSSIMForChannel(
            const vector<vector<Pixel>>& originalBlock,
            const vector<vector<Pixel>>& compressedBlock,
            int x, int y, int width, int height, int channel);

        Pixel calculateAverageColor(const vector<vector<Pixel>>& pixels,
int x, int y, int width, int height);
```

```
    };

#endif
```

3.2.7 ErrorCalculation.cpp

```
// include header file
#include "ErrorCalculator.hpp"


// handler
unique_ptr<ErrorCalculator> ErrorCalculator::create(ErrorMethod method)
{
    switch (method) {
        case ErrorMethod::VARIANCE:
            return make_unique<VarianceErrorCalculator>();
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            return make_unique<MADErrorCalculator>();
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            return make_unique<MaxPixelDifferenceCalculator>();
        case ErrorMethod::ENTROPY:
            return make_unique<EntropyCalculator>();
        case ErrorMethod::STRUCTURAL_SIMILARITY:
            return make_unique<SSIMCalculator>();
        default:
            return make_unique<VarianceErrorCalculator>();
    }
}

// rgb channel
unsigned char getChannelValue(const Pixel& pixel, int channel) {
    switch (channel) {
        case 0: return pixel.r;
        case 1: return pixel.g;
        case 2: return pixel.b;
        default: return 0;
    }
}

// variance sum up
double VarianceErrorCalculator::calculateError(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height) {
    return (calculateVarianceForChannel(pixels, x, y, width, height, 0)
+
            calculateVarianceForChannel(pixels, x, y, width, height, 1)
+
            calculateVarianceForChannel(pixels, x, y, width, height, 2))
/ 3.0;
}

// variance per channel
```

```
double VarianceErrorCalculator::calculateVarianceForChannel(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height, int
channel) {
    double sum = 0;
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i)
            sum += getChannelValue(pixels[j][i], channel);

    double mean = sum / (width * height);
    double variance = 0;
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i) {
            double diff = getChannelValue(pixels[j][i], channel) - mean;
            variance += diff * diff;
        }

    return variance / (width * height);
}

// MAD sum up
double MADErrorCalculator::calculateError(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height) {
    return (calculateMADForChannel(pixels, x, y, width, height, 0) +
            calculateMADForChannel(pixels, x, y, width, height, 1) +
            calculateMADForChannel(pixels, x, y, width, height, 2)) /
3.0;
}

// MAD per channel
double MADErrorCalculator::calculateMADForChannel(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height, int
channel) {
    double sum = 0;
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i)
            sum += getChannelValue(pixels[j][i], channel);

    double mean = sum / (width * height);
    double mad = 0;
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i)
            mad += abs(getChannelValue(pixels[j][i], channel) - mean);

    return mad / (width * height);
}

// Diff sum up
double MaxPixelDifferenceCalculator::calculateError(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height) {
    return (calculateMaxDiffForChannel(pixels, x, y, width, height, 0) +
            calculateMaxDiffForChannel(pixels, x, y, width, height, 1) +
            calculateMaxDiffForChannel(pixels, x, y, width, height, 2))
/ 3.0;
}

// Diff per channel
```

```
double MaxPixelDifferenceCalculator::calculateMaxDiffForChannel(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height, int
channel) {
    unsigned char minVal = 255, maxVal = 0;
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i) {
            auto val = getChannelValue(pixels[j][i], channel);
            minVal = min(minVal, val);
            maxVal = max(maxVal, val);
        }
    return static_cast<double>(maxVal - minVal);
}

// Entropy sum up
double EntropyCalculator::calculateError(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height) {
    return (calculateEntropyForChannel(pixels, x, y, width, height, 0) +
            calculateEntropyForChannel(pixels, x, y, width, height, 1) +
            calculateEntropyForChannel(pixels, x, y, width, height, 2))
/ 3.0;
}

// Entropy per channel
double EntropyCalculator::calculateEntropyForChannel(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int height, int
channel) {
    array<int,256> histogram{};
    for (int j = y; j < y + height; ++j)
        for (int i = x; i < x + width; ++i)
            histogram[getChannelValue(pixels[j][i], channel)]++;

    double entropy = 0, total = width * height;
    for (auto count : histogram)
        if (count)
            entropy -= (count/total) * log2(count/total);

    return entropy;
}

// SSIM sum up
double SSIMCalculator::calculateError(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height) {
    // Ciptakan blok gambar terkompresi (dengan warna rata-rata)
    Pixel avgColor = calculateAverageColor(pixels, x, y, width, height);
    vector<vector<Pixel>> compressedBlock(height, vector<Pixel>(width,
avgColor));

    // Hitung SSIM untuk setiap kanal warna
    double ssim_r = calculateSSIMForChannel(pixels, compressedBlock, x,
y, width, height, 0);
    double ssim_g = calculateSSIMForChannel(pixels, compressedBlock, x,
y, width, height, 1);
    double ssim_b = calculateSSIMForChannel(pixels, compressedBlock, x,
y, width, height, 2);

    // Rata-rata SSIM untuk semua kanal (bobot seragam)
```

```
    double avg_ssim = (ssim_r + ssim_g + ssim_b) / 3.0;

    // Definisi "error" = 1 - SSIM, makin besar SSIM makin kecil error
    return 1.0 - avg_ssim;
}

// SSIM per channel
double SSIMCalculator::calculateSSIMForChannel(const
vector<vector<Pixel>>& originalBlock, const vector<vector<Pixel>>&
compressedBlock, int x, int y, int width, int height, int channel){
    const double L = 255.0;
    const double K1 = 0.01;
    const double K2 = 0.03;
    const double C1 = (K1 * L) * (K1 * L);
    const double C2 = (K2 * L) * (K2 * L);

    int N = width * height;
    if (N < 1) {
        return 1.0;
    }

    // mean for original and compressed blocks
    double sum_x = 0.0, sum_y = 0.0;
    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            sum_x += getChannelValue(originalBlock[y+j][x+i], channel);
            sum_y += getChannelValue(compressedBlock[j][i], channel);
        }
    }

    double mu_x = sum_x / N;
    double mu_y = sum_y / N;

    // variance and covar
    double var_x = 0.0, var_y = 0.0, covar_xy = 0.0;

    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            double x_val = getChannelValue(originalBlock[y+j][x+i],
channel);
            double y_val = getChannelValue(compressedBlock[j][i],
channel);

            double diff_x = x_val - mu_x;
            double diff_y = y_val - mu_y;

            var_x += diff_x * diff_x;
            var_y += diff_y * diff_y;
            covar_xy += diff_x * diff_y;
        }
    }

    if (N > 1) {
        var_x /= (N - 1);
        var_y /= (N - 1);
        covar_xy /= (N - 1);
```

```
    } else {
        var_x = var_y = covar_xy = 0.0;
    }

    //    SSIM(x,y) = ((2μxμy + C1)(2σxy + C2)) / ((μx² + μy² + C1)(σx²
+ σy² + C2))
    double numerator = (2.0 * mu_x * mu_y + C1) * (2.0 * covar_xy + C2);
    double denominator = (mu_x * mu_x + mu_y * mu_y + C1) * (var_x +
var_y + C2);

    if (denominator < 1e-10) {
        return 1.0;
    }

    double ssim = numerator / denominator;

    return ssim;
}

// compressed block
Pixel SSIMCalculator::calculateAverageColor(const vector<vector<Pixel>>&
pixels, int x, int y, int width, int height) {
    int totalR = 0, totalG = 0, totalB = 0;
    int count = width * height;

    for (int j = y; j < y + height; ++j) {
        for (int i = x; i < x + width; ++i) {
            totalR += pixels[j][i].r;
            totalG += pixels[j][i].g;
            totalB += pixels[j][i].b;
        }
    }

    return Pixel(
        static_cast<unsigned char>(totalR / count),
        static_cast<unsigned char>(totalG / count),
        static_cast<unsigned char>(totalB / count)
    );
}
```

3.2.8 GifGenerator.hpp

```
#ifndef _GIF_GENERATOR_HPP
#define _GIF_GENERATOR_HPP


// include lib files
#include <string>
#include <vector>
#include <cstdlib>
#include <iomanip>
#include <sstream>
#include <iostream>
```

```
// include header file
#include "QuadTree.hpp"


// Include OpenCV
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>


// namespace
using namespace std;

class GifGenerator {
    public:
        GifGenerator(); // Ctor
        ~GifGenerator(); // Dtor

        // Generate GIF from QuadTree
        bool generateGif(const QuadTree& quadTree, const string&
outputPath);

    private:
        // Internal frame storage
        struct Frame {
            vector<vector<Pixel>> pixels;
            int width;
            int height;
        };

        // Helper methods
        void renderTreeAtDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node, int targetDepth);
        void renderTreeAtDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node, int targetDepth, int currentDepth);
        void renderPartialDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node,
                                int baseDepth, int nextDepth, float
progress);
        void drawNode(Frame& frame, const shared_ptr<QuadTreeNode>&
node);
        int getNodeDepth(const shared_ptr<QuadTreeNode>& node);

        vector<Frame> frames;
        int imageWidth;
        int imageHeight;
};

#endif
```

3.2.9 GifGenerator.cpp

```
// include header file
#include "GifGenerator.hpp"
```

```
GifGenerator::GifGenerator(): imageWidth(0), imageHeight(0) {
    // cons
}

GifGenerator::~GifGenerator() {
    // dtor
}

bool GifGenerator::generateGif(const QuadTree& quadTree, const string&
outputPath) {
    if (!quadTree.getRoot()) {
        return false;
    }

    auto root = quadTree.getRoot();
    imageWidth = root->getWidth();
    imageHeight = root->getHeight();

    try {
        string tempDir;

        // simpan gambar perframe di temporary folder
        #ifdef _WIN32 // windows
            tempDir = "temp_quadtree_frames";
            system(("if exist " + tempDir + " rmdir /s /q " +
tempDir).c_str());
            system(("mkdir " + tempDir).c_str());
        #else // linux
            tempDir = "/tmp/quadtree_frames";
            system(("rm -rf " + tempDir + " && mkdir -p " +
tempDir).c_str());
        #endif

        frames.clear();

        int maxFrames = 80;
        int depthLimit = quadTree.getDepth();

        for (int depth = 0; depth <= depthLimit; ++depth) {
            // buat frame sesuai dengan depth
            Frame frame;
            frame.width = imageWidth;
            frame.height = imageHeight;

            frame.pixels.resize(imageHeight, vector<Pixel>(imageWidth,
Pixel(255, 255, 255)));

            // Fill frame sesuai sama node relatif terhadap depth
            renderTreeAtDepth(frame, quadTree.getRoot(), depth);
            frames.push_back(frame);
        }

        // Save frames
        for (size_t i = 0; i < frames.size(); ++i) {
```

```
            cv::Mat cvFrame(frames[i].height, frames[i].width, CV_8UC3);

            for (int y = 0; y < frames[i].height; ++y) {
                for (int x = 0; x < frames[i].width; ++x) {
                    cv::Vec3b& color = cvFrame.at<cv::Vec3b>(y, x);
                    color[0] = frames[i].pixels[y][x].b;
                    color[1] = frames[i].pixels[y][x].g;
                    color[2] = frames[i].pixels[y][x].r;
                }
            }

            stringstream ss;
            ss << tempDir << "/frame_" << setw(5) << setfill('0') << i
<< ".png";
            cv::imwrite(ss.str(), cvFrame);
        }

        string cmd;
        #ifdef _WIN32 // windows
            string imCheck = "where magick >nul 2>&1";
            int imResult = system(imCheck.c_str());
            if (imResult == 0) {
                cmd = "magick -delay 50 -loop 0 " + tempDir +
"\\frame_*.png \"" + outputPath + "\"";
            } else {
                string ffmpegCheck = "where ffmpeg >nul 2>&1";
                int ffmpegResult = system(ffmpegCheck.c_str());
                if (ffmpegResult == 0) {
                    cmd = "ffmpeg -y -framerate 2 -i " + tempDir +
"\\frame_%05d.png -vf \"scale=trunc(iw/2)*2:trunc(ih/2)*2\" \"" +
outputPath + "\"";
                } else {
                    cerr << "Neither ImageMagick nor FFmpeg found on
Windows. Please install one of them to create GIFs." << endl;
                    cerr << "ImageMagick:
https://imagemagick.org/script/download.php" << endl;
                    cerr << "FFmpeg: https://ffmpeg.org/download.html"
<< endl;

                    system(("rmdir /s /q " + tempDir).c_str());
                    return false;
                }
            }
        #else // linux
            string imCheck = "which convert >/dev/null 2>&1";
            int imResult = system(imCheck.c_str());
            if (imResult == 0) {
                cmd = "convert -delay 50 -loop 0 " + tempDir +
"/frame_*.png \"" + outputPath + "\"";
            } else {
                string ffmpegCheck = "which ffmpeg >/dev/null 2>&1";
                int ffmpegResult = system(ffmpegCheck.c_str());
                if (ffmpegResult == 0) {
                    cmd = "ffmpeg -y -framerate 2 -i " + tempDir +
"/frame_%05d.png -vf \"scale=trunc(iw/2)*2:trunc(ih/2)*2\" \"" +
outputPath + "\"";
```

```
                } else {
                    cerr << "Neither ImageMagick nor FFmpeg found on
Linux. Please install one of them to create GIFs." << endl;
                    cerr << "ImageMagick: sudo apt-get install
imagemagick" << endl;
                    cerr << "FFmpeg: sudo apt-get install ffmpeg" <<
endl;

                    system(("rm -rf " + tempDir).c_str());
                    return false;
                }
            }
        #endif

        int result = system(cmd.c_str());

        if (result != 0) {
            cerr << "Failed to create GIF. Error code: " << result <<
endl;

            // Cleaning up the mess, hehe
            #ifdef _WIN32 // windows
                system(("rmdir /s /q " + tempDir).c_str());
            #else // linux
                system(("rm -rf " + tempDir).c_str());
            #endif

            return false;
        }

        #ifdef _WIN32
            system(("rmdir /s /q " + tempDir).c_str());
        #else
            system(("rm -rf " + tempDir).c_str());
        #endif

        cout << "GIF successfully created at: " << outputPath << endl;

        return true;
    } catch (const exception& e) {
        cerr << "Error generating GIF: " << e.what() << endl;
        return false;
    }
}

void GifGenerator::renderTreeAtDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node, int targetDepth, int currentDepth) {
    if (!node) return;

    // If we've reached a leaf node or the target depth, draw this node
    if (node->isLeaf() || currentDepth == targetDepth) {
        drawNode(frame, node);
    }

    // continue recursing if not
    else if (currentDepth < targetDepth) {
```

```
        for (const auto& child : node->getChildren()) {
            renderTreeAtDepth(frame, child, targetDepth, currentDepth +
1);
        }
    }
}

void GifGenerator::renderTreeAtDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node, int targetDepth) {
    renderTreeAtDepth(frame, node, targetDepth, 0);
}

void GifGenerator::renderPartialDepth(Frame& frame, const
shared_ptr<QuadTreeNode>& node, int baseDepth, int nextDepth, float
progress) {
    if (!node) return;

    if (node->isLeaf()) {
        drawNode(frame, node);
        return;
    }

    int nodeDepth = getNodeDepth(node);

    // Draw all nodes up to baseDepth
    if (nodeDepth <= baseDepth) {
        if (nodeDepth == baseDepth) {
            // At the boundary depth, decide if this node should be
subdivided
            if (progress < 0.5f) {
                drawNode(frame, node);
            } else {
                // show children
                for (const auto& child : node->getChildren()) {
                    drawNode(frame, child);
                }
            }
        } else {
            // if below base depth, recurse to children
            for (const auto& child : node->getChildren()) {
                renderPartialDepth(frame, child, baseDepth, nextDepth,
progress);
            }
        }
    } else {
        drawNode(frame, node);
    }
}

int GifGenerator::getNodeDepth(const shared_ptr<QuadTreeNode>& node) {
    if (!node) return 0;
    if (node->isLeaf()) return 0;

    int maxChildDepth = 0;
    for (const auto& child : node->getChildren()) {
        maxChildDepth = max(maxChildDepth, getNodeDepth(child));
```

```
    }

    return 1 + maxChildDepth;
}

void GifGenerator::drawNode(Frame& frame, const
shared_ptr<QuadTreeNode>& node) {
    if (!node) return;

    Pixel color = node->getColor();
    int x = node->getX();
    int y = node->getY();
    int width = node->getWidth();
    int height = node->getHeight();

    for (int j = y; j < y + height; ++j) {
        for (int i = x; i < x + width; ++i) {
            if (j >= 0 && i >= 0 && j < frame.height && i < frame.width)
{
                frame.pixels[j][i] = color;
            }
        }
    }
}
```

3.2.10 ImageProcessor.hpp

```
#ifndef _IMAGE_PROCESSOR_HPP
#define _IMAGE_PROCESSOR_HPP

// include header files
#include "Pixel.hpp"
#include "QuadTree.hpp"
#include "ErrorCalculator.hpp"
#include "CompressionParams.hpp"

// include lib files
#include <cmath>
#include <mutex>
#include <future>
#include <limits>
#include <memory>
#include <string>
#include <vector>
#include <fstream>
#include <iomanip>
#include <iostream>
#include <algorithm>

// namespace
using namespace std;
```

```
// include OpenCV for image processing
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>


class ImageProcessor {
    public:
        ImageProcessor(const CompressionParams& params); // Ctor
        ~ImageProcessor(); // Dtor

        // Image operations
        bool loadImage(const string& imagePath);
        QuadTree compressImage();
        bool saveCompressedImage(const string& outputPath);
        bool saveCompressedImageToBuffer(const string& extension,
vector<unsigned char>& buffer);
        size_t calculateTheoricalCompressedSize(const QuadTree& tree)
const;

        // Getters
        size_t getOriginalImageSize() const { return originalImageSize;
}
        size_t getCompressedImageSize() const { return
compressedImageSize; }
        int getImageWidth() const { return imageWidth; }
        int getImageHeight() const { return imageHeight; }

    private:
        CompressionParams params;
        int imageWidth;
        int imageHeight;
        vector<vector<Pixel>> pixels;
        unique_ptr<ErrorCalculator> errorCalculator;
        size_t originalImageSize;
        size_t compressedImageSize;
        QuadTree quadTree;

        // Helper methods
        void adjustMinimumBlockSize();
        size_t getFileSize(const string& filename) const;
        void initializeErrorCalculator();
        shared_ptr<QuadTreeNode> buildQuadTree(int x, int y, int width,
int height, int depth);
        bool shouldSubdivide(int x, int y, int width, int height,
double& error);
        Pixel calculateAverageColor(int x, int y, int width, int
height);

        // Target compression methods (bonus)
        double findThresholdForTargetCompression(double
targetPercentage);
        double compressWithThreshold(double threshold);

        // Validator
```

```
        bool isValidRegion(int x, int y, int width, int height) const;
};

#endif
```

3.2.11 ImageProcessor.cpp

```
// include header file
#include "ImageProcessor.hpp"


ImageProcessor::ImageProcessor(const CompressionParams& params):
params(params), imageWidth(0), imageHeight(0), originalImageSize(0),
compressedImageSize(0) {
    // cons

    initializeErrorCalculator();
}

ImageProcessor::~ImageProcessor() {
    // dtor
}


// file size getter
size_t ImageProcessor::getFileSize(const string& filename) const {
    ifstream file(filename, ios::binary | ios::ate);
    if (!file.is_open()) return 0;
    return file.tellg();
}


// take error calculation
void ImageProcessor::initializeErrorCalculator() {
    errorCalculator = ErrorCalculator::create(params.errorMethod);
    if (!errorCalculator) {
        cerr << "Failed to create error calculator. Using default
Variance method." << endl;
        errorCalculator = make_unique<VarianceErrorCalculator>();
    }
}


// adjust minimum block size
void ImageProcessor::adjustMinimumBlockSize() {
    if (imageWidth <= 0 || imageHeight <= 0) {
        return;
    }

    // Calculate the maximum possible area for a block (the entire
image)
```

```
    int maxPossibleArea = imageWidth * imageHeight;

    // Calculate reasonable maximum minimum block size (1/4 of the image
area)
    int recommendedMaxMinBlockSize = maxPossibleArea / 4;

    if (params.minBlockSize > maxPossibleArea) {
        cout << "Warning: Minimum block size (" << params.minBlockSize
             << " square pixels) is larger than the image area ("
             << maxPossibleArea << " square pixels)." << endl;
        cout << "Adjusting to the image area." << endl;
        params.minBlockSize = maxPossibleArea;
    }
    else if (params.minBlockSize > recommendedMaxMinBlockSize) {
        cout << "Warning: Minimum block size (" << params.minBlockSize
             << " square pixels) is very large relative to the image
size." << endl;
        cout << "This may result in poor compression performance." <<
endl;
    }
}


// image loader
bool ImageProcessor::loadImage(const string& imagePath) {
    try {
        cout << "Loading image: " << imagePath << endl;

        cv::Mat image = cv::imread(imagePath, cv::IMREAD_COLOR);

        if (image.empty()) {
            cerr << "Failed to load image: " << imagePath << endl;
            return false;
        }

        // dimensions
        imageWidth = image.cols;
        imageHeight = image.rows;

        cout << "Image loaded: " << imageWidth << "x" << imageHeight <<
" pixels" << endl;

        if (imageWidth <= 0 || imageHeight <= 0) {
            cerr << "Invalid image dimensions" << endl;
            return false;
        }

        originalImageSize = getFileSize(imagePath);

        // Convert to pixel
        pixels.clear();
        pixels.resize(imageHeight, vector<Pixel>(imageWidth));

        for (int y = 0; y < imageHeight; ++y) {
            for (int x = 0; x < imageWidth; ++x) {
                cv::Vec3b color = image.at<cv::Vec3b>(y, x);
```

```
                pixels[y][x] = Pixel(color[2], color[1], color[0]);
            }
        }

        cout << "Image converted to internal format" << endl;

        adjustMinimumBlockSize();
        return true;
    } catch (const exception& e) {
        cerr << "Exception loading image: " << e.what() << endl;
        return false;
    } catch (...) {
        cerr << "Unknown exception loading image" << endl;
        return false;
    }
}

// compressor
QuadTree ImageProcessor::compressImage() {
    cout << "Starting image compression..." << endl;
    QuadTree tree;

    try {
        if (imageWidth <= 0 || imageHeight <= 0 || pixels.empty()) {
            cerr << "No valid image data to compress" << endl;
            return tree;
        }

        if (params.targetCompressionPercentage > 0.0) {
            cout << "Using target compression: " <<
(params.targetCompressionPercentage * 100) << "%" << endl;
            params.threshold =
findThresholdForTargetCompression(params.targetCompressionPercentage);
            cout << "Adjusted threshold: " << params.threshold << endl;
        }

        cout << "Building quadtree..." << endl;
        shared_ptr<QuadTreeNode> root = buildQuadTree(0, 0, imageWidth,
imageHeight, 0);

        if (!root) {
            cerr << "Failed to build quadtree root" << endl;
            return tree;
        }

        tree.setRoot(root);
        tree.calculateDepthAndNodeCount();

        cout << "QuadTree built successfully: "
                  << "depth=" << tree.getDepth()
                  << ", nodes=" << tree.getNodeCount() << endl;

        compressedImageSize = calculateTheoricalCompressedSize(tree);

        quadTree = tree;
```

```
        return tree;
    } catch (const exception& e) {
        cerr << "Exception during compression: " << e.what() << endl;
        return tree;
    } catch (...) {
        cerr << "Unknown exception during compression" << endl;
        return tree;
    }
}

// region checker
bool ImageProcessor::isValidRegion(int x, int y, int width, int height)
const {
    return x >= 0 && y >= 0 && width > 0 && height > 0 && x + width <=
imageWidth && y + height <= imageHeight;
}


// main algo (recursive quadtree compression)
shared_ptr<QuadTreeNode> ImageProcessor::buildQuadTree(int x, int y, int
width, int height, int depth) {
    // Create a new node for this region
    auto node = make_shared<QuadTreeNode>(x, y, width, height);

    // Set the average color for this node ()
    node->setColor(calculateAverageColor(x, y, width, height));

    double error = 0.0;
    bool shouldDivide = shouldSubdivide(x, y, width, height, error); //
checker for subdivide, relatif berdasarkan threshold

    // partisi blok
    int halfWidth = width / 2;
    int remainderWidth = width - halfWidth;
    int halfHeight = height / 2;
    int remainderHeight = height - halfHeight;
    int subBlockArea = halfWidth * halfHeight;

    // subdivision algo
    if (shouldDivide && subBlockArea >= params.minBlockSize) {
        auto topLeft = buildQuadTree(x, y, halfWidth, halfHeight, depth
+ 1);
        auto topRight = buildQuadTree(x + halfWidth, y, remainderWidth,
halfHeight, depth + 1);
        auto bottomLeft = buildQuadTree(x, y + halfHeight, halfWidth,
remainderHeight, depth + 1);
        auto bottomRight = buildQuadTree(x + halfWidth, y + halfHeight,
remainderWidth, remainderHeight, depth + 1);

        // valid children
        if (topLeft) node->addChild(topLeft);
        if (topRight) node->addChild(topRight);
        if (bottomLeft) node->addChild(bottomLeft);
        if (bottomRight) node->addChild(bottomRight);
    }
```

```
    return node;
}

// checker if region should be subdivided
bool ImageProcessor::shouldSubdivide(int x, int y, int width, int
height, double& error) {
    if (!errorCalculator) {
        cerr << "Error calculator is null" << endl;
        return false;
    }

    if (!isValidRegion(x, y, width, height)) {
        return false;
    }

    try {
        error = errorCalculator->calculateError(pixels, x, y, width,
height);

        return error > params.threshold;
    } catch (const exception& e) {
        cerr << "Exception in shouldSubdivide: " << e.what() << endl;
        return false;
    } catch (...) {
        cerr << "Unknown exception in shouldSubdivide" << endl;
        return false;
    }
}

// color normalization
Pixel ImageProcessor::calculateAverageColor(int x, int y, int width, int
height) {
    if (!isValidRegion(x, y, width, height)) {
        return Pixel(0, 0, 0);
    }

    long totalR = 0, totalG = 0, totalB = 0;
    int count = width * height;

    if (count <= 0) {
        return Pixel(0, 0, 0);
    }

    try {
        for (int j = y; j < y + height; ++j) {
            for (int i = x; i < x + width; ++i) {
                totalR += pixels[j][i].r;
                totalG += pixels[j][i].g;
                totalB += pixels[j][i].b;
            }
        }

        return Pixel(
            static_cast<unsigned char>(totalR / count),
            static_cast<unsigned char>(totalG / count),
            static_cast<unsigned char>(totalB / count)
```

```
        );
    } catch (const exception& e) {
        cerr << "Exception in calculateAverageColor: " << e.what() <<
endl;
        return Pixel(0, 0, 0);
    }
}


// compress fuzz (for targetted compression)
double ImageProcessor::compressWithThreshold(double threshold) {
    // Save original threshold (from input)
    double originalThreshold = params.threshold;

    params.threshold = threshold;

    // Build test tree
    shared_ptr<QuadTreeNode> tempRoot = buildQuadTree(0, 0, imageWidth,
imageHeight, 0);

    if (!tempRoot) {
        params.threshold = originalThreshold;
        return 0.0;
    }

    QuadTree testTree;
    testTree.setRoot(tempRoot);
    testTree.calculateDepthAndNodeCount();

    // Calculate the theoretical compressed size
    size_t estimatedCompressedSize =
calculateTheoricalCompressedSize(testTree);

    const size_t MIN_FILE_SIZE = 1024;
    estimatedCompressedSize = max(estimatedCompressedSize,
MIN_FILE_SIZE);

    // Calculate compression percentage
    double compressionRatio = 0.0;
    if (originalImageSize > 0) {
        compressionRatio = 1.0 -
(static_cast<double>(estimatedCompressedSize) /
static_cast<double>(originalImageSize));
    }

    // Apply a correction factor based on empirical observation

    double correctionFactor = 0.7;  // Reduce expected compression
    compressionRatio *= correctionFactor;

    // Bound the result to valid range
    compressionRatio = max(0.0, min(0.99, compressionRatio));

    // Restore original threshold
    params.threshold = originalThreshold;
```

```
    return compressionRatio;
}


// main algo for targetted compression
double ImageProcessor::findThresholdForTargetCompression(double
targetPercentage) {
    // Save original threshold
    double originalThreshold = params.threshold;
    string outputPath = params.outputImagePath;
    string extension = outputPath.substr(outputPath.find_last_of("."));

    double lowT = 0.0, highT = 0.0;
    switch (params.errorMethod) {
        case ErrorMethod::VARIANCE:
            lowT = 0.0;   highT = 16256.25;
            break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            lowT = 0.0;   highT = 127.5;
            break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            lowT = 0.0;   highT = 255.0;
            break;
        case ErrorMethod::ENTROPY:
            lowT = 0.0;  highT = 8.0;
            break;
        case ErrorMethod::STRUCTURAL_SIMILARITY:
            lowT = 0.0;  highT = 1.0;
            break;
    }

    mutex mtx;

    unsigned int systemThreads = thread::hardware_concurrency();
    unsigned int numThreads = max(3u, systemThreads > 1 ? systemThreads
- 1 : 1);

    auto getCompressionRatio = [this, &extension, &mtx](double thresh)
-> double {
        lock_guard<mutex> lock(mtx);
        double oldThreshold = params.threshold;
        params.threshold = thresh;

        shared_ptr<QuadTreeNode> localRoot = buildQuadTree(0, 0,
imageWidth, imageHeight, 0);
        if (!localRoot) {
            params.threshold = oldThreshold;
            return -1.0;
        }

        QuadTree localTree;
        localTree.setRoot(localRoot);
        localTree.calculateDepthAndNodeCount();
        this->quadTree = localTree;

        vector<unsigned char> buffer;
```

```
        if (!saveCompressedImageToBuffer(extension, buffer)) {
            params.threshold = oldThreshold;
            return -1.0;
        }

        double compressionRatio = 1.0 -
(static_cast<double>(buffer.size()) / originalImageSize);
        params.threshold = oldThreshold;
        return compressionRatio;
    };

    map<double, double> cache;

    const double tolerance = 1e-6; // tolerance
    const int maxIterations = 25; // capper for the computation time

    vector<double> initialPoints;
    initialPoints.push_back(lowT);
    initialPoints.push_back((lowT + highT) / 2.0);
    initialPoints.push_back(highT);

    // Add more points based on available threads
    if (numThreads > 3) {
        initialPoints.push_back(lowT + (highT - lowT) / 4.0);
        initialPoints.push_back(lowT + 3 * (highT - lowT) / 4.0);
    }

    // Launch initial evaluations in parallel
    vector<future<double>> initialFutures;
    for (double point : initialPoints) {
        initialFutures.push_back(async(launch::async,
getCompressionRatio, point));
    }

    // Collect initial results
    for (size_t i = 0; i < initialPoints.size(); i++) {
        double ratio = initialFutures[i].get();
        cache[initialPoints[i]] = ratio;

        /*
        cout << "Initial point: threshold = " << initialPoints[i]
             << ", compression ratio = " << ratio << endl;
        */
    }

    // Checker if the target is achiaveable (dari min max)
    double minRatio = numeric_limits<double>::max();
    double maxRatio = numeric_limits<double>::lowest();
    for (const auto& entry : cache) {
        minRatio = min(minRatio, entry.second);
        maxRatio = max(maxRatio, entry.second);
    }

    if (targetPercentage < minRatio || targetPercentage > maxRatio) {
        cout << "Target compression of " << (targetPercentage * 100) <<
"% is not achievable within threshold range. Using inputted
```

```
threshold..." << endl;
        return originalThreshold;
    }

    // Track best threshold found so far
    double bestThreshold = initialPoints[0];
    double bestDiff = abs(cache[initialPoints[0]] - targetPercentage);

    for (const auto& entry : cache) {
        double diff = abs(entry.second - targetPercentage);
        if (diff < bestDiff) {
            bestDiff = diff;
            bestThreshold = entry.first;
        }
    }

    // Binary search w/ multithread
    int iteration = 0;
    while (iteration++ < maxIterations && bestDiff > tolerance) {
        vector<pair<double, double>> points;
        for (const auto& entry : cache) {
            points.push_back({entry.first, entry.second});
        }
        sort(points.begin(), points.end());

        // Find the interval where our target compression ratio falls
        bool foundInterval = false;
        double leftT = lowT, rightT = highT;

        for (size_t i = 0; i < points.size() - 1; i++) {
            double t1 = points[i].first;
            double r1 = points[i].second;
            double t2 = points[i + 1].first;
            double r2 = points[i + 1].second;

            bool inRange = (r1 <= r2 && targetPercentage >= r1 &&
targetPercentage <= r2) ||
                           (r1 >= r2 && targetPercentage <= r1 &&
targetPercentage >= r2);

            if (inRange) {
                leftT = t1;
                rightT = t2;
                foundInterval = true;
                break;
            }
        }

        // Try to add more evaluation points
        if (!foundInterval) {
            vector<double> newPoints;
            for (size_t i = 0; i < points.size() - 1; i++) {
                double mid = (points[i].first + points[i + 1].first) /
2.0;
                if (cache.find(mid) == cache.end()) {
                    newPoints.push_back(mid);
```

```
                }
            }

            while (newPoints.size() > numThreads) {
                // remove points from the smallest intervals first :3

                size_t minSpacingIdx = 0;
                double minSpacing = numeric_limits<double>::max();

                for (size_t i = 0; i < newPoints.size(); i++) {
                    size_t idx = 0;
                    while (idx < points.size() && points[idx].first <
newPoints[i]) {
                        idx++;
                    }

                    double spacing = 0.0;
                    if (idx > 0 && idx < points.size()) {
                        spacing = points[idx].first - points[idx -
1].first;
                    }

                    if (spacing < minSpacing) {
                        minSpacing = spacing;
                        minSpacingIdx = i;
                    }
                }

                newPoints.erase(newPoints.begin() + minSpacingIdx);
            }

            if (newPoints.empty()) {
                break;
            }

            // Evaluate new points in parallel
            vector<future<double>> futures;
            for (double point : newPoints) {
                futures.push_back(async(launch::async,
getCompressionRatio, point));
            }

            // Collect results
            for (size_t i = 0; i < newPoints.size(); i++) {
                double ratio = futures[i].get();
                cache[newPoints[i]] = ratio;

                double diff = abs(ratio - targetPercentage);
                if (diff < bestDiff) {
                    bestDiff = diff;
                    bestThreshold = newPoints[i];
                }

                /*
                cout << "Iteration " << iteration << ", threshold = " <<
newPoints[i]
```

```
                    << ", compression = " << ratio << ", diff = " <<
diff << endl;
                */
            }

            continue;
        }

        // Update search range (iteration)
        lowT = leftT;
        highT = rightT;

        // Generate test points within the current interval
        vector<double> testPoints;

        double midT = (lowT + highT) / 2.0;
        if (cache.find(midT) == cache.end()) {
            testPoints.push_back(midT);
        }

        // Add quarter points if interval is large enough
        double range = highT - lowT;
        if (range > (initialPoints.back() - initialPoints.front()) /
100.0) {
            double quarterT = lowT + range / 4.0;
            double threeQuarterT = lowT + 3.0 * range / 4.0;

            if (cache.find(quarterT) == cache.end()) {
                testPoints.push_back(quarterT);
            }

            if (cache.find(threeQuarterT) == cache.end()) {
                testPoints.push_back(threeQuarterT);
            }
        }

        if (testPoints.empty()) {
            break;
        }

        // Evaluate test points in parallel
        vector<future<double>> futures;
        for (double point : testPoints) {
            futures.push_back(async(launch::async, getCompressionRatio,
point));
        }

        // Collect results
        for (size_t i = 0; i < testPoints.size(); i++) {
            double ratio = futures[i].get();
            cache[testPoints[i]] = ratio;

            double diff = abs(ratio - targetPercentage);
            if (diff < bestDiff) {
                bestDiff = diff;
                bestThreshold = testPoints[i];
```

```
            }

            /*
            cout << "Iteration " << iteration << ", threshold = " <<
testPoints[i]
                 << ", compression = " << ratio << ", diff = " << diff
<< endl;
            */
        }
    }

    cout << "Best threshold found: " << bestThreshold
         << " (iteration " << iteration << "/" << maxIterations
         << ", difference: " << bestDiff << ")" << endl;

    return bestThreshold;
}


// calculating compressed size theoretically (helper for targetted
compression - might help? hehe)
size_t ImageProcessor::calculateTheoricalCompressedSize(const QuadTree&
tree) const {
    if (!tree.getRoot()) {
        return 0;
    }

    // Count leaf nodes
    int leafCount = 0;

    // Function to recursively count leaf nodes
    function<void(shared_ptr<QuadTreeNode>)> countLeaves =
[&](shared_ptr<QuadTreeNode> node) {
        if (!node) return;

        if (node->isLeaf()) {
            leafCount++;
        } else {
            for (const auto& child : node->getChildren()) {
                countLeaves(child);
            }
        }
    };

    countLeaves(tree.getRoot());

    // Each leaf node needs: (theoretically)
    // - Position (x,y): 2 integers = 8 bytes
    // - Size (width,height): 2 integers = 8 bytes
    // - Color (r,g,b): 3 bytes
    const size_t BYTES_PER_LEAF = 24;

    // Add header
    const size_t HEADER_SIZE = 64;

    // Calculate base size
```

```
    size_t baseSize = HEADER_SIZE + (leafCount * BYTES_PER_LEAF);

    string outputFormat =
params.outputImagePath.substr(params.outputImagePath.find_last_of(".") +
1);

    double formatFactor = 1.0;
    if (outputFormat == "jpg" || outputFormat == "jpeg") {
        formatFactor = 1.5;
    } else if (outputFormat == "png") {
        formatFactor = 2.0;
    }

    double blockSizeFactor = 16.0 / params.minBlockSize;

    return static_cast<size_t>(baseSize * formatFactor *
blockSizeFactor);
}


// converter from compressed to image format
bool ImageProcessor::saveCompressedImage(const string& outputPath) {
    if (!quadTree.getRoot()) {
        cerr << "No quadtree to save" << endl;
        return false;
    }

    // Periksa apakah outputPath adalah path temporary
    bool isTemp = (outputPath.find("/tmp/tucil_temp") != string::npos);

    try {
        if (!isTemp) {
            cout << "Saving compressed image to: " << outputPath <<
endl;
        }

        // Buat gambar baru dengan dimensi yang sama
        cv::Mat outputImage(imageHeight, imageWidth, CV_8UC3,
cv::Scalar(0, 0, 0));

        // Fungsi rekursif untuk merender QuadTree ke gambar
        function<void(shared_ptr<QuadTreeNode>)> renderNode =
[&](shared_ptr<QuadTreeNode> node) {
            if (!node) return;

            if (node->isLeaf()) {
                // Gambar blok dengan warna rata-rata
                Pixel color = node->getColor();
                cv::Scalar pixelColor(color.b, color.g, color.r);
                int x = max(0, node->getX());
                int y = max(0, node->getY());
                int width = min(node->getWidth(), imageWidth - x);
                int height = min(node->getHeight(), imageHeight - y);
                if (width > 0 && height > 0) {
                    cv::rectangle(outputImage, cv::Point(x, y),
cv::Point(x + width, y + height), pixelColor, -1);
```

```
                }
            } else {
                for (const auto& child : node->getChildren()) {
                    renderNode(child);
                }
            }
        };

        // Render QuadTree ke dalam outputImage
        renderNode(quadTree.getRoot());

        // Tentukan parameter kompresi berdasarkan ekstensi file
        vector<int> compression_params;
        if (outputPath.find(".jpg") != string::npos ||
outputPath.find(".jpeg") != string::npos) {
            compression_params.push_back(cv::IMWRITE_JPEG_QUALITY);
            compression_params.push_back(85);
        } else if (outputPath.find(".png") != string::npos) {
            compression_params.push_back(cv::IMWRITE_PNG_COMPRESSION);
            compression_params.push_back(9);
        }

        bool success = cv::imwrite(outputPath, outputImage,
compression_params);
        if (!success) {
            if (!isTemp) {
                cerr << "Failed to save image to: " << outputPath <<
endl;
            }
            return false;
        }

        compressedImageSize = getFileSize(outputPath);

        if (!isTemp) {
            cout << "Image saved successfully" << endl;
        }
        return true;

    } catch (const exception& e) {
        if (!isTemp) {
            cerr << "Exception saving image: " << e.what() << endl;
        }
        return false;
    } catch (...) {
        if (!isTemp) {
            cerr << "Unknown exception saving image" << endl;
        }
        return false;
    }
}


// converter from compressed to buffer (helper for targetted compress)
bool ImageProcessor::saveCompressedImageToBuffer(const string&
extension, vector<unsigned char>& buffer) {
```

```
    if (!quadTree.getRoot()) {
        cerr << "No quadtree to save" << endl;
        return false;
    }

    try {
        cv::Mat outputImage(imageHeight, imageWidth, CV_8UC3,
cv::Scalar(0, 0, 0));

        // render quadtree to image
        function<void(shared_ptr<QuadTreeNode>)> renderNode =
[&](shared_ptr<QuadTreeNode> node) {
            if (!node) return;
            if (node->isLeaf()) {
                Pixel color = node->getColor();
                cv::Scalar pixelColor(color.b, color.g, color.r);
                int x = max(0, node->getX());
                int y = max(0, node->getY());
                int width = min(node->getWidth(), imageWidth - x);
                int height = min(node->getHeight(), imageHeight - y);
                if (width > 0 && height > 0) {
                    cv::rectangle(outputImage, cv::Point(x, y),
cv::Point(x + width, y + height), pixelColor, -1);
                }
            } else {
                for (const auto& child : node->getChildren())
                    renderNode(child);
            }
        };

        renderNode(quadTree.getRoot());

        // extension
        vector<int> compression_params;
        if (extension == ".jpg" || extension == ".jpeg") {
            compression_params.push_back(cv::IMWRITE_JPEG_QUALITY);
            compression_params.push_back(85);
        } else if (extension == ".png") {
            compression_params.push_back(cv::IMWRITE_PNG_COMPRESSION);
            compression_params.push_back(9);
        }

        bool success = cv::imencode(extension, outputImage, buffer,
compression_params);
        if (!success) {
            cerr << "Failed to encode image to memory buffer" << endl;
            return false;
        }

        // Update file size
        compressedImageSize = buffer.size();
        return true;

    } catch (const exception& e) {
        cerr << "Exception in saveCompressedImageToBuffer: " << e.what()
<< endl;
```

```
        return false;
    } catch (...) {
        cerr << "Unknown exception in saveCompressedImageToBuffer" <<
endl;
        return false;
    }
}
```

3.2.12 InputManager.hpp

```
#ifndef _INPUT_MANAGER_HPP
#define _INPUT_MANAGER_HPP

// include lib files
#include <string>
#include <limits>
#include <cstdlib>
#include <fstream>
#include <iostream>
#include <iostream>
#include <algorithm>
#include <filesystem>

// include header files
#include "CompressionParams.hpp"

// include opencv
#include <opencv2/opencv.hpp>
#include <opencv2/imgcodecs.hpp>
#include <opencv2/highgui.hpp>


// namespace
using namespace std;
namespace fs = std::filesystem;


// Input class
class InputManager {
    public:
        InputManager(); // Ctor
        ~InputManager(); // Dtor

        CompressionParams getCompressionParams(); //getter

    private:
        // Data for compression
        enum class InputPage {
            INPUT_IMAGE_PATH = 0,
            ERROR_METHOD = 1,
            THRESHOLD = 2,
```

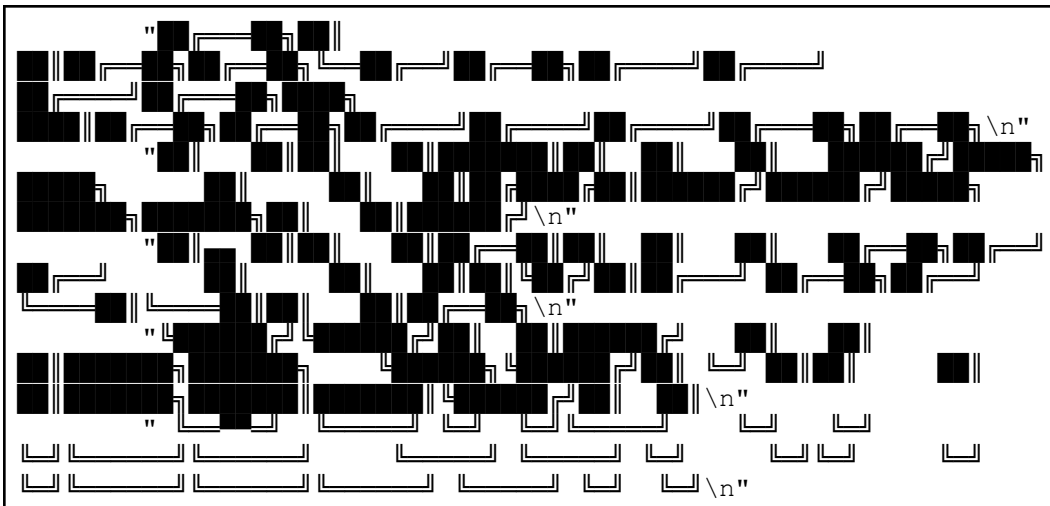
```
            MIN_BLOCK_SIZE = 3,
            TARGET_COMPRESSION = 4,
            OUTPUT_IMAGE_PATH = 5,
            GIF_OUTPUT_PATH = 6,
            CONFIRMATION = 7
        };

        CompressionParams params;
        InputPage currentPage;

        // Page display methods
        void displayPageHeader(const string& title);
        void displayPageFooter();
        void clearScreen();

        // Input Handler methods
        bool handleInputImagePathPage();
        bool handleErrorMethodPage();
        bool handleThresholdPage();
        bool handleMinBlockSizePage();
        bool handleTargetCompressionPage();
        bool handleOutputImagePathPage();
        bool handleGifOutputPathPage();
        bool handleConfirmationPage();

        // Validation methods
        bool fileExists(const string& filePath);
        bool isImageFile(const string& filePath);
        bool validatePath(const string& path, bool mustExist);
        bool validateErrorMethod(int method);
        bool validateThreshold(double threshold);
        bool validateMinBlockSize(int size);
        bool validateTargetCompression(double percentage);

        // Input getter methods
        string getStringInput(const string& prompt);
        int getIntInput(const string& prompt, int min, int max);
        double getDoubleInput(const string& prompt, double min, double
max);
        bool getBoolInput(const string& prompt);
};

#endif
```

3.2.13 InputManager.cpp

```
// include header file
#include "InputManager.hpp"


// ANSI color definitions
#define ANSI_RESET    "\033[0m"
#define ANSI_BOLD     "\033[1m"
#define ANSI_RED      "\033[31m"
```

```
#define ANSI_GREEN    "\033[32m"
#define ANSI_YELLOW   "\033[33m"
#define ANSI_BLUE     "\033[34m"
#define ANSI_MAGENTA  "\033[35m"
#define ANSI_CYAN     "\033[36m"
#define ANSI_WHITE    "\033[37m"


InputManager::InputManager() : currentPage(InputPage::INPUT_IMAGE_PATH)
{
    // constructor
}

InputManager::~InputManager() {
    // dtor
}


// Page handler (untuk paging)
CompressionParams InputManager::getCompressionParams() {
    bool goNext = true;

    while (true) {
        switch (currentPage) {
            case InputPage::INPUT_IMAGE_PATH:
                goNext = handleInputImagePathPage();
                break;
            case InputPage::ERROR_METHOD:
                goNext = handleErrorMethodPage();
                break;
            case InputPage::THRESHOLD:
                goNext = handleThresholdPage();
                break;
            case InputPage::MIN_BLOCK_SIZE:
                goNext = handleMinBlockSizePage();
                break;
            case InputPage::TARGET_COMPRESSION:
                goNext = handleTargetCompressionPage();
                break;
            case InputPage::OUTPUT_IMAGE_PATH:
                goNext = handleOutputImagePathPage();
                break;
            case InputPage::GIF_OUTPUT_PATH:
                goNext = handleGifOutputPathPage();
                break;
            case InputPage::CONFIRMATION:
                goNext = handleConfirmationPage();
                if (goNext) {
                    return params;
                }
                break;
        }

        // Handle navigation
        if (goNext) {
            currentPage =
```

```
static_cast<InputPage>(static_cast<int>(currentPage) + 1);
        } else {
            if (currentPage == InputPage::INPUT_IMAGE_PATH) {
                continue;
            }
            currentPage =
static_cast<InputPage>(static_cast<int>(currentPage) - 1);
        }
    }
}

// Screen flusher untuk refresh
void InputManager::clearScreen() {
    #ifdef _WIN32
        system("cls");
    #else
        system("clear");
    #endif
}

// spacing
void addVerticalSpacing(int lines = 10) {
    for (int i = 0; i < lines; ++i) {
        cout << endl;
    }
}

// Header untuk setiap halaman
void InputManager::displayPageHeader(const string& title) {
    clearScreen();
    #ifdef _WIN32
        cout <<
"    ____  _    _         _____ _______ _____  _______ _______    _____
____  __ __ _____  _____  _______  _____ _____  _____  _____  \n"
"  / __ \\| |  | |  /\\   |  __ \\__   __|  __ \\|  ____|  ____|  /
____/ __ \\|  \\/  |  __ \\|  __ \\|  ____|/ ____/ ____|/ __ \\|  __ \\
\n"
" | |  | | |  | | /  \\  | |  | | | |  | |__) | |__  | |__     | |    | |
| | \\  / | |__) | |__) | |__  | (___| (___ | |  | | |__) |\n"
" | |  | | |  | |/ /\\ \\ | |  | | | |  |  _  /|  __| |  __|    | |    | |
| | |\\/| |  ___/|  _  /|  __|  \\___ \\\\___ \\| |  | |  _  / \n"
" | |__| | |__| / ____ \\| |__| | | |  | | \\ \\| |____| |____   | |___|
|__| | |   | | |    | | \\ \\| |____ ____) |___) | |__| | | \\ \\ \n"
"  \\____\\____/_/    \\_\\\\_____/  |_|  |_|  \\_\\\\______|______|
\\_____\\____/|_|  |_|_|    |_|  \\_\\\\______|_____/_____/ \\____/|_|
\\_\\\n"
"
\n"
"
\n";
    #else
        // Mau pakai ini buat windows juga tapi gak support :(
        cout <<
        " ▄▄▄▄▄▄▄ ▄▄▄     ▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄
▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄
▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ ▄▄▄▄▄▄▄ \n"
```



```
        "\n";
    #endif

    cout << ANSI_YELLOW << ANSI_BOLD << "  QuadTree Image Compression -
" << title << ANSI_RESET << endl;
    cout << ANSI_GREEN << "Page: [ " << (static_cast<int>(currentPage) +
1) << "/8 ]" << ANSI_RESET << endl;
    addVerticalSpacing();
    cout << "type";
    cout << ANSI_MAGENTA << " 'help' " << ANSI_RESET;
    cout << "for list of commands" << endl;
    cout << endl;
}

void InputManager::displayPageFooter() {
    cout << endl;
    cout << "----------------------------------------" << endl;
    if (currentPage == InputPage::INPUT_IMAGE_PATH) {
        cout << ANSI_MAGENTA << "[Enter] Next" << ANSI_RESET << endl;
    } else if (currentPage == InputPage::CONFIRMATION) {
        cout << ANSI_MAGENTA << "[B] Back   [Enter] Confirm and Process"
<< ANSI_RESET << endl;
    } else {
        cout << ANSI_MAGENTA << "[B] Back   [Enter] Next" << ANSI_RESET
<< endl;
    }
    cout << "========================================" << endl;
}


// handle input image
bool InputManager::handleInputImagePathPage() {
    clearScreen();
    displayPageHeader("Input Image Path");
    cout << "Please enter the absolute path to the image to
compress:\n\n";

    while (true) {
        string path = getStringInput(" ");
```

```
        if (path.empty()) {
            cout << ANSI_RED << "Error: Path cannot be empty. Please
enter a valid file path." << ANSI_RESET << endl;
            continue;
        }

        string lowerInput = path;
        transform(lowerInput.begin(), lowerInput.end(),
lowerInput.begin(), ::tolower);

        if (lowerInput == "b") {
            return false;
        }
        else if (lowerInput == "clear") {
            clearScreen();
            displayPageHeader("Input Image Path");
            cout << "Please enter the absolute path to the image to
compress:\n\n";
            continue;
        }

        if (!fileExists(path)) {
            cout << ANSI_RED << "Error: File does not exist." <<
ANSI_RESET << endl;
            continue;
        }
        if (!isImageFile(path)) {
            cout << ANSI_RED << "Error: File is not a supported image
type. Use .jpg, .png, or .jpeg." << ANSI_RESET << endl;
            continue;
        }

        params.inputImagePath = path;
        return true;
    }
}

// handle input error method
bool InputManager::handleErrorMethodPage() {
    clearScreen();
    displayPageHeader("Error Measurement Method");
    cout << "Select the error measurement method:" << endl;
    cout << ANSI_BLUE << "1. Variance" << ANSI_RESET << endl;
    cout << ANSI_BLUE << "2. Mean Absolute Deviation (MAD)" <<
ANSI_RESET << endl;
    cout << ANSI_BLUE << "3. Max Pixel Difference" << ANSI_RESET <<
endl;
    cout << ANSI_BLUE << "4. Entropy" << ANSI_RESET << endl;
    cout << ANSI_BLUE << "5. Structural Similarity Index (SSIM) [Bonus]"
<< ANSI_RESET << endl;

    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            cout << ANSI_RED << "Error: Selection cannot be empty.
```

```
Please choose a method (1-5)." << ANSI_RESET << endl;
            continue;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("Error Measurement Method");
            cout << "Select the error measurement method:" << endl;
            cout << ANSI_BLUE << "1. Variance" << ANSI_RESET << endl;
            cout << ANSI_BLUE << "2. Mean Absolute Deviation (MAD)" <<
ANSI_RESET << endl;
            cout << ANSI_BLUE << "3. Max Pixel Difference" << ANSI_RESET
<< endl;
            cout << ANSI_BLUE << "4. Entropy" << ANSI_RESET << endl;
            cout << ANSI_BLUE << "5. Structural Similarity Index (SSIM)
[Bonus]" << ANSI_RESET << endl;
            continue;
        }

        try {
            int method = stoi(input);
            if (method < 1 || method > 5) {
                cout << ANSI_RED << "Please enter a value between 1 and
5." << ANSI_RESET << endl;
                continue;
            }
            params.errorMethod = static_cast<ErrorMethod>(method);
            return true;
        } catch (const exception&) {
            cout << ANSI_RED << "Invalid input. Please enter a number."
<< ANSI_RESET << endl;
        }
    }
}

// handle input threshold per error method
bool InputManager::handleThresholdPage() {
    clearScreen();
    displayPageHeader("Threshold Value");
    cout << "Enter the threshold value for error." << endl;
    cout << "This value determines when a block should be subdivided."
<< endl;

    double min = 0.0;
    double max = 255.0;
    double defaultValue = 50.0;
    string rangeDescription;

    switch (params.errorMethod) {
        case ErrorMethod::VARIANCE:
```

```
            min = 0.0;
            max = 16256.25; // (127.5)²
            defaultValue = 500.0;
            rangeDescription = "For Variance method (valid range:
0.0-16256.25)\n"
                               "* Low threshold (high detail): 50-500\n"
                               "* Medium threshold: 500-2000\n"
                               "* High threshold (less detail):
2000-5000++";
            break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            min = 0.0;
            max = 127.5; // Maximum possible MAD
            defaultValue = 15.0;
            rangeDescription = "For MAD method (valid range:
0.0-127.5)\n"
                               "* Low threshold (high detail): 5-15\n"
                               "* Medium threshold: 15-30\n"
                               "* High threshold (less detail): 30-50++";
            break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            min = 0.0;
            max = 255.0; // Maximum pixel difference
            defaultValue = 30.0;
            rangeDescription = "For Max Pixel Difference (valid range:
0.0-255.0)\n"
                               "* Low threshold (high detail): 10-30\n"
                               "* Medium threshold: 30-60\n"
                               "* High threshold (less detail):
60-100++";
            break;
        case ErrorMethod::ENTROPY:
            min = 0.0;
            max = 8.0; // Maximum entropy for 8-bit channels
            defaultValue = 1.0;
            rangeDescription = "For Entropy method (valid range:
0.0-8.0)\n"
                               "* Low threshold (high detail): 0.1-1.0\n"
                               "* Medium threshold: 1.0-2.0\n"
                               "* High threshold (less detail):
2.0-4.0++";
            break;
        case ErrorMethod::STRUCTURAL_SIMILARITY:
            min = 0.0;
            max = 1.0; // SSIM error range
            defaultValue = 0.05;
            rangeDescription = "For SSIM method (valid range:
0.0-1.0)\n"
                               "* Low threshold (high detail):
0.01-0.05\n"
                               "* Medium threshold: 0.05-0.15\n"
                               "* High threshold (less detail):
0.15-0.3++";
            break;
    }
```

```
    cout << ANSI_BOLD << ANSI_RED << "\nVALID THRESHOLD RANGE: " << min
<< " to " << max << ANSI_RESET << endl;
    cout << rangeDescription << endl;
    cout << "Suggested value: " << defaultValue << endl;

    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            cout << ANSI_RED << "Error: Threshold value cannot be empty.
Please enter a value within the range "
                 << min << " to " << max << "." << ANSI_RESET << endl;
            continue;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("Threshold Value");
            cout << "Enter the threshold value for error." << endl;
            cout << "This value determines when a block should be
subdivided." << endl;
            cout << ANSI_BOLD << ANSI_RED << "\nVALID THRESHOLD RANGE: "
<< min << " to " << max << ANSI_RESET << endl;
            cout << rangeDescription << endl;
            cout << "Suggested value: " << defaultValue << endl;
            continue;
        }

        try {
            double threshold = stod(input);
            if (threshold < min || threshold > max) {
                cout << ANSI_RED << "ERROR: The value " << threshold <<
" is outside the valid range of "
                     << min << " to " << max << " for the selected error
method." << ANSI_RESET << endl;
                cout << "Please enter a value within the valid range."
<< endl;
                continue;
            }
            params.threshold = threshold;
            return true;
        } catch (const exception&) {
            cout << ANSI_RED << "Invalid input. Please enter a number."
<< ANSI_RESET << endl;
        }
    }
}

// handle input minimum block size sesuai dengan dimensi gambar
bool InputManager::handleMinBlockSizePage() {
```

```
    clearScreen();
    displayPageHeader("Minimum Block Size");
    cout << "Enter the minimum block size in square pixels (area, not
length)" << endl;
    cout << "This is the smallest area (width x height) that will be
used during compression." << endl;

    int minValue = 1;
    int maxValue = 65536; // 256×256 default

    // cari max area dari gambar (buat maksimum min block size)
    if (!params.inputImagePath.empty() &&
fs::exists(params.inputImagePath)) {
        try {
            cv::Mat image = cv::imread(params.inputImagePath,
cv::IMREAD_COLOR);
            if (!image.empty()) {
                int imageWidth = image.cols;
                int imageHeight = image.rows;
                maxValue = imageWidth * imageHeight;
            }
        } catch (...) {
        }
    }

    cout << ANSI_GREEN << "\nAllowed range: " << minValue << "-" <<
maxValue
         << " square pixels" << ANSI_RESET << endl;

    cout << "\nRecommended values (only applies if the dimensions are
relatively big):" << endl;
    cout << "* 4 (2x2) - High detail" << endl;
    cout << "* 16 (4x4) - Good detail" << endl;
    cout << "* 64 (8x8) - Medium detail" << endl;
    cout << "* 256 (16x16++) - Low detail" << endl;

    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            cout << ANSI_RED << "Error: Minimum block size cannot be
empty. Please enter a positive integer value." << ANSI_RESET << endl;
            continue;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            return handleMinBlockSizePage();
        }

        try {
```

```
            int blockSize = stoi(input);
            if (blockSize < minValue) {
                cout << ANSI_RED << "Error: Block size must be at least
" << minValue << " square pixel." << ANSI_RESET << endl;
                continue;
            }

            if (blockSize > maxValue) {
                cout << ANSI_RED << "Error: Block size cannot exceed "
<< maxValue
                     << " square pixels (the image area)." << ANSI_RESET
<< endl;
                continue;
            }

            params.minBlockSize = blockSize;
            return true;
        } catch (const exception&) {
            cout << ANSI_RED << "Invalid input. Please enter a number."
<< ANSI_RESET << endl;
        }
    }
}

// handle input target compression percentage (bonus)
bool InputManager::handleTargetCompressionPage() {
    clearScreen();
    displayPageHeader("Target Compression Percentage");
    cout << "Enter the target compression percentage (0.0-1.0)" << endl;
    cout << "This is a bonus feature that automatically adjusts the
threshold." << endl;
    cout << "Enter 0 to disable automatic threshold adjustment." <<
endl;

    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            cout << ANSI_RED << "Error: Target compression value cannot
be empty. Please enter a value between 0.0 and 1.0." << ANSI_RESET <<
endl;
            continue;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("Target Compression Percentage");
            cout << "Enter the target compression percentage (0.0-1.0)"
<< endl;
            cout << "This is a bonus feature that automatically adjusts
```

```
the threshold." << endl;
            cout << "Enter 0 to disable automatic threshold adjustment."
<< endl;
            continue;
        }

        try {
            double percentage = stod(input);
            if (percentage < 0.0 || percentage > 1.0) {
                cout << ANSI_RED << "Please enter a value between 0.0
and 1.0." << ANSI_RESET << endl;
                continue;
            }
            params.targetCompressionPercentage = percentage;
            return true;
        } catch (const exception&) {
            cout << ANSI_RED << "Invalid input. Please enter a number."
<< ANSI_RESET << endl;
        }
    }
}

// handle output pathing
bool InputManager::handleOutputImagePathPage() {
    clearScreen();
    displayPageHeader("Output Image Path");

    fs::path inputPath(params.inputImagePath);
    string inputExt = inputPath.extension().string();
    transform(inputExt.begin(), inputExt.end(), inputExt.begin(),
::tolower);

    cout << "Enter the output path for the compressed image" << endl;
    cout << ANSI_BOLD << ANSI_RED << "NOTE: The output file MUST use the
same extension as the input file ("
         << inputExt << ")" << ANSI_RESET << endl;

    while (true) {
        string path = getStringInput(" ");

        if (path.empty()) {
            cout << ANSI_RED << "Error: Output path cannot be empty.
Please specify a valid file path with the "
                 << inputExt << " extension." << ANSI_RESET << endl;
            continue;
        }

        string lower = path;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("Output Image Path");
```

```
            cout << "Enter the output path for the compressed image" <<
endl;
            cout << ANSI_BOLD << ANSI_RED << "NOTE: The output file MUST
use the same extension as the input file ("
                 << inputExt << ")" << ANSI_RESET << endl;
            continue;
        }

        fs::path outputPath = path;
        fs::path parentPath = outputPath.parent_path();
        if (!parentPath.empty() && !fs::exists(parentPath)) {
            cout << ANSI_RED << "Error: Output directory does not exist.
Please enter a valid path." << ANSI_RESET << endl;
            continue;
        }

        string providedExt = "";
        size_t extPos = path.rfind('.');
        if (extPos != string::npos) {
            providedExt = path.substr(extPos);
            transform(providedExt.begin(), providedExt.end(),
providedExt.begin(), ::tolower);

            if (providedExt != inputExt) {
                cout << ANSI_RED << "ERROR: Invalid file extension: " <<
providedExt << endl;
                cout << "The output file extension must match the input
file extension: " << inputExt << ANSI_RESET << endl;
                cout << "Please try again with the correct extension."
<< endl;
                continue;
            }

            params.outputImagePath = path;
        } else {
            cout << ANSI_RED << "ERROR: No file extension provided." <<
endl;
            cout << "The output file must have the same extension as the
input file: " << inputExt << ANSI_RESET << endl;
            cout << "Please try again with the correct extension." <<
endl;
            continue;
        }

        cout << ANSI_GREEN << "Output path set to: " <<
params.outputImagePath << ANSI_RESET << endl;
        return true;
    }
}

// handle gif output path (bonus)
bool InputManager::handleGifOutputPathPage() {
    clearScreen();
    displayPageHeader("GIF Output Path (Bonus)");
    cout << "Do you want to generate a GIF showing the compression
process? (y/n)" << endl;
```

```
    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            cout << ANSI_RED << "Error: Selection cannot be empty.
Please enter 'y' or 'n'." << ANSI_RESET << endl;
            continue;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("GIF Output Path (Bonus)");
            cout << "Do you want to generate a GIF showing the
compression process? (y/n)" << endl;
            continue;
        }

        if (lower == "y" || lower == "yes") {
            params.generateGif = true;
            break;
        } else if (lower == "n" || lower == "no") {
            params.generateGif = false;
            return true;
        } else {
            cout << ANSI_RED << "Please enter 'y' or 'n'." << ANSI_RESET
<< endl;
        }
    }

    clearScreen();
    displayPageHeader("GIF Output Path (Bonus)");
    cout << "Enter the output path for the GIF file:" << endl;

    while (true) {
        string path = getStringInput(" ");

        if (path.empty()) {
            cout << ANSI_RED << "Error: GIF output path cannot be empty.
Please specify a valid file path with .gif extension." << ANSI_RESET <<
endl;
            continue;
        }

        string lower = path;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
```

```
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("GIF Output Path (Bonus)");
            cout << "Enter the output path for the GIF file:" << endl;
            continue;
        }

        fs::path outputPath = path;
        fs::path parentPath = outputPath.parent_path();
        if (!parentPath.empty() && !fs::exists(parentPath)) {
            cout << ANSI_RED << "Error: Output directory does not exist.
Please enter a valid path." << ANSI_RESET << endl;
            continue;
        }
        string extension = outputPath.extension().string();
        transform(extension.begin(), extension.end(), extension.begin(),
::tolower);
        if (extension != ".gif") {
            cout << ANSI_RED << "Error: Output must be .gif. Please
enter a valid path." << ANSI_RESET << endl;
            continue;
        }

        params.gifOutputPath = path;
        return true;
    }
}

// handle konfirmasi sebelum proses kompresi
bool InputManager::handleConfirmationPage() {
    clearScreen();
    displayPageHeader("Confirmation");

    cout << "Please review your settings:" << endl;
    cout << ANSI_BOLD << "Input Image Path: " << ANSI_RESET <<
params.inputImagePath << endl;

    cout << ANSI_BOLD << "Error Method: " << ANSI_RESET;
    switch (params.errorMethod) {
        case ErrorMethod::VARIANCE:
            cout << "Variance";
            break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            cout << "Mean Absolute Deviation (MAD)";
            break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            cout << "Max Pixel Difference";
            break;
        case ErrorMethod::ENTROPY:
            cout << "Entropy";
            break;
        case ErrorMethod::STRUCTURAL_SIMILARITY:
            cout << "Structural Similarity Index (SSIM)";
            break;
    }
    cout << endl;
```

```
    cout << ANSI_BOLD << "Threshold: " << ANSI_RESET << params.threshold
<< endl;
    cout << ANSI_BOLD << "Minimum Block Size: " << ANSI_RESET <<
params.minBlockSize << endl;

    if (params.targetCompressionPercentage > 0.0) {
        cout << ANSI_BOLD << "Target Compression: " << ANSI_RESET <<
(params.targetCompressionPercentage * 100) << "%" << endl;
    } else {
        cout << ANSI_BOLD << "Target Compression: " << ANSI_RESET <<
"Disabled" << endl;
    }

    cout << ANSI_BOLD << "Output Image Path: " << ANSI_RESET <<
params.outputImagePath << endl;

    if (params.generateGif) {
        cout << ANSI_BOLD << "GIF Output Path: " << ANSI_RESET <<
params.gifOutputPath << endl;
    } else {
        cout << ANSI_BOLD << "GIF Generation: " << ANSI_RESET <<
"Disabled" << endl;
    }

    cout << endl << "Press ENTER to confirm and start compression, or
type 'B' to go back." << endl;

    while (true) {
        string input = getStringInput(" ");

        if (input.empty()) {
            return true;
        }

        string lower = input;
        transform(lower.begin(), lower.end(), lower.begin(), ::tolower);

        if (lower == "b") {
            return false;
        }
        if (lower == "clear") {
            clearScreen();
            displayPageHeader("Confirmation");
            cout << "Please review your settings:" << endl;
            cout << ANSI_BOLD << "Input Image Path: " << ANSI_RESET <<
params.inputImagePath << endl;

            cout << ANSI_BOLD << "Error Method: " << ANSI_RESET;
            switch (params.errorMethod) {
                case ErrorMethod::VARIANCE:
                    cout << "Variance";
                    break;
                case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
                    cout << "Mean Absolute Deviation (MAD)";
                    break;
```

```
                case ErrorMethod::MAX_PIXEL_DIFFERENCE:
                    cout << "Max Pixel Difference";
                    break;
                case ErrorMethod::ENTROPY:
                    cout << "Entropy";
                    break;
                case ErrorMethod::STRUCTURAL_SIMILARITY:
                    cout << "Structural Similarity Index (SSIM)";
                    break;
            }
            cout << endl;
            cout << ANSI_BOLD << "Threshold: " << ANSI_RESET <<
params.threshold << endl;
            cout << ANSI_BOLD << "Minimum Block Size: " << ANSI_RESET <<
params.minBlockSize << endl;

            if (params.targetCompressionPercentage > 0.0) {
                cout << ANSI_BOLD << "Target Compression: " <<
ANSI_RESET << (params.targetCompressionPercentage * 100) << "%" << endl;
            } else {
                cout << ANSI_BOLD << "Target Compression: " <<
ANSI_RESET << "Disabled" << endl;
            }

            cout << ANSI_BOLD << "Output Image Path: " << ANSI_RESET <<
params.outputImagePath << endl;

            if (params.generateGif) {
                cout << ANSI_BOLD << "GIF Output Path: " << ANSI_RESET
<< params.gifOutputPath << endl;
            } else {
                cout << ANSI_BOLD << "GIF Generation: " << ANSI_RESET <<
"Disabled" << endl;
            }

            cout << endl << "Press ENTER to confirm and start
compression, or type 'B' to go back." << endl;
            continue;
        }
        return true;
    }
}

bool InputManager::fileExists(const string& filePath) {
    // checker buat mastiin file ada atau enggak

    return fs::exists(filePath);
}

bool InputManager::isImageFile(const string& filePath) {
    // checker buat mastiin file image atau bukan

    fs::path path(filePath);
    string ext = path.extension().string();
    transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
```

```
    return (ext == ".jpg" || ext == ".jpeg" || ext == ".png"); // .jpg,
.jpeg, .png
}


bool InputManager::validatePath(const string& path, bool mustExist) {
    // checker untuk absolute path

    if (path.empty()) {
        return false;
    }

    if (mustExist && !fileExists(path)) {
        return false;
    }

    return true;
}

bool InputManager::validateErrorMethod(int method) {
    // checker untuk error method (1-5)

    return method >= 1 && method <= 5;
}

bool InputManager::validateThreshold(double threshold) {
    // checker untuk threshold (relatif dengan error method)

    return threshold >= 0.0;
}

bool InputManager::validateMinBlockSize(int size) {
    // checker untuk minimum block size (relatif dengan dimensi gambar)

    return size >= 1;
}

bool InputManager::validateTargetCompression(double percentage) {
    // checker untuk target compression percentage (0%-100%)

    return percentage >= 0.0 && percentage <= 1.0;
}

string InputManager::getStringInput(const string& prompt) {
    // string input handler

    if (!prompt.empty()) {
        cout << prompt;
    }
    cout << "> ";
    string input;
    getline(cin, input);

    if (input == "exit") {
        exit(0);
    }
```

```
    if (input == "help") {
        cout << "Available commands: clear - clear the input line, help
- show available commands, b - go back, exit - exit program" << endl;
        return getStringInput(prompt);
    }

    return input;
}

int InputManager::getIntInput(const string& prompt, int min, int max) {
    // integer input handler

    if (!prompt.empty()) {
        cout << prompt;
    }
    cout << "> ";
    string input;
    getline(cin, input);

    if (input.empty()) {
        cout << ANSI_RED << "Error: Input cannot be empty. Please enter
a number between " << min << " and " << max << "." << ANSI_RESET <<
endl;
        return getIntInput(prompt, min, max);
    }

    if (input == "exit") {
        exit(0);
    }

    if (input == "help") {
        cout << "Available commands: clear - clear the input line, help
- show available commands, b - go back, exit - exit program" << endl;
        return getIntInput(prompt, min, max);
    }

    if (input == "B" || input == "b") {
        return -1;
    }

    try {
        int value = stoi(input);
        if (value < min || value > max) {
            cout << "Please enter a value between " << min << " and " <<
max << "." << endl;
            return getIntInput("", min, max);
        }
        return value;
    } catch (const exception&) {
        cout << "Invalid input. Please enter a number." << endl;
        return getIntInput("", min, max);
    }
}

double InputManager::getDoubleInput(const string& prompt, double min,
```

```
double max) {
    // double input handler

    if (!prompt.empty()) {
        cout << prompt;
    }
    cout << "> ";
    string input;
    getline(cin, input);

    if (input.empty()) {
        cout << ANSI_RED << "Error: Input cannot be empty. Please enter
a number between " << min << " and " << max << "." << ANSI_RESET <<
endl;
        return getDoubleInput(prompt, min, max);
    }

    if (input == "exit") {
        exit(0);
    }

    if (input == "help") {
        cout << "Available commands: clear - clear the input line, help
- show available commands, b - go back, exit - exit program" << endl;
        return getDoubleInput(prompt, min, max);
    }

    if (input == "B" || input == "b") {
        return -1.0;
    }

    try {
        double value = stod(input);
        if (value < min || value > max) {
            cout << "Please enter a value between " << min << " and " <<
max << "." << endl;
            return getDoubleInput("", min, max);
        }
        return value;
    } catch (const exception&) {
        cout << "Invalid input. Please enter a number." << endl;
        return getDoubleInput("", min, max);
    }
}

bool InputManager::getBoolInput(const string& prompt) {
    // boolean input handler

    if (!prompt.empty()) {
        cout << prompt;
    }

    while (true) {
        cout << "> ";
        string input;
        getline(cin, input);
```

```
        if (input.empty()) {
            cout << ANSI_RED << "Error: Input cannot be empty. Please
enter 'y' for Yes or 'n' for No." << ANSI_RESET << endl;
            continue;
        }

        if (input == "exit") {
            exit(0);
        }

        if (input == "help") {
            cout << "Available commands: clear - clear the input line,
help - show available commands, b - go back, exit - exit program" <<
endl;
            continue;
        }

        if (input == "B" || input == "b") {
            return false;
        }

        if (input == "Y" || input == "y" || input == "yes" || input ==
"Yes") {
            return true;
        } else if (input == "N" || input == "n" || input == "no" ||
input == "No") {
            return false;
        } else {
            cout << "Please enter 'y' for Yes or 'n' for No: ";
        }
    }
}
```

3.2.14 Pixel.hpp

```
#ifndef _PIXEL_HPP
#define _PIXEL_HPP


// Handler for Pixel in image processing
struct Pixel {
    unsigned char r;
    unsigned char g;
    unsigned char b;

    // Default ctor (black)
    Pixel() : r(0), g(0), b(0) {
    }

    // rgb ctor
```

```
    Pixel(unsigned char r, unsigned char g, unsigned char b) : r(r),
g(g), b(b) {
    }

    // Overload operators for pixel calculations
    Pixel operator+(const Pixel& other) const {
        return Pixel(r + other.r, g + other.g, b + other.b);
    }

    Pixel operator-(const Pixel& other) const {
        return Pixel(r - other.r, g - other.g, b - other.b);
    }

    Pixel operator/(int divisor) const {
        return Pixel(r / divisor, g / divisor, b / divisor);
    }

    bool operator==(const Pixel& other) const {
        return r == other.r && g == other.g && b == other.b;
    }

    bool operator!=(const Pixel& other) const {
        return !(*this == other);
    }
};

#endif
```

3.2.15 QuadTree.hpp

```
#ifndef _QUADTREE_HPP
#define _QUADTREE_HPP


// include lib files
#include <memory>
#include <vector>
#include <iostream>
#include <cassert>
#include <algorithm>

// include Pixel header file
#include "Pixel.hpp"


// namespace
using namespace std;


// Node
class QuadTreeNode {
```

```
    public:
        QuadTreeNode(int x, int y, int width, int height); //Ctor
        ~QuadTreeNode() = default; // Dtor

        // Getters
        int getX() const { return x; }
        int getY() const { return y; }
        int getWidth() const { return width; }
        int getHeight() const { return height; }
        bool isLeaf() const { return children.empty(); }
        const Pixel& getColor() const { return color; }
        const vector<shared_ptr<QuadTreeNode>>& getChildren() const {
return children; }

        // Setters
        void setColor(const Pixel& newColor) { color = newColor; }
        void addChild(shared_ptr<QuadTreeNode> child);

    private:
        int x, y;              // Top-left corner
        int width, height;     // Dimensions of the node
        Pixel color;           // color of the node (average color for
leaf nodes)
        vector<shared_ptr<QuadTreeNode>> children; // Child nodes
};


// QuadTree
class QuadTree {
    public:
        QuadTree();
        ~QuadTree() = default;

        // Getters
        shared_ptr<QuadTreeNode> getRoot() const { return root; }
        int getDepth() const { return depth; }
        int getNodeCount() const { return nodeCount; }

        // Setters
        void setRoot(shared_ptr<QuadTreeNode> newRoot);

        // Depth and node count calculation
        void calculateDepthAndNodeCount();

    private:
        shared_ptr<QuadTreeNode> root;
        int depth;
        int nodeCount;

        // Method
        int calculateDepth(shared_ptr<QuadTreeNode> node);
        int countNodes(shared_ptr<QuadTreeNode> node);
};

#endif
```

3.2.16 QuadTree.cpp

```
// include header file
#include "QuadTree.hpp"


// Implementation
QuadTreeNode::QuadTreeNode(int x, int y, int width, int height): x(x),
y(y), width(width), height(height), color(0, 0, 0) {

    // Make sure that the image has valid dimensions
    assert(width > 0 && "Width must be greater than 0");
    assert(height > 0 && "Height must be greater than 0");
}

void QuadTreeNode::addChild(shared_ptr<QuadTreeNode> child) {

    // Child building
    if (children.size() < 4 && child) {
        children.push_back(child);
    }
}

QuadTree::QuadTree(): root(nullptr), depth(0), nodeCount(0) {
    // default constructor
}

void QuadTree::setRoot(shared_ptr<QuadTreeNode> newRoot) {
    root = newRoot;
    calculateDepthAndNodeCount();
}

void QuadTree::calculateDepthAndNodeCount() {
    // Method to calculate the depth and node count of the QuadTree

    if (root) {
        depth = calculateDepth(root);
        nodeCount = countNodes(root);
    } else {
        depth = 0;
        nodeCount = 0;
    }
}

int QuadTree::calculateDepth(shared_ptr<QuadTreeNode> node) {
    // Method to calculate the depth of the QuadTree

    if (!node) {
        return 0;
    }

    // if the node is a leaf
    if (node->isLeaf()) {
```

```
        return 1;
    }

    int maxChildDepth = 0;
    for (const auto& child : node->getChildren()) {
        if (child) {
            maxChildDepth = max(maxChildDepth, calculateDepth(child));
        }
    }

    return 1 + maxChildDepth;
}

int QuadTree::countNodes(shared_ptr<QuadTreeNode> node) {
    // Method to count the number of nodes in the QuadTree

    if (!node) {
        return 0;
    }

    int count = 1;

    for (const auto& child : node->getChildren()) {
        if (child) {
            count += countNodes(child);
        }
    }

    return count;
}
```

3.2.17 main.cpp

```
// include all necessary headers
#include <chrono>
#include "QuadTree.hpp"
#include "InputManager.hpp"
#include "BasicInputManager.hpp"
#include "ImageProcessor.hpp"
#include "CompressionAnalyzer.hpp"
#include "GifGenerator.hpp"

void printUsage() {
    cout << "Quadtree Image Compressor" << endl;
    cout << "=========================" << endl;
    cout << "Usage modes:" << endl;
    cout << "1. Basic Mode: Run with arg \"basic\"" << endl;
    cout << "   ./quadtree_compressor basic" << endl;
    cout << endl;
    cout << "2. Interactive Paging Mode: Run with arg \"page\"" << endl;
    cout << "   ./quadtree_compressor page" << endl;
```

```
    cout << endl;
}

int main(int argc, char* argv[]) {
    CompressionParams params;
    bool useBasicMode = false;

    if (argc > 1) {
        string arg1 = argv[1];
        if (arg1 == "-h" || arg1 == "--help" || arg1 == "help" || arg1
== "/?") {
            printUsage();
            return 0;
        } else if (arg1 == "basic") {  // basic mode
            useBasicMode = true;
        } else if (arg1 == "page") {   // paging mode
            useBasicMode = false;
        } else {
            cout << "Unknown argument: " << arg1 << endl;
            printUsage();
            return 1;
        }
    }

    // Get params
    if (useBasicMode) {
        cout << "=== Quadtree Image Compressor - Basic Mode ===" <<
endl;
        BasicInputManager basicInputManager;
        params = basicInputManager.getCompressionParams();
    } else {
        InputManager inputManager;
        params = inputManager.getCompressionParams();
    }

    // Process the image
    auto start = chrono::high_resolution_clock::now();

    ImageProcessor processor(params);
    if (!processor.loadImage(params.inputImagePath)) {
        cerr << "Failed to load image: " << params.inputImagePath <<
endl;
        return 1;
    }

    QuadTree quadTree = processor.compressImage();

    if (!processor.saveCompressedImage(params.outputImagePath)) {
        cerr << "Failed to save compressed image: " <<
params.outputImagePath << endl;
        return 1;
    }

    if (params.generateGif && !params.gifOutputPath.empty()) {
        GifGenerator gifGen;
        if (!gifGen.generateGif(quadTree, params.gifOutputPath)) {
```

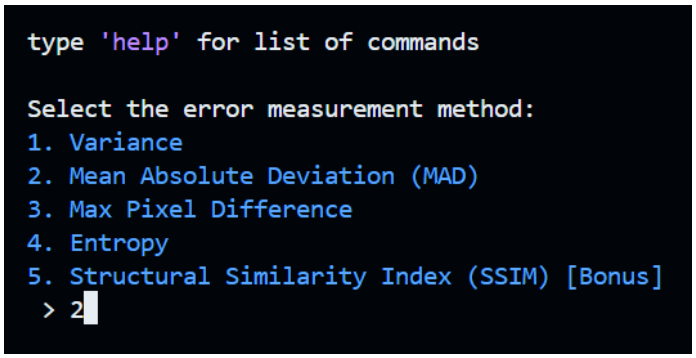
```
            cerr << "Warning: Failed to generate GIF: " <<
params.gifOutputPath << endl;
        }
    }

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> elapsed = end - start;

    // Display results
    CompressionAnalyzer analyzer(processor.getOriginalImageSize(),
processor.getCompressedImageSize(), quadTree);
    analyzer.displayResults(elapsed.count());

    return 0;
}
```

BAB IV: TEST CASES

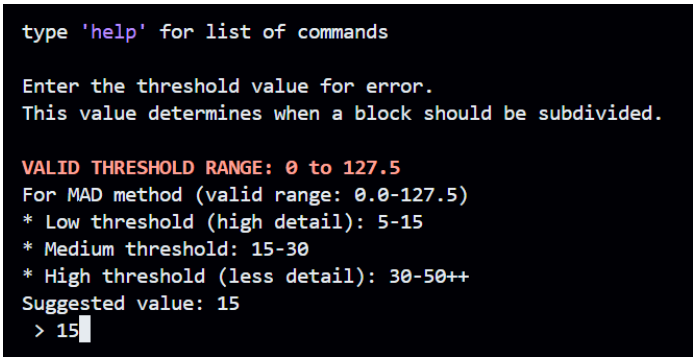
Berikut merupakan output dari percobaan masing-masing *test cases* dengan parameter berbeda yang kami lakukan. Input dari test case berada di dalam folder ../test/input sedangkan hasil dari test case berada di dalam folder ../test/output. Jika ingin mencoba program, format path adalah *absolute path* (sesuai dengan sistem operasi yang dipakai).

4.1. Input Valid - Kompresi Berhasil

Gambar 3. UI Input Gambar



Gambar 4. UI Input Error Measurement Method



Gambar 5. UI Input Threshold

```
Allowed range: 1-65536 square pixels

Recommended values (only applies if the dimensions are relatively big):
* 4 (2x2) - High detail
* 16 (4x4) - Good detail
* 64 (8x8) - Medium detail
* 256 (16x16++) - Low detail
 > 4
```

Gambar 6. UI Input Minimum Block Size

```
type 'help' for list of commands

Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/texas.jpg
Error Method: Mean Absolute Deviation (MAD)
Threshold: 15
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: texow.jpg
GIF Output Path: texgif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/texas.jpg
Image loaded: 1366x2048 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=10, nodes=84585
Saving compressed image to: texow.jpg
Image saved successfully
GIF successfully created at: texgif.gif
```

Gambar 7. UI Success Output

4.2. Variance

Input image (lappland.jpg):

Gambar 8. Input TC Variance

Output image (lapplow.jpg):

Gambar 9. Output TC Variance

```
Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/lappland.jpg
Error Method: Variance
Threshold: 500
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: lapplow.jpg
GIF Output Path: lappgif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/lappland.jpg
Image loaded: 2150x3035 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=11, nodes=161301
Saving compressed image to: lapplow.jpg
Image saved successfully
GIF successfully created at: lappgif.gif

========================================
       Compression Results
========================================
Execution Time: 16.903 seconds
Original Image Size: 703906 bytes (687.41 KB)
Compressed Image Size: 692203 bytes (675.98 KB)
Compression Percentage: 1.66%
QuadTree Depth: 11
QuadTree Node Count: 161301
========================================
```

Gambar 10. Output TC Variance Informations

4.3. Metode Mean Absolute Difference (MAD)

Input image (texas.jpg):

Gambar 11. Input TC MAD

Output image (texlow2.jpg):

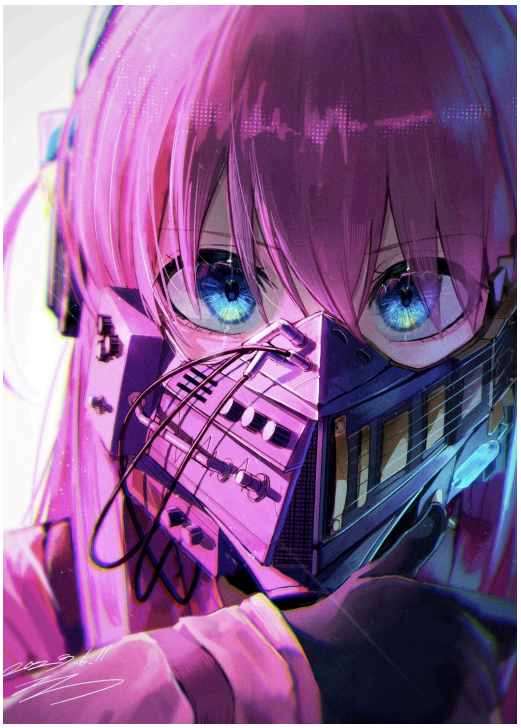
Gambar 12. Output TC MAD

```
Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/texas.jpg
Error Method: Mean Absolute Deviation (MAD)
Threshold: 15
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: texlow2.jpg
GIF Output Path: texlow2gif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/texas.jpg
Image loaded: 1366x2048 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=10, nodes=84585
Saving compressed image to: texlow2.jpg
Image saved successfully
GIF successfully created at: texlow2gif.gif

========================================
       Compression Results
========================================
Execution Time: 6.550 seconds
Original Image Size: 445585 bytes (435.14 KB)
Compressed Image Size: 365048 bytes (356.49 KB)
Compression Percentage: 18.07%
QuadTree Depth: 10
QuadTree Node Count: 84585
========================================
```

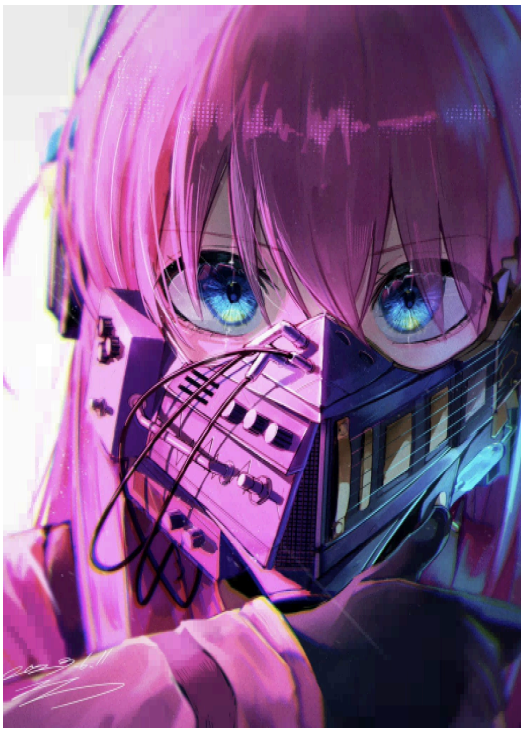
Gambar 13. Output TC MAD Informations

4.4. Max Pixel Difference

Input image (bocchi.jpg):



Output image (bochlow.jpg):



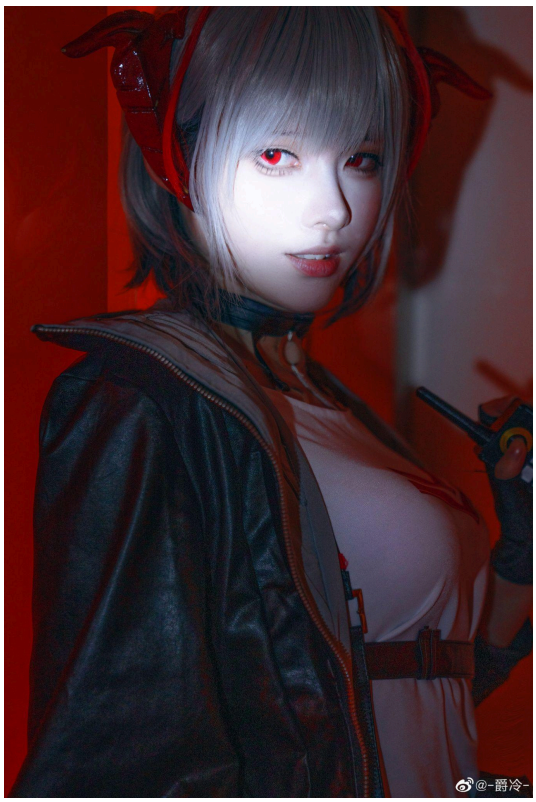
Gambar 14. Input TC Max Pixel Difference **Gambar 15.** Output TC Max Pixel Difference

```
Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/bocchi.jpg
Error Method: Max Pixel Difference
Threshold: 30
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: bochlow.jpg
GIF Output Path: bochlowgif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/bocchi.jpg
Image loaded: 1464x2048 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=10, nodes=191349
Saving compressed image to: bochlow.jpg
Image saved successfully
GIF successfully created at: bochlowgif.gif

=======================================
       Compression Results
=======================================
Execution Time: 12.781 seconds
Original Image Size: 587108 bytes (573.35 KB)
Compressed Image Size: 430622 bytes (420.53 KB)
Compression Percentage: 26.65%
QuadTree Depth: 10
QuadTree Node Count: 191349
=======================================
```

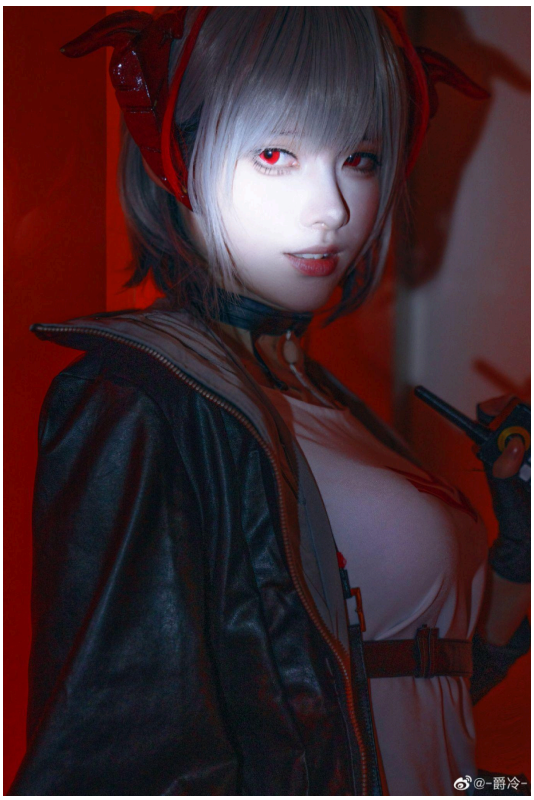
Gambar 16. Output TC Max Pixel Difference Informations

4.5. Entropy

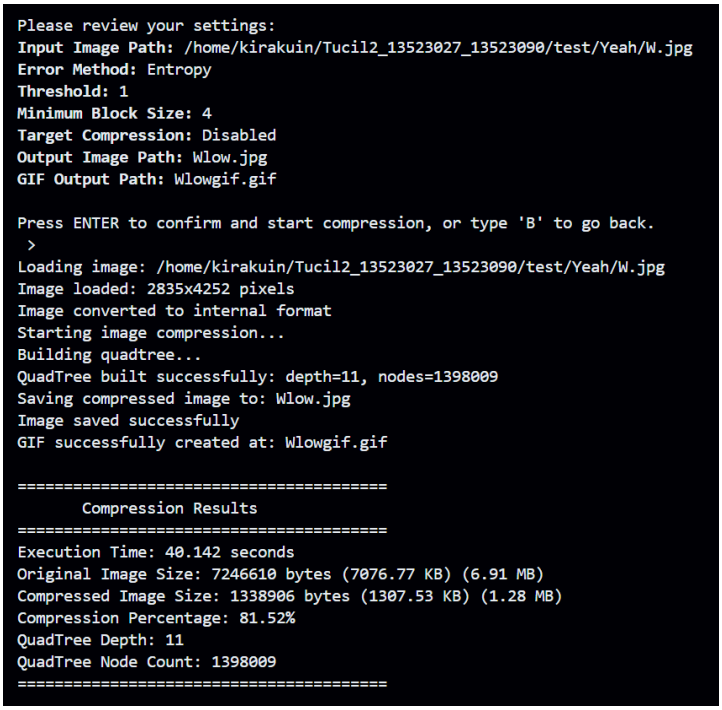
Input image (W.jpg): Output image (Wlow.jpg):



Gambar 17. Input TC Entropy



Gambar 18. Output TC Entropy



Gambar 19. Output TC Entropy Informations

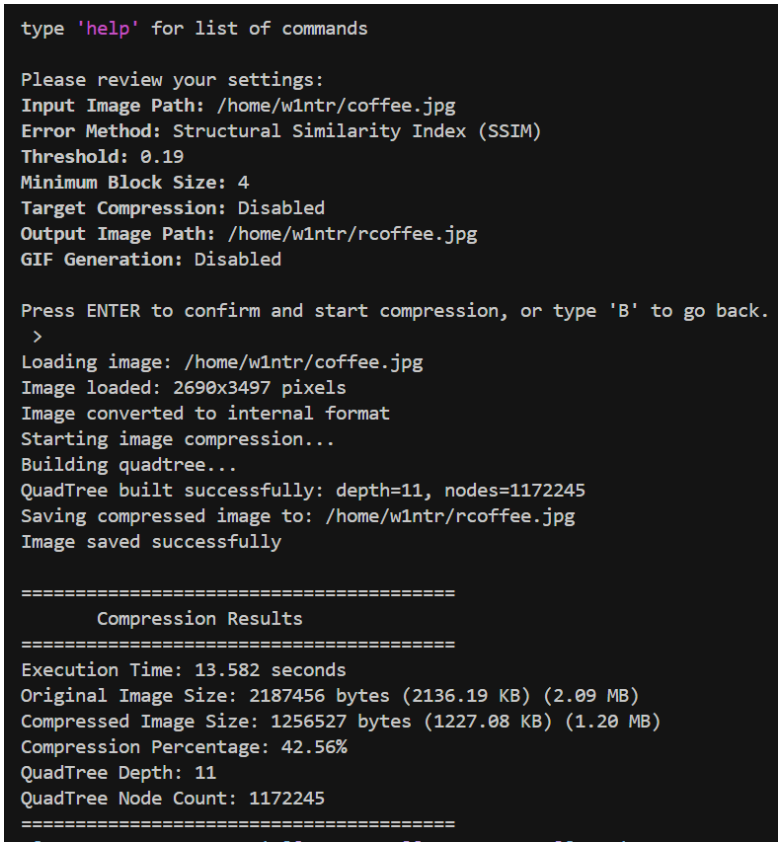
4.6. Structural Similarity Index (SSIM)

Input image (coffee.jpg):

Gambar 20. Input TC SSIM

Output image (rcoffee.jpg):

Gambar 21. Output TC SSIM



Gambar 22. Output TC SSIM Informations

4.7. Target Compression Percentage Tertentu

Input image (tragedy.png):

Gambar 23. Input TC Target Compression 1

Output image (rtragedy.png):

Gambar 24. Output TC Target Compression 1

```
type 'help' for list of commands

Please review your settings:
Input Image Path: /home/w1ntr/tragedy.png
Error Method: Structural Similarity Index (SSIM)
Threshold: 0.17
Minimum Block Size: 2
Target Compression: 69.69%
Output Image Path: /home/w1ntr/rtragedy.png
GIF Generation: Disabled

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/w1ntr/tragedy.png
Image loaded: 773x409 pixels
Image converted to internal format
Starting image compression...
Using target compression: 69.69%
Best threshold found: 0.966949 (iteration 26/25, difference: 1.96466e-05)
Adjusted threshold: 0.966949
Building quadtree...
QuadTree built successfully: depth=10, nodes=41405
Saving compressed image to: /home/w1ntr/rtragedy.png
Image saved successfully

=======================================
       Compression Results
=======================================
Execution Time: 14.319 seconds
Original Image Size: 221820 bytes (216.62 KB)
Compressed Image Size: 67238 bytes (65.66 KB)
Compression Percentage: 69.69%
QuadTree Depth: 10
QuadTree Node Count: 41405
=======================================
[ w1ntr from  lock-in][ 42.046s][ RAM: 1/7GB][ Friday at 9:20:09 PM][ main ≡]
[~/stima/Tucil2_13523027_13523090]
└─Δ
```

Gambar 25. Output TC Target Compression 1 Informations

Input image (kaneki.jpeg):

Gambar 26. Input TC Target Compression 2

Output image (rkaneki.jpeg):

Gambar 27. Output TC Target Compression 2

```
type 'help' for list of commands

Please review your settings:
Input Image Path: /home/w1ntr/kaneki.jpeg
Error Method: Max Pixel Difference
Threshold: 69
Minimum Block Size: 4
Target Compression: 40.2%
Output Image Path: /home/w1ntr/rkaneki.jpeg
GIF Generation: Disabled

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/w1ntr/kaneki.jpeg
Image loaded: 640x933 pixels
Image converted to internal format
Starting image compression...
Using target compression: 40.2%
Best threshold found: 18.9258 (iteration 26/25, difference: 2.77695e-05)
Adjusted threshold: 18.9258
Building quadtree...
QuadTree built successfully: depth=9, nodes=55457
Saving compressed image to: /home/w1ntr/rkaneki.jpeg
Image saved successfully

========================================
       Compression Results
========================================
Execution Time: 7.634 seconds
Original Image Size: 207278 bytes (202.42 KB)
Compressed Image Size: 123958 bytes (121.05 KB)
Compression Percentage: 40.20%
QuadTree Depth: 9
QuadTree Node Count: 55457
========================================
┌[ w1ntr from  lock-in][ 1:2.238s][ RAM: 1/7GB][ Friday at 9:32:53 PM][ main ≡]
└[~/stima/Tucil2_13523027_13523090]
○ └─△
```

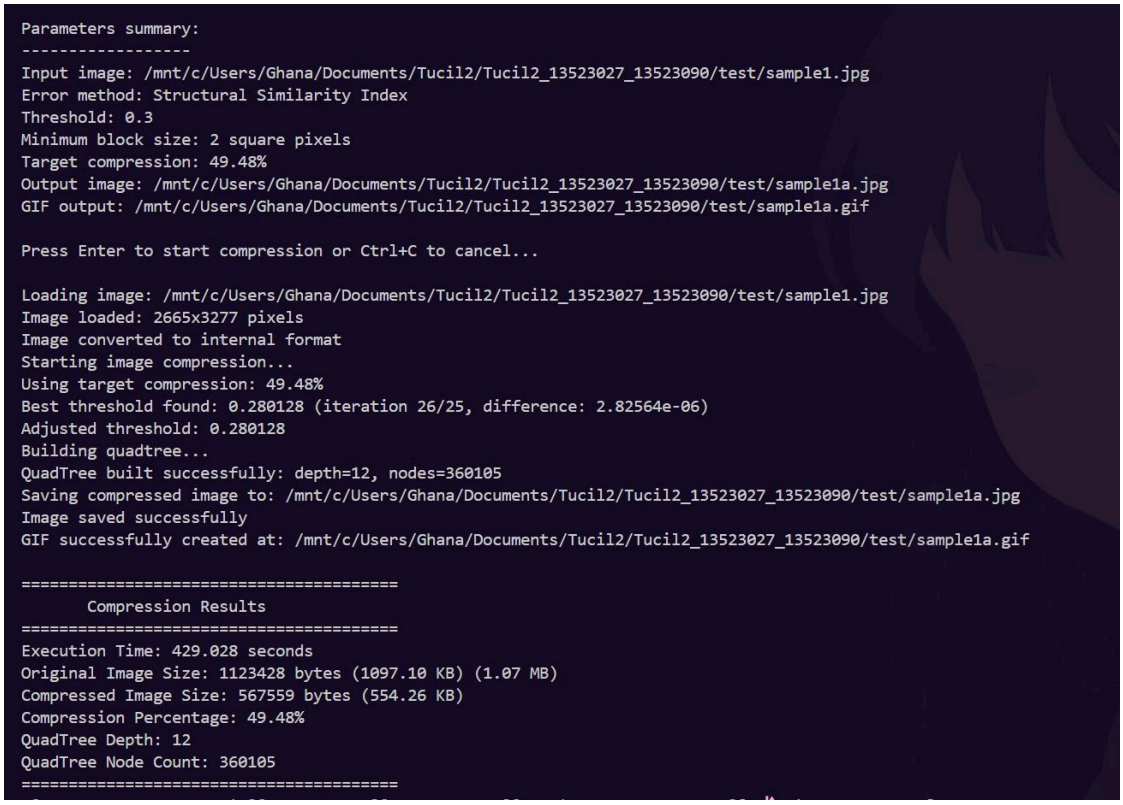
Gambar 28. Output TC Target Compression 2 Informations

Input image (sample.jpg):

Gambar 29. Input TC Target Compression 3

Output image (sample1a.jpg):

Gambar 30. Output TC Target Compression 3



Gambar 31. Output TC Target Compression 3 Informations

4.8. Nilai Threshold Berbeda

Input image (isagi.jpg):

Gambar 32. Input TC Threshold

Threshold 30

(Mean Absolute Difference)

Threshold 100
(Mean Absolute Difference)

Gambar 31. Output TC Threshold 30

Gambar 32. Output TC Threshold 100

```
type 'help' for list of commands

Please review your settings:
Input Image Path: /home/w1ntr/isagi.jpg
Error Method: Mean Absolute Deviation (MAD)
Threshold: 30
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: /home/w1ntr/mygoat.jpg
GIF Generation: Disabled

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/w1ntr/isagi.jpg
Image loaded: 1920x1080 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=10, nodes=9237
Saving compressed image to: /home/w1ntr/mygoat.jpg
Image saved successfully

========================================
       Compression Results
========================================
Execution Time: 0.683 seconds
Original Image Size: 135400 bytes (132.23 KB)
Compressed Image Size: 128127 bytes (125.12 KB)
Compression Percentage: 5.37%
QuadTree Depth: 10
QuadTree Node Count: 9237
========================================
```

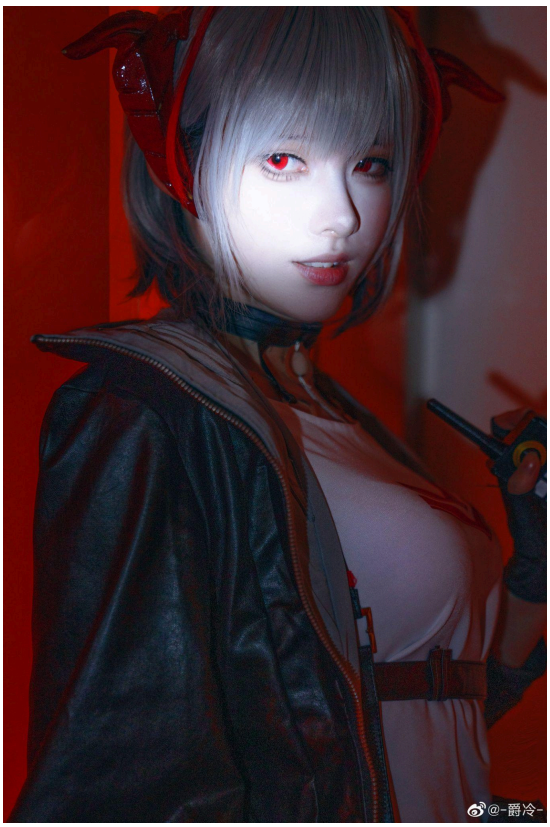
Gambar 33. Output TC Threshold 30 Informations

```
type 'help' for list of commands

Please review your settings:
Input Image Path: /home/w1ntr/isagi.jpg
Error Method: Mean Absolute Deviation (MAD)
Threshold: 127.5
Minimum Block Size: 4
Target Compression: Disabled
Output Image Path: /home/w1ntr/mygoat2.jpg
GIF Generation: Disabled

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/w1ntr/isagi.jpg
Image loaded: 1920x1080 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=1, nodes=1
Saving compressed image to: /home/w1ntr/mygoat2.jpg
Image saved successfully

========================================
       Compression Results
========================================
Execution Time: 0.209 seconds
Original Image Size: 135400 bytes (132.23 KB)
Compressed Image Size: 33268 bytes (32.49 KB)
Compression Percentage: 75.43%
QuadTree Depth: 1
QuadTree Node Count: 1
========================================
```

Gambar 34. Output TC Threshold 100 Informations

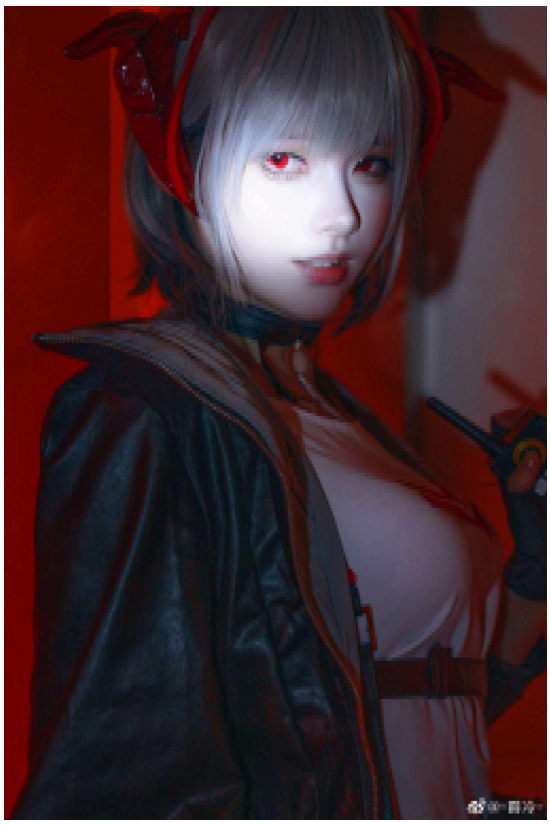
4.9. Nilai Minimum Block Size Berbeda

Input image (W.jpg):

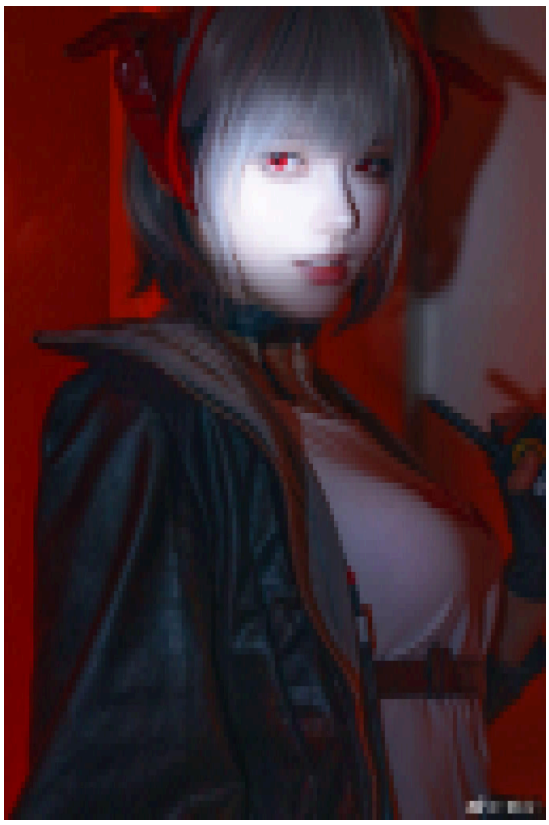


Gambar 34. Input TC Min Block Size

64 Block	256 Block
(Entropy)	(Entropy)



Gambar 34. Output TC Min Block Size 64



Gambar 35. Output TC Min Block Size 256

```
Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/W.jpg
Error Method: Entropy
Threshold: 1
Minimum Block Size: 64
Target Compression: Disabled
Output Image Path: W64.jpg
GIF Output Path: W64gif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/W.jpg
Image loaded: 2835x4252 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=9, nodes=87381
Saving compressed image to: W64.jpg
Image saved successfully
GIF successfully created at: W64gif.gif

========================================
       Compression Results
========================================
Execution Time: 30.785 seconds
Original Image Size: 7246610 bytes (7076.77 KB) (6.91 MB)
Compressed Image Size: 644802 bytes (629.69 KB)
Compression Percentage: 91.10%
QuadTree Depth: 9
QuadTree Node Count: 87381
========================================
```

Gambar 36. Output TC Min Block Size 64 Informations

```
Please review your settings:
Input Image Path: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/W.jpg
Error Method: Entropy
Threshold: 1
Minimum Block Size: 256
Target Compression: Disabled
Output Image Path: W256.jpg
GIF Output Path: W256gif.gif

Press ENTER to confirm and start compression, or type 'B' to go back.
 >
Loading image: /home/kirakuin/Tucil2_13523027_13523090/test/Yeah/W.jpg
Image loaded: 2835x4252 pixels
Image converted to internal format
Starting image compression...
Building quadtree...
QuadTree built successfully: depth=8, nodes=21845
Saving compressed image to: W256.jpg
Image saved successfully
GIF successfully created at: W256gif.gif

========================================
       Compression Results
========================================
Execution Time: 29.795 seconds
Original Image Size: 7246610 bytes (7076.77 KB) (6.91 MB)
Compressed Image Size: 505644 bytes (493.79 KB)
Compression Percentage: 93.02%
QuadTree Depth: 8
QuadTree Node Count: 21845
========================================
```

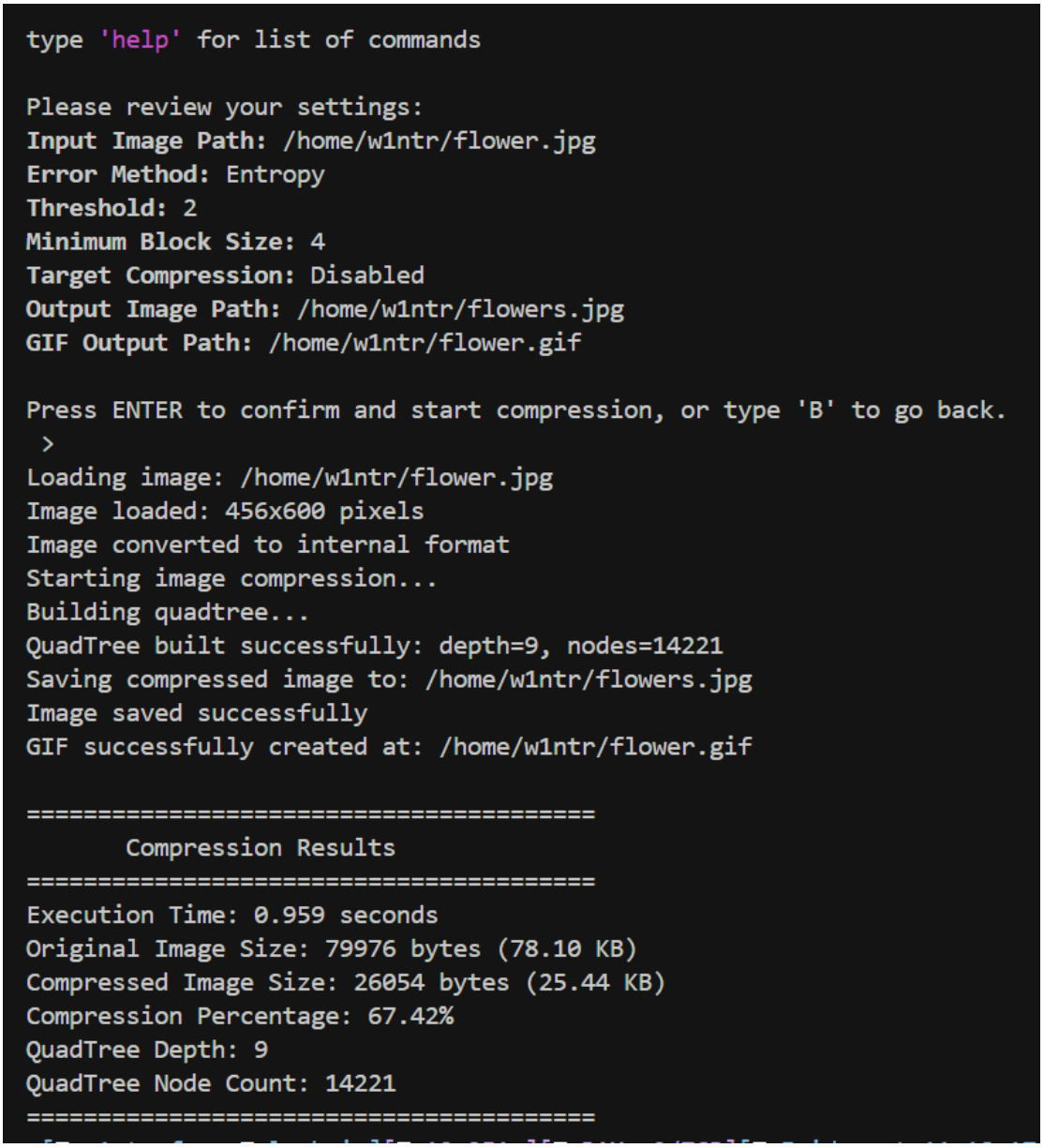
Gambar 37. Output TC Min Block Size 256 Informations

4.10. Output Proses Kompresi ke GIF

Input image (flower.jpg):

Gambar 38. Input TC Generate GIF

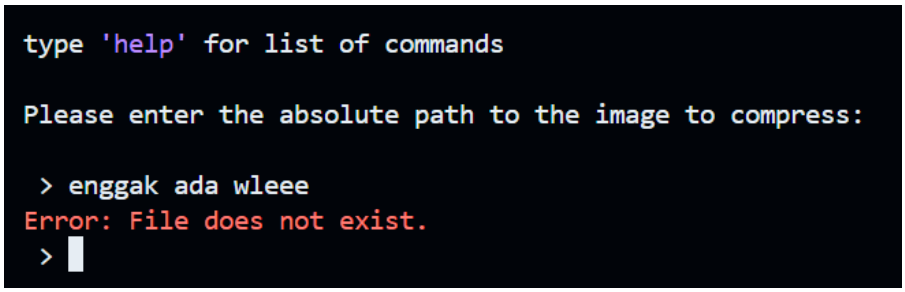
GIF:
https://raw.githubusercontent.com/Nayekah/Tucil2_13523027_13523090/refs/heads/main/test/output/flower.gif



Gambar 39. Output TC Generate GIF Informations

4.11. User Input Error

[Input path] Path ke input image tidak ada atau tidak valid:

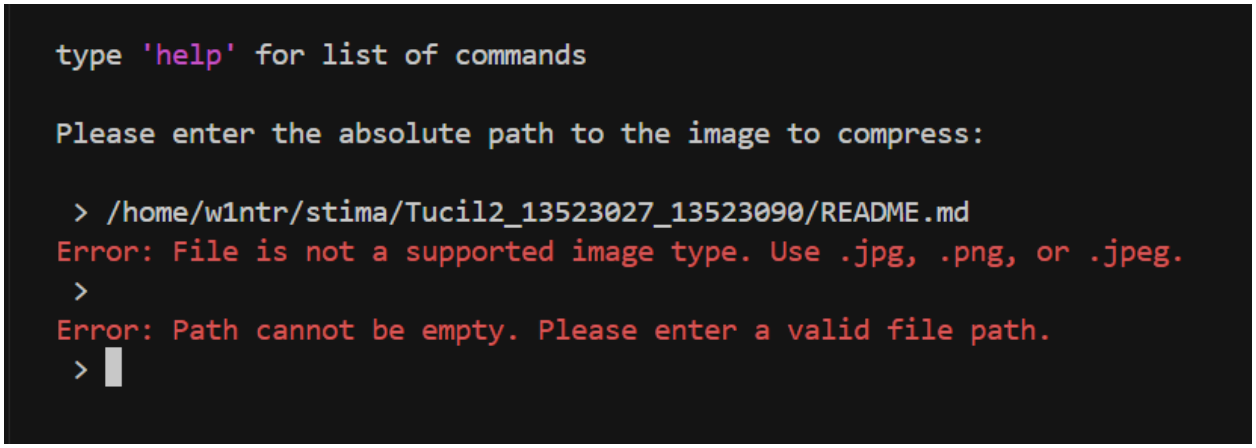


Gambar 40. Input Error 1

[Input path] Path ke input image valid tetapi formatnya tidak sesuai:

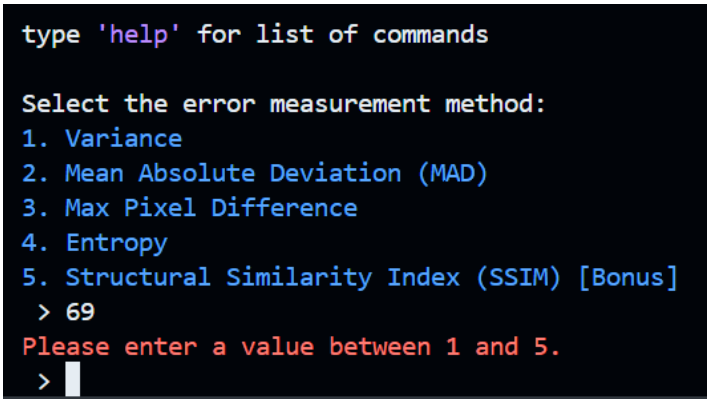
Gambar 41. Input Error 2

[Input path] Path kosong:



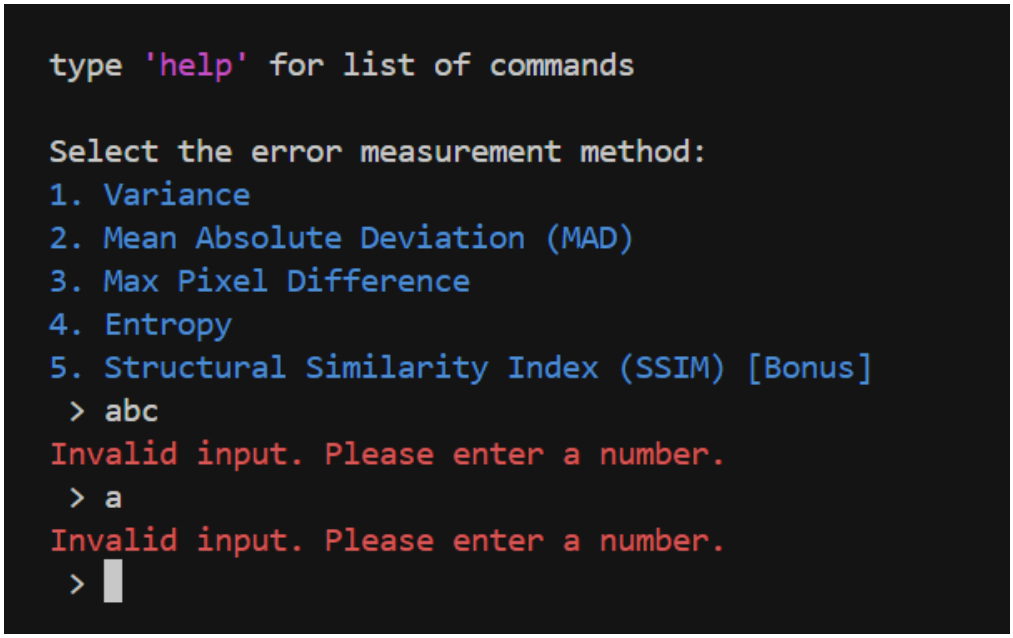
Gambar 42. Input Error 3

[Error method] Input di luar range:



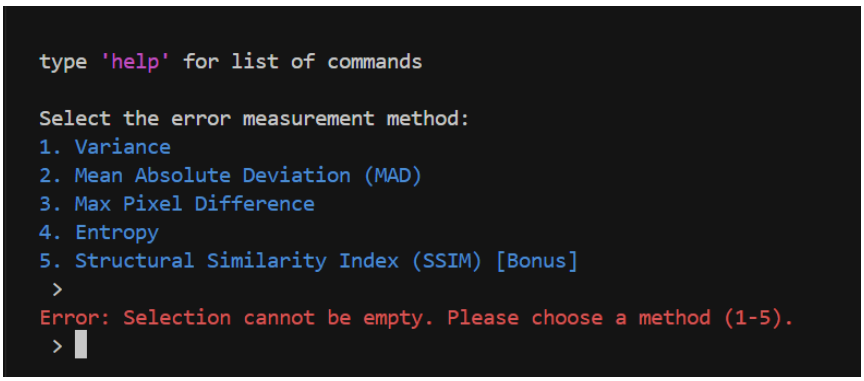
Gambar 43. Input Error 4

[Error method] Input tidak sesuai format:



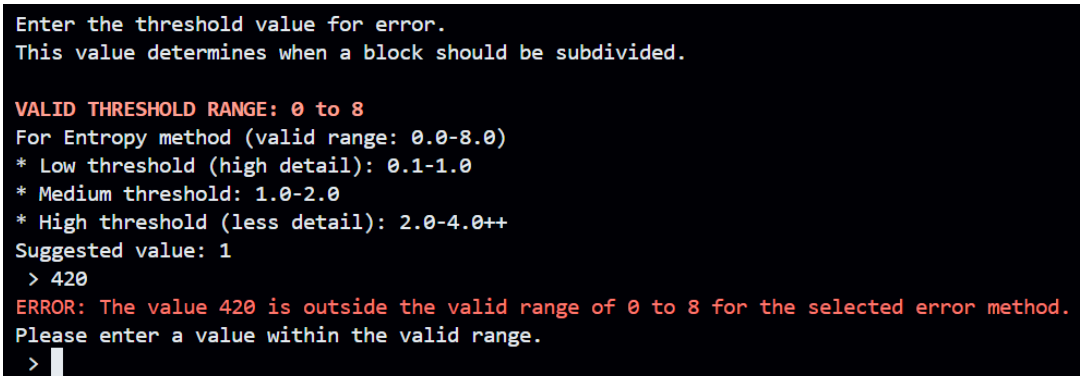
Gambar 44. Input Error 5

[Error method] Input kosong:



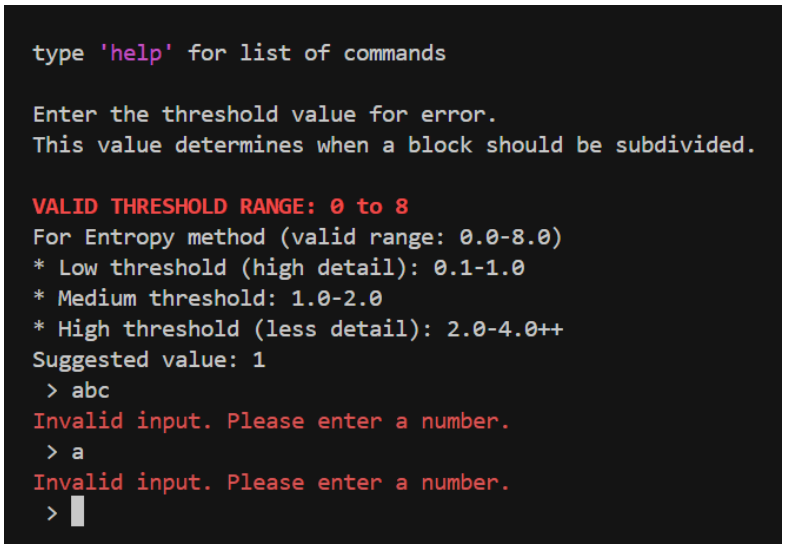
Gambar 45. Input Error 6

[Threshold] : Input di luar range:



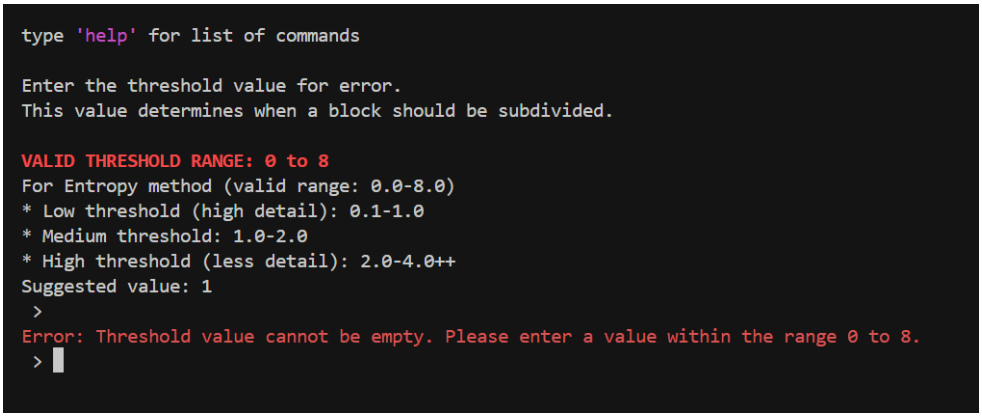
Gambar 46. Input Error 6

[Threshold] : Input tidak sesuai format:



Gambar 47. Input Error 7

[Threshold] : Input kosong:



Gambar 48. Input Error 8

[Minblocksize] : Input di luar range:

```
type 'help' for list of commands

Enter the minimum block size in square pixels (area, not length)
This is the smallest area (width x height) that will be used during compression.

Allowed range: 1-597120 square pixels

Recommended values (only applies if the dimensions are relatively big):
* 4 (2x2) - High detail
* 16 (4x4) - Good detail
* 64 (8x8) - Medium detail
* 256 (16x16++) - Low detail
 > 597121
Error: Block size cannot exceed 597120 square pixels (the image area).
 >
```

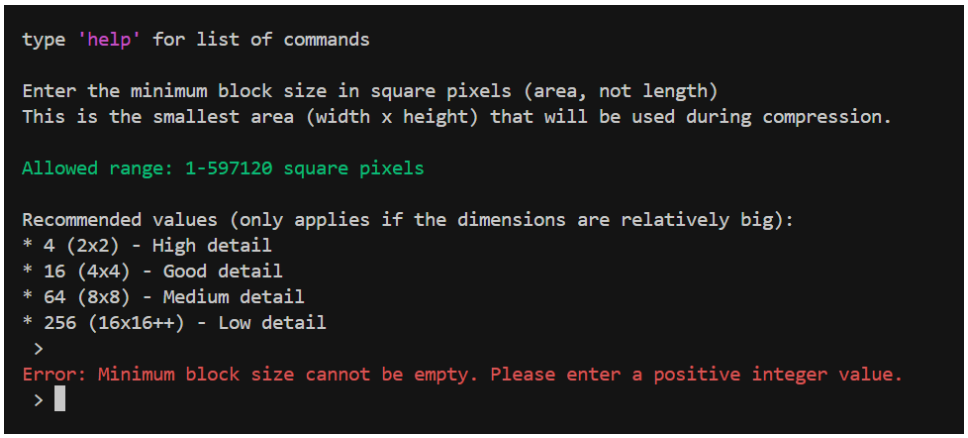
Gambar 49. Input Error 9

[Minblocksize] : Input tidak sesuai format:

```
Enter the minimum block size in square pixels (area, not length)
This is the smallest area (width x height) that will be used during compression.

Allowed range: 1-12054420 square pixels

Recommended values (only applies if the dimensions are relatively big):
* 4 (2x2) - High detail
* 16 (4x4) - Good detail
* 64 (8x8) - Medium detail
* 256 (16x16++) - Low detail
 > kurang tahu
Invalid input. Please enter a number.
 >
```

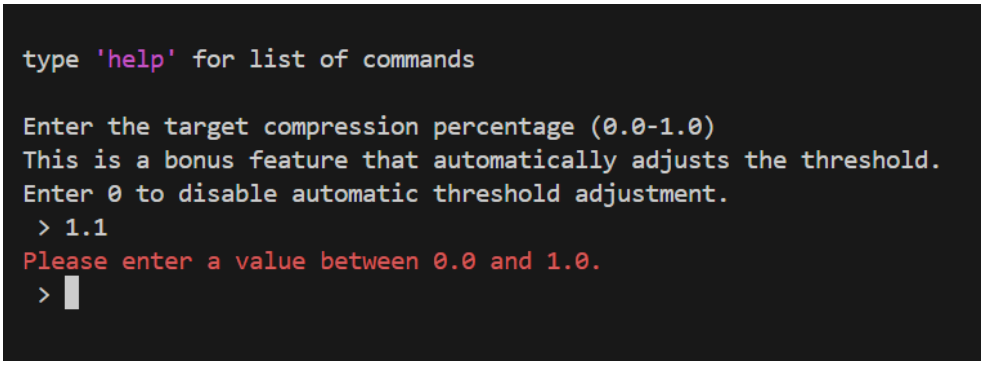
Gambar 50. Input Error 10

[Minblocksize] : Input kosong:



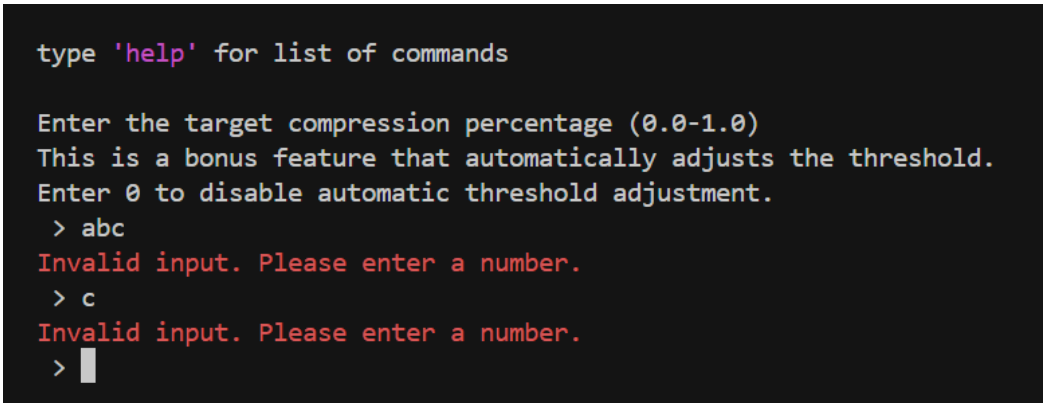
Gambar 51. Input Error 11

[Targeted compression] : Input di luar range:



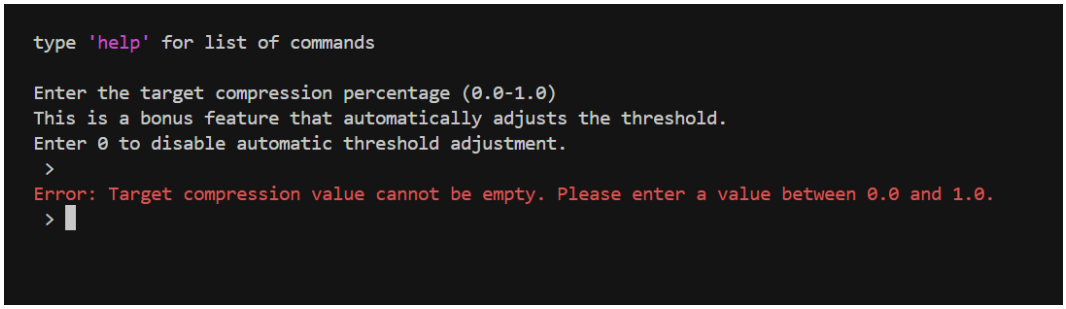
Gambar 52. Input Error 12

[Targeted compression] : Input tidak sesuai format:



Gambar 53. Input Error 13

[Targeted compression] : Input kosong:

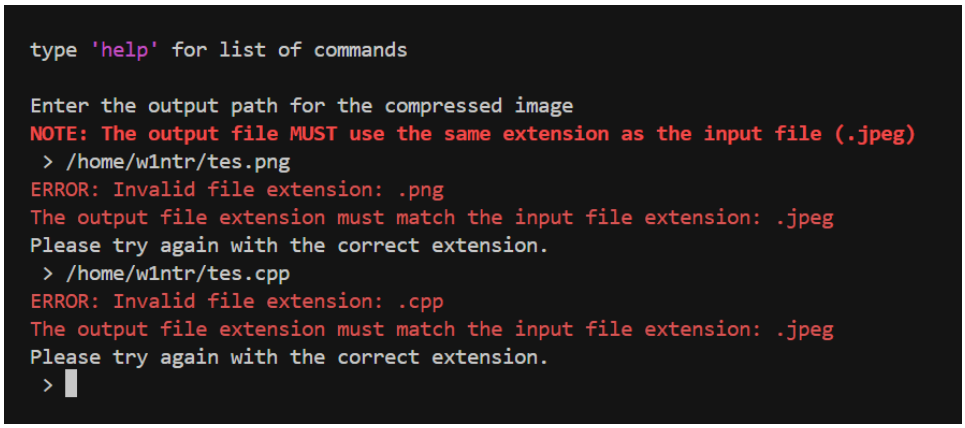


Gambar 54. Input Error 14

[Output path] : Path tidak valid:

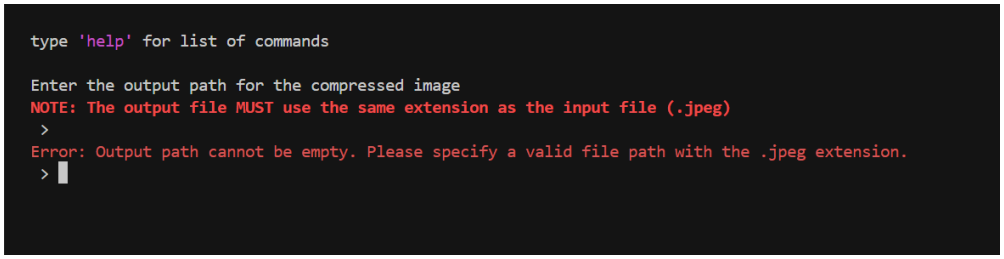
Gambar 55. Input Error 15

[Output path] : Path valid, tetapi extensionnya berbeda atau tidak support:



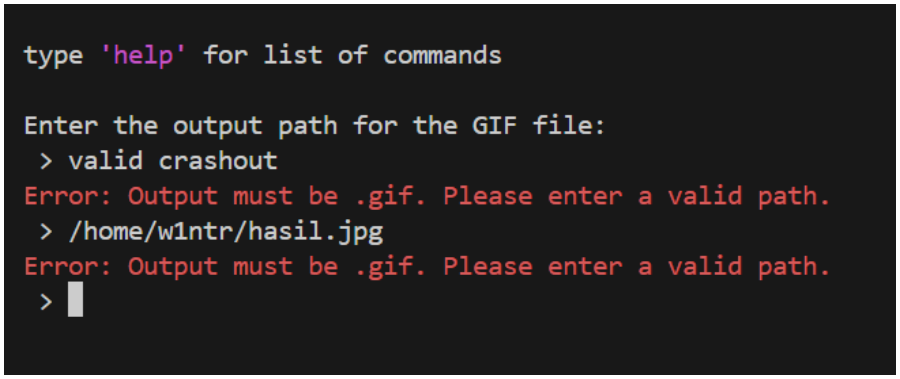
Gambar 56. Input Error 16

[Output path] : Path kosong:



Gambar 57. Input Error 17

[GIF] : Input tidak valid atau ekstensi tidak valid:

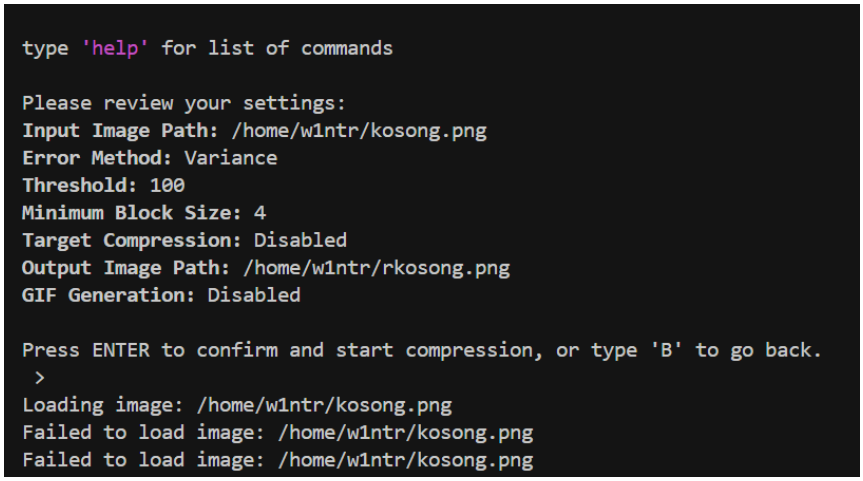


Gambar 58. Input Error 18

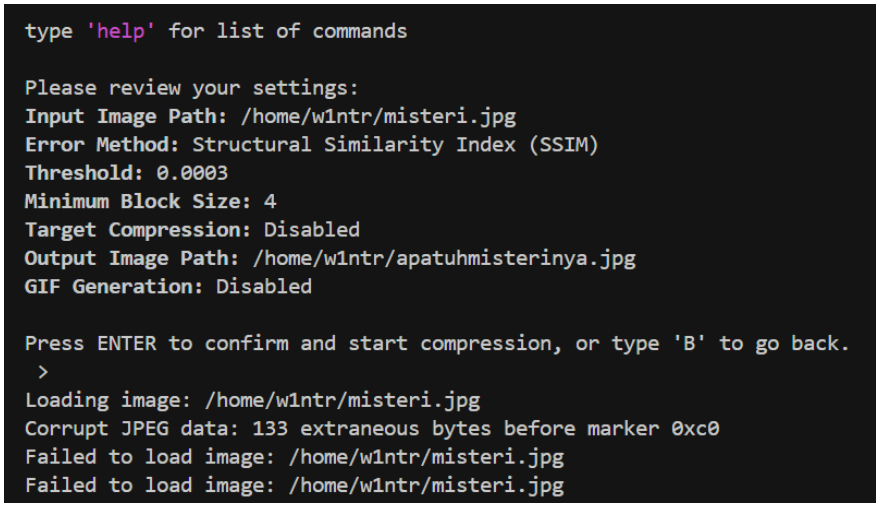
[GIF] : Input kosong:

Gambar 59. Input Error 19

[Compression] : Gambar kosong atau corrupt:



Gambar 60. Input Error 20



Gambar 61. Input Error 2

BAB V: HASIL ANALISIS PERCOBAAN

Berdasarkan konteks Tugas Kecil 2 (di mana gambar umumnya dianggap bertipe 24‑bit RGB dengan rentang nilai piksel 0–255 pada tiap kanal), berikut perkiraan rentang ambang batas (threshold) yang paling umum untuk keempat metode perhitungan error. Dalam penjelasan ini kita asumsikan ukuran blok minimal adalah lebih dari 1 piksel dan kanal R, G, B sama‑sama 8 bit.

1. **Variance**
   - **Formula ringkas**:
     $\sigma^2(c) = (1/N)\ \Sigma_i\ (P_i,c - \mu(c))^2$, kemudian $\sigma^2(RGB) = (\sigma^2(R) + \sigma^2(G) + \sigma^2(B)) / 3$.
   - **Nilai maksimum (8‑bit)**:
     Jika separuh piksel bernilai 0 dan separuh bernilai 255, rata‑ratanya ~127.5 dan variansinya mencapai ~16.256 (ribuan) per piksel. Perhitungannya secara detail kira-kira:
   - $E[X] = 127.5$, $E[X^2] = 32512.5$, sehingga $var = E[X^2] - (E[X])^2 = 16256.25$ untuk satu kanal.
   - Karena $\sigma^2(RGB)$ diambil rata‑rata tiga kanal, nilai maksimumnya masih ~16256.
   - **Rentang threshold**: 0 sampai ~16.256 (kali 10^3), atau lebih mudah dibulatkan menjadi 0 hingga ~16.300.

2. **Mean Absolute Deviation (MAD)**
   - **Formula ringkas**:
     $MAD(c) = (1/N)\ \Sigma_i\ |P_i,c - \mu(c)|$, kemudian $MAD(RGB) = (MAD(R) + MAD(G) + MAD(B)) / 3$.
   - **Nilai maksimum (8‑bit)**:
     Dengan skenario blok setengah piksel 0 dan setengah 255 (rata‑rata ~127.5), selisih mutlak setiap piksel ke rata‑rata adalah ~127.5. Jadi $MAD(c) \approx 127.5$ untuk kanal tersebut. Rata‑rata tiga kanal juga tetap ~127.5 jika semuanya punya pola ekstrim serupa.
   - **Rentang threshold**: 0 hingga ~127.5 (sering dibulatkan 0–128).

3. **Max Pixel Difference**
   - Formula ringkas:
     $D(c) = \max(P_i,c) - \min(P_i,c)$, lalu $D(RGB) = (D(R) + D(G) + D(B)) / 3$.
   - **Nilai maksimum (8‑bit)**:
     Untuk satu kanal, selisih maksimum = 255 − 0 = 255. Bila ketiga kanal sama‑sama memiliki selisih 255, maka $D(RGB) = (255 + 255 + 255) / 3 = 255$.
   - **Rentang threshold**: 0–255.

4. **Entropy**
    - **Formula ringkas**:

    $H(c) = -\Sigma_i P(c)(i) \times \log_2(P(c)(i))$, lalu $H(RGB) = (H(R) + H(G) + H(B)) / 3$.
    - **Nilai maksimum (8‑bit)**:
      Entropi per kanal berada di rentang 0–8 (karena $2^8 = 256$ kemungkinan nilai piksel), dengan maksimum 8 dicapai apabila distribusi piksel sangat merata (uniform). Tiga kanal dirata‑ratakan, sehingga maksimum H(RGB) juga 8.
    - **Rentang threshold**: 0–8.

Dengan demikian, jika gambar 8‑bit per kanal dan kita mengikuti definisi rumus Tabel 1 di naskah, rentang ambang batas yang lazim digunakan umumnya adalah:

- **Variance**: 0 sampai ±16.000‑an,
- **MAD**: 0 sampai ±128,
- **Max Pixel Difference**: 0 sampai 255,
- **Entropy**: 0 sampai 8.

Berikutnya akan dihitung kompleksitas dari *QuadTree compression* menggunakan *divide and conquer.* Algoritma utama adalah buildQuadTree() yang membangun QuadTree dari gambar yang dikompresi. Fungsi ini membagi gambar menjadi 4 bagian dan setiap bagian akan dibagi lagi menjadi 4 sub-bagian jika diperlukan secara rekursif.

Kita bisa menghitung kompleksitas algoritma buildQuadTree() terhadap banyaknya pixel pada gambar tersebut atau memandangnya sebagai panjang kali lebar (W×H). Untuk setiap level panggilan rekursi dilakukan aksi berikut:

- Membuat 1 node baru (node saat ini) -> O(1)
- Menghitung average color dengan calculateAverageColor() -> O(*width*×*height*) = O(N)
- Menghitung error dengan calculateError() -> O(*width*×*height*) = O(N)
- Jika masih perlu membagi maka membuat 4 *recursive call*, masing-masing pada setiap sub bagian.

Sehingga relasi rekurens pada setiap *recursive call*-nya adalah $T(N) = 4T(N/2) + O(N)$ yang dengan *master theorem* didapatkan bahwa $T(N) = O(N \times \log N)$ dengan N banyaknya pixel pada gambar asli atau $T(W,H) = O(W \times H \times \log_2(\max(W,H)))$. Digunakan $\log_2(\max(W,H))$ karena *recursive call* akan terus berlanjut hingga salah satu dari W atau H tidak bisa dibagi lagi.

BAB VI: BONUS

6.1. Targeted Compression

Bonus ini adalah bonus yang bertujuan untuk mencapai nilai target kompresi yang diinginkan dengan menyesuaikan nilai threshold. Untuk implementasi pada program ini, digunakan algoritma *decrease and conquer* yakni menggunakan *decrease by constant factor*. Multithreading juga digunakan pada implementasinya untuk membantu program menemukan solusi dengan lebih cepat. Secara garis besar, algoritma dari pencarian threshold dapat dilihat pada *pseudocode* di bawah ini:

```
procedure FindThresholdForTargetCompression(input
targetPercentage : real, output threshold : real)
{ Mencari nilai threshold untuk mencapai persentase kompresi
target
  Masukan: targetPercentage = persentase kompresi yang
diinginkan (0.0-1.0)
  Luaran: threshold = nilai threshold yang optimal }

Deklarasi:
  lowT, highT, bestThreshold, bestDiff : real
  cache : Map<real, real> { map untuk menyimpan threshold dan
rasio kompresi }
  initialPoints : Larik[real]
  maxIterations : integer { untuk membatasi jumlah iterasi }
  iteration : integer
  tolerance : real { toleransi minimum untuk perbedaan }

Algoritma:
  { Tentukan rentang threshold berdasarkan metode error }
  lowT ← 0.0
  highT ← nilai maksimum berdasarkan metode error

  { Inisialisasi parameter }
  maxIterations ← 25
  tolerance ← 1e-6
  bestDiff ← ∞
  bestThreshold ← lowT
  iteration ← 0

  { Pilih beberapa titik awal untuk pengujian }
  initialPoints ← [lowT, (lowT + highT)/2, highT]

  { Tambahkan lebih banyak titik jika memungkinkan }
```

```
  if numThreads > 3 then
    initialPoints ← initialPoints ∪ [lowT + (highT - lowT)/4,
lowT + 3*(highT - lowT)/4]
  endif

  { Evaluasi titik-titik awal secara paralel }
  for each point in initialPoints do
    ratio ← compressWithThreshold(point)
    cache[point] ← ratio

    { Cek apakah threshold ini yang terbaik sejauh ini }
    if |ratio - targetPercentage| < bestDiff then
      bestDiff ← |ratio - targetPercentage|
      bestThreshold ← point
    endif
  endfor

  { Periksa apakah target dapat dicapai }
  minRatio ← min(semua ratio dalam cache)
  maxRatio ← max(semua ratio dalam cache)

  if targetPercentage < minRatio or targetPercentage > maxRatio
then
    return originalThreshold
  endif

  { Binary search dengan multithreading }
  while iteration < maxIterations and bestDiff > tolerance do
    { Urutkan points berdasarkan threshold }
    points ← sort(cache.keys())

    { Cari interval dimana target berada }
    foundInterval ← false
    leftT ← lowT
    rightT ← highT

    for i ← 0 to points.size()-2 do
      t1 ← points[i]
      r1 ← cache[t1]
      t2 ← points[i+1]
      r2 ← cache[t2]

      { Periksa apakah target berada dalam interval ini }
      if (r1 ≤ r2 and targetPercentage ≥ r1 and
targetPercentage ≤ r2) or
         (r1 ≥ r2 and targetPercentage ≤ r1 and
```

```
targetPercentage ≥ r2) then
        leftT ← t1
        rightT ← t2
        foundInterval ← true
        break
      endif
    endfor

    if not foundInterval then
      { Coba tambahkan titik evaluasi baru }
      newPoints ← empty list
      for i ← 0 to points.size()-2 do
        mid ← (points[i] + points[i+1])/2
        if mid not in cache then
          newPoints ← newPoints ∪ {mid}
        endif
      endfor

      { Batasi jumlah titik baru berdasarkan jumlah thread }
      while newPoints.size() > numThreads do
        { Hapus titik dari interval terkecil }
        newPoints ← newPoints - {titik dari interval terkecil}
      endwhile

      if newPoints.empty() then
        break
      endif

      { Evaluasi titik baru secara paralel }
      for each point in newPoints do
        ratio ← compressWithThreshold(point)
        cache[point] ← ratio

        if |ratio - targetPercentage| < bestDiff then
          bestDiff ← |ratio - targetPercentage|
          bestThreshold ← point
        endif
      endfor
    else
      { Perbarui rentang pencarian }
      lowT ← leftT
      rightT ← rightT

      { Ciptakan titik uji di dalam interval baru }
      testPoints ← empty list
      midT ← (lowT + rightT)/2
```

```
      if midT not in cache then
        testPoints ← testPoints ∪ {midT}
      endif

      { Tambahkan titik-titik tambahan jika interval cukup
besar }
      range ← highT - lowT
      if range > (initialPoints[last] -
initialPoints[first])/100 then
        quarterT ← lowT + range/4
        threeQuarterT ← lowT + 3*range/4

        if quarterT not in cache then
          testPoints ← testPoints ∪ {quarterT}
        endif

        if threeQuarterT not in cache then
          testPoints ← testPoints ∪ {threeQuarterT}
        endif
      endif

      if testPoints.empty() then
        break
      endif

      { Evaluasi titik uji secara paralel }
      for each point in testPoints do
        ratio ← compressWithThreshold(point)
        cache[point] ← ratio

        if |ratio - targetPercentage| < bestDiff then
          bestDiff ← |ratio - targetPercentage|
          bestThreshold ← point
        endif
      endfor
    endif

    iteration ← iteration + 1
  endwhile

  return bestThreshold
```

Pertama, program akan membaca nilai minimum dan maksimum dari threshold berdasarkan metode kalkulasi error yang dipilih. Karena sudah dipastikan hubungan threshold dengan kompresi berbanding lurus (semakin besar threshold maka subdivisi blok menjadi lebih sedikit, menyebabkan kompresi akan semakin besar), maka penulis membagi rentang threshold menjadi 3 (secara default), yakni kiri, tengah, dan kanan. Jika perangkat yang digunakan bisa mendukung lebih banyak thread, maka akan dibentuk lagi beberapa region pada rentang threshold.

Setelah mendapatkan semua region, lakukan perhitungan secara teoritis secara paralel dengan metode kalkulasi error yang dipilih, dan coba buat buffer nyatanya untuk melakukan evaluasi teori dengan hasil aslinya. Dari proses kalkulasi ini, kita mendapatkan suatu *expected compression result* dari tiap region, cek target berada di region yang mana, dan update rentang pencarian sesuai dengan region yang telah dipersempit.

Agar program tidak berjalan lama dalam melakukan evaluasi target, ditetapkan suatu toleransi galat, yakni sebesar 10^{-6} dari target. Selain itu, penulis juga membatasi iterasi hanya sampai 25 kali iterasi agar program tidak berjalan terlalu lama, tetapi tetap mempertimbangkan akurasi dari kompresi. Karena penulis menggunakan algoritma *decrease and conquer*, *worst case* dari hasilnya masih dekat dengan target kompresi.

Jika iterasi sudah mencapai *expected compression result* yang sangat dekat dengan target, program akan mencoba untuk membuat titik uji, yang akan mencoba untuk mendekati nilai sedekat-dekatnya sehingga ditemukan suatu nilai yang cocok. Jika sudah menemukan yang cocok, maka nilai threshold akan dikembalikan ke algoritma kompresi untuk diproses lebih lanjut.

Untuk penggunaan, input dari bonus ini hanyalah suatu nilai desimal, dari rentang 0 sampai 1 (merepresentasikan 0-100%) yang merupakan nilai dari target kompresi yang diinginkan. Tetapi, program terlebih dahulu akan mengecek *expected compression result* dari minimum dan maksimum threshold sesuai dengan metode kalkulasi error yang dipilih. Jika target di luar rentang/region yang akan dibentuk, maka program akan mengeluarkan pesan “tidak mungkin untuk mencapai target kompresi”, dan sebagai alternatif, program akan melakukan kompresi dengan threshold yang telah dipilih oleh user di awal.

6.2. Structural Similarity Index Measure (SSIM)

Structural Similarity Index (SSIM) adalah metrik perseptual yang mengukur penurunan kualitas gambar yang disebabkan oleh pemrosesan, seperti kompresi data atau karena kehilangan dalam transmisi data. Ini adalah metrik referensi lengkap yang memerlukan dua gambar dari pengambilan gambar yang sama - gambar referensi dan gambar yang diproses. Gambar yang diproses biasanya dikompresi, dalam implementasi ini adalah blok yang dihitung rata-rata pixelnya.

.

Pada program ini, kami mengimplementasikan SSIM dengan persamaan:

$$SSIM_c(x,y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

Perhitungannya pun cukup sederhana, pertama adalah perhitungan konstanta, ada dua konstanta yang perlu dihitung, yakni C1 dan C2. Nilai C1 digunakan untuk menghindari ketidakstabilan perhitungan ketika nilai $\mu_{x,c}^2 + \mu y, c^2$ mendekati nol. Untuk setiap konstanta, digunakan persamaan:

$$C_1 = (K_1 L)^2$$

Dan

$$C_2 = (K_2 L)^2$$

Dengan L merupakan nilai dari *dynamic range* pixel, dengan nilai 255 untuk 8-bit gambar grayscale. Selanjutnya, K adalah suatu konstan yang kita pilih dengan nilainya lebih kecil dari 1. Tetapi, karena dari percobaan didapatkan fakta bahwa nilai K ini bersifat insignifikan terhadap hasil perhitungan error, maka kita bebas untuk mendefinisikannya, sehingga kami menetapkan nilai K1 sebesar 0.01 dan K2 sebesar 0.03. Sehingga jika dihitung, nilai C1 adalah sebesar 6.5025, dan nilai C2 sebesar 58.5225.

Setelah mendapatkan nilai C1 dan C2, hitung nilai dari rata-rata dan varians (kovarian) dari blok asli dan blok terkompresi. Untuk blok terkompresi, kami mengambil sample dari blok yang tiap nilai pixelnya (RGB) sudah dinormalisasi dengan rata-rata, atau dapat dilihat pada kode di bawah ini:

```
Pixel SSIMCalculator::calculateAverageColor(const
vector<vector<Pixel>>& pixels, int x, int y, int width, int
```

```
height) {
    int totalR = 0, totalG = 0, totalB = 0;
    int count = width * height;

    for (int j = y; j < y + height; ++j) {
        for (int i = x; i < x + width; ++i) {
            totalR += pixels[j][i].r;
            totalG += pixels[j][i].g;
            totalB += pixels[j][i].b;
        }
    }

    return Pixel(
        static_cast<unsigned char>(totalR / count),
        static_cast<unsigned char>(totalG / count),
        static_cast<unsigned char>(totalB / count)
    );
}
```

Setelah mendapatkan sample dari blok terkompresi, hitung nilai rata-rata dan variansnya (kovariannya), dengan x merepresentasikan blok asli dan y merepresentasikan blok terkompresi, untuk implementasinya dapat dilihat pada kode di bawah ini:

```
double SSIMCalculator::calculateSSIMForChannel(const
vector<vector<Pixel>>& originalBlock, const
vector<vector<Pixel>>& compressedBlock, int x, int y, int
width, int height, int channel){
    const double L = 255.0;
    const double K1 = 0.01;
    const double K2 = 0.03;
    const double C1 = (K1 * L) * (K1 * L);
    const double C2 = (K2 * L) * (K2 * L);

    int N = width * height;
    if (N < 1) {
        return 1.0;
    }

    // mean for original and compressed blocks
    double sum_x = 0.0, sum_y = 0.0;
    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            sum_x += getChannelValue(originalBlock[y+j][x+i],
channel);
            sum_y += getChannelValue(compressedBlock[j][i],
```

```
channel);
        }
    }

    double mu_x = sum_x / N;
    double mu_y = sum_y / N;

    // variance and covar
    double var_x = 0.0, var_y = 0.0, covar_xy = 0.0;

    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            double x_val =
getChannelValue(originalBlock[y+j][x+i], channel);
            double y_val =
getChannelValue(compressedBlock[j][i], channel);

            double diff_x = x_val - mu_x;
            double diff_y = y_val - mu_y;

            var_x += diff_x * diff_x;
            var_y += diff_y * diff_y;
            covar_xy += diff_x * diff_y;
        }
    }

    if (N > 1) {
        var_x /= (N - 1);
        var_y /= (N - 1);
        covar_xy /= (N - 1);
    } else {
        var_x = var_y = covar_xy = 0.0;
    }

    //    SSIM(x,y) = ((2μxμy + C1)(2σxy + C2)) / ((μx² + μy² +
C1)(σx² + σy² + C2))
    double numerator = (2.0 * mu_x * mu_y + C1) * (2.0 *
covar_xy + C2);
    double denominator = (mu_x * mu_x + mu_y * mu_y + C1) *
(var_x + var_y + C2);

    if (denominator < 1e-10) {
        return 1.0;
    }

    double ssim = numerator / denominator;
```

```
    return ssim;
}
```

Dengan menghitung nilai rata-rata dan varians(kovariannya), kita dapat menghitung nilai SSIMnya. Setelah mendapatkan nilai SSIM, selanjutnya adalah menghitung error, dengan persamaan berikut:

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Pada persamaan di atas, dilakukan pembobotan nilai SSIM pada tiap kanal warna. Pada kode ini, pembobotan yang dilakukan adalah setara, yakni R sebesar ⅓, G sebesar ⅓ , dan B sebesar ⅓. Nilai ini dipilih untuk memastikan semua warna pada gambar dapat di-*handle* dengan baik dan mengurangi potensi adanya penurunan kualitas gambar.

Setelah mendapatkan nilai SSIM, selanjutnya adalah mengurangi nilai tersebut dengan 1, hal ini bertujuan untuk memastikan konsistensi dan definisi penulis bahwa semakin besar threshold maka kualitas gambar akan menurun (nilai kompresinya semakin besar). Hal ini juga dilakukan secara sengaja untuk memudahkan dalam perhitungan teoritis pada bonus *targeted compression* yang dilakukan dengan pendekatan *decrease and conquer*.

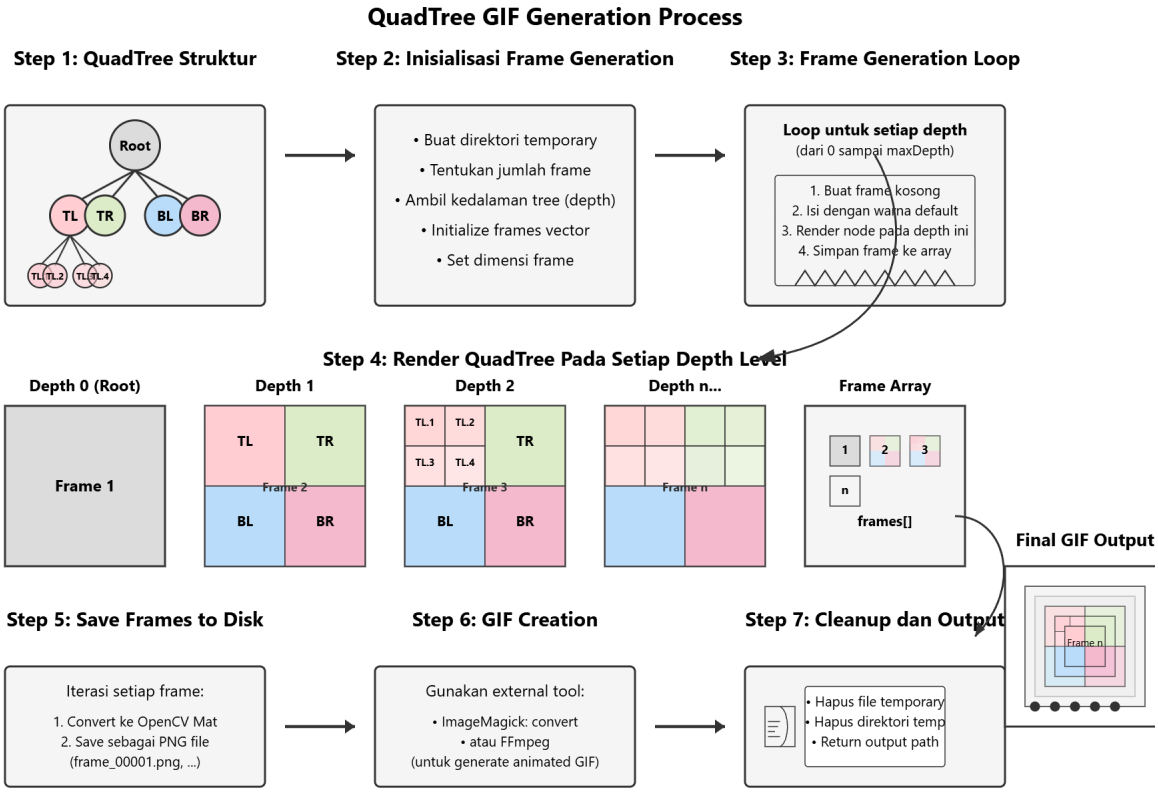
Terakhir, adalah penentuan rentang threshold. Karena SSIM adalah konsep untuk mencari kemiripan, maka parameter yang kami gunakan adalah parameter (0-100)%, yang direpresentasikan dengan nilai 0-1 dan bernilai desimal.

6.3. Graphics Interchange Format (GIF)

Pada program ini, kami mengimplementasikan GIF dengan menggunakan bantuan 3 library, yakni OpenCV, FFmpeg, dan ImageMagick, langkah untuk membuatnya pun sederhana, dengan langkah sebagai berikut:

1. Ambil QuadTree hasil dari kompresi.
2. Tentukan jumlah frame dari GIF, dan ambil data panjang dan lebar gambar untuk menjadi panjang dan lebar GIF.
3. Ambil data kedalaman (*depth*) dari QuadTree untuk menentukan jumlah *frame* dari GIF.
4. Untuk setiap kedalaman, gambarlah representasi dari gambarnya dengan menggunakan data *node* dan *root* dari QuadTree.
5. Simpan hasil gambar pada buffer sementara.
6. Gabungkan semua frame dalam bentuk gambar secara berurutan dan bertahap, definisikan juga delay per framenya.
7. Bentuk menjadi format GIF.
8. Bersihkan semua buffer sementara.

Untuk memudahkan dalam memahami langkah, penulis juga menyediakan visualisasi yang dapat dilihat sebagai berikut.



Gambar 62. Visualisasi GIF Generation

LAMPIRAN

Github Repository
Program dapat diakses pada https://github.com/Nayekah/Tucil2_13523027_13523090

Miscellaneous (Tabel Poin)

Tabel 2. Poin yang dikerjakan

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4	Mengimplementasi seluruh metode perhitungan error wajib	✓	
5	**[Bonus]** Implementasi persentase kompresi sebagai parameter tambahan	✓	
6	**[Bonus]** Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7	**[Bonus]** Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	

DAFTAR PUSTAKA

Geeksforgeeks. Introduction to Divide and Conquer Algorithm– Data Structure and Algorithm Tutorials. Diakses pada 9 April 2025 dari https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm-data-structure-and-algorithm-tutorials/

Rinaldi Munir. Algoritma Divide and Conquer (Bagian 1). Diakses pada 9 April 2025 dari https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf

Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861. keywords: {Image quality;Humans;Transform coding;Visual system;Visual perception;Data mining;Layout;Quality assessment;Degradation;Indexes}

AKHIR KATA

Sekian, *deliverables* dari kami, semoga AC, dan semangat kak asisten :3!

…

“I told one single lie…”

https://youtu.be/jhpfRNCpb5c?si=3YzHbHrZBqfkFymL

-> Nayaka.

“Merindink”

“Choco Minto!”

-> Fajar.
