

TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Pemanfaatan Algoritma *Divide and Conquer* dalam Kompresi
Gambar dengan Metode Quadtree



Disusun Oleh:
Aloisius Adrian Stevan Gunawan 13523054
Albertus Christian Poandy 13523077

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2025

Daftar Isi

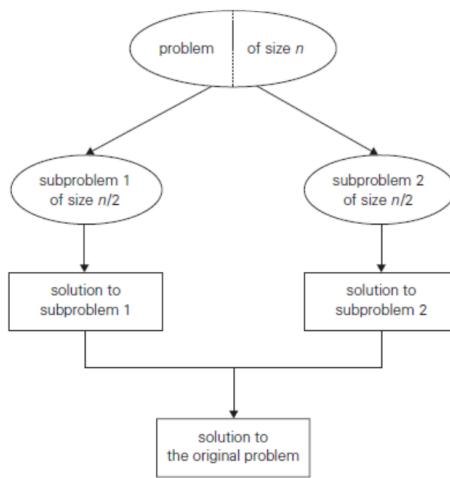
Daftar Isi	2
BAB I	3
DESKRIPSI MASALAH DAN ALGORITMA	3
1.1 Algoritma Divide and Conquer	3
1.2 Kompresi Gambar dengan Metode Quadtree	3
1.3 Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree	4
BAB II	6
SOURCE CODE	6
2.1 Struktur program dengan paradigma OOP	6
2.2 Kelas RGB	6
2.3 Kelas QuadTree	7
BAB III	17
PENGUJIAN	17
3.1 Contoh dengan metode error Variansi	17
3.2 Contoh dengan metode error Variansi 2	19
3.3 Contoh dengan metode error Mean Absolute Deviation (MAD)	21
3.4 Contoh dengan metode error Mean Absolute Deviation (MAD) 2	23
3.5 Contoh dengan metode error Max Pixel Difference	26
3.6 Contoh dengan metode error Max Pixel Difference 2	28
3.7 Contoh dengan metode error Entropy	30
3.8 Contoh dengan metode error Entropy 2	32
3.9 [Bonus] Contoh dengan metode error SSIM	34
3.10 [Bonus] Contoh dengan metode error SSIM 2	37
3.11 [Bonus] Contoh hasil gif	40
3.12 [Bonus] Contoh hasil gif 2	43
3.13 [Bonus] Contoh target compression	45
3.14 [Bonus] Contoh target compression	49
BAB IV	52
ANALISIS	52
4.1 Kompleksitas Algoritma Divide and Conquer	52
4.2 Analisis Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree	53
BAB V	53
BONUS	54
5.1 Target Persentase Kompresi	54
5.2 Structural Similarity Index (SSIM)	54
5.3 GIF	55
LAMPIRAN	57
DAFTAR PUSTAKA	58

BAB I

DESKRIPSI MASALAH DAN ALGORITMA

1.1 Algoritma Divide and Conquer

Algoritma *Divide and Conquer* adalah algoritma yang terdiri dari tiga bagian utama, yaitu *Divide*, *Conquer*, dan *Combine*. *Divide* adalah proses membagi persoalan menjadi beberapa upa-persoalan (*sub-problem*) yang memiliki kemiripan dengan persoalan semula, tetapi memiliki ukuran yang lebih kecil. *Conquer* adalah proses menyelesaikan setiap upa-persoalan yang telah dibagi sebelumnya, di mana proses penyelesaian ini dapat diselesaikan secara langsung jika telah berukuran kecil ataupun secara rekursif jika masih berukuran besar. *Combine* adalah proses menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi dari persoalan semula.



Gambar 1. Ilustrasi Algoritma Divide and Conquer

Algoritma ini sangat cocok digunakan untuk masalah yang dapat direduksi menjadi versi-versi yang lebih kecil tetapi memiliki struktur yang serupa, misalnya persoalan pengurutan (algoritma *merge sort*, *quicksort*), pencarian nilai minimum-maksimum, perpangkatan, ataupun persoalan geometri seperti *closest pair* dan *convex hull*. Karena proses pembagian dan penyelesaian dilakukan secara rekursif, algoritma ini juga umumnya dinyatakan dalam bentuk rekursif. Keunggulan dari pendekatan ini adalah efisiensi waktu yang umumnya lebih baik dibandingkan metode *brute force*, terutama untuk input yang besar.

1.2 Kompresi Gambar dengan Metode *Quadtree*

Quadtree adalah struktur data hierarkis berbentuk pohon di mana setiap simpulnya memiliki empat anak (atau nol anak jika merupakan simpul daun). Struktur ini digunakan untuk membagi ruang

dua dimensi secara rekursif menjadi empat bagian kuadran yang lebih kecil dengan ukuran yang semirip mungkin. Setiap pembagian dilakukan berdasarkan kriteria tertentu, dan struktur ini sangat cocok digunakan untuk merepresentasikan data spasial seperti citra. Quadtree bekerja dengan memetakan area dua dimensi ke dalam blok-blok yang bisa dianalisis dan diproses secara efisien.

Dalam pengolahan gambar, quadtree dapat digunakan untuk melakukan kompresi gambar, yaitu dengan membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian. Jika terdapat bagian yang memiliki tingkat keseragaman warna terlalu rendah, bagian tersebut akan dibagi lagi menjadi empat bagian yang lebih kecil. Proses ini berlanjut secara rekursif hingga seluruh bagian gambar cukup seragam atau tidak bisa dibagi lebih kecil lagi. Blok-blok yang sudah tidak dibagi lagi direpresentasikan sebagai simpul daun dalam quadtree. Setiap area gambar yang merupakan simpul daun pada quadtree akan digantikan menggunakan satu nilai warna saja, yaitu nilai rata-rata warna pada 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.

1.3 Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree

Paradigma *divide and conquer* merupakan kunci dalam melakukan *image compression* dengan metode *quadtree*. Algoritma Divide and Conquer dalam kompresi gambar dengan metode quadtree bekerja dengan cara membagi gambar menjadi bagian-bagian yang lebih kecil secara rekursif, berdasarkan tingkat keseragaman warna dalam blok piksel. Proses diawali dengan membagi gambar menjadi empat kuadran. Untuk setiap kuadran, dihitung nilai error atau ketidakseuniforman menggunakan metode yang dipilih (seperti *Variance*, *MAD*, *Max Pixel Difference*, *Entropy*, dan *SSIM*). Jika nilai error tersebut lebih besar dari ambang batas (*threshold*), dan ukuran blok masih lebih besar dari ukuran minimum yang diperbolehkan, maka blok tersebut dibagi lagi menjadi empat bagian. Proses ini diulang secara rekursif untuk setiap sub-blok hingga setiap blok yang ada tidak dapat dibagi menjadi lebih kecil lagi. Ini adalah tahap *Divide* dalam algoritma Divide and Conquer.

Setelah blok-blok tidak bisa dibagi lagi, baik karena sudah cukup seragam atau ukuran minimumnya tercapai, blok tersebut dianggap sebagai simpul daun dan disimpan bersama nilai warna rata-ratanya. Tahapan ini mencerminkan langkah Conquer dalam algoritma. Terakhir, untuk membentuk kembali gambar hasil kompresi, semua simpul daun digunakan untuk mengisi kembali blok-blok gambar dengan warna rata-rata masing-masing. Proses ini mencerminkan tahap *Combine*, di mana struktur quadtree disatukan kembali menjadi gambar baru yang telah dikompresi. Dengan cara ini, algoritma divide and conquer memungkinkan pengurangan ukuran gambar secara signifikan sambil tetap mempertahankan bentuk visual secara keseluruhan. Lebih detailnya, algoritma Divide and Conquer pada kompresi gambar dengan QuadTree adalah sebagai berikut:

Prosedur KompresiQuadTree(Gambar, x, y, width, height)

Algoritma:

1. Tinjauan kondisi
Hitung nilai *error* blok pada Gambar dengan posisi kiri-atas di (**x**, **y**) dan dengan ukuran **width** × **height** menggunakan metode perhitungan yang dipilih

2. Basis (kasus blok kecil atau warna seragam)

Untuk kasus $\text{error} < \text{threshold}$ atau $(\text{ukuran blok}/2) < \text{ukuran minimum}$, maka:

- Hitung rata-rata warna nilai RGB pada blok
- Tandai blok sebagai simpul daun
- Lakukan normalisasi warna blok sesuai dengan rata-rata nilai RGB blok

3. Rekursi (kasus blok tidak seragam dan masih bisa dibagi)

Untuk kasus $\text{error} \geq \text{threshold}$ dan $(\text{ukuran blok}/2) \geq \text{ukuran minimum}$, maka:

a. DIVIDE

Bagi blok menjadi empat sub-blok berukuran sama, dengan posisi kiri atas sebagai berikut:

- Kuadran 1: (x, y)
- Kuadran 2: $(x + \text{width}/2, y)$
- Kuadran 3: $(x, y + \text{height}/2)$
- Kuadran 4: $(x + \text{width}/2, y + \text{height}/2)$

b. CONQUER

Panggil prosedur `KompresiQuadtree` secara rekursif pada setiap kuadran:

```
KompresiQuadtree(Gambar, x, y, width/2, height/2)
KompresiQuadtree(Gambar, x + width/2, y, width/2, height/2)
KompresiQuadtree(Gambar, x, y + height/2, width/2, height/2)
KompresiQuadtree(Gambar, x + width/2, y + height/2, width/2, height/2)
```

c. COMBINE

Struktur pohon quadtree akan terbentuk secara otomatis dari pemanggilan rekursif, sehingga gambar hasil kompresi secara otomatis berhasil dibentuk dari proses normalisasi simpul daun.

BAB II

SOURCE CODE

2.1 Struktur program dengan paradigma OOP

Implementasi dari program ini dilakukan dengan menerapkan prinsip OOP untuk memudahkan modifikasi dan keterbacaan kode. Terdapat juga beberapa fungsi pembantu di luar objek, yang digunakan untuk memudahkan penggeraan dan keterbacaan kode.

Terdapat dua kelas yang akan digunakan dalam implementasi ini, yaitu kelas RGB yang berisi nilai dari sebuah pixel, dan kelas QuadTree yang bisa merepresentasikan sebuah simpul pada tree.

Pada setiap simpul, akan terdapat informasi posisi blok, warna pada blok tersebut (jika merupakan blok yang tidak dapat dipisah lagi), dan referensi ke keempat simpul anak, apabila blok ini masih dapat dipisah. Dengan desain ini, solusi *Divide and Conquer* dapat diimplementasikan secara rekursif dengan lebih efisien.

2.2 Kelas RGB

2.2.1 Struktur kelas RGB

```
class RGB
{
private:
public:
    int red;
    int green;
    int blue;
    int alpha;
    RGB();
    RGB(int r, int g, int b);
    RGB(int r, int g, int b, int a);
    RGB(const RGB& other);
    friend RGB min(RGB a, RGB b);
    friend RGB max(RGB a, RGB b);
    RGB& operator=(const RGB& other);
    RGB& operator+=(const RGB& other);
    RGB& operator/=(const int x);
    ~RGB();
};
```

2.3 Kelas QuadTree

2.3.1 Struktur kelas QuadTree

```
class QuadTree
{
private:
    double error;
    QuadTree* topLeftChild;
    QuadTree* topRightChild;
    QuadTree* bottomLeftChild;
    QuadTree* bottomRightChild;
    int startHeight, startWidth, currentHeight, currentWidth;
    bool isSmallest;

public:
    static double threshold;
    static RGB* block;
    static int height, width, numNodes;
    static int errorChoice, minimumBlockSize;
    static std::vector<std::vector<QuadTree*>> nodesAtDepth;

    QuadTree();
    QuadTree(int currentH, int currentW, int startH, int startW);
    ~QuadTree();

    static void setValue(int h, int w, RGB value);

    void setValue(int h, int w, RGB value, RGB* Block, unsigned char* imageData, bool gif);

    static RGB getValue(int h, int w);

    RGB getValue(int h, int w, RGB* Block);

    RGB getMean(RGB* Block);
    RGB getMin(RGB* Block);
    RGB getMax(RGB* Block);

    double variance(RGB* Block);
```

```

    double meanAbsoluteDeviation(RGB* Block);
    double maxPixelDifference(RGB* Block);
    double entropy(RGB* Block);
    double structuralSimilarityIndex(RGB* Block); // BONUS
    double getError(RGB* Block);

    void divConq();
    void divConq(double currentThreshold, RGB* referenceBlock);
    int compressImage(string exportPath, RGB* image, double
targetCompression, int originalFileSize);

    void colorNormalization();
    void colorNormalization(RGB* referenceBlock, RGB* Block, unsigned char*
ImageData, bool gif = false);

    int getDepth();

    void buildNodesAtDepth(int depth = 0);

    void generateGIF(RGB* image, std::string outputPath);

    static RGB* copyBlock();

    static void copyBlock(RGB* &Block);

};


```

2.3.2 Implementasi *error measurement*

Terdapat 5 metode dalam memperkirakan nilai error:

1. Variance

```

double QuadTree::variance(RGB* Block)
{
    double variance = 0;
    RGB mean = getMean(Block);
    for (int i = startHeight; i < startHeight + currentHeight; i++)
    {
        for (int j = startWidth; j < startWidth + currentWidth; j++)
        {
            variance += pow(getValue(i, j, Block).red - mean.red, 2);
            variance += pow(getValue(i, j, Block).green - mean.green,
2);
        }
    }
}


```

```

        variance += pow(getValue(i, j, Block).blue - mean.blue,
2);
    }
}
int total_pixel = currentHeight * currentWidth;
variance /= (double) total_pixel;
variance /= (double) 3;

return variance;
}

```

2. Mean Absolute Deviation

```

double QuadTree::meanAbsoluteDeviation(RGB* Block)
{
    double mean_absolute_deviation = 0;
    RGB mean = getMean(Block);
    for (int i = startHeight; i < startHeight + currentHeight; i++)
    {
        for (int j = startWidth; j < startWidth + currentWidth; j++)
        {
            mean_absolute_deviation += abs(getValue(i, j, Block).red
- mean.red);
            mean_absolute_deviation += abs(getValue(i, j,
Block).green - mean.green);
            mean_absolute_deviation += abs(getValue(i, j, Block).blue
- mean.blue);
        }
    }
    int total_pixel = currentHeight * currentWidth;
    mean_absolute_deviation /= total_pixel;
    mean_absolute_deviation /= 3;
    return mean_absolute_deviation;
}

```

3. Max Pixel Difference

```

double QuadTree::maxPixelDifference(RGB* Block)
{
    double max_pixel_difference = 0;
    RGB min = getMin(Block);
    RGB max = getMax(Block);
    max_pixel_difference += abs(max.red - min.red); //abs disini ga

```

```

wajib si
    max_pixel_difference += abs(max.green - min.green);
    max_pixel_difference += abs(max.blue - min.blue);
    max_pixel_difference /= 3;
    return max_pixel_difference;
}

```

4. Entropy

```

double QuadTree::entropy(RGB* Block)
{
    vector<int> redFrequency(256, 0);
    vector<int> greenFrequency(256, 0);
    vector<int> blueFrequency(256, 0);

    double entropy = 0.0;
    int total_pixel = currentHeight * currentWidth;
    // count frequency
    for (int i = startHeight; i < startHeight + currentHeight; i++)
    {
        for (int j = startWidth; j < startWidth + currentWidth; j++)
        {
            redFrequency[getValue(i, j, Block).red]++;
            greenFrequency[getValue(i, j, Block).green]++;
            blueFrequency[getValue(i, j, Block).blue]++;
        }
    }
    // calc entropy
    for (int i = 0; i < 256; i++)
    {
        if (redFrequency[i] > 0)
        {
            double p = static_cast<double>(redFrequency[i]) /
total_pixel;
            entropy -= p * log2(p);
        }
        if (greenFrequency[i] > 0)
        {
            double p = static_cast<double>(greenFrequency[i]) /
total_pixel;
            entropy -= p * log2(p);
        }
        if (blueFrequency[i] > 0)

```

```

    {
        double p = static_cast<double>(blueFrequency[i]) /
total_pixel;
        entropy -= p * log2(p);
    }
}
entropy /= 3;
return entropy;
}

```

5. Structural Similarity Index (SSIM) [Bonus]

```

double QuadTree::structuralSimilarityIndex(RGB* Block)
{
    double structural_similarity_index = 0;
    RGB mean = getMean(Block);
    int total_pixel = currentHeight * currentWidth;
    double sum_x_2_red = 0, sum_x_2_blue = 0, sum_x_2_green = 0;

    for (int i = startHeight; i < startHeight + currentHeight; i++)
    {
        for (int j = startWidth; j < startWidth + currentWidth; j++)
        {
            sum_x_2_red += pow(getValue(i, j, Block).red, 2);

            sum_x_2_green += pow(getValue(i, j, Block).green, 2);

            sum_x_2_blue += pow(getValue(i, j, Block).blue, 2);
        }
    }

    double var_x_red = sum_x_2_red / total_pixel - mean.red *
mean.red;
    double var_x_green = sum_x_2_green / total_pixel - mean.green *
mean.green;
    double var_x_blue = sum_x_2_blue / total_pixel - mean.blue *
mean.blue;
    double c2 = 58.5225;

    double n_red = c2;
    double d_red = (var_x_red + c2);

    double Wred = 0.2989, Wgreen = 0.5870, Wblue = 0.1140;

```

```

if(d_red == 0)
{
    structural_similarity_index += 1 * Wred;
}
else
{
    structural_similarity_index += (n_red / d_red) * Wred;
}

double n_green = c2;
double d_green = (var_x_green + c2);
structural_similarity_index += n_green / d_green;
if(d_green == 0)
{
    structural_similarity_index += 1 * Wgreen;
}
else
{
    structural_similarity_index += (n_green / d_green) * Wgreen;
}

double n_blue = c2;
double d_blue = (var_x_blue + c2);
structural_similarity_index += n_blue / d_blue;
if(d_blue == 0)
{
    structural_similarity_index += 1 * Wblue;
}
else
{
    structural_similarity_index += (n_blue / d_blue) * Wblue;
}

return structural_similarity_index;
}

```

2.3.3 Implementasi Rekursif

Berikut adalah metode yang digunakan untuk proses rekursif:

1. Divide and Conquer ketika target kompresi tidak ada

```

void QuadTree::divConq()
{

```

```

        double Error = getError(block);
        if(Error >= threshold && currentHeight/2 >= minimumBlockSize &&
currentWidth/2 >= minimumBlockSize)
        {
            isSmallest = false;
            //bagi block jadi 4
            int halfHeight = currentHeight / 2;
            int halfWidth = currentWidth / 2;
            int remainingHeight = currentHeight - halfHeight;
            int remainingWidth = currentWidth - halfWidth;

            topLeftChild = new QuadTree(halfHeight, halfWidth,
startHeight, startWidth);
            topRightChild = new QuadTree(halfHeight, remainingWidth,
startHeight, startWidth + halfWidth);
            bottomLeftChild = new QuadTree(remainingHeight, halfWidth,
startHeight + halfHeight, startWidth);
            bottomRightChild = new QuadTree(remainingHeight,
remainingWidth, startHeight + halfHeight, startWidth + halfWidth);

            topLeftChild->divConq();
            topRightChild->divConq();
            bottomLeftChild->divConq();
            bottomRightChild->divConq();
        }
        else colorNormalization();
    }
}

```

2. Divide and Conquer ketika ada target kompresi [Bonus]

```

void QuadTree::divConq(double currentThreshold, RGB* referenceBlock){
    double Error = getError(referenceBlock);
    if(Error >= currentThreshold && currentHeight/2 >=
minimumBlockSize && currentWidth/2 >= minimumBlockSize)
    {
        if(isSmallest){
            isSmallest = false;
        }
        //bagi block jadi 4
        int halfHeight = currentHeight / 2;
        int halfWidth = currentWidth / 2;
        int remainingHeight = currentHeight - halfHeight;
        int remainingWidth = currentWidth - halfWidth;
    }
}

```

```

        if (topLeftChild == nullptr)
            topLeftChild = new QuadTree(halfHeight, halfWidth,
startHeight, startWidth);
        if (topRightChild == nullptr)
            topRightChild = new QuadTree(halfHeight, remainingWidth,
startHeight, startWidth + halfWidth);
        if (bottomLeftChild == nullptr)
            bottomLeftChild = new QuadTree(remainingHeight,
halfWidth, startHeight + halfHeight, startWidth);
        if (bottomRightChild == nullptr)
            bottomRightChild = new QuadTree(remainingHeight,
remainingWidth, startHeight + halfHeight, startWidth + halfWidth);

        topLeftChild->divConq(currentThreshold, referenceBlock);
        topRightChild->divConq(currentThreshold, referenceBlock);
        bottomLeftChild->divConq(currentThreshold, referenceBlock);
        bottomRightChild->divConq(currentThreshold, referenceBlock);
    }
    else {
        isSmallest = true;
        colorNormalization(referenceBlock, block, imageData);
    }
}

```

3. Implementasi metode compressImage()

```

int QuadTree::compressImage(string exportPath, RGB* image, double
targetCompression, int originalFileSize)
{
    if (targetCompression == 0) {
        divConq();
        return exportImage(exportPath, block, width, height);
    }
    double percentageTarget = targetCompression;
    targetCompression = originalFileSize -
targetCompression*originalFileSize;
    int currNodeNum = 0;
    double l = 0, r = getError(block);
    int fileSize;

    while (r - l > 0.001) {
        threshold = (r+l)/2;

```

```

        divConq(threshold, image);

        fileSize = exportImage(exportPath, block, width, height);
        double percentage = (double) (originalFileSize - fileSize) / originalFileSize;
        // cout << threshold << ' ' << currNodeNum << ' ' << numNodes
        // << ' ' << targetCompression << ' ' << fileSize << ' ' << percentage
        // << endl;
        if (abs(percentage - percentageTarget) < 0.001) {
            break;
        }
        if (fileSize > targetCompression){
            l = threshold;
        }
        else{
            r = threshold;
        }
    }

    cout << "✓ Gambar berhasil dieksport ke: "<< exportPath << endl;
    return fileSize;
}

```

2.3.4 Implementasi GIF

Pada program ini, file gif.h digunakan untuk mendukung proses penggerjaan dalam menyimpan gambar berupa GIF. Library ini dikembangkan oleh Charlie Tangora, dan terbuka untuk publik. Berikut adalah implementasi kode menggunakan library gif.h:

```

void QuadTree::generateGIF(RGB* image, string outputPath){
    RGB* gifImage = new RGB[width * height];
    unsigned char* gifImageData = new unsigned char[width * height * 4];
    for (int i = 0; i < width * height * 4; i += 4){
        gifImageData[i] = 255;
    }

    GifWriter gif;
    GifBegin(&gif, outputPath.c_str(), width, height, 50);

    for (int i = 0; i < nodesAtDepth.size(); i++){
        for (QuadTree* qt: nodesAtDepth[i]){
            qt->colorNormalization(image, gifImage, gifImageData,

```

```
    true);
}
GifWriteFrame(&gif, gifImageData, width, height, 50);
}

GifEnd(&gif);

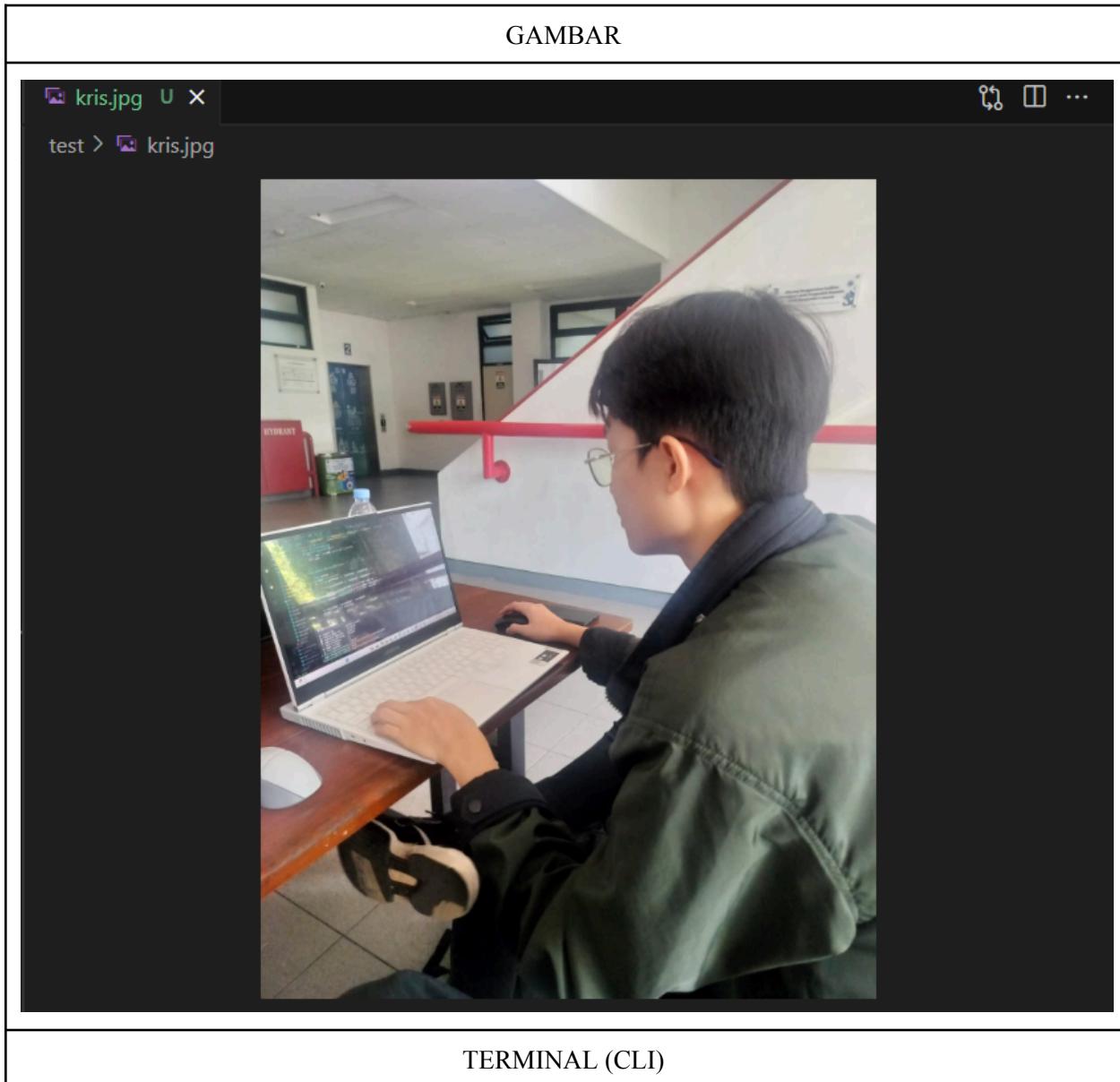
cout << "✓ GIF berhasil digenerate ke: "<< outputPath << endl;
delete[] gifImage;
delete[] gifImageData;
}
```

BAB III

PENGUJIAN

3.1 Contoh dengan metode error Variansi

-INPUT:



Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/kris.jpg
 Gambar berhasil dimuat!

Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 1

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 75

Masukkan ukuran blok minimum: 10

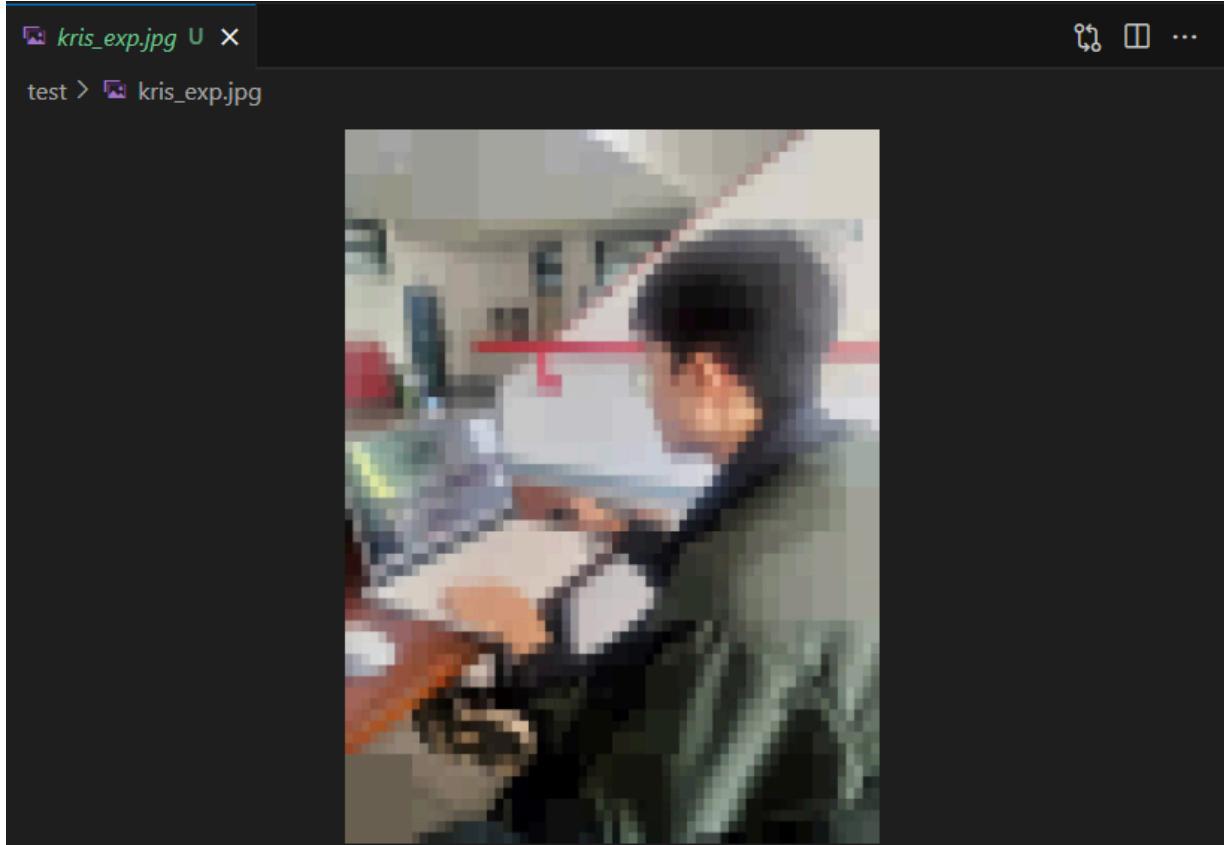
Export

Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/kris_exp.jpg

Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/kris_exp.gif

-OUTPUT:

GAMBAR



The image shows a highly compressed version of a photograph of a person's face, appearing very pixelated and distorted.

TERMINAL (CLI)

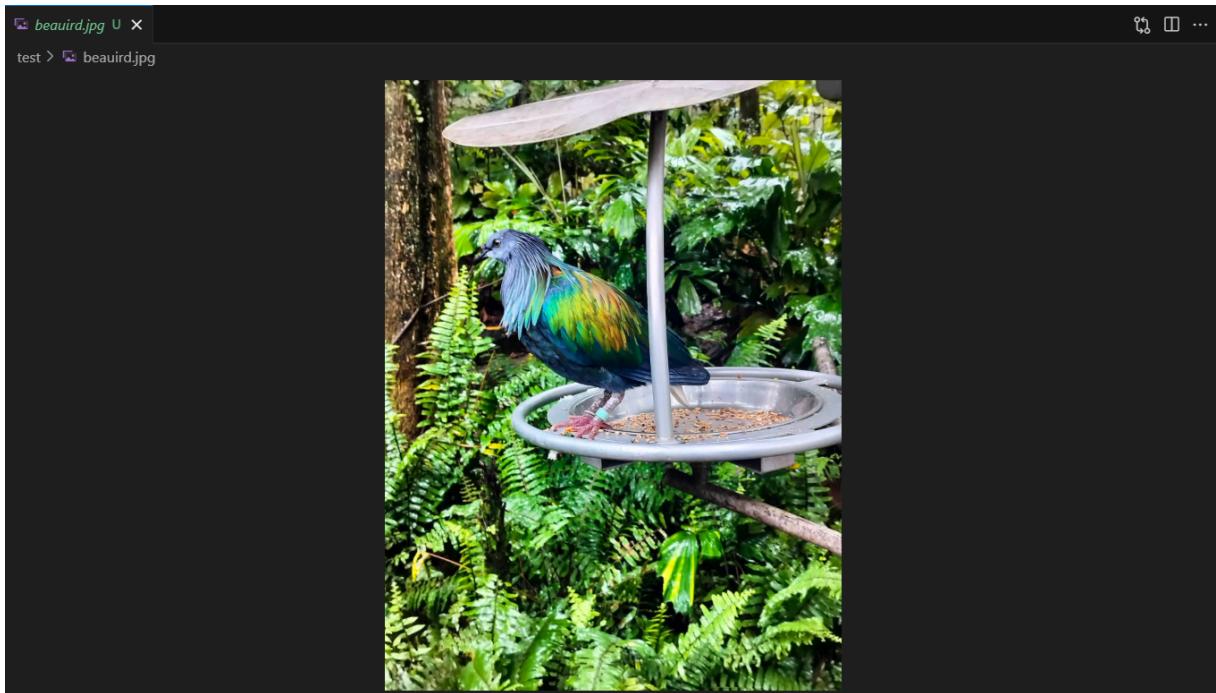
```
* Compressing Image...
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/kris_exp.gif
✓ Compression completed!

* Compression Summary
-----
⌚ Elapsed time      : 1.63056 seconds
💾 Original image size : 130.663 KB
💾 Compressed image size : 101.139 KB
📦 Compression percentage : 22.5955%
🌳 Depth of quadtree   : 6
🔢 Number of nodes     : 3709
```

3.2 Contoh dengan metode error Variansi 2

-INPUT:

GAMBAR



TERMINAL (CLI)

📁 Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/beaurid.jpg
 Gambar berhasil dimuat!

⚙️ Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 1

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 2000

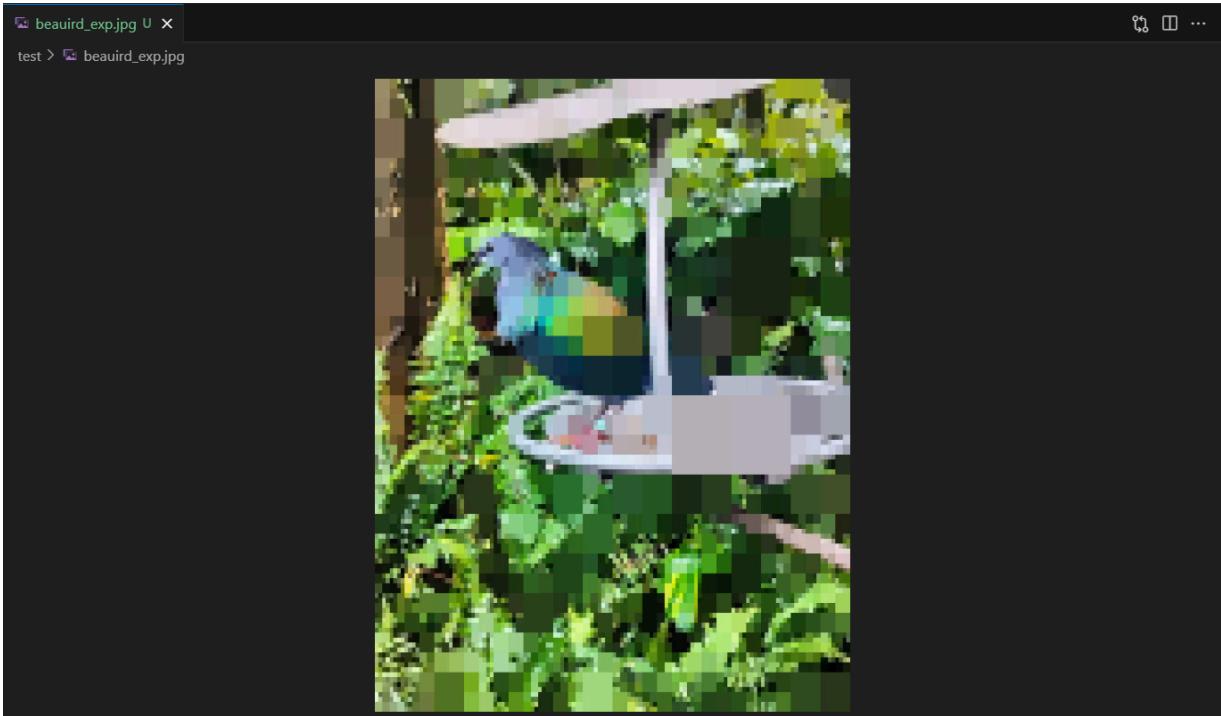
Masukkan ukuran blok minimum: 6

💾 Export

Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/beaurid_exp.jpg
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/beaurid_exp.gif

-OUTPUT:

GAMBAR



TERMINAL (CLI)

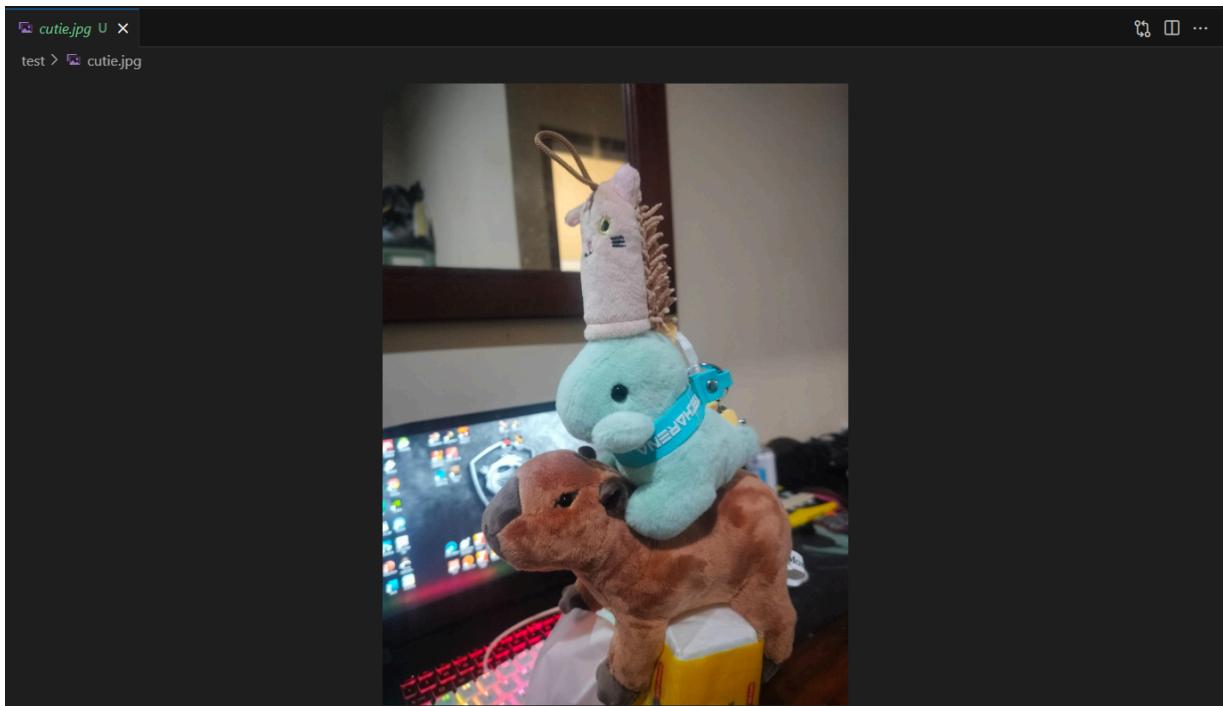
```
🛠️ Compressing Image...
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/beauird_exp.gif
✓ Compression completed!

📊 Compression Summary
-----
⌚ Elapsed time      : 1.34923 seconds
🕒 Original image size : 247.894 KB
🖼️ Compressed image size : 124.872 KB
📦 Compression percentage : 49.6269%
🌳 Depth of quadtree    : 7
🔢 Number of nodes     : 8205
```

3.3 Contoh dengan metode error Mean Absolute Deviation (MAD)

-INPUT:

GAMBAR



TERMINAL (CLI)

```
📁 Input
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/cutie.jpg
 Gambar berhasil dimuat!

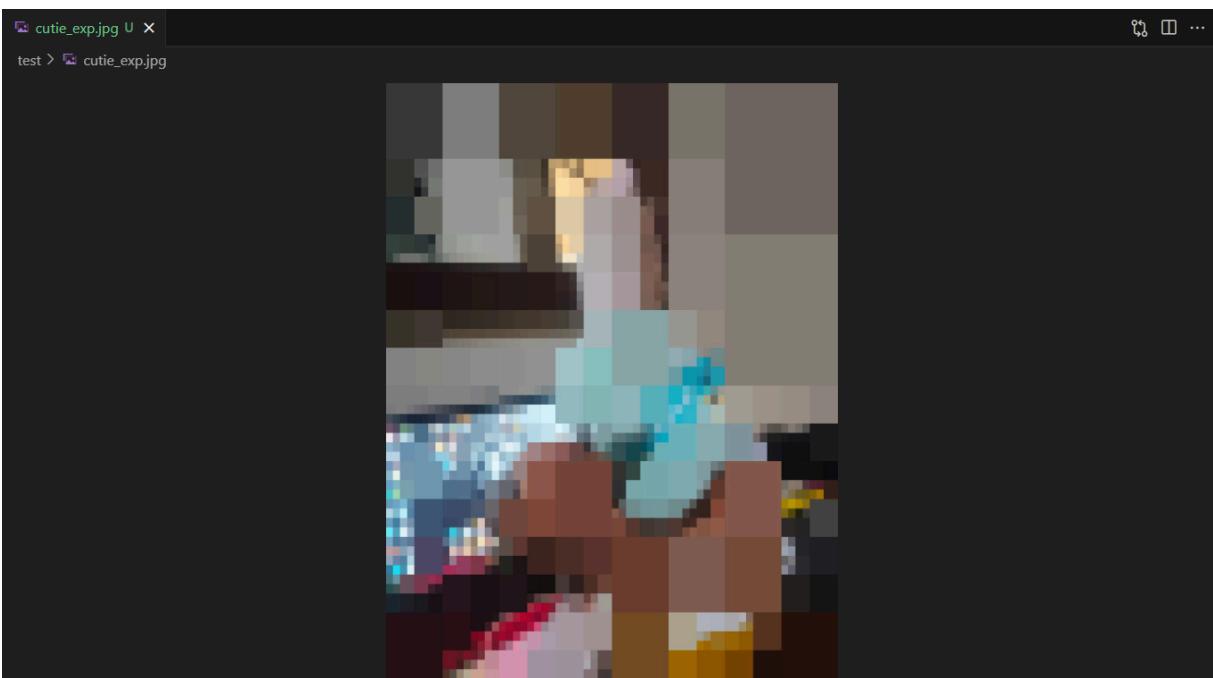
⚙️ Compression Settings
Pilih metode error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
Masukkan pilihan [1-5]: 2

Masukkan target kompresi [0.0-1.0]: 0
Masukkan ambang batas error: 25
Masukkan ukuran blok minimum: 10

💾 Export
Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/cutie_exp.jpg
Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/cutie_exp.gif
```

-OUTPUT:

GAMBAR



TERMINAL (CLI)

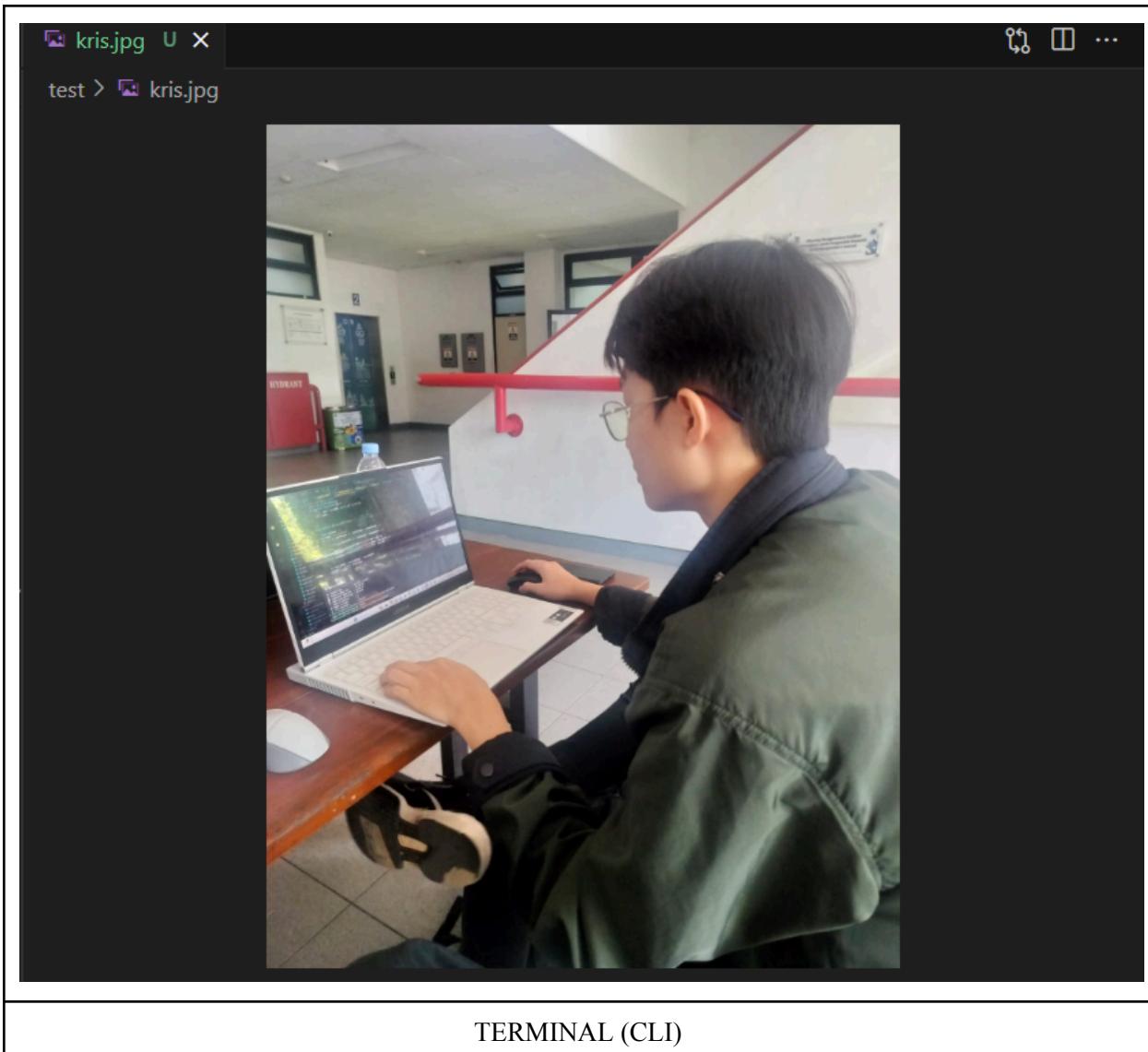
```
🛠️ Compressing Image...
✓ GIF berhasil degenerate ke: /home/crai/Tucil2_13523054_13523077/test/cutie_exp.gif
✓ Compression completed!

📊 Compression Summary
-----
⌚ Elapsed time      : 1.20813 seconds
💾 Original image size : 137.09 KB
🖼️ Compressed image size : 73.014 KB
📦 Compression percentage : 46.7401%
🌳 Depth of quadtree   : 6
🔢 Number of nodes     : 1137
```

3.4 Contoh dengan metode error Mean Absolute Deviation (MAD) 2

-INPUT:

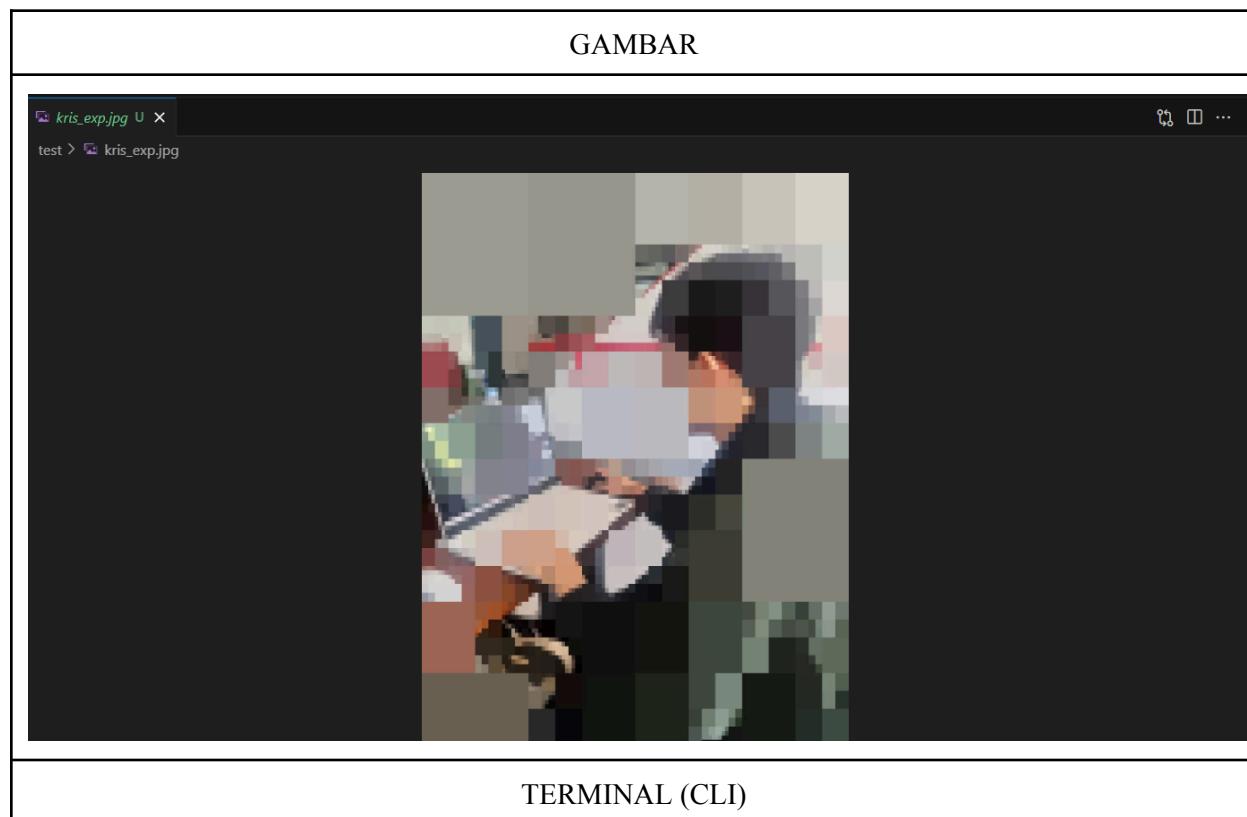
GAMBAR



TERMINAL (CLI)

```
📁 Input  
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/kris.jpg  
✓ Gambar berhasil dimuat!  
⚙️ Compression Settings  
Pilih metode error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (SSIM)  
Masukkan pilihan [1-5]: 2  
Masukkan target kompresi [0.0-1.0]: 0  
Masukkan ambang batas error: 25  
Masukkan ukuran blok minimum: 6  
📅 Export  
Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/kris_exp.jpg  
Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/kris_exp.gif
```

-OUTPUT:



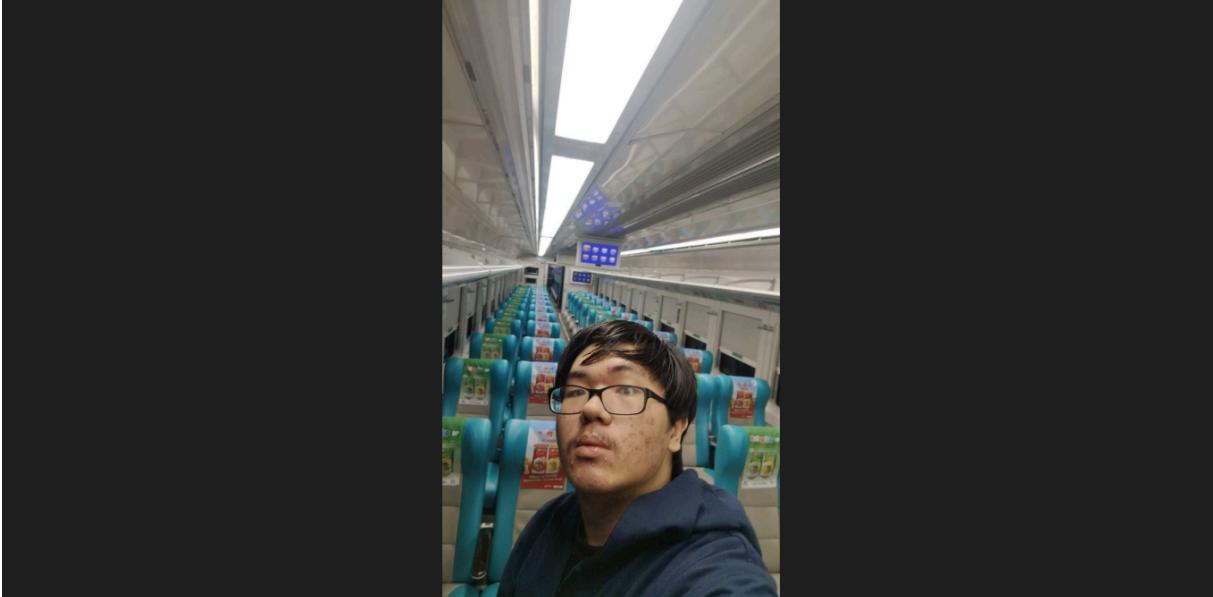
```
🛠️ Compressing Image...

✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/kris_exp.gif
✓ Compression completed!

📊 Compression Summary
-----
⌚ Elapsed time      : 1.31804 seconds
📷 Original image size : 130.663 KB
📝 Compressed image size : 77.212 KB
📦 Compression percentage : 40.9075%
🌳 Depth of quadtree    : 7
🕒 Number of nodes       : 2121
```

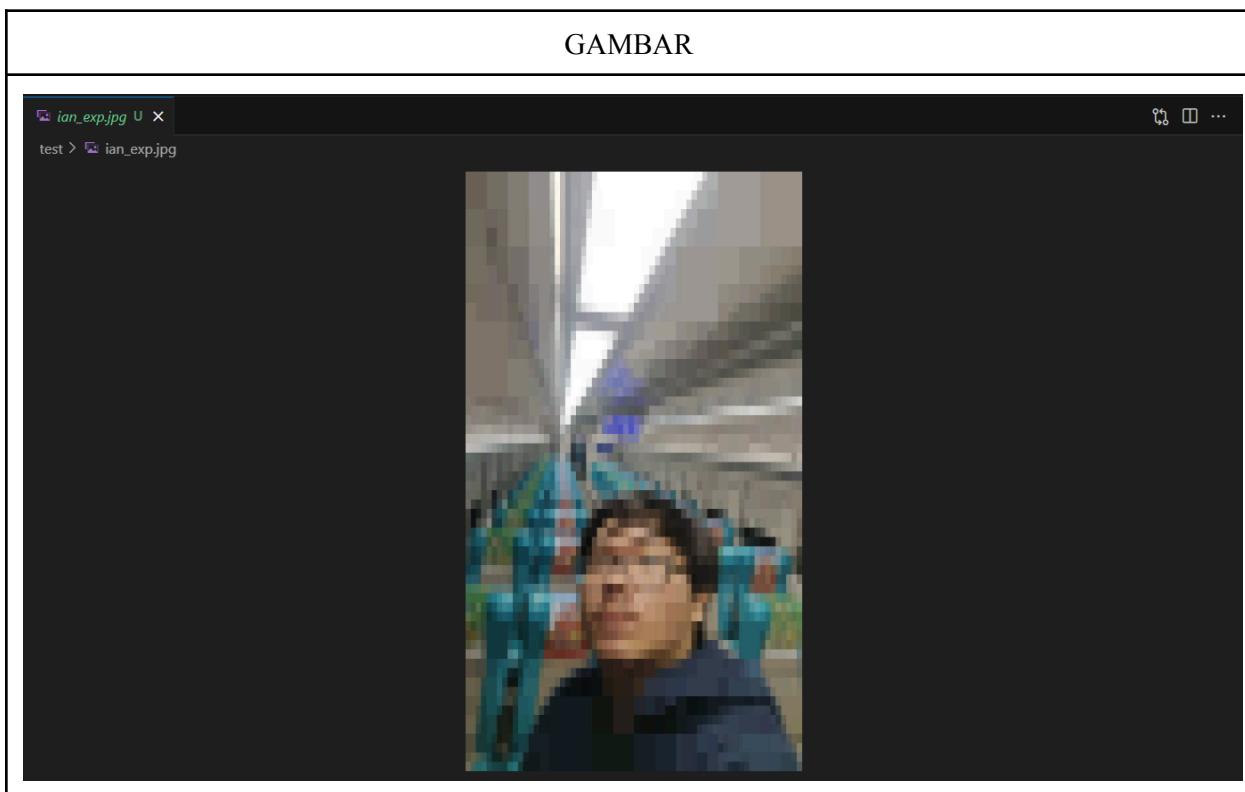
3.5 Contoh dengan metode error Max Pixel Difference

-INPUT:

GAMBAR	TERMINAL (CLI)
	

```
📁 Input  
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/ian.jpg  
✓ Gambar berhasil dimuat!  
⚙️ Compression Settings  
Pilih metode error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (SSIM)  
Masukkan pilihan [1-5]: 3  
Masukkan target kompresi [0.0-1.0]: 0  
Masukkan ambang batas error: 55  
Masukkan ukuran blok minimum: 10  
💾 Export  
Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/ian_exp.jpg  
Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/ian_exp.gif
```

-OUTPUT:



TERMINAL (CLI)

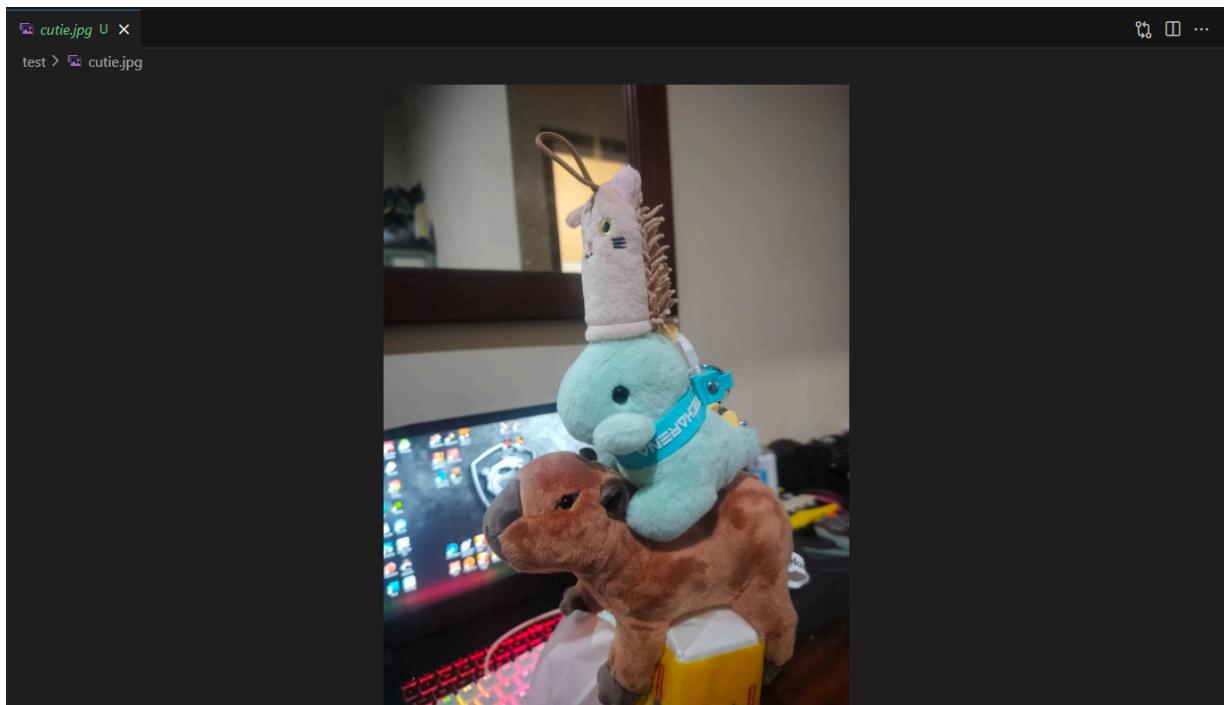
```
# Compressing Image...
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/ian_exp.gif
✓ Compression completed!

# Compression Summary
-----
⌚ Elapsed time      : 0.836961 seconds
🕒 Original image size : 87.895 KB
📈 Compressed image size : 52.845 KB
📦 Compression percentage : 39.8771%
🌳 Depth of quadtree    : 6
🌐 Number of nodes      : 3449
```

3.6 Contoh dengan metode error Max Pixel Difference 2

-INPUT:

GAMBAR



TERMINAL (CLI)

Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/cutie.jpg
 Gambar berhasil dimuat!

Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 3

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 125

Masukkan ukuran blok minimum: 6

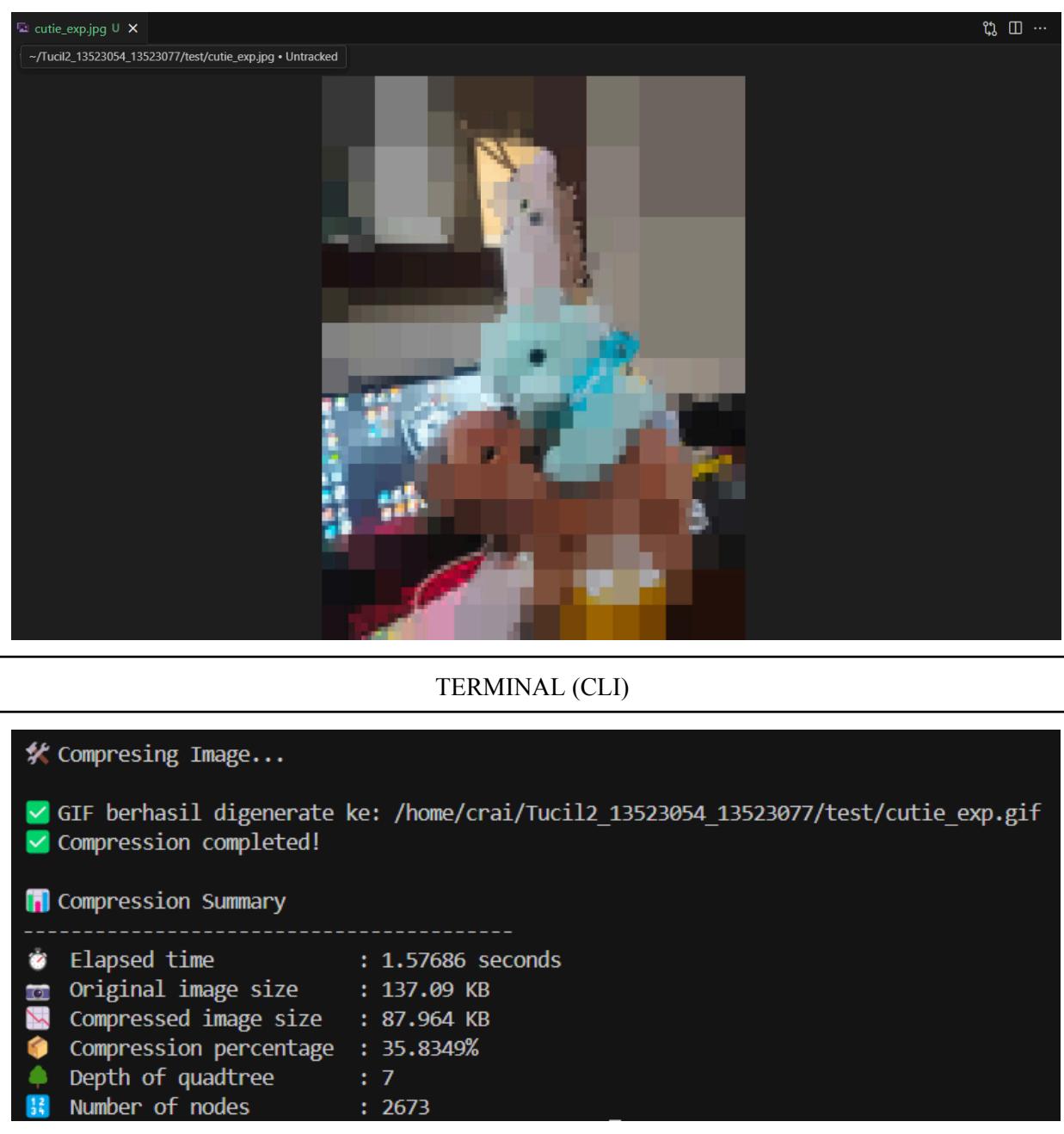
Export

Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cutie_exp.jpg
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cutie_exp.gif

Compressing Image...

-OUTPUT:

GAMBAR



terminal window showing the compression of an image named "cutie_exp.jpg" into a GIF file. The terminal output includes a success message and a detailed compression summary table.

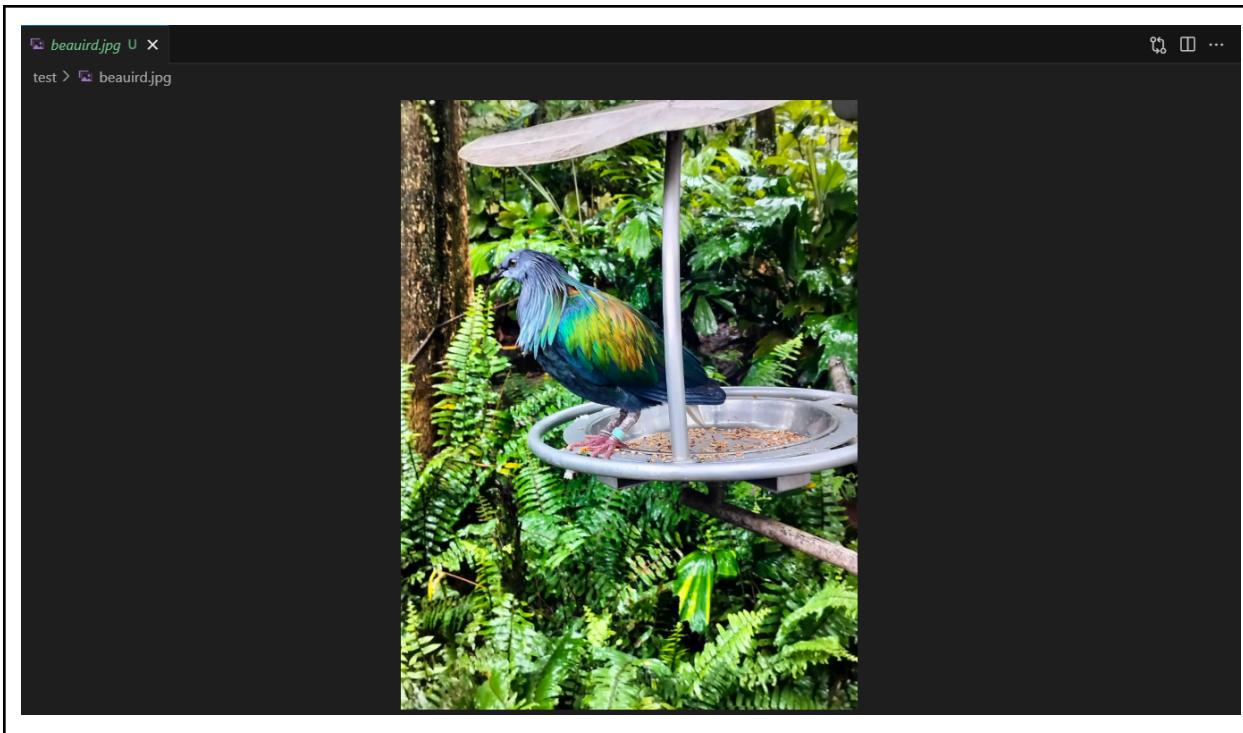
```
terminal window showing the compression of an image named "cutie_exp.jpg" into a GIF file. The terminal output includes a success message and a detailed compression summary table.
```

Elapsed time	: 1.57686 seconds
Original image size	: 137.09 KB
Compressed image size	: 87.964 KB
Compression percentage	: 35.8349%
Depth of quadtree	: 7
Number of nodes	: 2673

3.7 Contoh dengan metode error Entropy

-INPUT:

GAMBAR



TERMINAL (CLI)

📁 Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/beauird.jpg
 Gambar berhasil dimuat!

⚙️ Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 4

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 6

Masukkan ukuran blok minimum: 10

💾 Export

Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/beauird_exp.jpg

Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/beauird_exp.gif

-OUTPUT:

GAMBAR

The screenshot shows a terminal window with two tabs: 'beaurd_exp.jpg' (selected) and 'test'. The main area displays a highly compressed, low-resolution version of a photograph of a garden or patio area with a white fence and greenery. Below the image, the terminal window has a title bar labeled 'TERMINAL (CLI)'.

```
🛠️ Compressing Image...
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/beaurd_exp.gif
✓ Compression completed!

📊 Compression Summary
-----
⌚ Elapsed time      : 1.22479 seconds
🕒 Original image size : 247.894 KB
📝 Compressed image size : 109.163 KB
📦 Compression percentage : 55.9638%
🌳 Depth of quadtree   : 6
💾 Number of nodes     : 4861
```

3.8 Contoh dengan metode error Entropy 2

-INPUT:

GAMBAR

The screenshot shows a terminal window with a dark theme. At the top, there is a file viewer showing a photo of a man with glasses sitting in a train seat. Below the viewer, the terminal title is "TERMINAL (CLI)". The terminal content is as follows:

```
📁 Input
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/ian.jpg
 Gambar berhasil dimuat!

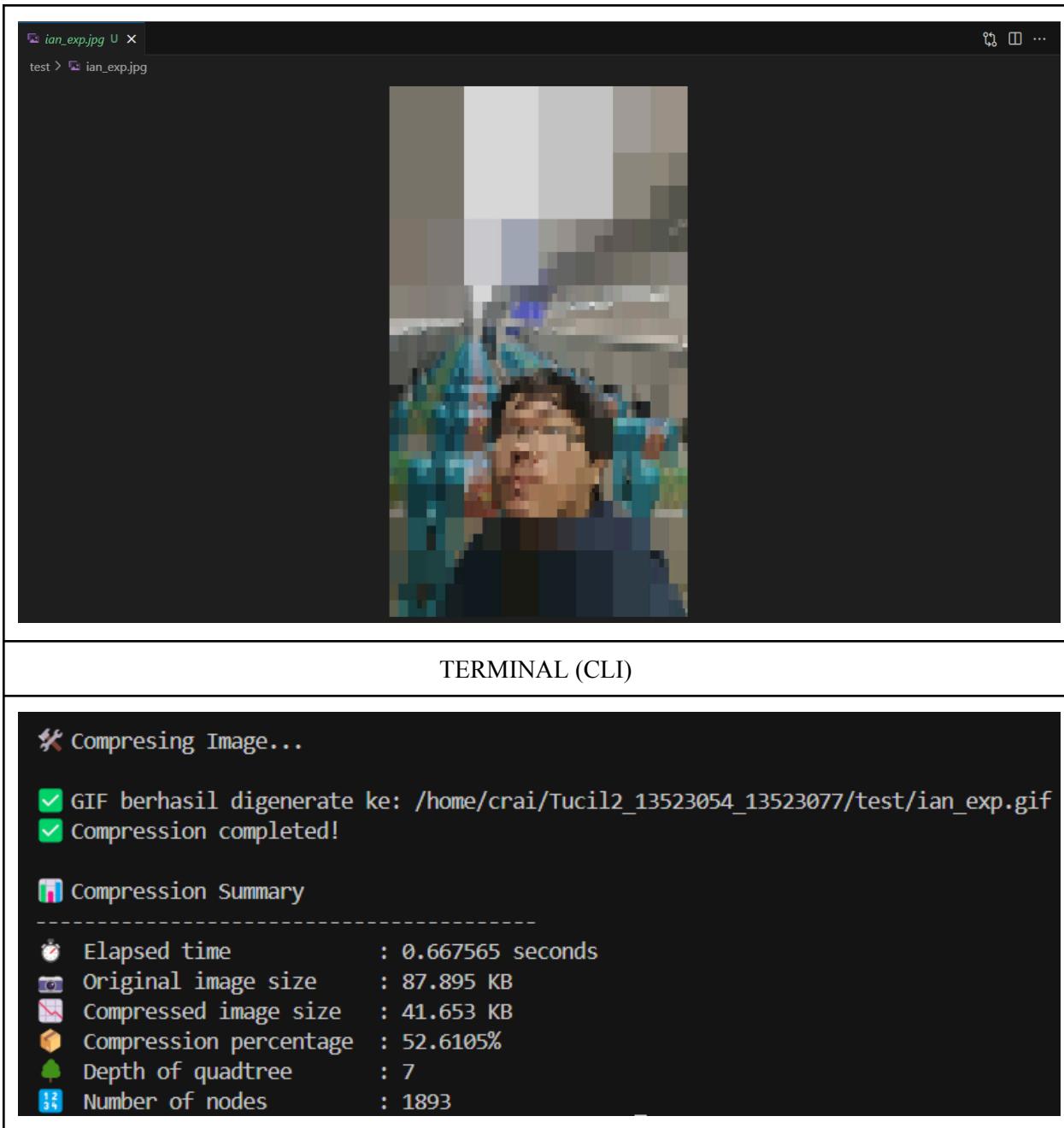
⚙️ Compression Settings
Pilih metode error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
Masukkan pilihan [1-5]: 4

Masukkan target kompresi [0.0-1.0]: 0
Masukkan ambang batas error: 6
Masukkan ukuran blok minimum: 6

📅 Export
Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/ian_exp.jpg
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/ian_exp.gif
```

-OUTPUT:

GAMBAR



3.9 [Bonus] Contoh dengan metode error SSIM

INPUT:

GAMBAR

🖼 patung.jpg ✎ ✕

🔍 ⌂ ⋮

test > 🖼 patung.jpg



TERMINAL (CLI)

Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/patung.jpg
 Gambar berhasil dimuat!

Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 5

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 0.08

Masukkan ukuran blok minimum: 6

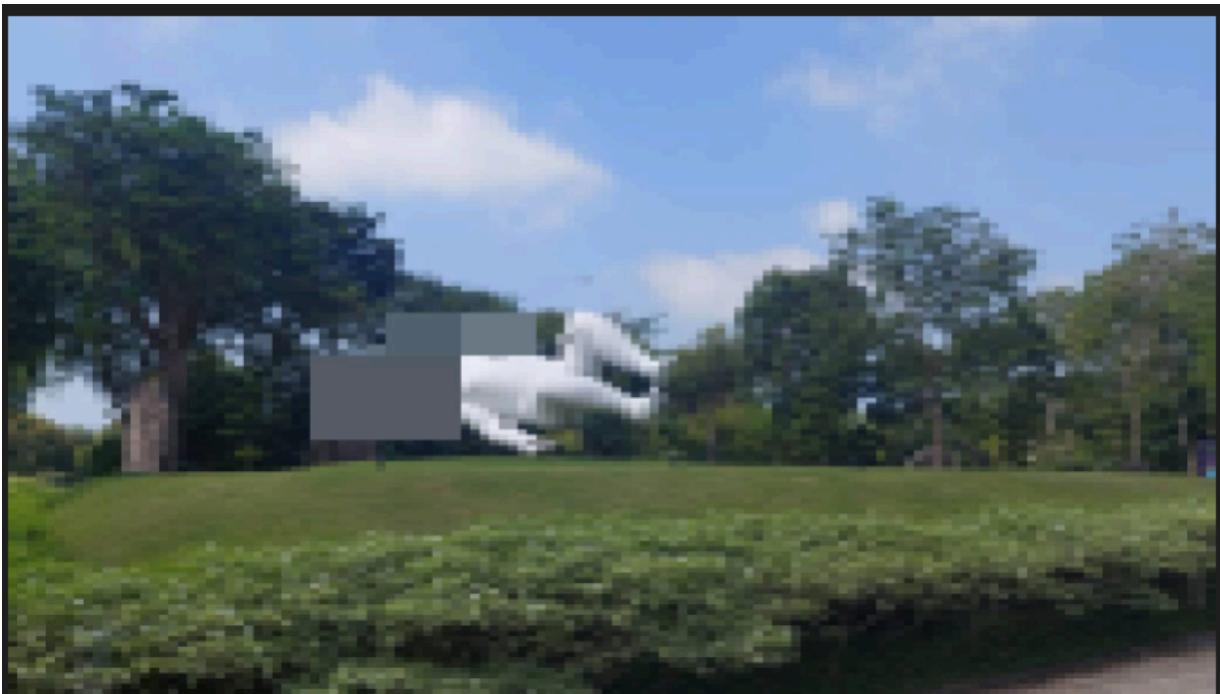
Export

Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/patung_exp.jpg

Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/patung_exp.gif

OUTPUT:

GAMBAR



TERMINAL (CLI)

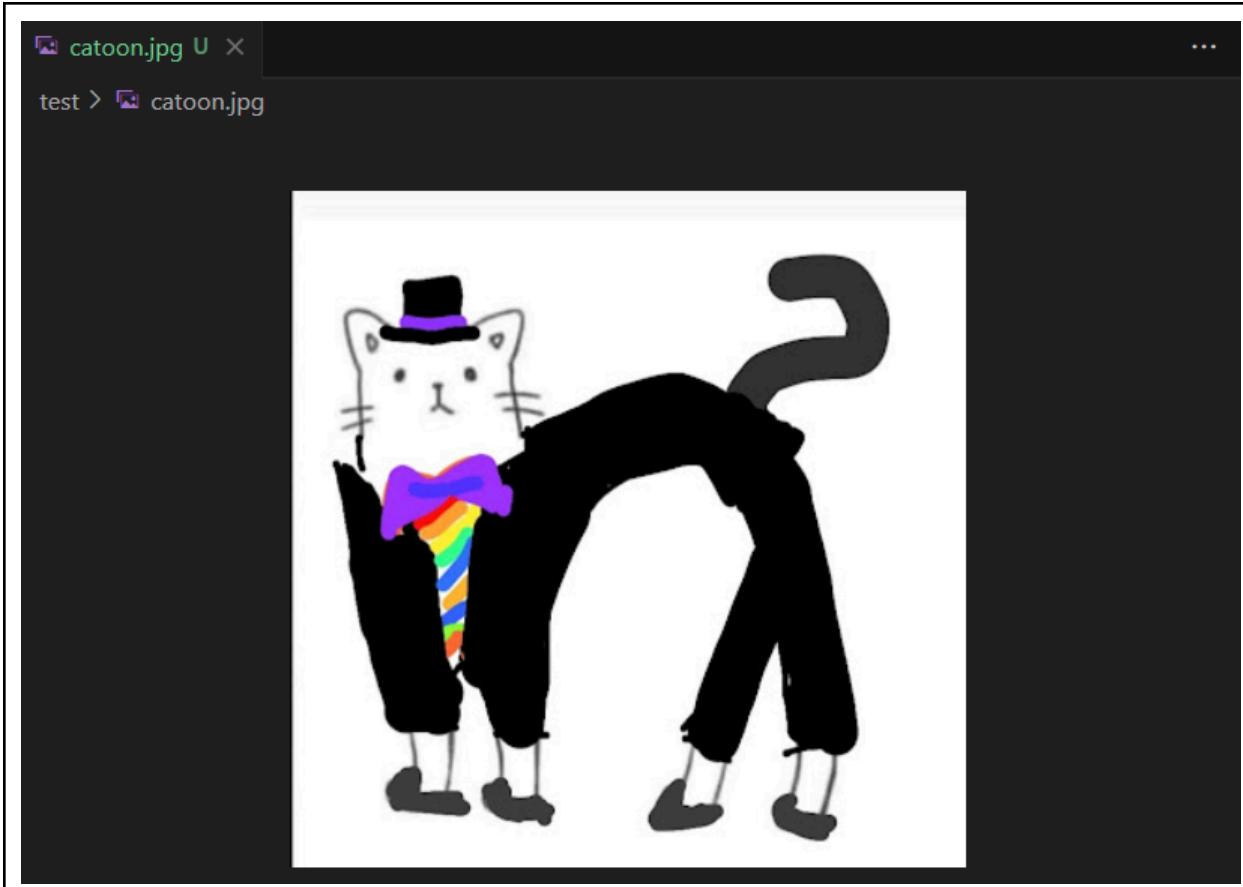
```
🛠️ Compressing Image...
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/patung_exp.gif
✓ Compression completed!

📊 Compression Summary
-----
⌚ Elapsed time      : 1.63834 seconds
🕒 Original image size : 221.508 KB
📝 Compressed image size : 105.265 KB
📦 Compression percentage : 52.478%
🌳 Depth of quadtree   : 7
🔢 Number of nodes     : 21313
```

3.10 [Bonus] Contoh dengan metode error SSIM 2

INPUT:

GAMBAR



TERMINAL (CLI)

📁 Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/cat.jpg
 Gambar berhasil dimuat!

⚙️ Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 5

Masukkan target kompresi [0.0-1.0]: 0

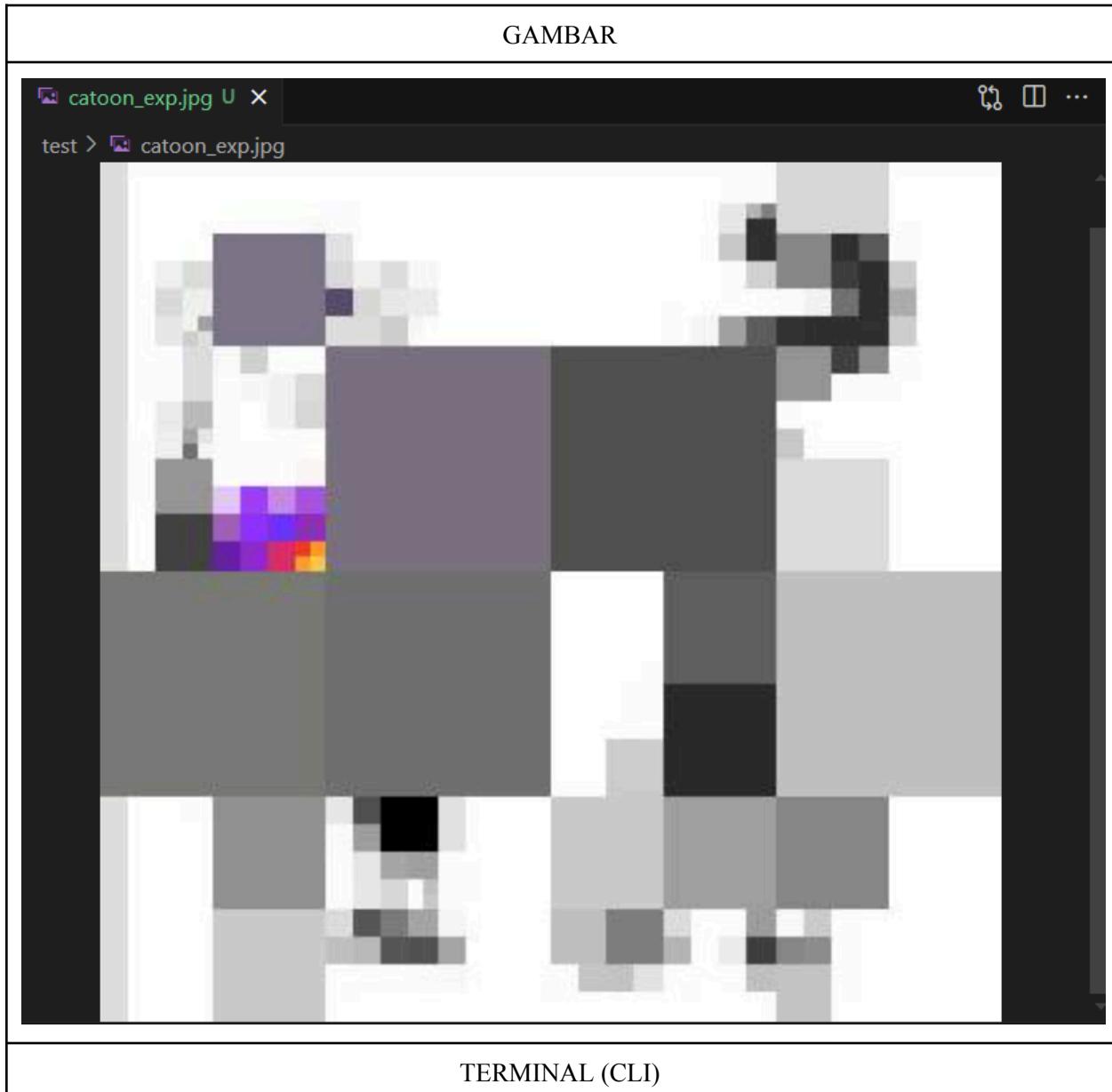
Masukkan ambang batas error: 0.05

Masukkan ukuran blok minimum: 6

💾 Export

Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cat_exp.jpg
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cat_exp.gif

OUTPUT:



```
🛠️ Compressing Image...

✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/catoot_exp.gif
✓ Compression completed!

📊 Compression Summary
-----  
⌚ Elapsed time : 0.0758606 seconds
📷 Original image size : 20.817 KB
📝 Compressed image size : 9.889 KB
📦 Compression percentage : 52.4956%
🌳 Depth of quadtree : 6
🕒 Number of nodes : 837
```

3.11 [Bonus] Contoh hasil gif

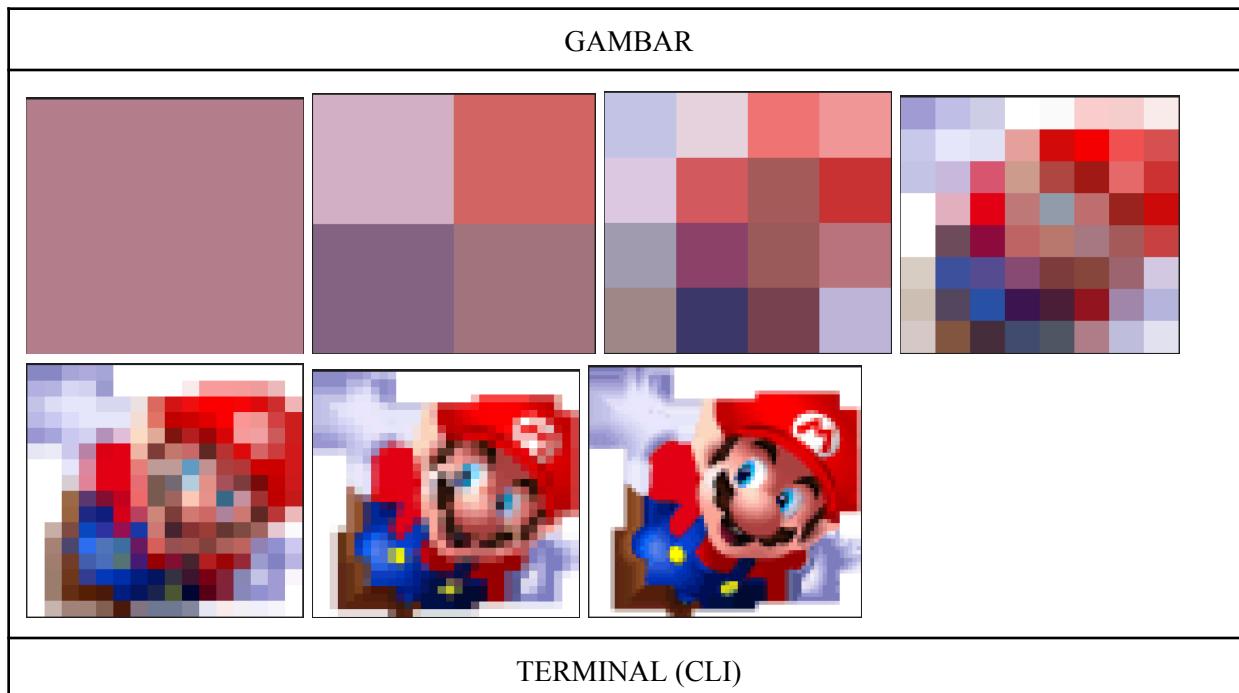
INPUT:

GAMBAR



```
📁 Input  
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/mario.png  
✔ Gambar berhasil dimuat!  
⚙️ Compression Settings  
Pilih metode error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (SSIM)  
Masukkan pilihan [1-5]: 4  
Masukkan target kompresi [0.0-1.0]: 0.7  
Masukkan ukuran blok minimum: 10  
💾 Export  
Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/mario_exp.png  
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/mario_exp.gif
```

OUTPUT:



```
✓ Gambar berhasil diekspor ke: /home/crai/Tucil2_13523054_13523077/test/mario_exp.png
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/mario_exp.gif
✓ Compression completed!

 Compression Summary
-----
⌚ Elapsed time : 231.475 seconds
💾 Original image size : 513.59 KB
📈 Compressed image size : 91.902 KB
📦 Compression percentage : 82.106%
🌳 Depth of quadtree : 6
🔢 Number of nodes : 5461
⚠️ Error threshold : 0.0463058
```

3.12 [Bonus] Contoh hasil gif 2

INPUT:



TERMINAL (CLI)

📁 Input

Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/cat0on.jpg
✓ Gambar berhasil dimuat!

⚙️ Compression Settings

Pilih metode error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Masukkan pilihan [1-5]: 5

Masukkan target kompresi [0.0-1.0]: 0

Masukkan ambang batas error: 0.05

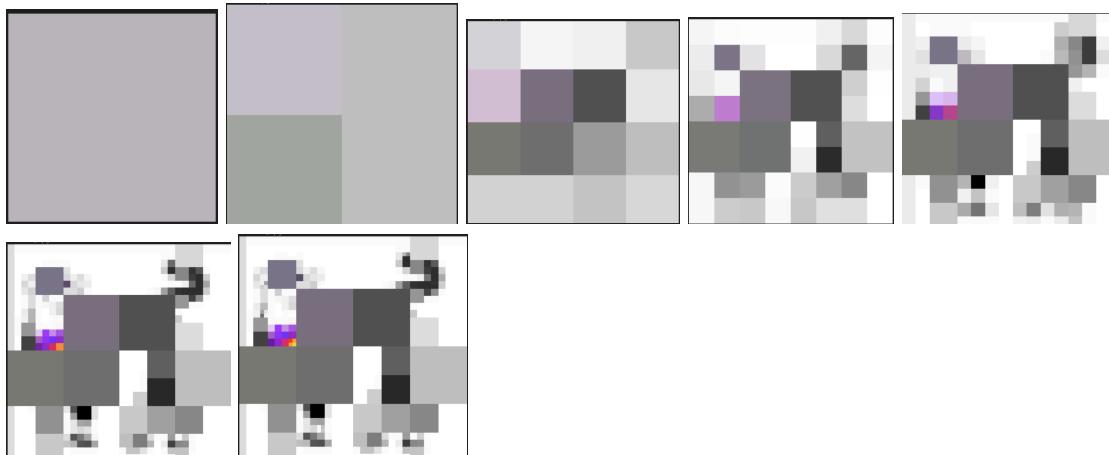
Masukkan ukuran blok minimum: 6

💾 Export

Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cat0on_exp.jpg
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/cat0on_exp.gif

OUTPUT:

GAMBAR



TERMINAL (CLI)

```
🛠️ Compressing Image...

✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/catoot_exp.gif
✓ Compression completed!

📊 Compression Summary
-----  
⌚ Elapsed time : 0.0758606 seconds
📷 Original image size : 20.817 KB
📝 Compressed image size : 9.889 KB
📦 Compression percentage : 52.4956%
🌳 Depth of quadtree : 6
🕒 Number of nodes : 837
```

3.13 [Bonus] Contoh target compression

INPUT:

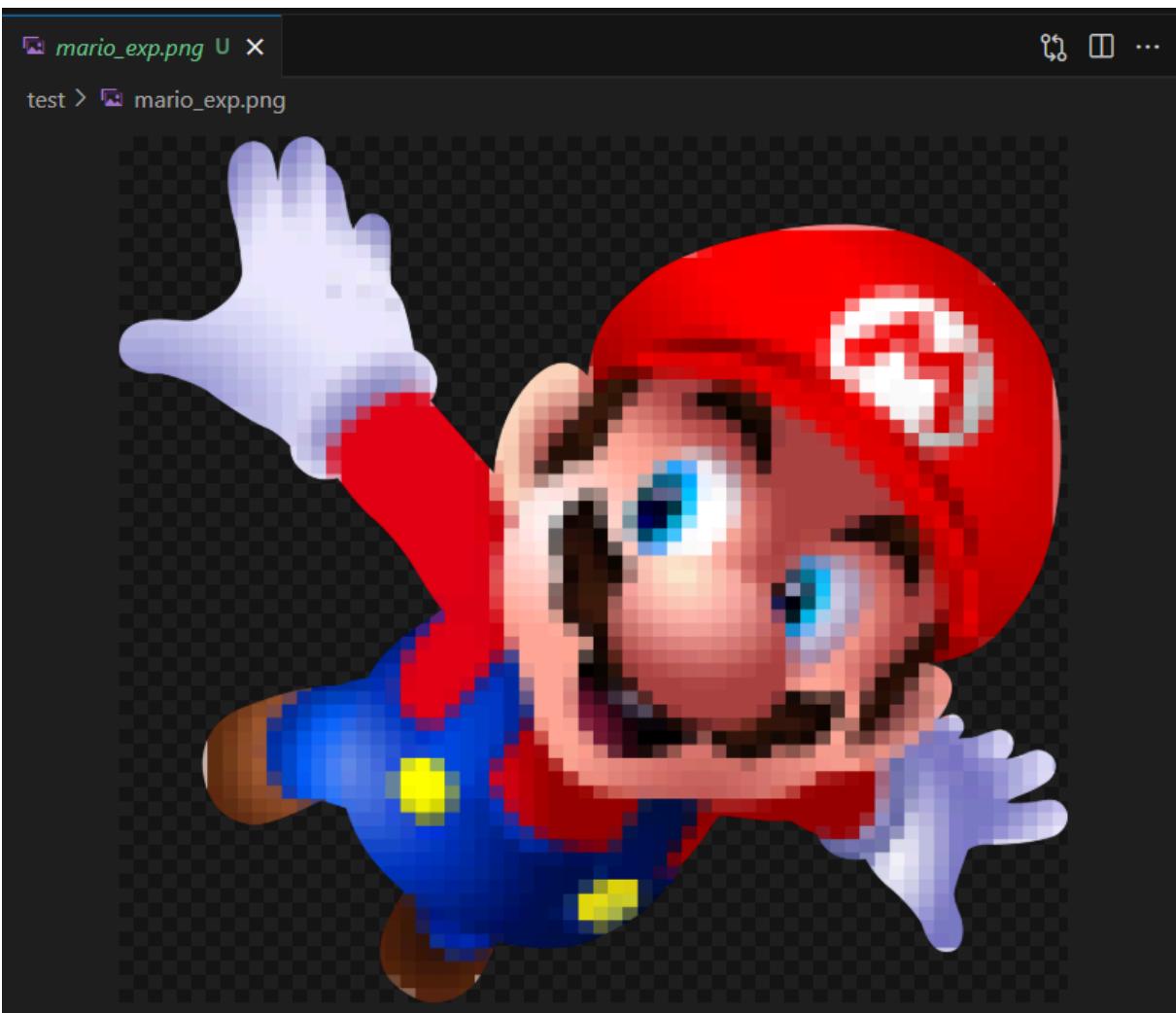
GAMBAR



```
📁 Input  
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/mario.png  
✅ Gambar berhasil dimuat!  
⚙️ Compression Settings  
Pilih metode error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (SSIM)  
Masukkan pilihan [1-5]: 4  
Masukkan target kompresi [0.0-1.0]: 0.7  
Masukkan ukuran blok minimum: 10  
💾 Export  
Masukkan alamat gambar yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/mario_exp.png  
Masukkan alamat GIF yang akan dieksport: /home/crai/Tucil2_13523054_13523077/test/mario_exp.gif
```

OUTPUT:

GAMBAR



TERMINAL (CLI)

```
report...  
✓ Gambar berhasil diekspor ke: /home/crai/Tucil2_13523054_13523077/test/mario_exp.png  
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/mario_exp.gif  
✓ Compression completed!  
  
Compression Summary  
-----  
⌚ Elapsed time : 231.475 seconds  
💽 Original image size : 513.59 KB  
☒ Compressed image size : 91.902 KB  
📦 Compression percentage : 82.106%  
🌳 Depth of quadtree : 6  
💾 Number of nodes : 5461  
⚠ Error threshold : 0.0463058
```

3.14 [Bonus] Contoh target compression

INPUT:

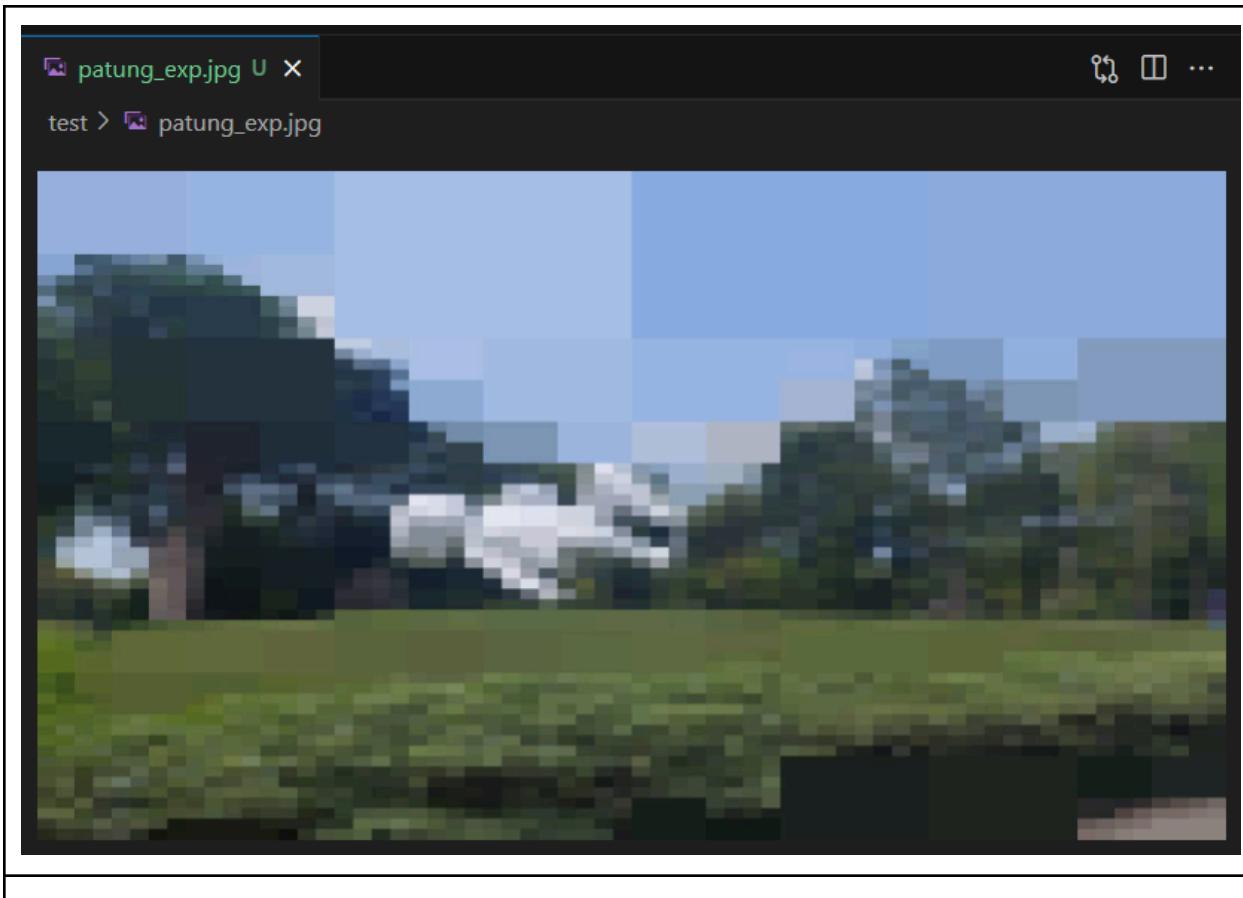
GAMBAR

TERMINAL (CLI)

```
📁 Input  
Masukkan alamat gambar yang akan dikompresi: /home/crai/Tucil2_13523054_13523077/test/patung.jpg  
✓ Gambar berhasil dimuat!  
⚙️ Compression Settings  
Pilih metode error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (SSIM)  
Masukkan pilihan [1-5]: 4  
Masukkan target kompresi [0.0-1.0]: 0.7  
Masukkan ukuran blok minimum: 10  
📅 Export  
Masukkan alamat gambar yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/patung_exp.jpg  
Masukkan alamat GIF yang akan diekspor: /home/crai/Tucil2_13523054_13523077/test/patung_exp.gif
```

OUTPUT:

GAMBAR



TERMINAL (CLI)

```
✓ Gambar berhasil diekspor ke: /home/crai/Tucil2_13523054_13523077/test/patung_exp.jpg
✓ GIF berhasil digenerate ke: /home/crai/Tucil2_13523054_13523077/test/patung_exp.gif
✓ Compression completed!

Compression Summary
-----
⌚ Elapsed time      : 40.8175 seconds
🕒 Original image size : 221.508 KB
📝 Compressed image size : 66.472 KB
📦 Compression percentage : 69.9912%
🌳 Depth of quadtree   : 6
💡 Number of nodes     : 2741
⚠️ Error threshold    : 5.76203
```

BAB IV

ANALISIS

4.1 Kompleksitas Algoritma Divide and Conquer

Misalkan n adalah jumlah total piksel dalam gambar yang terdiri dari r baris dan c kolom, sehingga $n = r \times c$. Kompresi terhadap gambar tersebut dilakukan menggunakan algoritma Quadtree berbasis metode *Divide and Conquer*, dengan parameter ambang batas homogenitas (*threshold*) sebesar t , dan ukuran blok minimum k . Untuk menentukan apakah suatu blok akan dibagi lebih lanjut (rekursif), didefinisikan sebuah fungsi f , yang bernilai:

$$f(x, y, err) = \begin{cases} 0, & \frac{x}{2} < k \vee \frac{y}{2} < k \vee err < t \\ 1, & \text{otherwise} \end{cases}$$

Fungsi ini mengembalikan nilai 1 jika suatu blok masih perlu dibagi (*divide*), dan 0 jika tidak (dilakukan normalisasi warna RGB). Berdasarkan fungsi tersebut, kompleksitas waktu dari algoritma divide and conquer pada kompresi gambar ini dapat dimodelkan dalam bentuk rekurens sebagai berikut:

$$T(n) = \begin{cases} n, & f(r, c, err) = 0 \\ 4 \cdot T(n/4) + n, & f(r, c, err) = 1 \end{cases}$$

Rekurens yang diperoleh dapat dianalisis menggunakan Teorema Master, yaitu sebuah metode yang umum digunakan untuk menentukan kompleksitas waktu dari algoritma rekursif, khususnya algoritma Divide and Conquer. Teorema Master menyelesaikan rekurens berbentuk umum:

$$T(n) = a \cdot T(n/b) + cn^d$$

Teorema ini terbagi menjadi tiga kasus utama:

1. Jika $a < b^d$, maka $T(n) = O(n^d)$
2. Jika $a = b^d$, maka $T(n) = O(n^d \log n)$
3. Jika $a > b^d$, maka $T(n) = O(n^{\log_b a})$

Maka, pada kasus kompleksitas algoritma *divide and conquer* sebelumnya, diperoleh $a = 4$, $b = 4$, $c = 1$, $d = 1$. Kasus yang terpenuhi adalah kasus ke-dua, yaitu $a = 4 = b^d = 4^1$, sehingga diperoleh:

$$T(n) = O(n \log n)$$

4.2 Analisis Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree

Kompleksitas algoritma menggambarkan bahwa algoritma dapat dikatakan cukup efisien dalam mengkompresi gambar dengan ukuran yang cukup besar. Namun hal yang akan mempengaruhi efisiensi waktu bukan hanya ukuran gambar, tetapi juga *threshold* dan ukuran blok minimum. Semakin tinggi *threshold* atau semakin kecil ukuran blok minimum, maka semakin dalam pohon Quadtree terbentuk, yang berarti jumlah node meningkat dan waktu proses juga ikut bertambah. Sebaliknya, *threshold* kecil dan ukuran blok minimum besar akan menghentikan proses *divide* dengan lebih cepat, yang membuat waktu kompresi lebih singkat.

BAB V

BONUS

5.1 Target Persentase Kompresi

Memungkinkan pengguna menentukan target persentase kompresi berupa nilai floating point (contoh: $1.0 = 100\%$). Jika pengguna memberikan nilai 0, mode ini akan dinonaktifkan. Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai threshold secara otomatis untuk mencapai target persentase kompresi yang ditentukan. Penyesuaian threshold ini dilakukan secara dinamis, memberikan fleksibilitas dalam proses kompresi untuk memenuhi target efisiensi sambil mempertahankan kualitas gambar.

Implementasi bonus ini menggunakan algoritma binary search, yaitu algoritma dengan pendekatan decrease and conquer, untuk mencari nilai threshold paling optimal yang mendekati target persentase kompresi yang diminta. Langkah-langkah algoritma yang dilakukan adalah sebagai berikut:

1. Inisialisasi Batas

Lakukan inisialisasi batas atas dan batas bawah nilai *threshold* yang mungkin, di mana batas atas *threshold* adalah nilai *error* dari blok gambar penuh, dan batas bawah adalah nol.

2. Pencarian Threshold dengan Binary Search

Lakukan langkah iteratif, di mana pada setiap iterasi:

- Pilih nilai tengah, yaitu $(l + r)/2$ sebagai threshold sementara, di mana l adalah batas bawah dan r adalah batas atas
- Lakukan proses *divide and conquer* untuk melakukan kompresi gambar menggunakan Quadtree dengan threshold sementara yang telah ditentukan
- Gambar hasil kompresi diekspor, lalu ukurannya dibandingkan dengan target persentase kompresi

3. Pengambilan Keputusan

- Jika ukuran file saat ini masih terlalu besar (lebih besar dari target yang diinginkan), maka *threshold* diperbesar (lakukan kompresi yang lebih signifikan), yaitu dengan memindahkan batas bawah menjadi *threshold* sementara $l = currentThreshold$
- Jika ukuran file saat ini masih terlalu kecil, maka *threshold* diperkecil (kurangi signifikansi kompresi), yaitu dengan memindahkan batas atas menjadi *threshold* sementara $r = currentThreshold$
- Proses berhenti ketika persentase kompresi sudah cukup dekat dengan target, yaitu selisih < 0.001 atau interval batas atas dan batas bawah terlalu dekat, yaitu $r - l > 0.001$

5.2 Structural Similarity Index (SSIM)

SSIM adalah sebuah teknik untuk membandingkan kemiripan dari dua gambar. Metode ini lebih berfokus kepada cara manusia memproses gambar visual. Metode ini dapat digunakan untuk mengecek kelayakan gambar ini untuk dipecah menjadi sub-block atau tidak perlu.

Pada projek ini, SSIM akan digunakan untuk membandingkan block sebelum diubah, dengan block setelah diubah(*compressed*). Pada umumnya, apabila nilai SSIM tinggi (mendekati 1) dan memenuhi batas toleransi, maka kedua block memiliki kemiripan, sehingga tidak perlu dipisah.

Secara umum, rumus SSIM adalah:

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

Dengan:

- μ_x, μ_y : rata-rata dari blok x dan y
- σ_x^2, σ_y^2 : varians dari masing-masing blok
- σ_{xy} : kovarians antara x dan y
- C_1, C_2 : konstanta kecil untuk menstabilkan pembagian jika penyebut mendekati nol

Namun, dalam program ini, kompresi gambar menggunakan rata-rata warna dari blok, sehingga rumus di atas dapat disederhanakan karena:

- $\mu_y = \mu_x$
- $\sigma_y^2 = 0$ (karena semua nilainya sama)
- $\sigma_{xy} = 0$ (karena tidak ada variasi yang bisa dikorelasikan)

Sehingga, kita dapat menarik kesimpulan bahwa rumus diatas dapat disederhanakan menjadi:

$$SSIM(x, y) = \frac{C_2}{(\sigma_x^2 + C_2)}$$

Rumus diatas kemudian digunakan untuk mencari nilai SSIM dari setiap channel (R,G,B) pada sebuah block, yang kemudian akan dikalikan dengan sebuah konstanta. Jumlah dari setiap channel kemudian akan menjadi nilai error, yang akan dibandingkan dengan threshold.

5.3 GIF

Sebagai bagian dari implementasi bonus, program juga dilengkapi dengan fitur untuk menghasilkan GIF animasi yang memvisualisasikan proses kompresi gambar secara bertahap menggunakan Quadtree. Fitur ini memungkinkan kita untuk melihat bagaimana blok-blok gambar disederhanakan/dinormalisasi secara bertahap berdasarkan kedalaman pohon Quadtree. Implementasi bonus ini menggunakan library gif.h yang dibuat oleh Charlie Tangora. Langkah-langkah algoritma yang dilakukan pada implementasi adalah sebagai berikut:

1. Bangun Quadtree menggunakan algoritma *divide and conquer*
Sebelum membentuk GIF, struktur quadtree harus sudah terbentuk sebelumnya dari proses kompresi gambar.
2. Inisialisasi Buffer Gambar

Alokasikan buffer yang akan digunakan untuk menyimpan gambar yang nantinya akan dieksport sebagai *frame-frame* dari GIF yang ingin dibentuk.

3. Simpan Node tiap Depth

Cari dan simpan seluruh node pohon Quadtree yang telah terbentuk dalam `nodesAtDepth`, yaitu larik yang berisi larik node Quadtree berdasarkan kedalamannya (depth).

4. Proses Iteratif untuk Tiap Kedalaman Quadtree

Lakukan iterasi untuk setiap depth pohon Quadtree, dimulai dari depth ke-0. Untuk tiap kedalaman:

- Dapatkan node-node pada kedalaman saat ini dari larik `nodesAtDepth` yang telah dibentuk sebelumnya.
- Lakukan normalisasi warna pada semua node (ingat bahwa sebuah node pada dasarnya adalah suatu sub-blok dari gambar) pada kedalaman saat ini dengan nilai rata-ratanya.
- Hasilnya langsung dituliskan ke buffer gambar yang telah diinisialisasi sebelumnya.
- Setelah semua node pada kedalaman saat ini telah dinormalisasi dan ditulis ke buffer gambar, ekspor buffer gambar sebagai satu *frame* baru pada GIF yang akan dibuat.

LAMPIRAN

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	✓	

Repository : https://github.com/mimiCrai/Tucil2_13523054_13523077

DAFTAR PUSTAKA

1. C. Tangora, gif-h: Single-Header C library for creating GIF images. [Online]. Available: <https://github.com/charlietangora/gif-h>
2. S. Barrett, stb_image_write.h: Public domain C library for writing images. [Online]. Available: https://github.com/nothings/stb/blob/master/stb_image_write.h
3. S. Barrett, stb_image.h: Public domain C library for loading images. [Online]. Available: https://github.com/nothings/stb/blob/master/stb_image.h
4. R. Munir, *Algoritma Divide and Conquer (2025) Bagian 1*. Program Studi Teknik Informatika, STEI ITB. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)
5. R. Munir, *Algoritma Divide and Conquer (2025) Bagian 2*. Program Studi Teknik Informatika, STEI ITB. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf)
6. R. Munir, *Algoritma Decrease and Conquer (2025) Bagian 1*. Program Studi Teknik Informatika, STEI ITB. [Online]. Available: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/11-Algoritma-Decrease-and-Conquer-2025-Bagian1.pdf>